Clean Room Monitoring System for Pharmaceutical Companies

Anastasia Mourka

Kongens Lyngby 2013
IMM-MSc-2013-94

Technical University of Denmark
Informatics and Mathematical
Modelling, DTU Compute
Building 321, DK-2800 Kongens Lyngby,
Denmark Phone +45 45253351, Fax +45
45882673 reception@imm.dtu.dk
www.imm.dtu.dk IMM-MSc-2013-94

Summary

Many pharmaceutical companies, nowadays, are using Facility Monitoring Systems to have an overview of the environmental conditions inside clean rooms, where the pharmaceuticals production takes place. Their conditions are measured with sensors and they are transferred through an FMS system, such as SCADA, to a log server, where they are saved. This network architecture is wired, soft-real time and its components are coupled. This doesn't allow the system to scale easily, because we need to perform validation to all the components of the system, new and old ones, to ensure they comply with FDA regulations.
The goal of the thesis is to propose a solution to the above problem, which would allow the system to scale easily, without requiring extensive re-validation of the entire system. For this purpose, we chose the decoupled architecture of publish-subscribe paradigm, where the sensors publish their measurements in an event bus, and the server log subscribes to them. Instead of an FMS system, we suggest a decoupled Alarm System, which also subscribes to the sensors' data and publishes an alarm event, when the data exceed a threshold. After transferring the data to the server, we apply a security scheme to them, which saves them in an encrypted way and detects when an attacker tampers with the log files. This way we ensure the safety of the data, after being received at the server.

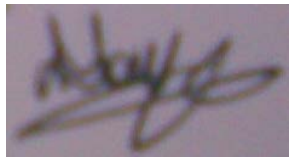
Preface

This thesis was prepared at the department of Informatics and Mathematical Modelling or DTU Compute at the Technical University of Denmark in fulfillment of the requirements for acquiring an M.Sc. in Computer Science and Engineering. The work on this thesis was conducted under the supervision of Associate Professor in DTU Christian Damsgaard Jensen and of Automation Specialist in NNE Pharmaplan Stig Allan Poulsen

The thesis deals with proposing a solution for pharmaceutical companies, which want to measure the environmental conditions in clean rooms, but the systems they are using are coupled and, thus, difficult and costly to scale, because their systems need to comply with the FDA regulations. The solution we describe considers a decoupled system architecture to facilitate the message exchange between the sensors, the alarm system and the log server. Furthermore, we apply a security scheme in the log server, so that the data it receives are saved in an encrypted way, easy to detect changes on them.

The thesis consists of an introduction presenting the problem, the proposed solutions and how we plan to solve it (Chapter 1); the state-of-the-art to the current message exchange methods and secure logging schemes (Chapter 2); an analysis of the FDA regulations, user requirements and sensor specifications (Chapter 3); a problem analysis which led us to choosing the appropriate solutions for it (Chapter 4); a description of the system architecture of our solution (Chapter 5); a description of the implementation of the prototype we built to demonstrate the functionality of our proposed solution (Chapter 6); the evaluation of the prototype performance and security efficiency (Chapter 7); the conclusion we extracted (Chapter 8); and possible future developments for our prototype (Chapter 9).

Lyngby, 09-August-2013



Anastasia
Mourka

Acknowledgements

I would like to thank the people who helped in completing this master thesis; my supervisors, Associate Professor in DTU, Christian Damsgaard Jensen and Automation Specialist in NNE Pharmaplan Stig Allan Poulsen for their continuous feedback regarding the overall guidance directions on the thesis and their advice for the implementation challenges; my family and friends for their support and understanding.

Table of Contents

Contents

CHAPTER 1

Introduction

Clean rooms are rooms where products highly sensitive to environmental conditions are produced and, therefore, these conditions need to be monitored. Clean rooms are used extensively by pharmaceutical companies for the production of medical products which should be very accurate and controlled. For this purpose the clean rooms contain sensors which measure the environmental values that are of interest for each production. Usually these are Temperature, Pressure Difference (between the inside and the outside of the room), humidity and number of particles. When an undesirable measurement appears then an alarm or a warning is raised, depending on how much the value change can affect the drug production. The measurements as well as the alarms are logged in a database for future reference.
The architecture used by many pharmaceutical companies nowadays includes the sensors connected to PLCs and through them sending their measurements to the monitoring system, usually a SCADA system. This system evaluates the values and raises the alarms if necessary. Furthermore, it sends the final data to the database to be saved. The systems used normally are wired, time-triggered, soft-real time systems and some of them involve a lot of additional costs and paper work when the system needs to scale. More specifically, the transferring and logging procedures of the sensor data should comply with the FDA regulations suitable for pharmaceutical companies producing drugs and medical devices. Therefore, extending the current network with adding new sensors includes evaluation/validation in terms of compliance with the FDA regulations. This evaluation applies not only to the new sensor, but also to the SCADA system as it part of the data transferring. This procedure takes a lot of time and paper work and is costly.

The solution we propose in this thesis is a decoupled system network that doesn't include the monitoring SCADA system between the sensor – server communication and therefore its FDA validation. A decoupled network has the advantage that can be easily extended with new sensors, without affecting the rest of the system architecture and functionality. The new

network will also avoid the SCADA interference in the data transferring and therefore its validation in case the system extends.
So, we assume the existence of a distributed system of wired sensors in a clean room area. We assume that our sensors collect the data mentioned previously and that they contain a clock to timestamp the data when collected. The alarm and the warning (flashlight) are also time stamped. We wish to examine the applicability of the publish/subscribe paradigm in order to develop a decoupled event-based system, which is easier to extend, than the one in use now.
The sensors would publish their measurements to an event bus, which would route them to their final destination. There is a log server and an alarm system subscribed to events of the event bus and therefore receiving the ones they are interested in. The alarm system subscribes to all the events and examines whether any measurement values above the permitted limits exist, but he doesn't store anything. In case of exceeding values, he publishes back to the event bus an alarm event. Both sensor measurements and alarm events would be logged in the log server, which would subscribe to both types of events. The logging should be done in a secure way so that no one can tamper later with the log file undetected.

To validate the proposed solution in this thesis we have gone through the FDA regulations to see if and how they would comply with the publish-subscribe paradigm. We also investigated the possible ways to implement the publish-subscribe paradigm and chose the one that would best fit our solution functionality. We, then, investigated the possible attacks in such kind of logging system and chose the security scheme that would prevent those attacks so that we can provide in the end a secure, tamper-free log in accordance with FDA requirements. To show how our suggested solution would work, we have developed a prototype that demonstrates this functionality.

From the results of this thesis we've learned that it is possible to decouple the existing communication system between sensors and database from the monitoring one. So, the use of a new network architecture based on the publish-subscribe method, makes the FDA validation easier and quicker, which implies that our system can scale easier.

CHAPTER 2

State of the Art

In the clean rooms of many pharmaceutical companies nowadays the data exchange is performed over a wired, soft-real time system. The term real-time, refers to a system where the correctness of its behavior depends not only on its results, but also on the physical time. Soft-real time is called a system that has to meet no hard deadlines of when a result is produced. [46] The sensors are connected to Programmable Logic Controllers (PLCs) and, usually, a Facility Monitoring System (MFS) is placed between the PLCs and the server log. The FMS, which is SCADA based, is a control system that reads the sensor value was sent to the PLCs and then analyses the receiving data in real time. In case a measured value exceeds a predefined limit for a certain amount of time, which could render the drug product unsuitable for use, an alarm is produced. Afterwards, both the data and the alarm notifications are sent to the database server to be saved.

2.1 Message exchange methods

In this section we provide a short description of the methods currently used regarding loosely coupled systems, unlike the ones used in the communication networks of the pharmaceutical companies where the interacting components are not always decoupled.

Distributed shared memory

This mechanism is used for sharing data in a system without inner-process communication. All the processes share the same memory, which means that any change to a value there by a process can be seen by any other process of the system. There are two approaches of the memory from a programming point of view. One is called Shared virtual memory and is very common in mono-processor systems which use paged virtual memory. Every process has a physical memory, but all these distributed memories share a single address space and consequently act as one total memory. This approach is used in the operating system layer. The other approach is called Object DSM and can be used in message exchange communications. The user defines

which objects will be shared and the DSM system takes care of the handling of these objects, i.e. creation, modification, access [30].
The advantages of the DSM mechanism are that it provides scalability to the system, it hides the messages exchanged between processes and it can handle large data bases without creating double messages or sending messages to processes. On the other hand, it has also some disadvantages; the programmer doesn't have much control on the messages being generated by DSM, there may also exist problems in the mechanism performance and there should be a protection mechanism against simultaneous access to shared data [31]. Furthermore, the interacting components of a shared space model are decoupled in space and time. 'Space decoupling' means that the interacting components are not aware of each other's existence. 'Time decoupling' means that the publisher and the subscribers don't need to run simultaneously [39].

Message Passing (point-to-point)
This message passing mechanism requires two parties, a sender and a receiver of the message. The messages are exchanged asynchronously, which means that the sender can send one or more messages without waiting for a reception notification and the receiver can receive one or more messages, even if the sender is not currently transmitting any. The messages transferred are kept pending in a queue, until they are received. The receiver processes are aware of all pending messages, but each message can be received by only one receiver. Senders usually send messages asynchronously, while receivers work synchronously. So, there is coupling of the communicating components in space and time, but not in synchronization. Point-to-point communication is provided by the Message Passing Interface (MPI) library, which helps in implementing data exchange between processes [32, 39].

Remote method Invocation (RMI)
This method is used to enable computers in a distributed network to communicate with each other through objects. It is implemented in Java and is a way of remote procedure call (RPC), but in our case one or more objects are sent along with the request. The call mechanism is called object serialization and an RMI request invokes the method of a remote object [33].
The advantages of this method are that it is easy to implement and, therefore, the applications using it are more flexible and robust. Moreover, no installation is needed on the client side and thus, the server-client communication is decoupled. However, a disadvantage is considered that it can be unsafe when using Dynamic class loading, because the client-side methods can be overwritten by a malicious programmer. Moreover, the object sending mechanism is less efficient than the Socket objects. Apart from this,

when a connection is broken while the remote method is executed, then this method won't return and further action should be taken to prevent this incident [34, 35]. Remote procedure calls are coupled in space, time and synchronization, and this is also reflected in this method. 'Coupling in synchronization' implies that the senders' and receivers' functions are temporarily paused when they send or receive notifications respectively [39].

Finally, another communication method used in loosely-coupled system is the publish-subscribe mechanism, which is described in the following section.

2.2 Publish-Subscribe

This technology was used by middleware products which required asynchronous communication of distributed systems in an autonomous and decoupled way. The idea is that applications, called publishers just send messages without any previous knowledge of the receivers' identity. Similarly, receivers called subscribers, receive only messages they are interested int. Receivers use subscription as a way to keep up-to-date with the latest updates on a subject of their interest. This provides a low-latency update processing but on the other hand the subscription management could be of great complexity, depending on the application. There is usually an entity between publishers and subscribers that provides this service [37].
A disadvantage is that there are few ways for the subscribers to define in detail their interest criteria on a topic-based subscription model. [38].
The advantages of this paradigm are that it is scalable due to the loosely coupled relation between the components. The decoupling property is provided in both space and time, i.e. the communicating components don't know each other and don't need to run simultaneously. Moreover, any other functions/operations they may perform are not temporarily paused by the publish-subscribe operation. It allows also the use of database which provides a big capacity and its response time is considered good. Additionally, a subscriber can register to more than one message types on a topic-based subscription model. [39]

2.2.1 Event Bus

Event Bus is ideal for communication of decoupled components, because its architecture is very loosely coupled. Thus, we can easily add and remove modules of the network during run time without affecting its core functionality. It can send directly or broadcast messages. It is easy to be tested and is currently used in many applications [11] [21].
The architecture of the network that the event bus supports can be:

- Publish-Subscribe: The idea is that components do not communicate and depend on each other. They don't even know each other's

existence. They just publish (send) events, which are heading to the event bus. These events after the event bus will be distributed to any subscriber which has subscribed to them [13].

- Broadcast: a message/event sent from the event bus will be delivered to all modules/recipients
- Point-to-point: a message/event is sent to only one module [21].

The event bus is considered an improvement of the observer pattern, where a module, called 'subject', retains a list of the observers'/subscribers' modules to events. The event bus keeps the loose coupling and broadcasting properties of the observer pattern, but avoids others such as code duplication, since the list of observers is kept in the event bus and not in each module separately. The event management is performed by a few classes, which makes debugging easier. Finally, the event wiring is easier, since observers don't search for subjects and the other way round, but everything is based on event types [22].

Java Event Bus

It implements the publish/subscribe paradigm between java objects. Objects don't need to refer to each other to communicate. Subscribing to an event includes class semantics and string matching. The version currently used and tested for a long period is 1.x (e.g. in applications such as Swing, SWT, and Apache Pivot), but version 2.x is also under development to include more toolkits than the previous one (e.g. Ajax, GWT, and Flex) [11]. The latest version currently available is 1.4, but it is considered to be relatively slow. The listeners are defined through annotations and we have both strong and weak references to store the subscribed listeners. It supports both synchronous and asynchronous event dispatching. In the synchronous dispatching the publication method is blocked until the event it sent is received by all handlers, while in the asynchronous dispatching the publication method returns immediately after the event is sent and the event delivery runs in another thread in an asynchronous manner. Event bus probably has a static filtering mechanism and also it makes it possible to influence the handlers order by ordering criteria [12].

The JavaEventBus can be used by both user interface (or SWING) applications and non-Swing applications. The basic interface in the JavaEventBus library is the *EventService*. It can be implemented either as a *ThreadSafeEventService* or as *SwingEventService*. In *ThreadSafeEventService* implementation publishing and subscription of an event is done on the same thread. We can use this implementation for multi-threaded (or threadsafe) applications. The *SwingEventService* extends the *ThreadSafeEventService*, providing the possibility of an event not published

on a *EventDispatchThread* (EDT), to be later subscribed to it. The *SwingEventService* implementation is the one used by default and should be avoided in non-Swing applications, if we want them to perform as expected. We can avoid this either by configuration or by using the API. In both cases, we change the setting concerning the *EventServiceLocator*, which is a registry of the *EventServices* [5]. Alternatively, we could create a new *EventService* implementation, which "*overrides the publish method and routes the call to a thread in a thread pool*" [4].
The *EventService* has two types of publications/subscriptions - Classes and Strings. In the class-based type, a component publishes an event as an object on the *EventBus* and then any component can be subscribed to the class of that object in order to be notified. In the topic-based type, a component publishes an event object using a String, called "topic". Any other component subscribed to that specific "topic" on the *EventBus*/*EventService*, will be notified and receive the corresponding object.
Finally, there are two ways of subscription, weak and strong. Weak subscription, which is used by default, is implemented by using *Weak References* to manage the listeners and is applied when using the *subscribe()* method. The disadvantage of this method is that the subscriber can get garbage collected, even during his subscription. Strong subscription avoids this problem and is implemented with the *subscribeStrongly()* method. Its disadvantage, however, is that it generates memory leaks, if the subscribers are not unsubscribed when needed [6].

Simpleeventbus
It implements also the publish/subscribe model, so that components can be decoupled with each other. This way the communication among them is much simpler and stable. It uses weak references to store internal subscriber objects, which, on one hand avoids garbage to be collected in the event bus, as it removes the objects if the subscriber misses its strong references, but on the other hand it can result in events that are lost for subscribers without strong references [13]. It is quite easy to implement and offers a fast delivery of the events. The characteristic that differentiates it from the other implementations is that the event delivery can be cancelled. The listeners are stated via annotations. The event dispatching is only asynchronous and there is no event filtering applied. Finally, we cannot apply ordering criteria to the event handlers [12].

Google Guava Event Bus
It offers another way to implement the publish/subscribe architecture, following the Observer Pattern instead of the Event Listener (followed in Java). The difference here is that objects do not subscribe to other objects or

the event bus, but to event types [14]. Components do not register with one another and therefore are not aware of each other's existence. This renders this event bus not of general type [15]. Listeners are defined here also via annotations. There is synchronous event dispatching, while the asynchronous one is performed only via specialized classes. There is no event filtering and handler prioritization. Finally a big disadvantage is that it uses strong reference types, i.e. the client should take care of the unsubscription of the subscribed listeners [12].

GWT (Google Web Toolkit) Event Bus

It is an event bus built as a GWT application, using usually the mvp4g framework. As a web application it uses the Model-View-Controller pattern. The Model, which contains data and methods, expects to receive requests by the Controller and the View, to which it responds. The View renders the Model. When we have more than one Views that have to cooperate with each other, instead of having their Controllers communicate with one another, we have them communicate via an event bus. Thus, the system is decoupled. A Controller creates events and event handlers and provides the connection for the Model and View to communicate. Below in figure 2.1 we can see the model of a single view [16] [17].

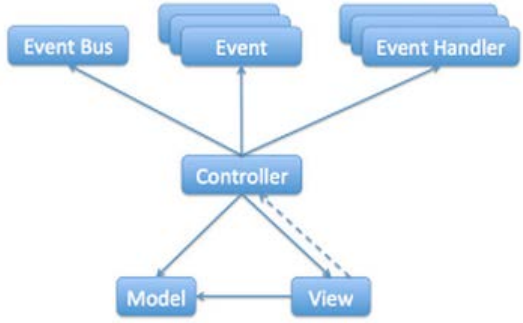


Figure 2.1: A single view of the model of GWT Event Bus

MBassador

It is an event bus that models the publish-subscribe paradigm. It is not heavy for the system and contains a lot of features. It is efficient in performance and resource use. It copes well with concurrent applications. Some of its features are: it annotates its message handlers (listeners); the events can be of any type and their hierarchy is defined upon delivery; the event publishing can be done either synchronously (the method blocks until the message is delivered) or asynchronously (once the message is sent, we forget about it); it uses static filtering and handlers prioritization; it uses weak references, thus there

is no need to unsubscribe listeners when not used; it is designed in a way extensible in the future for custom applications [18] [12].

Gum Tree Event Bus

It is based on the publish-subscribe method. Events are created upon state change and then the event bus sends a *StateChangeEvent* to all the subscribers. The limitation of this event bus is that it can only have one thread in its pool. So, a delay may occur if one task is running for a long time inside a *HandleEvent* method. It is better for the programmer to create his own private event bus for handling critical events, rather than the global gum tree [19].

JQuery as Global Event Bus

This event bus is mainly used in web development and not so often in other systems. It uses also the publish-subscribe paradigm but it is used for example to inform objects of a page about changes happening in an element of the web page [20].

Other Event Bus implementations (UI toolkits):

Flex – Mate

Mate is a framework which is based on tags and its purpose is to handle events created by our Flex application. It helps us find who handles the events, whether we need to retrieve data from the server or to trigger an event. Currently version 9.0 is released. The main functionality of Mate is based on the *EventMap*, which consists of *EventHandlers* for each type of event distributed, and it is usually implemented as a stand-alone *mxml* file [23].

Java Server Faces

The JSF user interface is responsible for checking that an event occurred. It creates then an instance of the event class related to this event and adds it to an event list. A UI component fires an event to signal user actions. For firing the event, a checking of the list of the event listeners is required and the already registered handler or listener is notified.

The difference of JSF from the event model for standalone applications is that since the user acts in a client, which doesn't communicate always with the server, some events arrive delayed (when there is a new connection). To handle this, the creation and handling of events is performed in different phases [24, 25].

2.2.2 Java Messaging Service (JMS)

Messaging is a method used by different software components/clients, which want to communicate. However, it is not necessary that the application clients know with which ones to exchange messages, but only the message format and destination. This way we achieve a loosely coupled communication. There is a messaging agent, to connect with, that is responsible for creating, reading, sending and receiving messages among components.

JMS is a Java application that provides interfaces and related semantics, which can be used by client applications to exchange messages. JMS acts as a messaging agent, i.e. creates, reads, sends and receives these messages. The application clients should also be implemented in Java, in order to use this technology. The communication it provides is loosely coupled, asynchronous and reliable. Asynchronous messaging means that the JMS provider delivers messages to the client, the moment they arrive and not upon the client's request. Reliable messaging means that a message is delivered one time. However, it is possible to set up a messaging service, which allows lost or double message delivery, if needed.

The current version of JMS is 1.1.

The JMS application can be proved very beneficiary for enterprises, who want components communicating without any knowledge of other components, and therefore can be easily replaced. Components send messages and continue to operate without receiving any response. Finally, it could be the case that not all components operate at the same time.

The JMS was first introduced in 1998 and was used for applications to access messaging-oriented middleware systems (MOMs). However, nowadays it can provider many more capabilities useful for enterprises. When Java EE 1.3 platform was first released, JMS was a part of it. So, nowadays Java EE platform contains the following components used for JMS; application clients, Enterprise JavaBeans, web components, message-driven beans and messages. Application clients can exchange message in both synchronous and asynchronous way, while the Enterprise JavaBeans and the web components only in a synchronous way. The message-driven beans are used for asynchronous communication and finally message exchanges allow also the exchange of JMS operations as well as the performance of database access [7].

JMS Architecture

The architecture of the JMS API consists of the following components; a JMS provider, JMS clients, messages and Administrated Objects. The way these components interact is shown at figure 2.2. Administered objects are preconfigured JMS objects and can be of two types; destinations and connection factories. An administrator creates them to be used later by the

clients. The Administrative Tool binds these objects to a JNDI namespace. JMS clients are programs written in Java which produce and/or consume messages. They inject resources to access the administrative objects in the JNDI namespace. Finally the JMS clients establish a logical connection to the administrative objects through the JMS provider. "*The JMS Provider is a messaging system that implements the JMS interfaces and provides administrative and control features*". Both JMS Provider and JMS clients can be implemented in Java EE platform [8].

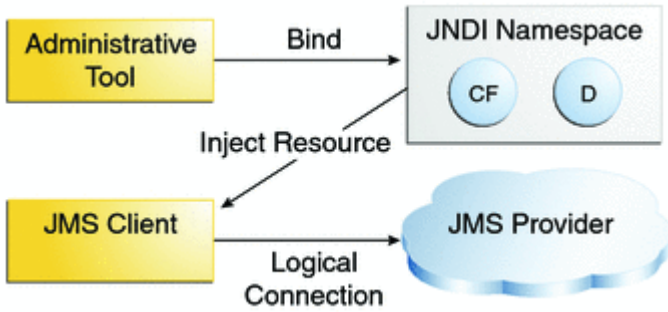


Figure 2.2: JMS API Architecture

Messaging Domains

The JMS API provides two messaging domains, point-to-point and publish-subscribe. A JMS provider can implement both of them and a client can use both.

A **point-to-point** application contains the following components; senders, clients and messaging queues. Senders send messages to a queue. Each queue accepts a specific type of messages and keeps them until they are consumed or expired. The client consumes the messages and acknowledges the fact, but doesn't have a time dependency with the producer. A message can be consumed only by one consumer.

A **publish-subscribe** application contains the publishers, who address messages to topics and the subscribers. The publishers are not aware of the subscribers' existence. There can be multiple publishers and multiple receivers. In this case there is a timing dependency among these two components, as a client needs to subscribe to a message topic, before the message is send. Furthermore, the message consumption is done only if the consumer is active. It is, however, possible to create durable subscriptions, which receive messages even if the consumers are not active. In figure 2.3 we can see a scheme that shows how the publish-subscribe paradigm works.

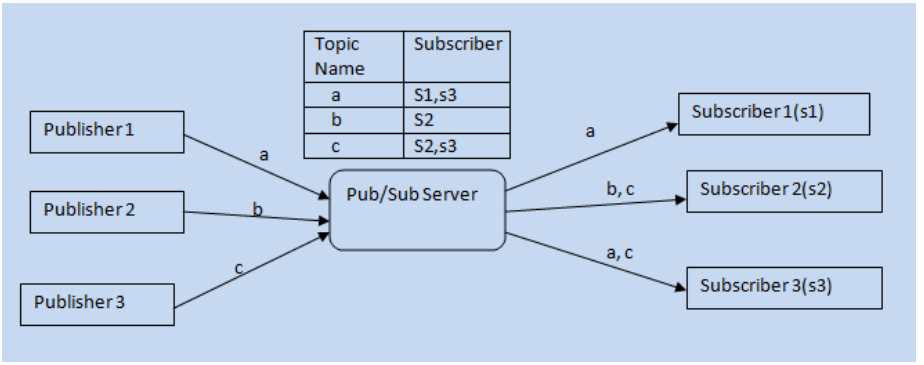


Figure 2.3: Topic-based Publish Subscribe paradigm [36]

The message consumption can be categorized as either synchronous or asynchronous. In synchronous consumption, the client fetches the message by calling the receiving method, which in turn blocks until the message arrives or a time out occurs. In asynchronous consumption, the client registers a message listener with a consumer. So, when a message arrives to a destination, then the JMS Provider calls the *onMessage* method on the client to deliver the message to the client [8].

JMS API Programming Model

We can see the JMS programming model in figure 2.4:

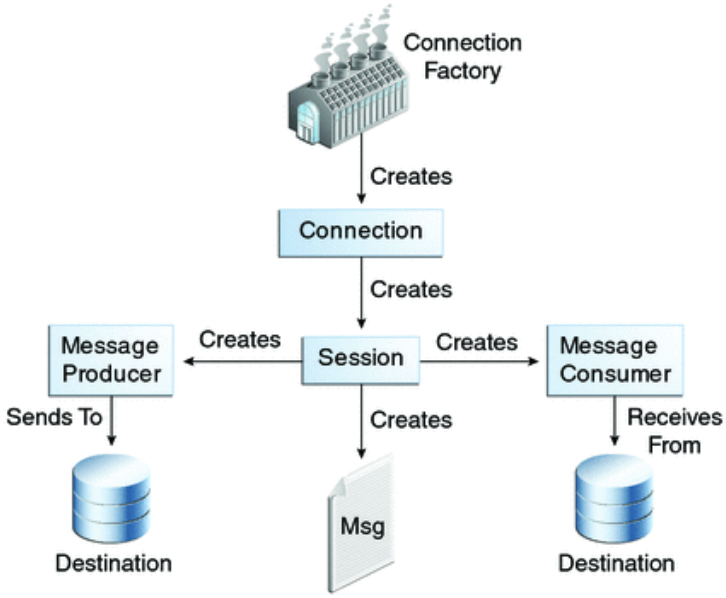


Figure 2.4: The JMS Programming Model

First of all, we create the administrative objects in the JMS Provider. We can use the Administration consol of Glassfish Server to do that.
The connection factory object can be created as an instance of either *ConnectionFactory.QueueConnectionFactory* or *ConnectionFactory*.*TopicConnectionFactory* interface. The client uses this object to create a connection to a provider. The client program includes a connection factory resource statement, which injects this resource to the connection factory. We should remember to close the connection, so that the JMS Provider releases the resources.
Using Glassfish we can also create a destination object (by specifying a JNDI name). This object is the place where the producers send their messages, and from where the receivers retrieve them. We call them queues in the PTP domain and topics in the publish-subscribe paradigm. A JMS application can use many queues or topics or even both. Apart from the destination defined in the JNDI, we also need to specify a physical destination. In the client program, we inject also a destination resource. "*The resource names are mapped to destination resources created in the JNDI namespace*". We should not forget that a destination is connected to only one domain; queue or topic. In order to allow both, we create an object of type "Destination".
"*A connection encapsulates a virtual connection with a JMS client*". It creates one or more sessions. However, this option is only available for client applications in Java EE.
A session is a "*single-threaded context*" and is used to create messages, producers, consumers, queue browsers, queues and topics. When creating a session, we can define it as transacted or not, and also specify whether the messages get automatically acknowledged by the session when received successfully or not [9].

2.3 Secure Logging

Audit log security includes protection from modification and deleting without permission, so that it is possible later to detect and investigate the security violations that occurred.
A very popular method for securing audit logs is Forward Security mechanism.
It was first introduced by **Bellare and Yee (1997)** and the general idea is using message authentication codes (MACs) to encrypt the logs in a way, that even if the storage system is compromised (i.e. a MAC key becomes known to the attacker) he cannot read or alter old entries. He can only delete them, but not without being detected. More specifically, time is divided in "epochs". So, if a system is compromised in time Tc of an epoch Ej, i.e. $Tc \in EEj = \{t:T_j<t<T_{j+1}\}$, then the attacker cannot see entries for time t<Tc.

An attacker would ideally wish to modify and erase logs without being detected. He tries to exploit bugs that a Trusted Computing Base (TCB) possibly has and gain privileges that let him exploit log data. One of the ways to prevent attacks till 1997, have been using *remote logging*, where the logging host sends the data to a remote host, which hopefully the attacker doesn't have enough knowledge to exploit. Another method which can be used additionally is *log replication*, where the data is copied from the host to more than one remote hosts, so any erase action can be easily detected, as the attacker won't probably be able to intrude to all the remote hosts to cover his actions. Of course, these two methods are based on assumptions. In case these assumptions are not valid, e.g. the intruder gains access to all logging hosts, then these two methods are not safe anymore. Another method used, was writing the audit log to a printer. This requires filtering the log entries in advance so as to avoid printer overwhelming. However, if the attacker gets physical access to the printer, then no security can be applied. The same thread exists for Write Once Read Multiple (WORM) drives, as long as the attacker can access them physically.

MAC keys are used usually in a sender-receiver communication, but we cannot apply the exact same scheme on a log stored locally, as the data can be at risk in case the attacker learns the secret MAC key. The way we will use the MAC coding is different now. Entries will be encrypted upon writing. We initialize a log system LOG within epochs. The data will be logged in the log system in an order. The MAC code will be different for every epoch. The MAC key Ki of an epoch Ei will be created by a non-reversible function from the key K_{i-1}, and when a new epoch starts, then, the key of the previous epoch K_{i-1} will be erased. The base key K0 will be able to verify all the log data and thus will be saved in a secure remote verifier, assuming to be uncompromised. This verifier is assumed to retain also all the keys and keeps its privacy and integrity properties. As we mentioned we divide the time in epochs, which are sequential and not overlapping in time. Every epoch has a first and last message and there are no messages on the boundaries of the epochs. The LOG creates a log entry for each message it receives, but they are not verified immediately. A log entry is of type (m_i, $FIMAC_j(m_i)$), where FIMAC is an authentication code which doesn't transform the messages, but just appends to them. The verifier computes the following:

$$\text{Valid}(m,j,x)= \begin{cases} true\_if\_x = FIMAC_j(m) \\ false\_otherwise \end{cases}$$

The logger can add any data he wishes to the data log. In the end of every epoch he adds an additional log entry. The key Ki of the FIMAC is generated from the previous key Ki-1 using a one-way function. Afterwards the key Ki-1 is erased from the system.

Let's assume an algorithm A running for time t and creating maximum q log messages of length maximum L. (q,t,L,e) *FI secure* is characterized a log system if there is not such an algorithm, that can make an "open" request to learn all secrets and outputs (m,j,s) with probability e, i.e. a false entry (m,s) for an epoch Ej earlier than the time he intruded. This term is concerned only with the attacker possibility of forging log entries but not deleting them. Deleting of log entries is a denial-of-service attack and can only be detected. A log system is characterized as (q,t,L,e) *deletion detecting-FI secure (DD-FI)* if it is FI secure, according to the previous definition, and also the log verifier can detect whether log entries were deleted, if he is given the log system output and the epoch, during which the attacker compromised the system.

We use the *prf-chain FI MAC* to design MAC algorithms with forward integrity (FI) property. The prf-chain functions by using block ciphers and standard MACs. The initials stand for pseudo-random functions. In each epoch Ej, the log system has a secret value sj. All the data messages of that epoch have the MAC code: MACkj(mi), where $kj=prf_{Sj-1(0)}$. When we move to a new epoch Ej+1, then we generate both the new secret $sj+1=prf_{Sj(1)}$ and the new MAC key $kj+1=prf_{Sj(0)}$, and finally we erase the secret sj and MAC key kj of the previous epoch from memory. "The prf family can be broken with probability d/n using 2 queries" and "its security downgrades linearly with the number of epochs".

To design *MAC schemes with DD-FI* security we need to include sequence numbers and create a marker to denote when the log changes epoch, before the change occurs. The log entries mj have a MACki(0|rel(j)|mj), where rel(j) is the message sequence number inside the epoch, i.e. j-first(epoch(j))+1. The symbol | shows concatenation. Upon a change to the next epoch we have a marker MACki(1|rel(last(i))) and afterwards the creation of the new secret and MAC key takes place. This marker is the way to detect when a deletion occurs in the log system by an attacker.

As we mentioned previously the secrets are shared among the log system and the verifier. This can be done encrypting them with the verifier's public key. Of course, this is not enough as an attacker can send false messages encrypted with the verifier's public key. So, we additionally use authentication of the log system in the beginning of the log with either a timestamp protocol or a signature mechanism with the forward non-reputability property using the public key. An alternative of using secrets in the log system is having the verifier to create the keys and send them to the log system by a cryptographic mechanism with forward security.

In an attack network latencies play an important role on the speed an attacker compromises a system and if his attack is automated. It is necessary that our FI log system changes epoch with high frequency, so that the FI scheme can be efficient. It may also be the case that the log entries have associated

requirements such as keeping data about failed log-in trials. These may lead to leaking the user's identity. If we ignore covert channel and traffic analysis attacks, then encrypting the log entries before sending them to the logger can be an adequate security measure [29].

Another approach to secure logging is given by **Schneier and Kelsey (1999)**. Their solution method manages to prevent an intruder to read and alter entries in a machine before the time he potentially compromises it and detecting his actions of deleting old data. Still no guarantee can be given for data logged after he has compromised the machine. We also assume that there is not a reliable and secure communication channel between the untrusted machine U, which initially stores the logs, and the trusted server T. They also assume the existence of a partially trusted verifier V, who can read but not change the records in U.
The method, in brief, is the following; U ensures a secure connection with T and shares a secret key to create a log file. After every log entry, the log authentication is hashed with a one-way function. Every new key is derived from the previous one with a one-way function and overwrites the old one. Every entry has an element in a hash chain, which can be used to authenticate all the previous log entries any time, even remotely. Finally, each log entry has a unique permission mask, which defines which entries can be accessed by whom partially trusted user, since each of them has different access rights.

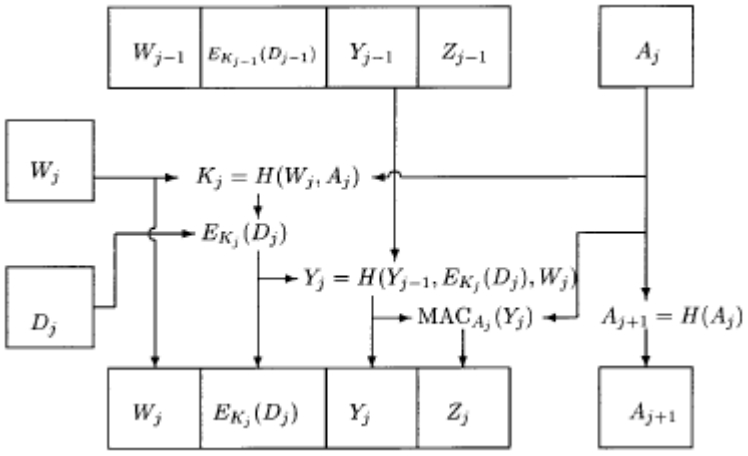


Figure 2.5: Adding an entry to the log

In figure 2.5 we can see a scheme that describes how the security mechanism works, i.e. how we can create the next secure log entry Lj based on the previous one L_{j-1}. The letter Dj represents the data which we want to enter in the next entry j. The letter Wj represents the log entry type, i.e. the

permission mask for the verifier V. T determines which log entry types Wj will V access. The letter Aj is the authentication key for the j^{th} log entry. The first key for the first log entry is Ao and is either generated by T and is securely sent to U or the other way round. The way to generate keys for the following entries is by applying the increment hash algorithm on the current key. Immediately after creating the new key Aj, the previous one A_{j-1} is deleted. (We could create an Abnormal Shutdown message and store it in nonvolatile storage. This message can be used in case of a system crash and will be copied to the log file after system recovery). The letter Kj is the result of the hash algorithm between the log entry type Wj and the authentication key Aj. Kj is used as the encryption key to perform symmetric encryption of the data Ekj(Dj) (e.g. DES, IDEA or Blowfish algorithm). Immediately after creating the new key Kj, the previous one K_{j-1} is deleted. Moreover, Kj is deleted also after its use (i.e. after encrypting the data). The letter Yj is the hash chain, generated by the hash algorithm applied on the previous hash chain Y_{j-1}, the encryption key of data Ekj(Dj) and the log entry type Wj. This chain allows V to verify parts of the log, since he is based on Ekj(Dj) and not Dj. In the beginning of the chain Y-1 is a 20-byte block of binary zeros. Finally, the letter Zj is computed by applying the symmetric message authentication code on the hash chain Yj, based on the authentication key Aj. So, a log entry Lj consists of the following information; Wj, Ekj(Dj), Yj and Zj. With this scheme, if an attacker compromised U at a time t, he will have a list of log entries and the key A_{t+1}. He cannot calculate the previous authentication keys before t, so he cannot read or modify the previous entries. He cannot create new ones either, but he can delete old entries and learn about the new ones. However, if he deletes old entries, T will detect when U will interact with it.

The start of this procedure is special, as U needs to establish a secure connection with T using Diffie Hellman protocol. U forms the following values: a random session key Ko=hash(Wo,Ao), a current timestamp d, a timestamp of his timeout d+, an identifier for the logfile IDlog, it's certificate from T Cu, a random starting key Ao and a value X=p,d,Cu,Ao, where p is the protocol step id. He sends then to T the message Mo=p,IDu, PKEpkT(Ko), Eko(Xo,SIGNSKu(Xo)), where PKE is the public key encryption under T's public key and can be implemented with RSA or ElGamal algorithm, SIGN is the digital signature with the private key of U and can be implemented with RSA or DSA algorithm and finally, E is the symmetric encryption under the Ko key. U then creates the first log entry Lo using Wo=LogfileInitializationType and Do=d,d+,IDlog,Mo and stores Ao and hash(Xo) locally, while he waits T's response. T verifies the Mo message he received from U. For the message to be correct it should decrypt correctly, include the valid signature of U and his valid certificate. In the case of a correct message, T creates a random session key K1 and sends back to U the message M1=p, IDt, PKEpku(K1),

Ek1(X1,SIGNskt(X1)) where X1=p, IDlog, hash(Xo). Finally, U verifies the message M1 he got from T and forms a new log entry Lj with log entry type Wj=ResponseMessageType and Dj=Mj. He also calculates A1=hash("Increment Hash", Ao). But in case M1 is invalid or sent after d+, then U forms a log entry with W1=AbnormalCloseType and D1 containing the current timestamp and a description of the reason of the closure. Subsequently the file is closed. The purpose of writing down this last type of message, is to prevent the case of an intruder deleting all log files of U after compromise and then claiming that U didn't ever receive M1 in the startup.
We have a specific procedure to end and close a Logfile. The final message we write has entry code Wf=NormalCloseMessage and as data Df a timestamp. The final authentication key Af and Kf are deleted and the file is then closed. Afterwards, T can receive and validate the complete log, since he has the hash chain and the Zf (because he knows Do). He can also find the encryption keys and read the log entries.
In case some person or machine V moderately trusted wants to read or verify some entries while still on U, then this is possible under the condition that T has sent message M1 to U, T has received a copy of the log from U and the log is not closed, yet. The reason for this is that if V is compromised in the startup procedure of U and T, then it will access all data in U. The procedure is the following; U sends a copy of the log entries to V, including message M1. V verifies each entry of the hash chain and, then, creates a secure connection with T, using a key. He makes a list Q of the log entries he wants to read, which contains their log entry type Wj and the entry index, i.e. Qj=j,Wj. Then he sends to T the message M2=p, IDlog, f, Yf, Zf, Q. T verifies that the log originated from U is correct and V has the rights to learn it. T can calculate Af from Ao and therefore verify that Zf=MACa(Yf). In case of a problem, he sends an error message to V and records the problem which could be caused either from the U's IDlog or by V. In no problem occurs, T creates a response list R for each Qj. The Rj contains either the decryption key (which he computed based on Wj) or the reason for its refusal. He then sends to V the message M3=p,R. V can read but not modify the log data. Finally, V deletes the connection key he had established before to communicate securely with T [40].

We also came into the approach suggested by **Ma and Tsudik (2009)**, which seems to find vulnerabilities in the two methods mentioned previously and so they suggest a new one, which overcomes them. They found out that the Schneier-Kelsey method has two security vulnerabilities; one is the truncation attack, which means that an attacker can compromise U and delete the last entries which are logged in U but not yet sent to T. And since the communication between U and T is not continuous but in most cases event-

driven, T cannot find out that some entries are missing from U. The other security drawback of this scheme is the delayed detection of the attacker. This is the occasion because V needs T to verify a log file and can happen if the attacker compromises U, before T receives the latest log file and before U closes the file. The attacker can then read the current authentication key Ai, generate MACs from the current time and modify the related messages Mi and values Yi, but not Zi. So, when V gets a copy of log entries and sends the final values of his file Yf and Zf to T, the later verifies them successfully. The attack will be detected only when U communicates with T, which can take a while. Furthermore, the Schneier-Kelsey scheme has two architectural limitations. There is a need of an online server, as we require continuous communication between U and T and between V and T. Secondly, the storage requirements of the scheme are inefficient, as we need in total at least 512 bits per log entry (256bit for the hash value Yi and 256 bits for the MAC Zi). This makes the scheme inappropriate for devices with limited storage capacities, such as sensors, implantable medical devices etc. Moreover, the use of multiple authentication tags, instead of a unique one, may cause truncation attacks. The same security drawbacks and architecture limitations characterize the Bellare-Yee scheme and therefore make it unsuitable for use.

Ma and Tsudik suggest a scheme that overcomes the above problems and is called “Forward Secure Sequential Aggregate (FssAgg) Authentication”. Additionally, this scheme provides: (a) data integrity, since the attacker cannot insert fake data or modify or delete the existing ones, (b) stream integrity, since he cannot re-order the log entries, and (c) forward security, since the signing key is produced by a one-way function and the attacker cannot recover previous keys and signatures from the current key. The components included in the scheme are FssAgg.Kg, FssAgg.Asig, FssAgg.Upd and FssAgg.Aver. The first one is an algorithm for creating pairs of public/private keys given the maximum number of time periods as input. The second one is an algorithm which takes as input a private key, a message and the aggregate signature so far and computes a new signature to sign the message and combines this signature with the signature so-far to generate the new aggregate signature. Then the FssAgg.Upd procedure runs, which creates a new signing key for the next time period, using as input the key for the current period. Finally, the fourth algorithm verifies whether a signature is valid or not, by taking as input an aggregate signature, a set of messages and a public key. This scheme satisfies also the property of correctness, as an aggregated signature created by FssAgg.Asig is verified by FssAgg.Aver, and the property of unforgeability, since an attacker cannot make a valid forgery even if he know the current signature key.

The scheme they suggest is implemented in two ways, each serving different needs; a *private-verifiable* method and a *public-verifiable* one. The first one uses again the players T, V, U but to avoid the online server, we use two MACs, one verified by V and one used by T to verify log entries. Therefore, any time an authenticated log file consists of two MACs and the log entry. In the beginning T is online and U commits two random symmetric keys A1 and B1, information about the log and the key update frequency UPD. This frequency is fixed by T or U and can depend on time or activity, but here we assume that we have an update for each log entry. We assume U is not compromised in the beginning. After T receives those, he goes off-line. He stores these two initial keys and knows that at least one log entry L1 exists in U, so as to avoid total deletion by an attacker. U after sending these two keys, he creates an entry log L1 which relates to a message, e.g. "START", creates two MACs from the two keys μ_{T1}=macA1(L1) and μ_{V1}=macB1(L1) and uses a one-way hash function F to generate new keys A2=F(A1) and B2=F(A1). The log entry generation process is as follows; before creating the ith entry, U contains the log entries L1 to L_{i-1}, the keys Ai and Bi and the MACs μ_{Ti-1} and μ_{Vi-1}. At the i^{th} entry, U creates the Li entry, the MAC for V macAi(Li) and the value μvi=H(μ_{Vi-1}||macAi(Li)), where H is a one-way hash aggregation function, which unrolls as μvi=H(H(…H(μ_{V1}||macA1(L1))…|| macAi(Li)). Similarly, U creates the MAC for T μ_{Ti}=H(μ_{Ti-1}||macBi(Li)). U then creates the keys A_{i+1}=F(Ai) and B_{i+1}=F(Bi) and deletes the keys Ai and Bi. Finally, to close a log file U creates a log message Lf, updates the μ_{Vf} and μ_{Tf}, deletes the keys Af and Bf and sends the closed file to T. When V wants to validate a series of entries, he receives key A1 from t and a copy of entries L1 to $L_{f'}$ and $\mu_{Vf'}$ from U. He computes keys A2 to $A_{f'}$ and $\mu'_{Vf'}$ and checks if it matches $\mu_{Vf'}$. T validates the final closed log file using B1 and μ_{Tf}. With this scheme V can verify a log file without T's help, which make our scheme efficient in matters of storage. However, we still cannot prevent a delayed deletion attack.

The public-verifiable scheme is generated to achieve this. Instead of T we use a Certification Authority (CA) to certify/register the public key of U. And V belongs to the public domain, which means that anyone with a copy of the log file can verify it. Therefore, it is suitable for scalable systems which require public verification. Here a log file contains the log entries and a signature $\sigma_{1,f}$. In the beginning U uses the FssAgg.Kg algorithm to create a pair of private sk1 and public pk keys. He registers the public one with a public CA. So, a certificate of a log file IDlog is CERT(IDlog)=SIGNca(U, IDlog, t, T, pk, timestamp, …), where U is the log creator, t the starting time and T the maximum number of key update periods. He then creates the log entry L1, sets it to the certificate, creates a signature $\sigma_{1,1}$ on L1 using FssAgg.Asig algorithm and the initial key sk1. He updates the key from sk1 to sk2 with

FssAgg.Upd and deletes sk1. When creating a log entry i, the log file contains so far the entries from L1 to L_{i-1}, a signature $\sigma_{1,i-1}$, his current private key ski. When we have a new event, U creates a log entry Li, creates a new signature $\sigma_{1,i}$ to sign Li, giving to the FssAgg.Asig algorithm as input Li, $\sigma_{1,i-1}$ and ski. He uses the FssAgg.Upd algorithm to update the secret key sk_{i+1} and deletes sk_i. In case we reach the T number of update periods, we can extend it. We can create a special log entry LT, which contains the new batch of public keys. We can create a public key for the new number T and use the initial skT (created in the creation of LT) to certify a new set of public keys to be used in the future. To close the log file U creates a respective log file Lf, updates the signature and deletes the secret key. Validation of the log by a verifier V can be done by extracting the public keys from the CERT(IDlog) value which is included in the initial log entry L1 and by verifying CA's signature on this value. He finally verifies the log with the aggregate algorithm FssAgg.Aver.
There has been the need to extend the FssAgg scheme in a way that it becomes immutable. The reason is that currently there is only one aggregate tag. The individual ones are deleted after integrated in the aggregate. Therefore, we can only verify the entire log and not individual logs. However, this can be inconvenient and costly if individual verification is desired. Moreover, using the aggregate verification we cannot derive any info about the authenticity of its version. A solution would be keeping individual signatures, but the aggregate methods are public. So, any revealing of either the individual signatures or the MACs implies truncation attack. So, we applying immutability to a private MAC scheme means that U will create a phantom MAC in the beginning and compute μ1=macA1(L1) on the "start" message L1, update A1 to A2, then compute μ2=macA2(L2) and aggregate $\mu_{V,2}$=H(μ1|| μ2). Both μ2 and $\mu_{V,2}$ are stored, A2 is updated to A3, then μ1 and A1 are deleted. Therefore, if a verifier wants to validate a specific log, then he can use just μi. In the case of a signature FssAgg scheme, immutability is applied differently. U computes both an individual signature σj for a log entry Lj and an umbrella signature σj* for k+1 entries of L_{j-k},..,Lj. The umbrella signature is aggregated with the aggregate so-far one and then erased. The time period this signature is created is called anchor point and the corresponding entries anchor log entries. The individual signature is the one used for individual verification of each log entry [41].

Ma and Tsudik provide the details on how to implement the FssAgg methods in [43]. First, they explain the FssAgg MAC scheme, where H is a one-way hash function of k-bit strings domain, Ha is also a one-way hash function of arbitrary length input and h is a MAC, which takes as input a k-bit key x and a message m and outputs a t-bit MAC hx(m). The FssAgg.Kg algorithm is implemented as a symmetric key generation algorithm which creates a k-bit

secret key. The FssAgg.Asig method, which takes a message Mi and an aggregate so-far signature as inputs, firstly generates a MAC signature on Mi using ski, σi=hski(Mi), and secondly calculates $\sigma_{1,i}$=Ha($\sigma_{1,i-1}$||σi). The method FssAgg.Upd functions as ski=H(sk_{i-1}). And finally, the FssAgg.Aver is used by the verifier who has the initial secret key sko. He calculates the other keys through the public update function, he then re-computes the signatures and compares them with the original ones. If they match then the log is verified, otherwise it is not.

As for the FssAgg signature scheme it is based on the BLS scheme. The last one uses bilinear mapping of cyclic groups *G*1 x G2 -> GT. The hash function H is used to generate the key Gi for each signer computing $v=gi^x$, where v is his public key and x his private one. Then the FssAgg.Kg algorithm would generate the key pair xi=H(x_{i-1}) and $v=gi^x$. The FssAgg.Asig first computes a signature $\sigma i=H^{Xi}$(index||Mi), where Mi is the message to be signed and index the position of Mi in storage, used for ordering. Then the method aggregates the signature by multiplying σ1,i= $\sigma_{1,i-1}$* σi and, afterwards, the signer updates the key. The FssAgg.Upd updates a secret key by the hash function xi=H(x_{i-1}). Finally, the FssAgg.Aver function uses e($\sigma_{1,n}$)=Πi=1,n e(hi, vi) and the public key to verify an aggregate signature. [43]

Ma has gone even further and explored the cons and pros of the above aggregate signature scheme. Both of its schemes resist the truncation attack. However, the one based on the MAC scheme provides message-level forward security but doesn't support non-repudiation and public transferrable verification. The one based on the signature scheme, also called as BLS-FssAgg provides interval-level forward security but requires a lot of space for storing in the verifier's side. In the case that an attacker obtains the signing key then he can add and remove tags. Moreover, it is not suitable for use in data intensive application such as databases. Instead he suggests two new schemes BM-FssAgg and AR-FssAgg, which are better than the BLS-FssAgg scheme in many factors, such as constant public key size, efficient aggregate verification etc. It then explores two ways of using these signatures; the first one is as soon as the message is generated which is not so secure in the scenario that we have sensors storing their secret key together with their data. The second way is to wait until we can move to the next interval and then sign the data. This method prohibits the attacker to obtain the secret signing key even if he compromises the system. However, the implementation of these two schemes BM-FssAgg and AR-FssAgg is trivial. The authors will search for a non-trivial one in the future. [44]

Stathopoulos et al. (2006) investigate the privacy and confidentiality maintenance in public network providers such as telephony, either fixed or mobile, and Internet Providers. They believe that secure logging plays a vital

role in detecting any security attacks by internal or external malicious users. Therefore, they suggest a secure logging method to ensure security in telecommunications of public networks. They have already read the security schemes for logging which have been proposed till that time by others, including the ones of Bellare and Yee (1997) and Schneier and Kelsey (1999). They noticed that most of these schemes assume that the host's Operating System is not compromised by an attacker.

The thread models considered by the previous schemes, include the log Generators and the log Server and are two: (a) "Trusted Generators and Marginally Trusted Log Server", where disclosure and modification attacks may occur in the log server, and (b) "Distributed Log Generators and Marginally Trusted Log Server", where we consider a distributed system vulnerable to impersonation attacks against log generators and transmitted log messages, as well as disclosure attacks on the messages transmission.

Stathopoulos et al. assume that all components involved in logging are semi-trusted, i.e. the generators, the log server and the transmission channel. Therefore, they also consider insider and collusion attacks between log generator and server, which are not investigated by the previous thread models. More specifically, they consider modification attacks (a) on saved logs by attackers who have compromised the server, (b) from corrupted generators and (c) from colluding generators and servers. They also assume that a trusted Regulatory Authority RA exists which ensures the security and privacy in the Provider. Finally, the security framework is considered to keep the properties of availability and integrity of the logging procedures and log files.

There are 5 phases that contribute to this framework constitution.

1. In the first phase, we define the network and operational events, describing what is important to be logged in the Provider. In order to define this, a Reference Model (figure 2.6) is used, where the logging needs are analyzed from three views, called Planes; (1) the Functional Plane, models the network and its operational events, without considering information about architecture, implementation, design and topology constraints. So, this plane is not limited only to a Provider environment. This plane links functions to the log files that monitor them. The data that should be logged here are security functions (such as system access control, password and user management), service management functions (such as monitoring and troubleshouting) and network management functions (such as network configuration, connectivity and routing). (2) The Service Plane models services executed within the network or the IT nodes, such as password management, dsl, snmp etc. It discriminates the system from the application services and it concerns the operating system platform, the communication protocols, interconnections and the

hardware. (3) The Logging Plane describes in log files, commands and events of each service.

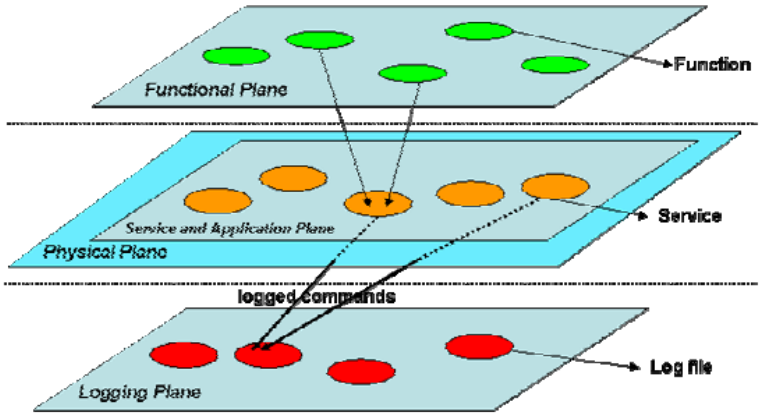


Figure 2.6: Log Reference Model

2. In the second phase of the framework, we define the operational and security requirements of each log file, which depend on the agreement between the Provider and the RA. More specifically, we define the file structure, i.e. fields, its generation frequency and its storage requirements, i.e. its form, storage duration and type (local or remote).
3. In the third phase, security measures against external attacks are taken. Here the scheme of Schneier and Kesley is suggested, as it helps to preserve forward integrity on the log files. This scheme holds under the assumption that the authentication key is not compromised and, therefore, modifications on log files don't remain undetected.
4. In the fourth phase, security measures against internal attacks are taken. We assume that the log server is compromised on time ti and so the attacker learns the authentication key at that moment and all the next ones. He decides though to make his files modification attack on time tj>ti. He won't be then detected, even if the MAC algorithms are replaced by digital signatures. Stathopoulos et al. suggest using the Schneier and Kesley security scheme enhanced with digital signatures in predefined and random time and a trusted RA, which stores them. Additionally, there will be limited interaction of the Provider with the RA. They assume that the communication is secure between the RA and the Provider, which has the server. They also assume that the server has two public/secret key pairs PK1/SK1 and PK2/SK2 and the relative certificates. The certificates are issued by a trusted certification authority, trusted by all other parties. The signing can be manual and automated.

a. The manual signing is performed at the log server by the administrator in an isolated environment. He uses the secret key SK1, which is not installed in the server, and the signatures are sent to the RA. RA acknowledges the reception of the signature and stores it. The procedure is performed periodically at recommended time periods ranging from a day up to a week. RA can later verify the signature with their certificate when he receives a copy of the log file of the server. This way RA can detect any modification on the log entries, which occurred after they have been signed by the Provider and received at RA, even if the authentication keys Ai have been compromised. However, it cannot detect the attack if it occurred before the log entry was signed.
b. The automated signing can be done in two ways. One way concerns signing of the event which RA has defined as critical, such as log file modifications, change of users and user privileges, system restart etc. When a critical event occurs, an alarm is generated. We assume the existence of an Alarm Service, which handles the alarms and communicates with RA. It signs the critical log entry and some of the following ones with the secret key SK2 and sends the signature to RA. The RA acknowledges the reception of the signature and saves it. The other automated signing way concerns the RA which requests at random times a signature from the current log entries of the log file. This request is handled by the Alarm Service as alarm. With automated signing, the probabilities that an attacker modifies a log entry undetected are reduced even more, because it minimizes the available time-period for an intruder to modify a log entry before it is stored in RA.

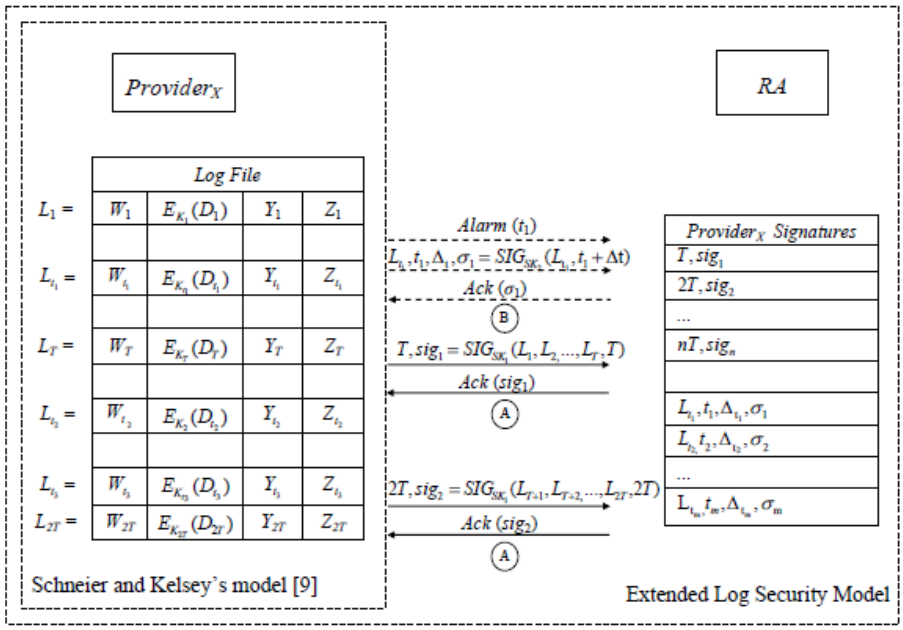


Figure 2.7: Extended secure log system

5. The fifth phase of the framework includes the implementation of the security scheme.
6. Finally, the sixth phase includes the verification of the log files. RA will compare the log files stored in his environment with those in the Provider environment. First RA checks the validity of signatures created with SK1 and when they are all verified, then it checks the ones created with SK2. If any signature is not verified, then this is evidence that the log file in the Provider has been modified.

In an extended system more than one Providers and Alarm Services may be used. So, if an intruder tried to perform an internal attack, he would have to compromise all of these systems, so as to interfere in the automated signing procedure. But even in that case, the probability that his actions will be detected is still high.

CHAPTER 3

System Analysis

In order to sell pharmaceuticals at the market, the producer must prove they are in compliance with the regulations of the market. For the U.S. market the regulations are set by FDA. Other markets will have regulations set by local authorities, but FDA regulations are among the strictest worldwide and when complying with them, most of the other markets are covered. FDA sets regulations for various topics (mechanical, chemical etc.), but in this thesis we will restrict ourselves to look at the IT/Automation part of pharmaceutical production only. The producer will "compile" the FDA requirements to specific requirements for the system he wants to build – the URS (User Requirements Specification). To help the vendor of the system to design a system which conforms to FDA regulations, ISPE has issued a comprehensive guideline, GAMP, which introduces terminology and methods to create a system which will be recognized as compliant during an inspection.

3.1 FDA Regulations

FDA are the Food and Drug Administration regulations in U.S. targeted to the pharmaceutical industries, therefore our systems should also comply with them. Furthermore, the relevant European Regulations on this field are included in the FDA, so we consider the second ones as more general and complete to follow.

Before we describe the regulations applied to our system, we should first define whether our system is considered open or closed. According to the definition in 21 CFR 11, "*closed system means an environment in which system access is controlled by persons who are responsible for the content of electronic records that are on the system*" [1]. When this is not the case, then the system is considered open.

3.1.1 Part 11 – Electronic Records; Electronic Signatures

One of the most prominent FDA regulations is securing of the electronic records through the processes and controls of the system. The term "*electronic records*" is defined as "*any combination of text, graphics, data, audio, pictorial, or other information representation in digital form that is created, modified, maintained, archived, retrieved, or distributed by a computer system*" [1]. The other term that we are going to use is defined as "*digital signatures*", i.e. "*electronic signature based upon cryptographic methods of originator authentication, computed by using a set of rules and a set of parameters such that the identity of the signer and the integrity of the data can be verified*" [1].

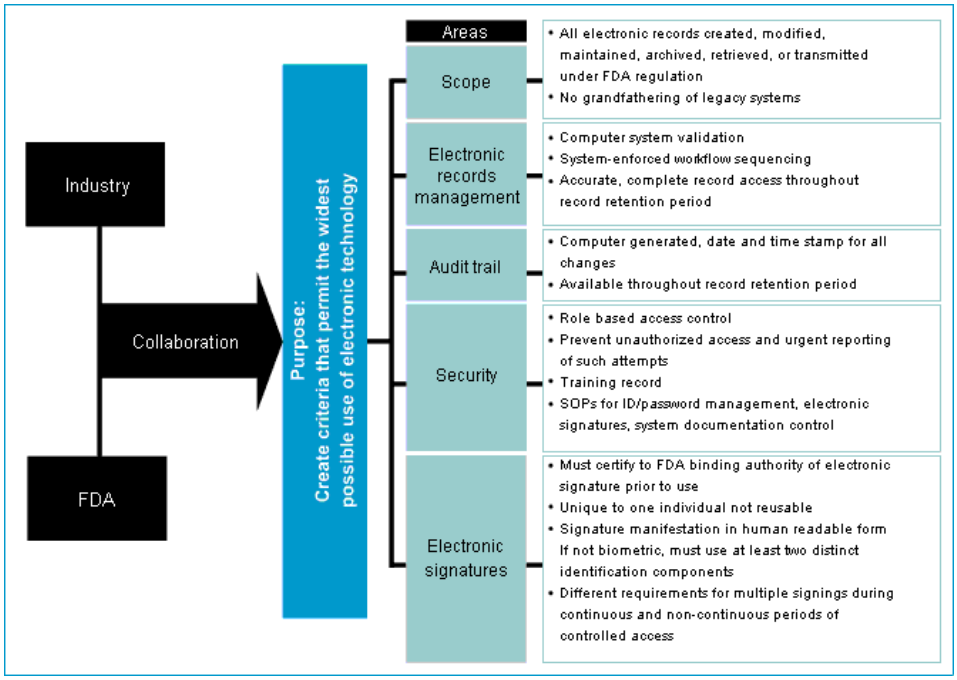


Figure 3.1: Scope of 21CFR Part 11 Requirements [41]

The scope of part 11 of the FDA regulations is to identify the criteria which characterize the electronic records as trustworthy, reliable and in general equivalent to paper records. The type of electronic records discussed here includes those created, modified, archived, retrieved, transmitted, or submitted to the FDA agency. But it does not refer to paper records. It also concerns electronic signatures related with records and requires that they are equivalent to hand written signatures. Finally, the scope includes the computer systems, the controls and the documentation, as they affect directly the electronic records, and therefore, should be available for inspection by FDA any time [1].

In case the electronic records are maintained but not submitted to the FDA agency, they can still be used, if they satisfy the requirements of this part. In case the records are submitted to the agency, they can be used, if, apart from satisfying the requirements of this part, they fall into the submission type accepted by the agency. More specifically it should be defined in the agency which type of submissions can be accepted without paper form and by which agency units they can be accepted. All these rules are documented, so that people can easily find out how (i.e. method and media of transmission, file format and technical protocols) and whether to process a record [1].
The requirements that should be fulfilled in an open system, according to 21 CFR 11 about electronic records and signatures, is ensuring their authenticity, integrity and confidentiality (from their creation until their reception). The way to do this is dealing with the following matters:

- We should be able to create accurate and complete copies of our records for any use, e.g. inspection and review by FDA.
- Any operator entries and action that may affect (create, alter, delete) the records should be also recorded with secure, time-stamped audit trails automatically created by a computer system. Moreover, any changes to records should not affect older entries.
- There should be written policies that account users responsible for their actions over the records.
- The users with access to the records should be of limited permissions.
- Persons who use, develop or maintain the records should have the appropriate education.
- The system should be validated so that its performance is accurate, reliable and consistent. It should also be able to find out whether a record is altered or invalid.
- We should ensure the sequencing of the events in the system by operational system checks.
- We should also perform authority checks to ensure that only authorized users can perform actions to the system, such as its use, signing or altering records, or accessing input and output devices.
- We should ensure the validity of the data source, by device checks.
- We should perform controls concerning the documentation of the system operation and maintenance and the way it is distributed, accessed and used. We should also ensure that its development in an audit trail is time-sequenced.
- We should use document encryption and digital signatures for the records. Signing relates a record with the signer’s name, the date and time this happened and the reason.

- The records should also be protected during their retention period, so that we can accurately retrieve them.

Finally, we mention the most important requirements for the electronic signatures:

- Electronic signatures should be unique and bound to an individual, which means only he can use them. And they cannot be reused by anyone else.
- The individual's identity should be verified before an organization establishes and certifies his signature.
- Each signature should consist of at least two distinct components, a code (id) and a password.
- In case a user performs a series of signing for a continuous period, then the fist signing must be using all signature components (i.e. both id and a password), but the subsequent ones can be performed using only one. In any other case, both signature components should be used for signing.
- Security and integrity of ids and passwords should be controlled by keeping their combination unique. They should also be checked, recalled or revised periodically. Their transaction should be secured to prevent their unauthorized use. Moreover, attempts for unauthorized use should be detected and reported.

3.1.2 Part 820 - Quality System Regulation

The scope of this part of FDA regulations is mainly focused on the production process of medical devices and not on drug products. This is also of interest since a Facility Monitoring System (FMS) system is responsible for device production as well.

It is mentioned that the systems that control the environmental conditions (i.e. sensors) should be periodically checked to ensure that they are functioning properly and accurately. The manufacturer should also ensure that they are properly placed and installed in the area to be measured.

Finally, all the computerized systems used for automated processing of data which are used in the production or quality control process should be validated. They should be working according to the established protocol. Moreover, in case of changes in the software, these systems should be validated again and their results should be documented [2].

3.1.3 Part 210 – Current Good Manufacturing Practice in Manufacturing, Processing, Packing, or Holding of Drugs; General

This section covers the methods, facilities and controls used for the manufacturing, processing, packing or holding of drugs or biological products, or human cells, tissues or their products (HCT/Ps). These processes also include packaging, labeling, testing and quality control of the drug. In case a failure to comply with these rules occurs, then the act, the drug and the person responsible will be handled according to the regulations. The rules in general should supplement and not conflict each other. In case of a conflict, though, the more specific case supersedes the more general one.
This section also provides some definitions of terms used later in sections 211-226. The terms that interest us are the following:
“Act” refers to the Federal Food, Drug and Cosmetic Act.
“Batch” is used to characterize the quantity of a drug or material with uniform behavior and quality (in specific limits). A batch is produced in a single manufacturing cycle.
“Component” refers to an ingredient which will be used in the manufacture of a drug product.
“Lot” means a batch or part of it or, in case of a drug, then it is an amount of it, which has uniform behavior or quality in a certain time or quantity of its production.
“Quality Control Unit” refers to any person or organization department responsible for the quality control. It ensures that the facilities, practices, records, personnel, methods and controls are following the regulations. Among its responsibilities is ensuring that any deviations from the standard procedures is done under proper authorization and is adequately documented. It should also assure that the reports are accurately describing the methods, procedures and raw data of the study. [27][10]

3.1.4 Part 211 – Current Good Manufacturing Practice for Finished Pharmaceuticals

The scope of this regulations part includes the minimum good manufacturing practice for drug and biological products, human cells, tissues and their products (HCT/Ps).
First of all, as concerns the personnel, there should be a quality control unit to examine the drug product during all its production phases and approve or reject it. This unit has the authority to review the electronic records of the production any time. Moreover, all the facilities for testing the product should

be available to this unit, which should examine all the procedures that impact the drug product on its identity, quality, purity and strength.

As concerns the facilities, there should be of suitable size so that the equipment installed there later should not risk any mixing or contamination of the drug product. Furthermore, there should be control systems in order to prevent product contamination. So, among the procedures to be controlled, the ones related to measuring equipment are:

- control procedures and laboratory operations, and
- aseptic procedures, which include the control of the temperature and the humidity in the room, filtering of the air, a system to disinfect the room, a system to monitor the environment conditions and a system, which maintains the previous one.

Among the various control systems, there should be systems to control the clean room temperature, humidity, air pressure, micro-organisms and dust during the manufacturing, processing, packing or holding of the drug product.

As concerns the equipment, its design, size and location should be appropriate for its use, cleaning and maintenance. Written procedures should be followed for cleaning and maintenance and the results should be kept in records. More specifically, automatic, mechanical or electronic equipment, and computers can be used in production, packing and holding of drugs. Therefore, they should be calibrated, inspected and checked to ensure that their performance is the expected one and keep records of the results. Any changes over the control and production records should be performed only by authorized users of the computerized system. The input and output of these systems should be checked for accuracy and the degree and frequency of their verification should be based on the complexity and reliability of the systems. We should keep a backup file of the data except for those eliminated by a computerization, which should be written records.

As concerns the production and process controls, they should assure that the product is produced properly. In case any changes occur, they should be approved by the quality control. The procedures followed during the drug production should be written and any deviation recorded and justified. Moreover, all equipment, processing lines and containers should be ready any time to identify their contents and the processing phase they are. The identification can be done by id number or name, in case of only one equipment piece.

We should take into consideration the laboratory controls used for test, sampling, specifications and standards. Among others, the instruments, apparatus, gauges and recording devices should be calibrated according to written procedures. These procedures would include any schedules, accuracy limits, remedial actions needed to be taken etc.

Moreover, among the records of the drug products, we should also include records for equipment cleaning, maintenance and use, which contain information about the date, time, product and lot number of each product batch [26].

3.2 User Requirements

In this section we describe shortly using an example of the user requirements specifications provided by a large pharmaceutical company concerning a Facility Monitoring System (FMS). The FMS system purpose is to document the temperature, humidity, differential pressure, number of particles etc. measured, and identify any excursion occurring. It consists of sensors connected to the FMS system and being stored in the FMS Server. The stored data can be viewed and operated by both the server and client interface. The sub-modules between sensors and server can buffer the data in case of failure in the communication of the other two modules. If a temperature excursion occurs users are informed by both sensors and server interface and the FMS system sends a notification to the Alarm system with the alarm cause and the sensor ID it originated from.

The user requirements are categorized in qualification (Q) and non-qualification, or in more correct terms into GMP-critical and non-GMP-critical. The characterization GMP-critical is given according to whether they need to be verified through qualification tests for GMP critical aspects. If GMP-critical requirements are not followed, we would result in a patient life-threatening situation. The previously mentioned requirements included in FDA 21 CFR part 11 and EU GMP Guide Annex 11 are considered GMP-critical.

For example, automation of periodic tasks and screen interface are non-GMP-critical requirements.

There are two types of measurements notifications, warnings and alarms. Warnings have stricter limits than alarms, and are recording unwanted but acceptable states, therefore are non-GMP-critical. Alarms are recording unacceptable states and, in case they occur, the storage conditions stated in GMP are not fulfilled and the product needs evaluation. So, they are considered GMP-critical requirements.

There should be also system alarms indicating failures in communication between modules of the FMS. As it is possible that records might get lost, so these alarms are considered GMP-critical.

One more type of alarm is power failure, which is also GMP-critical as records can be lost here, too.

Transmission of alarms to staff, though, is a non- GMP-critical requirement.

There are some things to be considered before we do the installation of an FMS system. These are

- the Configuration Item List,

- the Identification components,
- the equipment measuring temperature (and other environmental properties),
- the software,
- the backup of the records,
- the physical layout and location of the network, and
- situations of power failure.

Every FMS should have a **Configuration Item List (CIL)**, which consists a qualification requirement. This list consists of documentation, software and hardware components and settings of critical parameters on them (which can often change), list of users and the rights each one has and finally a graph scheme showing the layout of the hardware and software components. This list should be always kept updated and controlled and is considered a document for configuration management of FMS.

The **components identification** is also a qualification requirement and its purpose is to label clearly and uniquely the components, so as to trace their usage and maintenance, manage their configuration and control the change of the system.

The equipment for **measuring various environmental properties** is investigated in matters of accuracy and design. The accuracy of the sensors is considered a critical requirement and includes (a) that they are calibrated at values (e.g. temperatures) which are specified in the CIL and (b) that they follow the requirements for the Maximum Permissible Error in CIL. The design, however, of the equipment is a not critical requirement. The design includes all type of equipment used for transmission, i.e. sensors, cables, transmitters and aims at ensuring their usability.

As concerns the **software**, we should cover different aspects of it. The operating system of the FMS server could be, for example, Windows 7 or Windows Server 2003 and the one of the FMS client could be Windows 7. However, which operating system is used depends mainly on the company standard and how spread the system is and is not a Q-requirement. The FMS server and client software should be though in accordance with the CIL and thus are Q-requirements. The initiative behind this is that it can be under configuration management and change control and therefore, should be able to ensure that its usage and maintenance are traceable. Finally, the language of the log screens and event logs should be either in the local language or preferably in English to cover the usability needs of the operators. This requirement covers also the need to present the above information to the vendors and to any given inspector, such as US-FDA, EMA, Novo Nordic etc. So, it consists a critical requirement.

An automatic periodic **backup** should be performed at least once per week according to the user requirements, although not a critical one. We should be

though careful to physically place the backup media in a location with restricted access control and away from the FMS server, which contains the original data. This is a critical requirement.
As concerns the **physical location** of the various network components, they should be isolated from other systems with a firewall, and placed in a physical isolation, where only authorized users and system can access the FMS and its data. The FMS server should also be placed in a room with restricted access. These consist Q-requirements. Finally, it is desirable but not essential that the server is able to print out any data needed.
The ability to be robust against **power failures** of the FMS server, base and sub-modules is a critical requirement. The goals are that we don't miss any electronic records, the alarms are fired immediately and logged in the alarm log and that the monitoring and registration of environmental measurements still keep being recorded even during a power failure.

Functional Requirements
There is also a list of functional requirements that should be satisfied. These include system access, user groups, system reports, data security, system time, data screening, print out of transferred data, alarms and warnings, and staff alerting.
The access in the system should be authenticated through a username, and a password, unique for each user. The password should contain at least 8 characters, both numbers and letters, both small and capital, and should automatically expire after 70 days. This ensures data security and integrity and therefore, is a Q-requirement for our system.
There should be user groups with different privileges on the system, so as to avoid any misuse of the system either it is done intentionally or not. Therefore, this consists a Q-requirement. The role "Viewer" can only see the data of the FMS system displayed on a screen. The "Super User" can additionally acknowledge and activate/deactivate alarms/warnings and change their limits and print reports. Finally, the "Administrator" has all the privileges in the system, including the right to grant user roles to users and shut down the server.
It is critical that system reports exist, which contain information of the users who logged in and out of the system, the time this occurred, and what activities they performed, so that we can trace them later. Additionally, it should contain user configuration data, with their privileges and also event logs. User privileges should be reviewed periodically. Finally the reports should be printed in a way that the pages are uniquely identified, as required by the Good Documentation Practice. System report is Q-requirement.
Another requirement that is characterized as qualification one is the data security. We should prevent any loss of electronic records by keeping a

backup and/ or mirrored disks and by quickly restoring the system (Disaster Recovery). The system should be self monitored and trigger alarms in case of failures (e.g. lost communication), which should be logged in the alarm log.
Critical requirement is that the time of the system follows the local UTC time. The system should be able to change time (a) whenever a change is performed and (b) automatically from normal to daylight-saving time. Finally, the system should ensure that the changes in time won't result into overwriting the time-stamped electronic records.
As concerns the data screening, we should ensure that the data on the screen are valid. The information presented is the sensor ID, the value of the environmental property measured and its unit. This is a Q-requirement. Another important but not critical point is the format of data presentation, which should be both in table and graphical format. Both of them should be able to show alarms and moreover the value readings should at least contain one decimal place.
Another issue to be considered is the print out of the measured environmental property (e.g. temperature, humidity etc.). It is critical to be in a readable format. This means that we should be able to print reports sent from one or more sensors for a period of our choice. And such a report should contain data about the sensors id, date and time in both tables and curves format, with the related alarm events. It would be convenient for review reasons if it is configurable in a way that it can create periodic reports and if these reports can contain graphs collecting the measurements of the same property (e.g. temperature) from all sensors, distinguished by color. However, this is a non-Q requirement.
Alarms and warnings of measured values should also be considered. An alarm or a warning is activated when the value of a measured property exceeds the specified threshold limit for longer than a certain period (suppression time). The alarms should display on user's screen and get registered in the alarm log, with data of the sensor ID, measured value and time. It should be possible to change the alarm or warning threshold and the suppression time. The alarms are considered critical requirements because they should be working and displayed correctly. The warnings though are not GMP-critical and are only useful from a business perspective.
The last functional requirement mentioned in the URS is the alerting of the staff in case of an alarm/warning. This is not though GMP-critical. We should ensure that the alarm event is transferred from the FMS Base to the Gatekeeper/Watch Staff, who in turn have a procedure to follow to handle it and an updated call list of persons to inform about the alarm. There should be also an acoustic and visual indication triggered in case of an alarm to inform users about it. Apart from that, an sms should be sent to the personnel responsible for handling the alarms, containing the id of the component which

triggered the alarm, the date and time, a description of the reason of the alarm and a configurable text. We should also keep track of user actions concerning the alarms. This includes logging the username of the person who saw the alarm (login/logout) and the alarm acknowledgement, the relevant actions (alarm activation/deactivation) and the date and time of this action. This is stored in a separate place in the log server.

Performance Requirements

The data sent from the sensors should be stored in FMS server as electronic records and should contain information about the sensor ID, the measured value and a timestamp on it. The interval between two sensor readings should be of specific time period for time-triggered type of sensors. The time period depends on the type of the measurement, e.g. 5minutes for temperature, 10sec for differential pressure etc. However, this is not the case in event-triggered sensors, where the requirement is to receive a reading if the measurement varied for more than some specific time period. The logging and storage of data is Q-requirement. It is also desirable that the FMS system can handle at least 10 sensors, but it is not GMP-critical. In case the FMS system covers a big area, such as a factory, the number of sensors could be up to several hundreds.

Finally, there are some critical requirements concerning the electronic records, which are compliant with 21 CFR part 11 and GMP Annex 11. These requirements include that the system should create accurate and complete copies of electronic records, which contain audit trail and metadata, such as, timestamp, sensor ID, unit of measure). The format in which data are saved should be non-proprietary, such as .xml, .csv, .pdf, and the data presentation is preferred in a tabular form. Their retention period should be at least 18 months and they should be available for retrieval any time. The retention period reflects the time expected the drug to be used in total from its storage until it reaches the pharmacy and later the patient's home, which is on average 10-15 years. This time period could be split into period when one has immediate access to the product (e.g. 18 months) and a period where one has access to the product back-up. Of course, it is probable that in such a long retention period (of 10-15 years) the FMS system is replaced. However, the data should be kept in the database server and be available for retrieval even if the FMS is replaced. The access to the records should be restricted to authorized users. They should be the only ones who can create and modify an electronic record. And in case a user creates, changes or deletes an electronic record, then his Id and actions should be recorded and time stamped. Any change in a registration should not obscure previous ones. As long as a record is stored, the system should continue keeping an

audit trail. User passwords should automatically expire after 70 days and can also be changed, if needed, by the system administrator. Attempts of using the system after a long period of inactivity should be prohibited. Moreover, any unauthorized access attempts should be prevented and logged.

It is GMP-critical that there is written documentation about procedures of system operation and maintenance, which include the password change, the creation and removal of users in the system (FMS and Operating System), the backup of stored records, the re-establishment of a disaster recovery backup, the report generation of the user setup and the system time synchronization.
Finally, it would also be useful if the installed software, were be available on a CD-R/DVD-R, but this does not consist a Q-requirement [28].

3.3 GAMP

The GAMP (Good Automated Manufacturing Practice) guide has been based on the GxP and therefore describes how a system should be in order to comply with the FDA regulations through validation plans and operational controls. In other words it examines how the validation framework of a computerized system complies with GxP specifications through its life cycle. It ensures that the system is scalable and includes the control of Quality Risk Management. We can evaluate the risk either on the whole system or on each component separately. However, in case a computerized system is part of an automated or a big manufacturing system, the validation should be applied as a whole to the system and not on the computerized system as a separate component.
The figure 3.2 shows the approach through which a computerized system can comply with its intended use in its life cycle. We should note that specification procedure should always be addressed by verification. Of course factors such as risk, complexity and system novelty may affect and cause variations to the above approach.

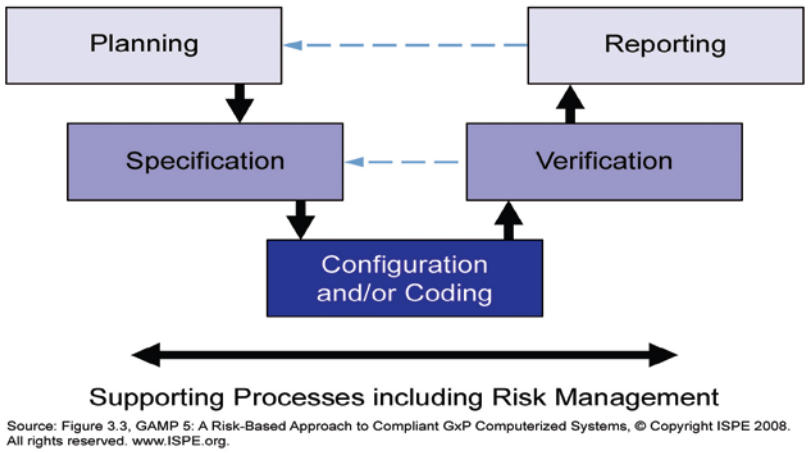


Figure 3.2: Life cycle of a computerized system

The **Life Cycle** of a computerized system, includes 4 phases; concept, project, operation and retirement.

1. **Concept** involves the activities needed to take place before starting a project, including initialization and justification. They are, however, out-of-scope for GAMP.
2. **Project** is divided in 4 sub-phases:
   - *Planning*:

   Here we specify all the user requirements (maybe we have made a short start at the concept level, but here is the main level where we develop them). They should take into account aspects such as risk assessment (which in our case translates into ensuring the data integrity) and the systems architecture, in matters of complexity and novelty. The requirements should also take into account responsibilities, procedures and timelines.
   - *Specification, configuration and coding*:

   The specification requirements should be provided by the customer and should in adequate detail so that we can be able to develop our system as we intended to, verify and maintain it. Configuration of the system should be performed repeatedly, while coding should comply with defined standards. The specification procedure could be either tightly coupled or separate from system configuration and coding, depending on the method the software is developed. In any case specifications should be maintained and controlled.
   - *Verification:*

   It includes all the reviews and testing done through the development of the system to ensure that the specifications have been met. Its purpose is

to identify errors and defects, so that they are corrected early. There are several types of testing and it is performed at different levels of the system. It should include various cases, such as the normal and the abnormal/invalid ones, system performance under normal and heavy load etc. All in all it should cover software, hardware, system configuration and acceptance.

- *Reporting and release*:

It involves the documentation process which should be approved before it is released. This documentation should include a summary of validation activities performed, any deviations from the plan and their corresponding corrective actions, and a statement of how suitable the system is for its intended use.

In any case, the requirements gathered should be of adequately detailed to be able to define our system. The more detailed they are, the more complicated and accurate our system will be.

Traceability has to do with the relation among products of the development process and its main purpose is to ensure that the requirements are met. Therefore, we should always keep traceability in mind when developing the user requirements. It can also help identifying the potential impact of a change in the system.

Functional Requirements:

They describe the system behavior and what functions and facilities it should provide. For every function in the system we should consider its aim and its impact on the other functions of the system, its performance (response, distributed processing etc), its security (failures, input and output values checking, access limitations, time-outs, redundancy, data recovery, self checking), traceability to user requirements, and any error conditions, failure actions, log files and diagnostics. As for the data handled in the system, we should consider first of all the manner in which they are defined, their access to data, the allowed range of their input and output values, any required fields, the relationships among the data and their validation checks, their integrity and security, their archiving and retention time. Finally, for the system interface we should consider how it will interact with the users, who will acquire roles to access it, how it will interact with equipment (e.g. sensors), and how with the other systems (concerning timing, communication policies etc.). For all kind of interfaces we should consider the “data transmitted and received, their type, format, range and allowed values, the interface timing, the transfer rate of the data, their communication protocol (initiation and order of execution), mechanisms for initiation and interruption, communication through parameters, common data areas, messages, direct

access to internal data, error handling, recovery and reporting, access and security".

The system software can be categorized by GAMP in three types, in order to select the appropriate life cycle activities for the system. The **main categories of computerized systems** are 1) infrastructure software (category 1), 2) non-configured product (category 3), 4) configured product (category 4), and 4) custom application (category 5).

- Category 1 involves infrastructure software, consisting of software parts linked together to form an environment, which supports other applications. This category can be divided in two categories of software; one is the "established or commercially available layered software", such as applications that run under the control of this software, operating systems, programming languages. The other category is "infrastructure software tools", such as network monitoring software, security software, antivirus.
- *Category 3* includes commercially available software, which runs on standard hardware components. These software products are used off-the-self, i.e. they are not configurable for a specific business process or we use the default configuration for this process (e.g. firmware-based software). The life cycle activities of this category include only User Requirements Specification and Requirements Testing.
- *Category 4* includes commercially available software, which runs on standard hardware components. It meets the user's requirements and the business process and the code is not altered. Examples of this category are the SCADA system, the data acquisition systems, simple human machine interfaces, etc. Their life cycle activities, apart from the activities for category 3, include also Functional Specification and Testing as well as Configuration Specification and Testing.
- Finally, *category 5* of GAMP, includes software products which are developed especially to meet/serve user requirements, for which no commercial solution is available. However, when we use custom software of hardware instead of standard then the risk of failure and defects increases. The life cycle activities followed in this category are as shown in figure 3.3 and include in general the specification of the user and functional requirements, the design of the system and the unit specification. Then we have the test that is relevant with each of the previous steps/stages.

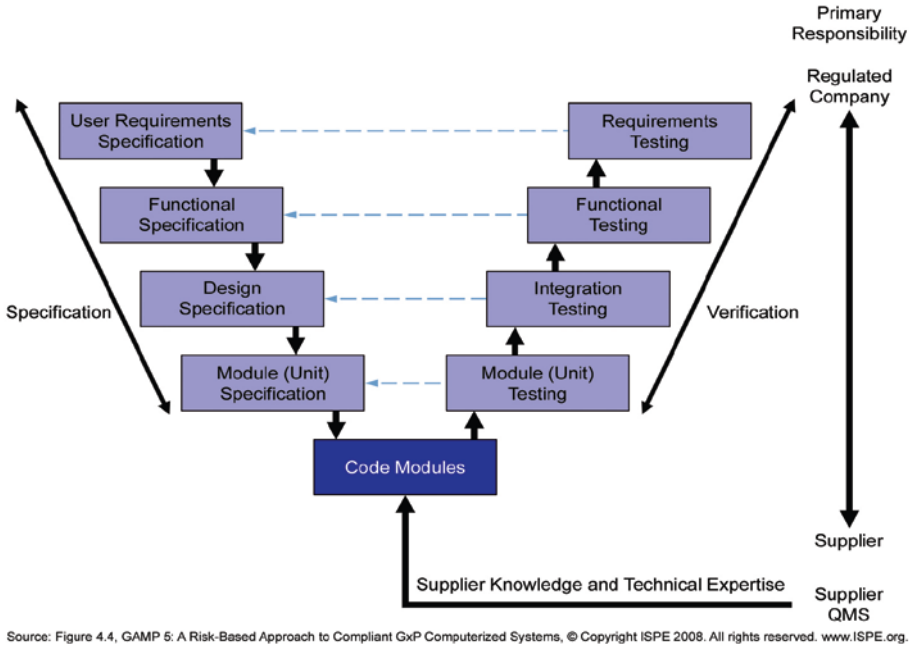


Figure 3.3: Life cycle activities of a computerized system of category 5

3. **Operation** involves ensuring that the operational procedures and plans of a system are implemented and work properly through the system's lifecycle. The operational processes that interest us mostly are those that affect or are related to electronic records.- One of them is management of support services for the maintenance of the system, and *monitoring of its performance* to detect system failures and avoid losing records. Another important process is incident management, which involves capturing events of abnormal behavior of the system. If a corrective or preventive action is possible then we use the CAPA (Corrective and Preventive Action) process to prevent the failure events from happening. Continuity management is one more operational process, which ensures the safe storing and easy retrieval of records as well as the existence of a business continuity and a disaster recovery plan in case of a system failure. We should also implement *security management*, including role-based access, system access with usernames and passwords in order to protect the data, with an administrator role being responsible for them. Finally, we should mention the *records management* process, which involves actions concerning records storing and retrieving as well as their retention.

   Let's take a closer look at some of these operational procedures:

- **Performance monitoring** is responsible for gathering data that can be useful for identifying system problems on time. The parameters, which should be monitored are all computerized systems (servers, PCs, control systems, workstations), the network and the applications. We should monitor computer systems in terms of CPU and cache use, their response and transmission time, system error messages, alarms, existence of critical processes etc. A network should be examined on its load and broadcasts and the availability of its components. Finally the applications should be monitored on their error messages, response times, availability and number of users using them concurrently. We should keep a monitoring plan for each parameter we examine, to keep data about its occurrence frequency, warning limit and notification mechanism, monitoring tool, secure storage of results and its time period.
- **Security Management** is the process for ensuring confidentiality, integrity and availability in an organization system. Its purpose is to minimize any security vulnerabilities. The goal is to comply with the GxP regulations and ensure that the data are protected against failures that might cause their loss or damage, and also from unauthorized editing. The general countermeasures that should be taken are defining user roles with different access permissions, performing periodic testing and security controlling of several security critical aspects of the system (e.g. system access), keeping an updated list of authorized users to our system, taking action against any security weakness of the system. Other topics that should be covered by security management are the physical safety of the data, the systems used for sending messages, the shared network and mobile computing resources, internet access and its use, communication with external computing systems and anti-virus policies.
- **Records Managements** involves the manipulation of the so called **Electronic Production Records** (EPR). These are divided into two categories Electronic Batch Records (EBR) and Electronic Device History Record (EDHR). Their purpose is to ensure that the data are collected as supposed to and that the equipment is manufactured accordingly. One method to perform this procedure is Review By Exception, where data from the manufacturing operation are screened, so that we can later check that everything is as expected (no exceptions, deviations etc). To implement this process we need to define the system functionality, accuracy and reliability, retain data intended for review, keep accurate computer means, review and check periodically the RBE functionality, check its communication

with other systems, check for any errors occurring. Any exception should include information sufficient to recover any records needed. Therefore, according to GAMP we should ensure the accurate recording of data and events, that processes are within defined limits, the monitoring of data and the process rates, correct generation of alarms, accurate, trustworthy and secure electronic records.

4. **Retirement** includes the procedures undertaken when the system is too old and needs to be replaced. This includes cancelling of all active operations (e.g. users deactivation), shutting down of the system, destruction of data, documentation, software and hardware, and, finally, transfer of data to the new system.

Changing the management of the computerized system can improve its **efficiency**, which includes several things such as our system being able to scale and keeping the specifications current through it whole working life etc. The system can become more efficient also by establishing requirements concerning data archiving structure and format; the database may archive records by retention period to make it easier retrieve data later on, but this may involve a complex design.

As we already mentioned in the first paragraph of this chapter, every computerized system should follow a validation plan to ensure that it is compliant with the GxP specifications. As part of the **validation plan** for a computerized system, we should have a system overview including its business purpose and intended use and, finally, a high level description of the system and its architecture.

Part of the validation procedure is the **Quality Risk Management**. Every system should be analyzed in terms of quality risk, as part of its life cycle activities. In order to perform Risk Analysis, generally, we should take into consideration the system impact on the patients' safety, product quality and the data integrity. This is accompanied by identifying also the controls for managing these risks, by either eliminating the probability that they happen or managing their negative impacts quickly and effectively. More specifically, the factors to be considered are (a) the possible failures of our system/automation process, (b) their consequences, and (c) their impact on the data integrity. We should also consider the way to detect the failure and how to prevent it from happening or at least eliminate its possibilities. First of all, we should verify if our system is GxP regulated. In this case, we should mention the specific regulations and which parts of our system they affect. This procedure is known as Initial Risk Assessment. Functional Risk Assessment is the process needed afterwards to identify the risks on the data integrity. It is focused mostly on the functions with the highest impact on the system. Later on, it investigates how easy it is or not to detect a function or

how probable it is to occur. This way we can use resources more effectively. After all, our aim is to reduce the risk of a failure. In case we cannot prevent a failure from happening, we then try to make it easier detectable. Usually system controls are automatic, e.g. alarms, restrictions to data fields, data prompts for identification/verification etc. Similar risk analysis we should perform when scaling the system activities (e.g. code reviews, design reviews etc), as it might impact the data integrity, the complexity of the system. We can see the full risk analysis plan on figure 3.4.

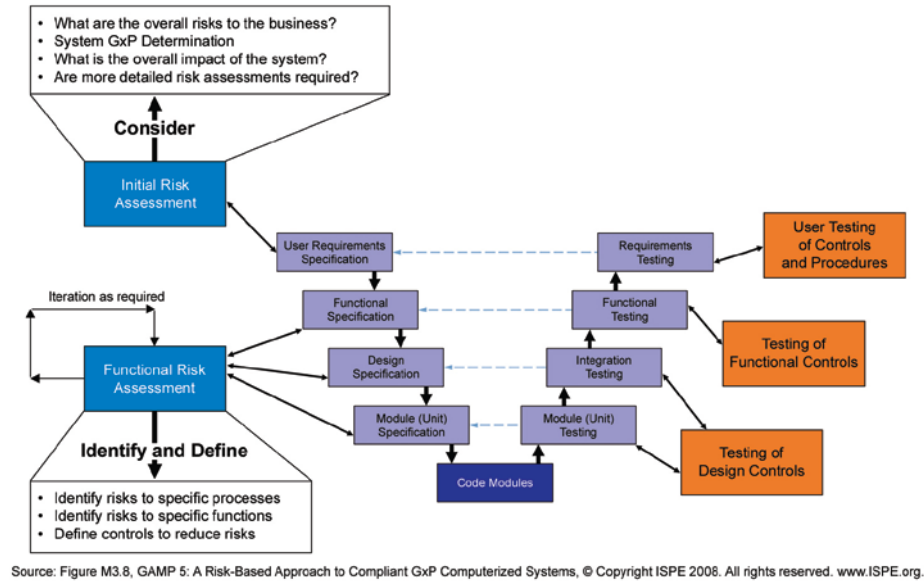


Figure 3.4: Risk analysis plan

An example of a risk, could be a failed recording. It could either have a medium impact as an inadequate change in documentation of a more severe one such as loosing old versions of data or inadequate attribution of data changes. Another example could be a failure of a function with acceptable data (in a specific range), which could be of low risk if we accept normal data only, of medium risk if we examine the boundary and null values, and of high risk if apart from these we also examine decimal precision and alpha characters [3].

3.4 Sensor Requirements

Here we describe the requirements of selected sensor and transmitter equipment from several Danish and international vendors. The equipment is just examples used to identify the requirements that they can impose to our system.

Testo

One famous equipment provider is Testo. More specifically, we will examine the ***Testo 6351 Ethernet differential pressure transmitter*** and the P2A software that accompanies it.

This transmitter is suitable for applications using Ethernet connection, such as “Clean rooms, Complex room climate applications, Monitoring flow velocities of volumetric flow rates in air conditioning systems” etc. Its functionality includes transmitting the signal via analog outputs to a control unit, recording, documenting and screening simultaneously the measured data and producing an alarm, if needed.

According to the Guide to the Expression of Uncertainty in Measurement (GUM) the measurement accuracy of the differential pressure has ±0.8% of the measuring range of the final value and ±0.3 Pa intrinsic error. The temperature deviation from the nominal temperature 22 ^{o}C is 0.03% of the measuring range per degree Kelvin. Finally, the corresponding temperature deviation for zeroing with solenoid valve is 0%.

The measurement cycle is 1 per second. The interface is given via the P2A software. The voltage supply for the 4 wires is 20 to 30 V AC/DC and the electric current consumption 300mA. Its resolution is 12bits. The operating temperature of the device is from -5 to 50^{o}C and the storage temperature from -20 to 60^{o}C.

The **relay** properties (collective alarm, switching limits and hysteresis) are set via the display or the P2A software. A Testo 3651 device has 4 relays. The connectivity options are NC=normaly closed contact/root/pin, C=closed, NO=normally open contact.

The parameters displayed by this equipment is differential pressure (in Pa, hPa, kPa, mbar, bar, mmH2O, inch H2O, inch HG, kg/cm2, PSI), flow rate (in m.s, ft/min) and volumetric flow rate (in m3/h, l/min, Nm3/h, NI/min). The flow rate is expressed as differential pressure >0.2Pa or 0.1% of the respective measuring range in order to avoid fluctuations of the flow rate at zero point.

As for the **scaling**, i.e. the minimum and maximum values, there are three types. One is the measuring range, which defines the maximum performance of the sensor included in it. However, any values outside it are not included, but only sent as messages. The other type is the standard scaling, which means that the output signals are assigned to the measuring range during delivery and that this range is applied as standard after the exchange of the unit. The transmitter keeps its scaling even if it is not connected to voltage. The third one is the maximum settings for performing manual scaling, where we can calculate the maximum limits as the difference between min and max value and it is also possible to scale beyond the measuring range to adjust to the standard values of a PLC.

There is a light indicating whether the transmitter is switched on and either expresses its status or is an alarm indication. It is not possible to monitor the alarm circuit through the switch.
We can connect an **Ethernet module** on the transmitter. It provides two functionalities which can be determined using a DIP switch. The one is that the transmitter becomes a so called Saveris subscriber, defined and set as default option by Testo, and the other one is that the transmitter can be integrated into any Ethernet system, as an XML server. For the Saveris subscriber an IP address should be set through the P2A software from its menu. For the XML server functionality, the user should know how to read an XML file, which will be downloaded and decoded. The communication in this case resembles the client-server paradigm. The **XML interface** is accessed through a URL, which gives us the IP address, the XML document path and any related parameters, if required. Using the Ethernet module we can read the measurements (value as numerical decimal number and unit in ASCII), the instrument type (device id as numerical whole number), the firmware date (year, month, day as numerical whole numbers) and version (in a 6 characters ASCII), the status (0 if no messages, otherwise 1) and related messages (messages in ASCII and their serial number which has 8 characters ASCII), alarm messages (message in ASCII and state as numerical whole number 0 or 1), service hour counter (numerical whole number), as well as both read and write adjustment data, analog outputs, relays data (channel, number 0 to 4 and delay as numerical whole numbers) and user settings (pressure, min, max, mean readings as numerical decimal numbers and connector and channel type in ASCII). An XML tag can also contain other information like the units of the measured values, like Pa for pressure, ^{o}C for temperature and %RH for humidity and all of them expressed as numerical, decimal numbers. An example of a measurement of these values is 50.0%RH humidity and 22.0^{o}C temperature.
The properties that identify a transmitter are, therefore, the instrument type (6351), the build number, the firmware version (1.02) and its serial number. Furthermore, a transmitter can display three types of messages; status, warning and error. They are all stored in the device with an operating hours stamp. The status messages are related to initialization of the device and changes in parameters, the warning messages are related to values limits being exceeded and the error messages apart from exceeded limits recognize also watchdog errors.
The **alarm** can be used in two modes; either to monitor limit values or as a collective alarm. When used for setting limits we choose the min or max and set the limit and delay of each alarm. A delay can be of 0 to 240 seconds and has no effect on the collective alarm. If someone turns off the alarm status then both visual alarm and relay connection are prevented. An alarm is

turned off upon acknowledgement. Triggering of a new alarm is not noticeable while the alarm status of another one is on. In case many alarms are triggered simultaneously, only the last one will be shown.

The **P2A software** is installed separately from the transmitter and the operating system required for the installation can be Windows 2000 SP4 or XP Home/Professional or Vista. The P2A software is responsible for creating an instrument and a parameter file. The first one contains historical logs of the parameters of a transmitter and can be edited. The second one doesn't relate to only one transmitter and doesn't contain any historic data. It contains information of the parameters of transmitters of the same type. These files are characterized by the instrument type, serial number and firmware version of the instrument and the current date/time stamp. The readings are updated every second. As for the analog output it is expressed in V or mA and has 1 decimal place. It also informs us whether the connection to the PLC is closed, so a value is transmitted or not. It saves the maximum and minimum values of a channel. Any parameterizations, adjustments and messages are time-stamped with the operating hours. The first two are also related to the user who performed them.

A stable current reading appears after around 20 seconds. [42]

CHAPTER 4

Problem Analysis

4.1 Problem Overview

We have clean rooms in a building facility which is responsible for producing pharmaceutical products. Every clean room contains a production line of a specific product. We need to maintain stable the environmental conditions of the room, which differ from room to room depending on the production phase of the drug or the drug itself. Therefore, each room contains a set of sensors to measure the environmental conditions in it and also some other equipment that upon user's action can change these conditions if necessary. The values these sensors measure include usually temperature, pressure difference (between the inside and the outside of the room), humidity and number of particles.

The network architecture that is currently used by many pharmaceutical companies is as follows; sets of sensors are connected to Programmable Logic Controllers (PLCs), which sample the analogue signal of the sensors. The sensors can measure one or more values at the same time, e.g. temperature and humidity. The data after the PLCs are delivered to the SCADA system, which is a control system that reads the value from the PLC and analyses the incoming data in real time. The SCADA and PLC reading frequencies are not synchronized and the second one runs much faster than the first one. The FMS system, which is SCADA based can produce diagrams showing the measured values which just arrived from the sensors and also produce a warning if a measured value is approaching a limit which would be dangerous for the drug in production. And in case the limit is extended for an undesirable amount of time then it produces an alarm. These alarms can be either flash light or a loud sound or a phone call to the personnel in charge of interfering. In any case the alarm event is transferred to the user who can interfere to save the product using the Control System. This system is separate from the FMS system. The measurements as well as the warning and alarm notifications are all transferred to a log server which contains a database, where they are stored and kept there for as long as the retention

period lasts. There are also other types of alarms, such as alarms for power failures and alarms for failures in communication between modules of the FMS.

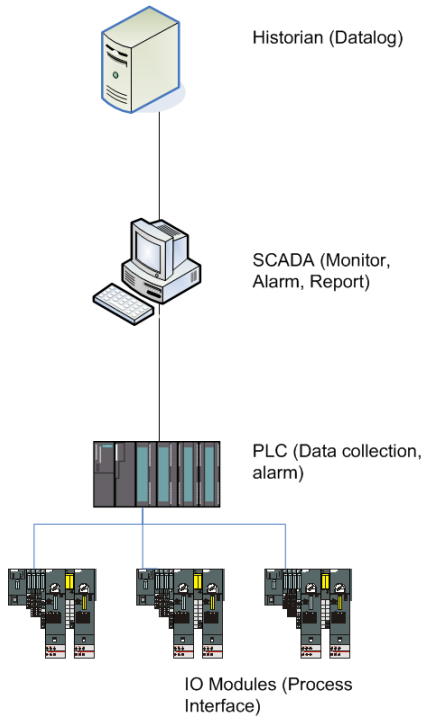


Figure 4.1: Current System Architecture

The system we just described is wired, time-triggered and soft-real time, but scaling is complicated. In case we need to extend the current system by adding additional sensors for example, then we need to ensure that the new system is compliant with the FDA regulations. This implies validation procedures not only to the new sensor device but also to the SCADA system it connects to until the measured values it sends are received by the database server. This involves additional costs, paper work and working hours and therefore is undesirable.

Once the data are transferred to the database, they are saved in the format they were received. No encryption mechanisms are applied. Only authorized users (e.g. administrators) have access to them, after they enter their user credentials. We consider that the users are trustworthy and don't tamper with the saved data. However, there is theoretically the possibility for a malicious user to log in and modify the saved data.

4.2 Possible Solution

4.2.1 Network

The solution we shall provide would be a wired system loosely coupled, that can be easily extended, and not include re-validation of the rest of the system to comply with the FDA regulations. For this purpose we will exclude the SCADA system from our network and consider an alarm application system which will check the measured values and raise an alarm in case they exceed their permitted limit. This alarm system would be separate from the database server and won't interfere between sensors and the log server. The system would also be asynchronous i.e. the messages are sent when they are published by the sensors and not upon the receivers request. Finally, we will apply a security scheme to encrypt the data upon arrival at the Log Server.

In the State of the Art (Chapter 2) we have identified the possible methods that would serve for the data transmission from the sensors to the database in a loosely-coupled way. The methods pointed out were the distributed shared memory, the message passing (point-to-point), the remote method invocation and the publish-subscribe.

The distributed shared memory is mainly used for communications in the system operation layer and is not suitable for distributed systems. The point-to-point message passing provides an asynchronous sending mechanism, and a synchronous receiving one. So, it is decoupled in the communicating components that offer the advantage of easily scaling, but it is not decoupled in synchronization. Furthermore, there can be multiple senders but only one receiver for each message. This would not serve for the implementation of the idea of having more than one possible receivers of the same message, i.e. the alarm system and the log server. The remote method invocation has many advantages such as easy implementation, flexibility and robustness and the sender-receiver communication is decoupled. However, the remote procedure calls used in the method are coupled in space, time and synchronization. For this reason, we avoid using this method. Finally, the publish-subscribe method provides an asynchronous communication in an autonomous manner. The communicating components are decoupled in both time and space. Moreover, each component can also perform other operations while communicating. This implies that it is scalable. Moreover, using the topic-based subscription the same message can be delivered to more than one receivers. It is also considered to work well with a database and provide good response times. The only disadvantage that we should keep in mind is that there are few ways for the subscribers/receivers to define in detail their interest criteria on a topic-based subscription model. However, this inefficiency cannot be of much impact if the criteria are simple.

Based on the above mentioned advantages and disadvantage of these 5 systems, we decided to use the publish-subscribe method as a communication method in our system. The sensors will be the publishers sending messages or in terms of this scheme we can say "publishing events".

These events will be actually the new values the sensors have measured from the environment. The reading of the values will be time-triggered from the sensors. Theoretically, there could exist the possibility that the sensors are event-triggered. But in the system we plan to build we will consider time-triggered ones. The alarm system and the log server will be the subscribers to the messages/events, according to some interest criteria. The publish-subscribe model will be topic-based, as we need more than one modules to receive the same message. We already know that there are few ways for the subscribers to define in detail their interest criteria on a topic-based model, but this doesn't consist a problem in our case, because the interest criteria we will use are quite simple.
After we had selected the communication, we examined in which way we would implement it. We have identified two ways to implement this; the java event bus and the java messaging service. In both of them the system components are very loosely coupled, which implies that they can be easily removed or added to a network at run time without affecting its core functionality. The communication is asynchronous and reliable, meaning that the event messages are sent when they occur and that they are delivered only once to each receiver who subscribed to them. However, both its publish() and subscribe() methods can only be called from the same thread, i.e. they are processed only locally by the application client that runs them. Also, we cannot have two separate applications calls, e.g. the sensor publishing the event and the server log subscribing to it. On the contrary, the java messaging service includes a messaging service which is responsible for creating, reading, sending and receiving messages among components, which connect to it. Moreover, it doesn't face the problem the event bus faces. So, this implementation will be chosen for our system in matters of communication.

4.2.2 Secure Logging

In the State of the Art (Chapter 2) we have identified the possible security schemes that could serve for logging the data in an encrypted way in the database, once they have arrived in the Log Server. The possible attack types that want to protect the log file from, are the following:

- insertion of a new entry
- alteration of an existing entry:
  - replacement of just a value of the entry (e.g. the measurement value, the sensor id)
  - replacement of the whole entry by another one (of e.g. the same time)
- reordering of entries
- deletion of an entry
- duplication of an entry

The security scheme of **Schneier and Kesley** has some vulnerabilities, which do not allow us to protect the log file from all the above attack types. An attacker, who modifies the log entries, is detected only under the assumption that the authentication key is not revealed to him. However, in case an attacker compromises the server log, referred to as untrusted in this scheme, it is unlikely that he learn the authentication key. So, he will then be able to modify the entries except for the MAC values. The attack will be finally detected by the Trusted Server, but only when he receives the log entries from the untrusted server to verify it. Moreover, if the attacker instead of modifying the entries in U, he just deletes the last ones, before they are sent to T, the second one will not discover that they are missing. However, as concerns the architectural requirements of this scheme, they are compliant with our system, since (a) it is not a problem to provide an online server, if necessary, and (b) the storage capacities a database offers should be enough to satisfy the storage requirements for the hash values and MACs of the log entries.

The **Bellare-Yee** security scheme has the same advantages and disadvantages as the one previously mentioned, but is older.

Ma and Tsudik suggests two security scheme implementations to overcome the limitations of the two schemes previously mentioned. One scheme is based on a private key and the other one on a public key. The scheme based on the private key is resilient to truncation attack, doesn't need an on-line server, it is storage efficient and provides message-level forward security. However, it is not resilient to delayed detection attack, and does not provide non-repudiation and public transferrable verification. The scheme based on the public key is resilient both to the truncation and the delayed detection attack, doesn't need an online server and provides interval-level forward security. Therefore, it should be preferred from the private one in case we encounter attacks which are detected late. However, it requires storage capacities on the verifier's side. It is possible for an attacker to tamper with tags, in case he obtains the signing key. Finally, this scheme is not suitable for applications where the data use is intensive, such as databases, and therefore is unsuitable for our system.

Instead **Ma** suggests two new schemes BM-FssAgg and AR-FssAgg, which are better than the previous ones in many factors, such as constant public key size, efficient aggregate verification etc. However, the implementation of these two schemes is trivial and the author is investigating a non-trivial one.

Finally, **Stathopoulos, Kotzanikolaou and Magkos** extend the security scheme of Schneier and Kesley so as to overcome its vulnerabilities. (a) They use manual off-line signatures (created by the system administrator in the Untrusted Server) to protect the log files from modifications after the file has been signed and the signatures have been sent to the Regulatory

Authority RA. This method protects our log files from external and some internal attacks, but there is no guarantee for the security of the file before it is signed. Therefore, in addition to (a), they suggest (b) automated signing of critical log events, which are sent to RA for storage. This way the time available for an attacker to modify the file before it is stored, is very low. Additionally, the RA requests, at random time periods, signed versions of the current log entries in the untrusted server. For an attacker to intercept the automated signing and alter the log files before they are sent to RA, means that he needs to compromise not only the Provider/-s, where the log server is, but also the Alarm Services, responsible for handling the critical events. And even if he compromises all these systems, it is still uncertain that his actions remain undetected.

To sum up, because of their security vulnerabilities, the schemes of Schneier and Kesley and Bellare-Yee are unsuitable for our log system. However, their architectural requirements are compliant with it. As concerns the scheme of Ma and Tsudik the version based on the private key seems to face also some security issues, but the one based on the public key overcomes them. However, the second one is not suitable for use in databases, and therefore cannot be used in our system. Finally, as concerns the scheme of Stathopoulos et al. it is based on the one of Schneier and Kesley, but extends it so as to overcome its security issues. Therefore, we decide to choose this scheme for our system.

4.3 Requirements Analysis

To validate the proposed solution we have presented the FDA regulations in the System Analysis (Chapter 3) of this report and we want to see how they would comply with the publish-subscribe paradigm and what they would require from the solution we wish to implement, which we present below.

The FDA regulations in part 21 CRF 11 define the characterization of the system into closed or open according to whether the system access is controlled by persons who are responsible for the content of its electronic records or not. In our case, the system access is controlled automatically by the system itself, therefore we consider it as open system. This part 11 includes also the definition of electronic records as a combination of various information, presented in a digital form, which is created, modified, maintained, archived, retrieved, or distributed by a computer system. In our case this includes mainly text and data, as this is the form the information is transferred from the sensors until it is saved into the database. As concerns the electronic signatures we will use them when the user performs specific actions, such as changing the alarm limits, acknowledging an alarm etc. And it will include the user's input of a code, apart from his authentication in the system.

Some more issues to consider related to 21 CRF 11 are:

1. Our system is considered open; since it is wired there is no need of encryption on the transmitted data, but security will be applied to the record entries when logging them. However, data integrity and authenticity must be validated, therefore digital signatures will be used when needed.
2. Our system should be validated, which means that it should be accurate and reliable and its performance should be consistent, so that it ensures the transmission of data. This ensures that the data will be successfully delivered to the server, without losses or alterations. Since we use a wired network, the probability of making alterations to already generated data while transmitted, is not very high in most situations. It could be however the case that, since all our system components are wired, someone with physical access to the sensors could connect a device to the network and provide fake data. This can be traced later, when the production is finished and we check the log files. We can then find out whether there has been an extra sensor which wasn't supposed to be there. Finally, it could be the case that an invalid record is transmitted to our server, for example a sensor malfunctioning and sending one value which is extremely high, while the rest of the values are normal and similar. In this case, the system should recognize the invalid record, both by the alarm system and by checking at the log files of the records for their reliability.
3. Another issue stated in FDA, related with the above, is the protection of data from alterations after they are stored in the database. To ensure that we develop a security mechanism when writing down the records. They are encrypted in a way that if someone makes a change it will be easily noticed. A factor that helps in this is the fact that our records are time-stamped and therefore we can detect if a record is missing.

Furthermore in the GAMP guide the life cycle of a computerized system is described in four phases. In this thesis report we include most parts that the included in the project phase. In the planning phase we have defined most of the user requirements. In the specification phase we include some of the specification requirements concerning the sensors, but not the rest parts of the system such as the database server. In this thesis the requirements are given as general guidelines averagely requested by most of the clients, but the further details depend each time on the client needs. This doesn't allow building a standard system for all cases, but on the other side it permits our system be more flexible. We also perform configuration and coding of a prototype showing the functionality of the final system to be developed later. For the verification sub-phase the software application needs to be tested to ensure it is working as expected. As for the reporting and release sub-phase

it will not be included in the scope of this thesis. However, there will be a summarizing validation report to check, if our software performs as expected. Moreover, the computerized systems in GAMP are categorized in three types to select the appropriate life cycle activities. Our software implementation would be a mix of different categories, including operating system, database, possible integrated Java applications. As concerns the *operation* phase, the operational processes that interest us mostly are those that affect or are related to electronic records.

CHAPTER 5

System Design

5.1 Network

The system we want to build should be decoupled in space and time and synchronization, which means that sensors can be added in and removed from it without affecting the rest of the system. This way it can be easily extended and the validation of the new system focuses only on validating the new sensors, and not the entire system with all its components.

5.1.1 Architecture

In the figure 5.1 it is displayed the architecture of our system, after we applied the publish-subscribe paradigm, which, in the Problem Analysis chapter, was chosen as an ideal solution for a loosely coupled system. Using this message exchange method, the messages are sent when they are published by the sensors and not upon the receivers request. As we see in figure 5.1, the sensors, the Alarm System and the Server communicate with the JMS, which is provided through GlassFish server. GlassFish acts as an application server and can run over any network, so we can connect (using wires) the rest of the system components to a server which is running GlassFish and provides the JMS.

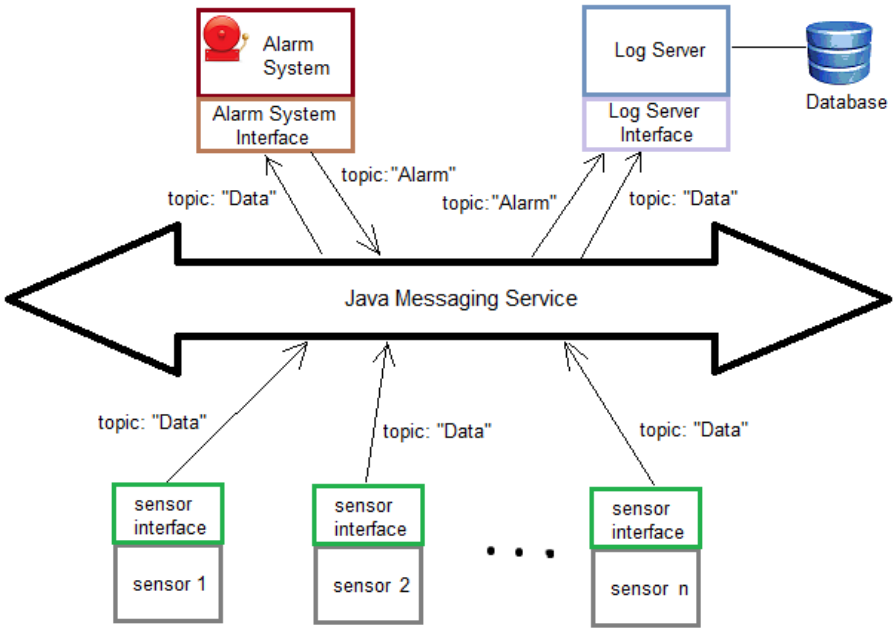


Figure 5.1: System architecture

5.1.2 Components

The components included in the above system architecture are described in detail below.

Sensors

The system will contain sensors which are wired. They can be of various types. In the market there are sensors which measure only one value type (e.g. temperature), and others which measure more than one value types at the same time (such as temperature and humidity). There are even more advanced sensors, which can evaluate their own measurements and send an alarm notification when a measurement exceeds a predefined threshold. So, it is possible more than one type of sensors to be in one room and function without causing any problems to the rest of the system. We also assume that they contain a clock to timestamp the data when collected. Thus, we will implement 3 different sensor applications that simulate the functionality of the sensor types we mentioned. We also assume that the sensor with an integrated alarm functionality, can notify only about alarms (but not about warnings).

Java Messaging Service

The Java Messaging Service (JMS) is a service provided by a server. In our case, we will use the Glassfish server, because, as mentioned in the JMS documentation, it works more efficiently with JMS. JMS acts as an event bus, routing the messages it receives to their destinations. We should not confuse JMS with the Java Event Bus. JMS is not aware of the ids or other details of the sensors that are communicating with it. It just knows the administrative objects in the JNDI namespace, which have been defined through the JMS Provider with the use of the Glassfish server interface. The one object is the connection factory, which we create as an instance of *ConnectionFactory*.*TopicConnectionFactory* interface, because we want a message in the JMS to be delivered to more than one subscriber. The other objects, that we define through Glassfish, are the destination objects, i.e. the objects where the publishers send their messages and from where the subscribers receive them. In the publish-subscribe paradigm these destination objects are called "topics". The topics we shall define will be simple, but able to serve our needs. We will use the name "Data" for any information being sent by any of the sensor types we mentioned and the name "Alarm" for any information sent by the Alarm System.

Alarm System

It is an application that reads all the sensor measurements after being sent by the sensors and evaluates them. If they exceed their warning limits, the Alarm System raises a warning notification. Similarly, if they exceed the alarm threshold for a time period more than the expected, it raises an alarm notification. These notifications can be a flash light, a horn sound, an email or sms notifying the person who is responsible for performing an action that would change the environmental conditions in the clean room and save the product before it is unsuitable for use. These kinds of actions are not performed through the alarm system application but another control system, which is separate from our architecture. The only action that could be related with the alarm system, could be the user acknowledging that he got informed about the alarm. This could be also an action performed in the control system. Apart from raising notifications the alarm system can also produce graphical representations of the incoming values. However, the incoming values are not saved in the alarm system, so these representations would be only real-time, thus, one cannot see the graphic scheme once it is out of the console output.

In the prototype we will build, we shall implement only the basic functionalities of the alarm system, which include receiving the sensor measurements, evaluating them and creating warning or alarm notifications when the measurements exceed their predefined thresholds. We also assume that the alarm and the warning (flashlight) are time stamped.

Log Server

The server receives all the data from the sensors and the alarm system and stores them in a database in an encrypted way. The goal is that no one can tamper with the logged data undetected. The data should be also available any time for retrieval by anyone upon request. The server should be able to present the data in a tabular format as well as a graphical representation. The server should also be connected to a printer to send a copy of its data. A back up of the logged data should be taken periodically.

In our prototype, we will implement only the most important of the functionalities described above, i.e. the server receiving the data sent by the sensors and the alarm system, and also the security scheme of Stathopoulos et al. that we chose in the Problem Analysis chapter. We will not use a database here, due to simplicity, but we will apply the scheme to txt files, where we will store our data.

5.1.3 Interfaces

Sensors

The sensor components would have a sensor interface application on top of them. This interface will have as input the sensor measurements and other data they provide and will publish these data, as a message to the JMS. A way to test this, could be by connecting one sensor to a PC, in a wired way, by using a USB Sensor Interface Module, such as those mentioned in [49] and [50]. Then in the pc we could have the application which serves as the sensor interface which will publish the data to JMS. However, considering that in a clean room we have hundreds of sensors, this suggestion would be a costly solution, so it is recommended only for testing reasons for a few sensors.

A more efficient solution would be developing a hardware module which contains the software that reads the sensor data and then connect to the JMS to publish them. This module could be added on top of the sensor components. Some sensor vendors, such as Testo, have already an Ethernet module, through which the sensor can send data. It could be an idea that we examined these kinds of modules and found out how we can use them to provide data to our sensor interface application. However, this is out of the scope of this thesis.

Alarm System

We will also develop an application which simulates some of the functionalities of an alarm system. This could be integrated in an already existing alarm system, to provide the interface for it to communicate with the JMS.

Server Log
The application we will develop will simulate the functionality of a sensor performing the basic actions we mentioned before. This application could be modified and part of it could be used by a server as an interface to communicate with the JMS. The security aspects of our prototype may be used in their present form, with the only exception of the file-based data storage, which should be replaced by a proper database server.

5.1.4 Interactions

Sensor-JMS
Sensors will be connected to the Java Messaging Service (JMS), which is provided through an application server running over any network. Therefore, the sensors need to connect online to the Glassfish server, through which the service is provided. The JMS is not aware of how many sensors are connected to it and its functionality is not affected when a sensor is added or removed from the system. Apart from the destination defined in the JNDI for the JMS, we also need to specify a physical destination for the clients to connect. In the client program, i.e. the interfaces of the sensors, the alarm system and the server, we inject also a destination resource, whose name is already defined in the respective resource in the JNDI namespace. These resources are the connection factory name and the topics names. Each sensor interacts with the JMS by publishing a message with one of the topic names defined. We decided that the messages published from the sensors will have the topic “Data”. A sensor doesn’t know what happens to a message after it will have been published. JMS takes care of its routing to the correct destination.

Alarm System – JMS
The Alarm System connects to the JMS to subscribe to the data it is interested in. It subscribes to messages of topic “Data”, i.e. all the messages the sensors send. After evaluating the sensor measurements, it publishes back to the JMS a message when a measurement exceeds the warning or alarm thresholds. The message contains the description of “WARNING” or “ALARM” and all the information that was fed in the sensor message. The messages the Alarm System publishes have the topic “Alarm”. The Alarm System is not aware of either how many sensors exist and send messages or who is subscribed to and reads the messages it publishes to the JMS.

Log Server - JMS

The Log Server also communicates with the JMS service and is subscribed to messages with topics “Data” or “Alarm”. Consequently, it receives the messages that the sensors and the Alarm System send, without, though, being aware of their existence.

Generally, we can connect any system to the JMS service, by just subscribing it to the data it is interested in. And if the component wants to send out some information, it can either publish it under one of the already existing topics, if their names describe it, or create a new topic in the JNDI namespace and publish data with this topic.

5.2 Secure Logging

In the figure 5.2 we see the complete secure logging scheme as described by Stathopoulos et al. and part of it by Schneier and Kesley. The Server Log mentioned above actually consists the so called Untrusted Server US in this scheme. In practice the US consists an application through which the log Server applies security to the log entries. It could be considered as a functionality of the log Server. We adopt the term “Untrusted Server”, because it is a server where the administrator and other users with access, can tamper with the log files. The Trusted Server is considered another server, separate from the Untrusted one, which is less likely to be compromised. This server establishes its first communication with the US before the second one starts saving the log entries. In this communication they create the first two initialization messages to be logged in the database. Moreover, TS learns the authentication key that US is going to use in his encryption mechanism. TS communicates again with US in the end of the logging procedure, to decrypt the data and verify the log entries according to Schneier and Kesley mechanism. The Regulatory Authority RA and the Alarm (Security) Service are also two different applications in different components, described by Stathopoulos et al. The RA creates accesses the log file in the database to create signatures of the current entries at random times. He receives also signatures of the log entries that the US administrator manually creates and sent to RA. Alarm (Security) Service creates automatically signatures of the current log entries when a critical event occurs, such as modification of the log files. He sends then these signatures to RA. In the end of the logging procedure, RA verifies all the signatures and detects if any attacks occurred to the log. All these systems can be wired among them. According to Stathopoulos et al. it is possible to have more than one Alarm Services, which detect critical events, such as log file modifications, system restart, modifications of users and their privileges etc. In our prototype we consider as critical event only the log file modifications.

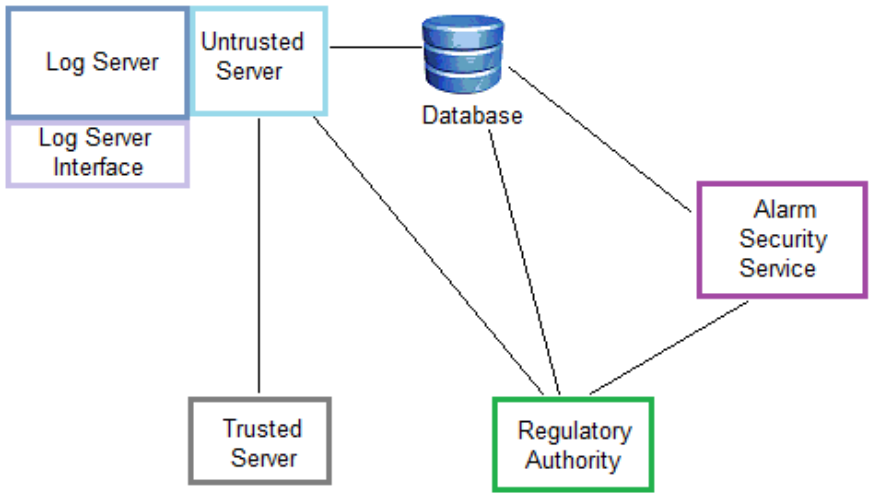


Figure 5.2: Secure logging scheme

Interactions

As we can notice in the scheme, the Untrusted Server communicates with the Trusted Server and with the RA (for the manual signing). The Alarm Security Service communicates with the RA and has also access to the storage area of the Untrusted Server, which contains the log entries. Finally, the RA component communicates with the Alarm Service as well as US and has access to its storage area (i.e. log files or database).

Assumptions

We consider the same assumptions as Stathopoulos et al. for their scheme. This means we assume that the communication interface between the Untrusted Server and the RA are safe. We assume also that the Untrusted Server has two public-private key pairs, which are not though stored in its memory storage because we want to avoid an intruder learning them if he compromises the Untrusted Server. In our prototype we have included these key pairs in the storage of the Alarm (Security) Service, as it uses them to sign the log entries of US. Finally, since in the prototype we will develop, we perform the communication of the above components with files exchange, we assume the existence of a common server or other computer area, where all the components of this scheme have access to. In our prototype, we will use Eclipse Workspace as this area and we will create separate folders representing the local storage for the rest of the components of the security scheme. In the future, in a complete system with database, we would consider another way to communicate the information between the different security components instead of files.

CHAPTER 6

Implementation

6.1 Network (JMS)

JMS Configurarion

We use JavaEE Eclipse with jre 7 and sdk 7. We also use Glassfish 3.1.2 as a server. After we have started the Glassfish server we use the Admin consol to access and define the administrative objects in the JMS Resources. We name the connection factory as:
JNDI name: “jms/GFConnectionFactory”
Physical name: “GFConnectionFactory”
And we characterize this object as an instance of “*ConnectionFactory*.*TopicConnectionFactory* “.
Then we name two destination objects:

1. JNDI name: “jms/Data”, physical name: “Data”, Type: “Topic”
2. JNDI name: “jms/Alarm”, physical name: “Alarm”, Type: “Topic”

Therefore, we have binded the administrative objects to a JNDI namespace. More information on how to install Glassfish and configure JMS can be found on section 6.3. [47]

6.1.1 SimpleSensor.java

The SensorObject.java file is an application client which acts as a publisher in the network, e.g. like a sensor publishing continuously data. We define the names of the resources to be mapped later at the destination resources in JNDI namespace

```
public static final String TOPIC1 = "jms/Data";
```

In the **main()** method we instantiate a sensor object and use it to get the initial context of the JNDI namespace [48]:

```
Context InitialContext = sensor.getInitialContext();
```

The function getInitialContext() sets some properties in order for this client application to connect to Glassfish server [48]:

```
properties.setProperty("java.naming.factory.initial",
"com.sun.enterprise.naming.SerialInitContextFactory");
```

```
properties.setProperty("java.naming.factory.url.pkgs",
"com.sun.enterprise.naming");
properties.setProperty("java.naming.provider.url",
"$$op://localhost:3700");
```

After connecting to Glassfish server, we inject (inside the main function) the administration objects created at the server, i.e. the destination resource "jms/Data" and the connection factory resource "GFConnectionFactory" [48]:

```
Topic topic1 = (Topic)
InitialContext.lookup(SensorObject.TOPIC1);
TopicConnectionFactory topicconnectionfactory =
(TopicConnectionFactory)
InitialContext.lookup("GFConnectionFactory");
```

Using these resources it is possible to create a logical connection (of type topic) to the JMS Provider, in other words after creating a connection factory object "topicconnectionfactory", we can use it to create a connection object. And afterwards, start the actual connection [48].

```
TopicConnection topicconnection  =
topicconnectionfactory.createTopicConnection();
topicconnection.start();
```

In the end of the main method we call the **publish (TopicConnection topicconnection, Topic topic)** method, which we have created providing the *topicconnection* object and the topic ("jms/Data") as parameters.
Our publish method uses the connection object ("*topicconnection*") to create a session object (*publishsession*). In the parameters of this method, we define that the session is not transacted (false) and that the acknowledgement of messages when received is done automatically (*Session.AUTO_ACKNOWLEDGE*) [48].

```
TopicSession publishsession =
topicconnection.createTopicSession(false,
Session.AUTO_ACKNOWLEDGE);
```

We use the session object to create a publisher object (*topicpublisher*). This line publishes an instance for the topic passed in the current method.

```
TopicPublisher topicpublisher =
publishsession.createPublisher(topic);
```

We then call the **getProperties()** method, which reads a *sensor1.properties* file to get the properties (id and transmission period) of a sensor.

```
properties.load(new FileInputStream(PropertiesFileName));
ensorid = Integer.parseInt(properties.getProperty("ID"));
period = Integer.parseInt(properties.getProperty("PERIOD"));
vtype = properties.getProperty("TYPE");
vunit = properties.getProperty("UNIT");
```

```
warnlimit =
Double.parseDouble(properties.getProperty("WARNING_THRESHOLD"))
;
alarmlimit =
Double.parseDouble(properties.getProperty("ALARM_THRESHOLD"));
alarmperiod =
Integer.parseInt(properties.getProperty("ALARM_PERIOD"));
```

The *sensor1.properties* file emulates a real sensor and the basic properties it might contain as a piece of hardware.

Then, in the **publish** method, we read a txt file which contains some measurement values in the way they would occur in a real environment. We have created 4 different types of files to simulate different situations occurring in a production environment. Assuming that we measure temperature we have the NormalTempValues.txt which contains measurements in the range of the normal temperature limits. The TestTemp.txt file is similar to the previous one, but contains fewer values. The AlarmTempValues.txt contains measurements which range from normal values up to values that exceed the permitted warning and alarm limit, causing initially a warning and afterwards an alarm. Finally, we have created theVariatingTempValues.txt files, which contain temperature measurements which range from normal values up to values of each the alarm limit, but quickly reduce, as if the system operator has changed the conditions to save the drug being produced.

```
FileInputStream fis = new
FileInputStream("C:\\Users\\Anastasia\\EclipseWSworkspace\\Inpu
t \\TestTemp.txt");
byte[] TByte = new byte[fis.available()];
fis.read(TByte);
fis.close();
```

After we read the file we save the input in a String.

```
String TMsg = new String(TByte);
```

We then use the break line character “\n” to separate the measurement values, which we save in a String array.

```
Tvalue = new String[0];
Tvalue = TMsg.split(new String("\n".getBytes()));
```

The publish method continues with a while loop, inside which we call the TemperatureSensor2() method to constantly get temperature values as a message:

```
message = TemperatureSensor2(count);
```

In case we don’t want to use the values from the files, but we wish to get random temperature values each time we can call the TemperatureSensor() which is now in comments:

```
message = TemperatureSensor();
```

And using again the session object we create a message object, which we instantiate with the sensor ID and message, which was just created. The

object is serializable, that means data are transferred as series of bytes. After we have instantiated the serial message, we save in it the values which we read from the sensor1.properties file about our sensor, such as the warning and alarm limits etc. Finally, we call the publish method with the publisher object we created before, and publish this message object. The message is sent to the JMS API characterized under the topic “jms/Data” [48].

```
objectmessage = publishsession.createObjectMessage();
serialmessage = new SerialMsgObject(sensorID,message);
serialmessage.setValue(value);
serialmessage.setAlarm(alarmlimit);
serialmessage.setAlarmPeriod(alarmperiod);
serialmessage.setWarning(warnlimit);
objectmessage.setObject(serialmessage);
topicpublisher.publish(objectmessage);
```

In case we have used the TemperatureSensor2() method, then the while loop continues to run until the measurement value it reads is “exit”. The message is sent to the JMS, then, we close the topic connection and exit and, thus, terminate the SimpleSensor.java application.

```
topicconnection.close();
System.exit(0);
```

In case we have used TemperatureSensor() method, the while loop continues to run until we stop manually the SimpleSensor.java application.

As we mentioned before, we have created a **TemperatureSensor()** method to take, periodically, temperature (and other) measurements, which are created in a random way. Below we present how this method works:

After the method is called we pause for a period*1000 miliseconds, which will be the time after reading each value:

```
Thread.sleep(period * 1000);
```

We then create an instance of a structure we defined called *temperatureReading*, which will be used to hold information about a temperature reading.

```
temperatureReading reading = new temperatureReading();
```

This entity contains information about the type of the value we measure, its unit and its measurement. In this example, we measure only temperature, so the type is “Temp”, but it can also be humidity (Humid), differential pressure (DiffPress) or particles (Particles).

Therefore, having this structure we can get the temperature value, calling the method **getMeasurement()**, which finds a random temperature with a mean of 25 and standard deviation of 5:

```
reading.measurement = getMeasurement();
```

Finally, we call the **printLogLine** method, giving as input parameters part of the information to be sent later, ie. the sensor id, the measurement type and the measurement value. The method returns the final message string, which is published as a message object.

```
msg = printLogLine(sensorid, reading.type,
Integer.toString(reading.measurement), reading.unit);
```

We have also created another similar method called **TemperatureSensor2 *(int c)***, which, periodically, uses measurements that we had previously read from the txt file and are now saved in a String array. This method reads a String array value, periodically, every period of *1000 miliseconds, as previously. And then the information is saved in a *temperatureReading* entity, too. We use a counter to go through the string array values and we increase by one the counter to read the next value of the string array every time the *TemperatureSensor2* method is called. Finally, we call the *printLogLine* method, as before. When we reach the end of the string array, we will read the "END" message. Then we save an empty measurement in the *temperatureReading* entity and produce an "exit" message which will be used later by the *publish* (*TopicConnection topicconnection, Topic topic*) method.

```
if(Tvalue[c].equals("END")){
	reading.measurement2 = "";
	msg = "exit";
} else {
	reading.measurement2 = Tvalue[c];
	count++;
	msg = printLogLine(sensorid, reading.type,
reading.measurement2, reading.unit);
}
```

Finally the ***printLogLine*** method creates a timestamp for the data to sent, emulating the fact of having timestamp the data sent by the sensor the moment it measures an environmental value (e.g. temperature).

```
SimpleDateFormat format = new SimpleDateFormat("yyyy/MM/dd
HHmmssSSSS");
Calendar calendar = Calendar.getInstance();
java.util.Date now = calendar.getTime();
String currentTime = format.format(now);
```

We then create an message object to hold locally all the values of a log line.

```
Logline log = new Logline();
```

We save in that object instance all the values of a final message [timestamp, sensor id, measurement type, value, measurement unit]. We use it, firstly to print the values to the console (so as to know what a sensor object publishes) and secondly to create the message string to be sent later as a serializable object.

```
String finalmessage = log.timestamp + "\t" + log.id + "\t" +
log.event_type + "\t" + log.value + "\t" + log.unit +"\n";
```

The Log line printed in the console, and therefore the message string sent has the format:

[timestamp (date and time) sensor_id measurement_type value measurement_unit]

For example,
`2013/08/02 1109290090 A11 Temp 22.0 Celsius`
where:
timestamp (date and time) = `2013/08/02 1109290090`
sensor_id = `A11`
measurement_type = `Temp`
value = `22.0`
measurement_unit = `Celsius`

6.1.2 SerialMsgObject.java

This file represents the object to be transmitted (sent and received) in a serialisable format. Its construction method saves the name of the client, who sent the message and the message itself in a string format.

```
public SerialMsgObject (String name, String message){
       this.name = name;
       this.message = message;
}
```

It contains data in String format, which are the name of the message sender, the message, the measurement value, the alarm period, the alarm and warning limits. It also contains set() and get() methods, that can be used by any other application to access these strings. This java file is exactly the same in all the applications that exchange messages between them, because it defines the format of the message, which should be exactly the same, when passed from one application to the other through the JMS API.

6.1.3 SensorwithAlarm.java

It contains almost the same code as the SimpleSensor.java file, but it reads another sensor properties file, *sensor2.properties*. It represents data sent/published by another sensor. It has a different sensor id but the rest of the properties could be either the same or different. This sensor application simulates a sensor which can also produce alarms, when the measured values exceed the alarm limit specified in the *sensor2.properties* file. This is implemented in the **printLogLine** method as shown below and determines the message being published:

```
if(Double.parseDouble(log.value)>(alarmlimit-1.0)){
     System.out.println(log.timestamp + "\t" + log.id + "\t" +
log.event_type + "\t" + log.value  + "\t" + log.unit +"\t"+
"SensorALARM"+ "\n");
```

```
     finalmessage = log.timestamp + "\t" + log.id + "\t" +
log.event_type + "\t" + log.value +"\t" + log.unit +"\t"+
"SensorALARM"+"\n";
}else{
     System.out.println(log.timestamp + "\t" + log.id + "\t" +
log.event_type + "\t" + log.value + "\t" + log.unit + "\n");
     finalmessage = log.timestamp + "\t" + log.id + "\t" +
log.event_type + "\t" + log.value +"\t" + log.unit + "\n";
}
```

The Log line printed in the console of a message, which contains an alarm notification, has the format:

[timestamp (date and time) sensor_id measurement_type value measurement_unit SensorALARM]

For example,

```
2013/08/01 2014370917     B12    Temp   29.1   Celsius
       SensorALARM
```

6.1.4 DoubleSensor.java

This sensor simulates the functionality of a sensor which measures and publishes two values, e.g. temperature and humidity. We provide to the application two different files: (i) sensor1a.properties, defining the properties for the temperature, and (ii) sensor1b.properties defining the properties for the humidity. The application publishes two different types of messages, one, containing information about temperature and another, carrying information about humidity. So, an example of the messages it sends could be:

```
2013/08/01 1836490300     C13    Temp   20.4   Celsius
2013/08/01 1836530303     C13    Humid  50.4   %RH
```

6.1.5 ServerObject.java

This application client represents the server who reads the messages from the JMS API. It is part of the ServerApp project and is located in the package *com.javamsgservice*. Its ***main*** method resembles a lot the corresponding one in the SensorObject.java. We instantiate here a server object instead.

```
ServerObject server = new ServerObject();
```

Before that, though, we also initialize an untrusted server object, which starts the procedures related with applying the security scheme we chose when we log the data.

```
us = new UntrustedServer();
```

But we also inject resources in the JNDI namespace to create a connection with the JMS Provider. The names of the administrative objects we define in

this file are the same as previously. We use also the getInitialContext method to connect to the GlassFish server and access the JNDI namespace.
The difference here is that instead of publishing data, we subscribe to them. More specifically inside the **main** method we use the server object to call a subscribe method we created

```
server.subscribe(topicconnection, topic1, topic2);
```

We pass the connection object (topicconnection) and the topic1 = “jms/Data” and topic2 = “jms/Alarm” as parameters to this method. The Server application subscribes to both topics, so as to receive messages published by the sensors under topic1 and to messages published by the alarm system under the topic2.
Inside the method we defined as **subscribe(TopicConnection topicconnection, Topic topic1, Topic topic2)**, we first create a session object (subscribesession) using the connection object. We define that the session is not transacted (false) and that the acknowledgement of messages when received is done automatically [48]:

```
TopicSession subscribesession =
topicconnection.createTopicSession(false,
Session.AUTO_ACKNOWLEDGE);
```

Then using the session object we create the subscriber object, which is an instance of the topic object (“jms/Data”) passed as parameter in this method [48]:

```
TopicSubscriber topicsubscriber =
subscribesession.createSubscriber(topic);
```

The subscriber object calls the onMessage method every time a message object is received [48]:

```
topicsubscriber.setMessageListener(this);
```

The same commands we use for topic2 [48]:

```
TopicSubscriber topicsubscriber2 =
subscribesession.createSubscriber(topic2);
topicsubscriber2.setMessageListener(this);
```

The **onMessage** method creates an instance of the serializable message object and reads its values [48].

```
ObjectMessage objectmessage = (ObjectMessage) message;
SerialMsgObject serialmessage = (SerialMsgObject)
objectmessage.getObject();
String text = serialmessage.getMessage();
```

We print these values in the console to see directly if they are the expected ones

```
System.out.println("Sender: "+ serialmessage.getName());
System.out.println("Message: "+ serialmessage.getMessage());
```

The output on the console should be of the format:
Sender: sensorID

Message: Timestamp (date and time) sensorID MeasurementType MeasurementValue
For example,
Sender: 11
Message: 2013/05/28 1123440115 11 Temp 30

Then, we save the received message as a string text line on a text file

```
output = new BufferedWriter(new FileWriter(file,true));
output.write(text);
```

Each log line saved should be of the save format as the way it is published on the console by the SensorObject.java
[Timestamp (date and time) sensorID MeasurementType MeasurementValue]
For example,
2013/05/28 1123440115 11 Temp 30.

Finally, we check each of the received messages, if they are the “exit” String. If they are not, then we give them as input parameter to the saveEntry() method of the untrusted server application, so as to save them in an encrypted way. When we get the 1st exit message, we mark the file as closed and we don’t let any more messages to be saved in the log file. Now the “END” string in the end of the measurements files is the one which triggers the “exit” message to be sent to the Server application. In real conditions of a drug production, this “exit” message could be triggered manually by the administrator.

```
if(text.equals("exit")){
   if(closed==false){
       us.saveEntry(text);  //save last entry, ignore next ones
      closed=true;
   }
} else {
   if(closed==false){      //save it only if file isn’t closed
       us.saveEntry(text);  // save entry using Untrusted Server
   }
}
```

6.1.6 AlarmObject.java

This application simulates the Alarm System, which is responsible for recognizing measured values,
which exceed a predefined value limit. It subscribes to JMS events of Topic "jms/Data" and evaluates them. If it recognizes an abnormal value it publishes the msg back to JMS under the Topic "jms/Alarm". The object messages it sends and receives are serializable.

We first initialize the alarm object inside the **main()** method, and then we use the getInitialContext() method to connect to the GlassFish server and access the JNDI namespace and, thus, the 2 topics and the GFConnectionFactory [48].

```
Context InitialContext = server.getInitialContext();
Topic topic1 = (Topic)
InitialContext.lookup(AlarmObject.TOPIC1);
topic2 = (Topic) InitialContext.lookup(AlarmObject.TOPIC2);
TopicConnectionFactory topicconnectionfactory =
(TopicConnectionFactory)
InitialContext.lookup("GFConnectionFactory");
```

Then we create and run a topic connection, which we use to subscribe to events of topic1 = "jms/Data" using the method *subscribe (TopicConnection topicconnection, Topic topic*).

```
topicconnection  =
topicconnectionfactory.createTopicConnection();
topicconnection.start();
server.subscribe(topicconnection, topic1);
```

In the **subscribe (TopicConnection topicconnection, Topic topic)** method we create a session, which we use to create a susbscriber object to call the onMessage() method [48].

```
TopicSession subscribesession =
topicconnection.createTopicSession(false,
Session.AUTO_ACKNOWLEDGE);
TopicSubscriber topicsubscriber =
subscribesession.createSubscriber(topic);
topicsubscriber.setMessageListener(this);
```

In the **onMessage()** method, we create an instance of the serializable message object and read its values [48].

```
ObjectMessage objectmessage = (ObjectMessage) message;
SerialMsgObject serialmessage = (SerialMsgObject)
objectmessage.getObject();
String text = serialmessage.getMessage();
String nameID = serialmessage.getName();
```

Among the values of the message object we get, are included the warning and alarm limits and the alarm period (where as alarm period we define the maximum time period we can have alarm conditions without exposing the drug production in danger).

We then save in an *AlarmSystemLog.txt* file the log entries the alarm system subscribes to.

Then we check for alarm events. If the measurement is above its warning limit, but below its alarm limit, then we call the *publish(topicconnection, topic2, serialmessage)* method, without counting continuous alarm events with the count value. If, however, the value is above its alarm limit, then we call the publish method and also save the value in *alarmtime* array, which maintains the timestamps of the continuous alarm messages. We access

them with the count value. If the time difference of the timestamp of the current message since the 1st alarm event (in a series of continuous alarm events) exceeds the alarm period, then we notify the administrator that the product has been exposed to unsuitable conditions for time longer than it can take and is potentially unsuitable for use. In any other case that the value measured is normal, nothing happens.

```
if((value>(warninglimit-1.0))&&(value<alarmlimit)){
       server.publish(topicconnection, topic2, serialmessage);
       alarmtime = new long[100];
       count = 0;
} else if(value>(alarmlimit-1.0)){
       server.publish(topicconnection, topic2, serialmessage);
       alarmtime[count] = Long.parseLong(log2[0]);
       long difference = alarmtime[count]-alarmtime[0];
       if(difference>(alarmperiod*10000)) {
              System.out.println("Product exposed to
unappropriate conditions for over its time limit. Product is
potentially unsuitable.");
       }
      count++;
} else {
       alarmtime = new long[100];
       count = 0;
}
```

Finally, the **publish (TopicConnection topicconnection, Topic topic, SerialMsgObject serialmessage)** method is called only on an alarm or warning event and publishes message objects under topic2 = “jms/Alarm”. Depending on whether the message informs about a warning or about an alarm being exceeded, a related message is added in front of the data string we received from the sensor.

```
if((value>(warninglimit-1.0))&&(value<alarmlimit)) {
       message = "AlarmSystem: WARNING: " +
serialmessage.getMessage();
} else {
    message = "AlarmSystem: ALARM: " +
serialmessage.getMessage();
}
```

Afterwards the alarm event is published, unless the data received from the sensor is the “exit” message, which we ignore.

```
if (message.equalsIgnoreCase("exit")){
       topicconnection.close();
       System.exit(0);
} else {
       objectmessage = publishsession.createObjectMessage();
       objectmessage.setObject(new
SerialMsgObject(sensorID,message));
       topicpublisher.publish(objectmessage);
}
```

6.2 Security

6.2.1 Package: com.Security.UntrustedServer

The following files are included in the package *com.Security.UntrustedServer.* We have created the folder USstorage to represent the local storage of the Untrusted Server. There we save some data such as keys that the Untrusted Server (US) creates and uses later while it runs. The communication of the US with Trusted Server is implemented with files exchange. The information each server keeps on its own storage cannot be read by others. The only exception consist the server public key, which can be accessed by everyone. When the two servers communicate they place files in the EclipseWorkspace. In a realistic scenario, instead of EclipseWorkspace we can use for example another server, where both servers have access to.

6.2.1.1 UntrustedServer.java

The core functionality of this application is to encrypt the entries received at the server. For encryption it follows the Schneier and Kesley security scheme. More specifically, it creates log entries:
Li = [LogEntryType, AES(Di)_Ki, Yi, Zi]
where

- Di = [current_timestamp, timeout_timestamp, ID_logentry, Message] where Message = the data we want to save
- AES(Di)_Ki = symmetric encryption of the Di string using the AES algorithm and the encryption key Ki
- Yi = hash(Yi-1, AES(Di)_Ki, LogEntryType) i.e. a value produced by applying a one-way hash function to 3 values
- Zi = MAC(Yi)_Ai i.e. Message Authentication Code, which is a one way function applied on the Yi value using the Ai authentication key.

The **constructor method** of the UntrustedServer class starts the secureserver.java application which waits input from the UntrustedServer.

```
Thread TS = new Thread() {
     public void run() {   secureserver.main();   }
 };
TS.start();
```

It then instantiates an InitializeLog object, which performs all the initial communication with the secureserver application and writes the first initial messages in the log file, before any entry is saved there. We will describe these methods in detail in the following sections.

```
InitializeLog ini = new InitializeLog();
ini.Init();
```

Before we continue further, we check whether the message M1 that the InitializeLog object received from the secure server application is verified.

```
M1verified = ini.getM1Ver();
```

If message M1 is not verified, then we get a notification and no logging activity takes place. If it is verified, then this means that the Trusted and Untrusted Servers exchanged their initial secret keys successfully and no intruder tampered with the initial messages they exchanged. So, the rest of the application continues running normally. Next the program asks the user to define the date and time the untrusted server is expected to timeout and stop working. This in real conditions could be the production end day.

```
System.out.println("Expected timestamp the Untrusted Server
will timeout (YYYY/MM/DD HHMM): ");
try{
    BufferedReader bufferRead = new BufferedReader(new
InputStreamReader(System.in));
    timeout = bufferRead.readLine();
    timeout = timeout + "000000";
    timeoutdate = new SimpleDateFormat("yyyy/MM/dd HHmmssSSSS",
Locale.ENGLISH).parse(timeout);
} catch(IOException | ParseException e){
       e.printStackTrace(); }
```

Then after preparing 4 different LogFiles we save the first two messages Mo and M1, which were created from the 1st communication of the Trusted Server (secureserver.java) with the UntrustedServer (InitializeLog.java).

```
saveEntry("MoMessage");
saveEntry("M1Message");
```

The main log file is the LogFile.txt. The others are just copies of it and are used for other applications to access the log file, without interfering with the functionality of saving log entries, because they result into problems when we want to sign entries later.

The core functionality of this class, though, is performed in the **saveEntry (String entry)** method. This method takes as input parameter a string, which contains the message to be logged. If the log file is not closed (for some reason), then we can proceed with encrypting and saving the entry. We then create a timestamp to define when the entry message is going to be saved.

```
SimpleDateFormat format = new SimpleDateFormat("yyyy/MM/dd
HHmmssSSSS");
Calendar calendar = Calendar.getInstance();
java.util.Date now = calendar.getTime();
timestamp = format.format(now);
```

We also use a counter i to enumerate the current log entry. This counter is very useful in other cases also, because there are special messages that should be produced in the first two entries. If i=0 then the MoMessage should be encrypted and saved and the log entry should be of type Wi = “LogfileInitializationType”. If i=1, then it should be the M1Message to be encrypted and saved and the log entry should be of type Wi =

“ResponseMessageType”. In case the M1Message is verified as false or the current timestamp exceeds the timestamp that the Untrusted Server will timeout, then the log entry should be of type Wi = “AbnormalCloseType”. If the entry string is the message “exit”, then we define the log entry of type Wi = "NormalCloseMessage". Finally, if we have a normal message entry string, then we characterize it as Wi = “LogEntryType”.

```
if(i==0){     //if it is MoMessage
       try {
              fis = new
FileInputStream("C:\\Users\\Anastasia\\EclipseWSworkspace\\MoMe
ssage");
              byte[] MoMsgByte = new byte[fis.available()];
              fis.read(MoMsgByte);
              fis.close();
              Message = new String(MoMsgByte);
       } catch ( IOException e) { e.printStackTrace(); }
              Wi = "LogfileInitializationType";  //LogEntryType
} else if (i==1){   //if it is M1Message
       try {
              fis = new
FileInputStream("C:\\Users\\Anastasia\\EclipseWSworkspace\\M1Me
ssage");
              byte[] M1MsgByte = new byte[fis.available()];
              fis.read(M1MsgByte);
              fis.close();
              Message = new String(M1MsgByte);
       } catch ( IOException e) { e.printStackTrace(); }
              Wi = "ResponseMessageType";
} else if (M1verified==false){
       Wi = "AbnormalCloseType";   //LogEntryType
       Message = ("M1 Message sent by Trusted Server is NOT
verified in Untrusted Server.");
       System.out.println(Message);
} else if (now.after(timeoutdate)) {
       Wi = "AbnormalCloseType";   //LogEntryType
       Message = "Timeout occured in Untrusted Server";
       System.out.println(Message);
} else if (entry.equals("exit")){
       Wi = "NormalCloseMessage";
       Message = "";
} else {
       Message = entry;   //String received in ServerObject
       Wi = "LogEntryType";
}
```

We then define the string to be logged in an unencrypted format yet. The information it includes are the current timestamp, the timestamp when the Untrusted Server will timeout, the id of the log entry and the message it received from either the sensor, or the alarm system, or an empty message in case of an abnormal or normal close of the log file. So the logging format is:

[log_timestamp timeout_timestamp logentry_id message]

```
String Di = timestamp +" "+timeout+ " "+ID_logentry+" "+
```

```
Message;
```
An example of such an entry is
```
2013/08/02 1109250079 2013/08/02 1130000000 4 2013/08/02
1109250079  A11   Temp  21.0  Celsius
```
where
Log_timestamp = `2013/08/02 1109250079`
Timeout_timestamp = `2013/08/02 1130000000`
Logentry_id = 4
Message = `2013/08/02 1109250079 A11   Temp  21.0  Celsius`

Then we check if it is the 1st log entry we are going to save (if i=0). If this is the case, then we read the Ao authentication key, which the InitializeLog.java had created previously and saved locally in the Untrusted Server storage. Furthermore we create a 20-byte array of zeros, which we are going to use as value Yi-1, which we are going to use a bit later.

```
byte[] Y_minus1 = new byte[20];
for(int j = 0 ; j < Y_minus1.length ; j++) {
       Y_minus1[j] = 0;    }
Yi = new String(Y_minus1);
```

We then create the encryption key Ki, applying a one-way hash function to the authentication key Ai and the log entry type Wi.

```
String Ki = Ai +" "+ Wi;
int encki = Ki.hashCode();
```

Having the Di string message and the encryption key Ki, we can produce the AES(Di)_Ki value, i.e. we can encrypt the Di value with the AES symmetric encryption algorithm using the key Ki. The encryption method returns as result a string with the encrypted data.

```
AES256 aesDi = new AES256();
try {
        AESDi_Ki = aesDi.encryptAES256(Di,
Integer.toString(encki));
} catch (Exception e) {    e.printStackTrace();       }
```

We then create the Yi value, by hashing the values Yi-1, the output of the AES encryption AESDi_Ki, and the key Ai.

```
String hashcurrent = Yi +" "+ AESDi_Ki+" "+Wi;
int Yint = hashcurrent.hashCode();
Yi = Integer.toString(Yint);
```

The last value we create for the encrypted log entry is a MAC (Message Authentication Code) of the value Yi, using the key Ai.

```
byte[] AikeyBytes=Ai.getBytes();
SecretKey key = new SecretKeySpec(AikeyBytes, "HmacSHA1");
```

```
Mac m;
try {
	m = Mac.getInstance("HmacSHA1");
	m.init(key);
	m.update(Yi.getBytes());
	mac = m.doFinal();
} catch ( NoSuchAlgorithmException | InvalidKeyException e) {
	e.printStackTrace();
}
```

Therefore an encrypted log entry will contain the following information

[log_entry_type, encrypted_message, value_Yi, value_MACi]

```
String Li = Wi +"   "+ AESDi_Ki +"   "+ Yi +"   "+ Zi + "\n\n";
```

For example,

LogEntryType
IJ6z0QgONqWemDWSqObwtvZ/DB0D6iFGVf3imoIp6i7AONa8IuJzP1o5w
kJEBNgtsXPpmrVjSw+2
AQT6nC09CuMwezCy15yafdzfX+IoY+5XbsT96EKbLxqDqnMcloWb -
834139319 1263724178

where

log_entry_type Wi = LogEntryType

encrypted_message AESDi_Ki=
IJ6z0QgONqWemDWSqObwtvZ/DB0D6iFGVf3imoIp6i7AONa8IuJzP1o5w
kJEBNgtsXPpmrVjSw+2
AQT6nC09CuMwezCy15yafdzfX+IoY+5XbsT96EKbLxqDqnMcloWb

value_Yi = -834139319

value_MACi = 1263724178

Afterwards, we save the encrypted log entry in the LogFile.txt and LogFile 2, 3, 4 which are copies of the first one. The reason we save the log entries in different files is to avoid simultaneous access to the same file by the US, the Regulatory Authority and the Alarm Service, which causes errors (e.g. incorrect signatures, when the Alarm Service tries to sign the logged entries). In the complete system, instead of files, we'll have a database to save our encrypted entries, so we'll have to manage access rights there instead.

Then we increment the authentication key Ai, replacing the one we just used. The new key will be used at the new entry. The new key is created by incrementing the old one and then hashing it.

```
int a1 = Integer.parseInt(Ai) + 1;    //Ai+1
String A1 = Integer.toString(a1);
int aa1 = A1.hashCode();
Ai = Integer.toString(aa1);
```

In the end of the saveEntry method we check if we had previously received an “exit” or an “AbnormalCloseType” message. In that case we log the

current entry as the last one in our log file, we delete the keys Ai and Ki and we mark the log file as closed.

```
if(entry.equals("exit")||Wi.equals("AbnormalCloseType")){
	Ai = "";
	Ki = ""; encki = 0;
	fileclosed = true;
}
```

6.2.1.2 InitializeLog.java

According to Schneier & Kelsey security scheme, before starting logging entries, we should establish communication between Unstrusted (US) & Trusted Server (TS). The InitializeLog class takes care of this. The actions that are performed here are:

1. US sends MoMessage = [protocol_stepID, ID_US, PKE(Ko)_pubTS, AES(Xo, Sign(Xo)_privUS)]
   where Xo = [protocol_stepID, timestamp, Ao]
   Ao = authentication key
   Ko = session key (US->TS)
2. Then US waits for the response of TS, ie. M1Message.
3. When US receives M1 it tries to verify it; it should decrypt correctly and contain a valid signature Sign(X1).

The core method of this class is **Init()**. The first action to do in this method is to create a private-public key pair for the Untrusted Server, which we store in USstorage.

```
Key_Pair pairUS = new Key_Pair();
pairUS.create("USstorage\\USsk", "USstorage\\USpk");
privUS = pairUS.getPrivKey();
```

Then we create the first authentication key Ao in a random way and save it in USstorage for later use.

```
Random rnd = new Random();
int ao = rnd.nextInt();
Ao = Integer.toString(ao);
```

Afterwards, we create a session key Ko to communicate 1st time with Trusted Server. Defining as first log entry type Wo = “LogfileInitializationType”, the key is generated by hashing Wo and Ao, i.e. Ko=hash(Wo,Ao).

```
String Wo= "LogfileInitializationType";
String hash = Wo + Ao;
sessionKo = hash.hashCode();
```

Then we create a timestamp of the message, to note when it was created and sent first time.

```
SimpleDateFormat format = new SimpleDateFormat("yyyy/MM/dd
HHmmssSSSS");
Calendar calendar = Calendar.getInstance();
```

```
java.util.Date now = calendar.getTime();
timestamp = format.format(now);
```
So, we are ready now to construct a part of the MoMessage, the Xo message: (protocol_step_ID, Start_timestamp, Authentication_Key_Ao)
```
String XoMessage = protocol_stepID + " "+ timestamp + " " + Ao;
```
and then sign it with the private key of US
```
signature si0 = new signature();
si0.sign(XoByteArray, privUS);
byte[] Sig = si0.returnSign();
```
In order for the signature to be read from a file correctly, it needs to be saved and read in a byte[] format. So, apart from the MoMessage file where we'll save it as a String, we save it also separately in a SignXo file, but we will let TS access this file, only if it decrypts correctly the MoMessage.

In the following steps, we read the public key of Trusted Server, using the local method readPK_TS().
```
pubKeyTS = readPK_TS();
```
We will use the public key of TS to perform a public key encryption PKE of the session key Ko, using the RSA algorithm: PKE(Ko)_pkTS
```
PKE_RSA pkeKo = new PKE_RSA();
pkeKo.encryptPKE_RSA(pubKeyTS, Integer.toString(sessionKo));
PKEbytearray = pkeKo.EncryptPKERSA_Result();
PKEtextmsg = new String(PKEbytearray);
```
After that, we perform symmetric encryption with the algorithm AES-256 of values Xo and its signature sign(Xo):
```
String signatureXo = new String(Sig);
String AESmsg = XoMessage + " " + signatureXo;
AES256 aesXoSignXo = new AES256();
AESresult = aesXoSignXo.encryptAES256(AESmsg,
Integer.toString(sessionKo));
```
The PKE(Ko)_pkTS value is saved also in a separate file PKE_Ko, because it faces the same problem as the sign(Xo).
Finally, we create the MoMessage consisting of the following values:
```
MoMessagePrint = protocol_stepID + "   " + ID_US + "   " +
PKEtextmsg + "   " + AESresult;
```
where ID_US is the id of the Untrusted Server.

Then the init() method waits until TS answers by creating the M1Message file. When US receives the M1Message, it tries to decrypt it. The encrypted values it contains are:
M1Message = [protocolstep_ID, TS_ID, PKE(K1)_pubUS, AES(X1,Sign(X1)_privTS)]
First we decrypt the message encrypted with PKE using the RSA algorithm and the public key of US:

```
PKE_RSA pkeK1 = new PKE_RSA();
```
To decrypt it we need the private key of US. The decrypted value we get is the session key K1 that TS created.
```
pkeK1.decryptPKE_RSA("USstorage\\USsk", pkek1);
sessionK1 = pkeK1.decryptPKERSA_Result();
```

Then we decrypt the message encrypted with AES256 (in value result[3]), using the session key K1.
```
AES256 aes2 = new AES256();
String X1SignX1 = aes2.decryptAES256(result[3], sessionK1);
```
The result we get is the message X1 and its signature sign(X1).
Afterwards, we verify the signature sign(X1) that TS created with its private key. To verify it, we use its public key TSpk. The result we get is a Boolean variable which returns true if the signature is verified or false otherwise.
```
signature s1 = new signature();
boolean X1SignVerif =
s1.SignVerif("TSstorage\\TSpk","sigX1",X1);
```
If the signature above verifies correctly, then we consider that the message M1Message is verified, too.
```
if (X1SignVerif == true){
     System.out.println("Message M1 sent signed by TS is
verified!\n");
     M1verified = true;
}
```
Previously, we had mentioned that the local method **readPK_TS()**, reads the encoded public key bytes of the Trusted Server. First, we read the public key of TS, which is located in its memory, TSstorage. In real conditions we should place the public key in an area/server where both TS and US have access and limit the permissions of TSstorage only to TS.
Then we define the key specification; the key was encoded according to the X.509 standard for RSA by SUN. We create then a KeyFactory object to instantiate a RSA public key from its encoding and finally we use the KeyFactory object to generate a PublicKey from the key specification.
```
X509EncodedKeySpec pubKeySpec = new
X509EncodedKeySpec(encKeyTS);
KeyFactory keyFactory = KeyFactory.getInstance("RSA");
pubKeyTS = keyFactory.generatePublic(pubKeySpec);
```
The method returns as result the public key of TS.

6.2.1.3 ManualSigning.java

This class performs the “manual signing" operation, as it is described in the security scheme of Stathopoulos, Kotzanikolaou, Magkos. It is performed by the administrator of US to manually sign log entries.
The recommended frequency for this operation is once per day or per week.

First we read the private key SK1 that the Alarm Service (for Security) has already generated. Then we create then a KeyFactory object to instantiate a RSA private key from its encoding and then we define the key specification; the private key was encoded according to PKCS8 standard.

```
KeyFactory keyFactory = KeyFactory.getInstance("RSA");
EncodedKeySpec privateKeySpec = new
PKCS8EncodedKeySpec(privateKeyBytes);
privateKey1 = keyFactory.generatePrivate(privateKeySpec);
```

Afterwards we read the encrypted log file with the local method **readFile()**, which reads actually a copy of the log file LogFile4.txt and save the messages it read on the String variable str.

```
readFile();
```

So, we use the str variable to sign the current content of the log file with the private key SK1.

```
signature s1 = new signature();
s1.sign(str.getBytes(), privateKey1);
```

We call the local method **save(byte[] Signature)**, to save the signed log entries in RAstorage (where RA stands for Regulatory Authority, as this is defined in Stathopoulos et al. security scheme).

```
save(realSigstr);
```

Finally, we wait for RA to acknowledge the reception of the signature above, i.e. to create an ACK1 file in Eclipse Workspace, i.e. the area of common access to both US and RA. All the signatures sent to RA, are stored in RAstorage and are enumerated.

6.2.2 Package: com.Security.TrustedServer

The following files are included in the package *com.Security.TrustedServer.* We have created the folder TSstorage to represent the local storage of the Trusted Server TS. There we save some data such as keys that the TS stores. When TS wants to send a message to US, it creates a file in Eclipse Workspace. In a realistic scenario, instead of Eclipse Workspace we can use for example another server, where they have access all servers that want to communicate.

6.2.2.1 secureserver.java

As we mentioned previously, according to Schneier & Kelsey security scheme, before starting logging entries, we should establish the communication between Unstrusted (US) & Trusted Server (TS). So, US first sends a MoMessage to TS. TS verifies MoMessage, i.e. checks if it decrypts correctly and if it has a valid signature. TS then creates M1Message and sends it to US. The variables M1Message contains are:

M1Message = [protocolstep_ID, TS_ID, PKE(K1)_pubUS, AES(X1,Sign(X1)_privTS]
where X1 = [protocolstep_ID, ID_log, hash(Xo)]
K1 = session key for TS sending a message to US
TS_ID = the id of TS

First, Trusted Server creates his private-public key-pair, which he stores in his memory TSstorage.

```
Key_Pair pairTS = new Key_Pair();
pairTS.create("TSstorage\\TSsk", "TSstorage\\TSpk");
```

Then TS waits until MoMessage is created by US in their common-access area (EclipseWorkspace). Once it is created, we read it and try to decrypt it.

a) We decrypt the message pke0, encrypted with PKE using the RSA algorithm and the private key of TS. The result we get is the session key Ko.

```
PKE_RSA pkeKo = new PKE_RSA();
pkeKo.decryptPKE_RSA("TSstorage\\TSsk", pke0);
sessionKo = pkeKo.decryptPKERSA_Result();
```

b) We decrypt the message encrypted with AES256 algorithm. To do that we use the session key Ko. As result we get the Xo message and its signature sign(Xo).

```
AES256 aesDo = new AES256();
String XoSignXo = aesDo.decryptAES256(result[3],
sessionKo);
```

Finally, we try to verify the signature of the sign(Xo). If the method returns true, then the signature is verified.

```
signature si = new signature();
boolean XoSignVerif =
si.SignVerif("USstorage\\USpk","sigXo",Xo);
```

If the above signature is verified, then TS starts forming the M1Message to reply to US.
First, it creates a random session key K1.

```
SecureRandom random = new SecureRandom();
sessionK1 = new BigInteger(80, random).toString(32);
```

Then it creates the X1 message, which contains the protocol_stepID, logID and hash(Xo).
The protocol step id is created by incrementing the respective value we received with MoMessage

```
int  p = Integer.parseInt(protocol_stepID);
p = p + 1;
protocol_stepID = Integer.toString(p);
```

And the hash value of Xo is created as

```
int hashXo = Xo.hashCode();
```

Then we can form our X1 message:

```
String X1 = protocol_stepID + " " +logID + " "+
Integer.toString(hashXo);
```

and sign it with the private key of TS

```
signature s1 = new signature();
s1.sign(X1.getBytes(), privTS);
realSigX1 = s1.returnSign();
```

Afterwards, we perform Public Key Encryption PKE (using RSA algorithm) of the session key K1 with Untrusted Server's public key: PKE(K1)_pkUS. First we read the public key of US, using the local method readPK_US().

```
pkUS = readPK_US();
PKE_RSA m1pke = new PKE_RSA();
m1pke.encryptPKE_RSA(pkUS, sessionK1);
M1_PKEbytearray = m1pke.EncryptPKERSA_Result();
```

After that, we perform symmetric encryption with AES-256 algorithm of the variables X1 and sign(X1):

```
try {
       String signatureX1 = new String(realSigX1);
       String AESmsgM1 = X1 + " " + signatureX1;
       AES256 m1aes = new AES256();
       AESresultM1 = m1aes.encryptAES256(AESmsgM1, sessionK1);

} catch (Exception e) {  e.printStackTrace();  }
```

So, we are, finally, ready to form and save the M1Message:

```
String M1msg = protocol_stepID + "   "+ Trusted_ID + "
"+M1_PKEtextmsg+ "   "+ AESresultM1;
```

6.2.2.2 VerifyLog.java

After the US has created the log file containing all entries encrypted, the TS server gets this file to decrypt it and verify it. An encrypted log entry contains the values [LogEntryType, AES(Di)_Ki, Yi, MACi]

(for more details see comments in UntrustedServer.java file). The TS learned the authentication key Ao, when he decrypted MoMessage with secureserver.java application. MoMessage contained Xo message, which contained Ao key. According to Schneier & Kelsey security scheme, the way to decrypt the log is the following:

1. Ki = hash(Wi, Ai), so he learns the encryption key Ki for AES algorithm
2. decrypt: AES(Di)_Ki, so he learns data Di, which contains among others Mi, ie. the core message
3. verification: recreates Yi = has(Yi-1, AES(Di)_Ki, Wi) and compares value to the received one
4. verification: recreates MACi = MAC(Yi)_Ai and compares value to the received one

The reason that TS recreates Yi and MACi values, instead of decrypting them is that the hash and MAC are 1-way functions and, thus, cannot be decrypted.

In the file we read the LogFile.txt (and its copies LogFiles 2,3,4) which contains the encrypted entries and try to verify each entry one-by-one.
If it is the first entry then the authentication key Ai is Ao. Otherwise, we calculate it by incrementing the previous key Ai-1 by 1 and hashing the sum.

```
int ai = Integer.parseInt(Ai) + 1;
String A1 = Integer.toString(ai);
int aa1 = A1.hashCode();
Ai = Integer.toString(aa1);
```

Then we calculate the key Ki by hashing the values entry log type Wi and the key Ai

```
String Ki =  Ai +" "+ Wi;
int keyi = Ki.hashCode();
```

After that, we decrypt the message encrypted with AES-256 algorithm, using the key Ki and get as output the decrypted data Di.

```
AES256 aesDi = new AES256();
Di = Di + aesDi.decryptAES256(AESDi_Ki,
Integer.toString(keyi));
```

If the message cannot be decrypted, then someone has tampered with the encrypted log entry in the log file.

Then we try to recreate value Yi. It is constructed as Yi = hash(Yi-1 , AES(Di)_Ki, Wi). If it is the first log entry then the Yi-1 value is a 20-byte array of zeros.

```
String hashcurrent = Yi +" "+ AESDi_Ki +" "+Wi;
int Yi_int = hashcurrent.hashCode();
Yi = Integer.toString(Yi_int);
```

After we have created Yi, we compare it with the one contained in the encrypted log entry. If they match, then Yi value is verified and we can continue trying to verify the MAC value. If not, then someone tampered with the encrypted log entry.

```
if(Yi_int==Integer.parseInt(entrypart[2])){
	System.out.println("Y["+i+"] = hash(Yi-1 ,AES(Di)_Ki, Wi)
is verified");
	Yi_verified = true;
}
```

If the Yi is verified, we create the MAC of the Yi value, using the Ai key:

```
byte[] AikeyBytes=Ai.getBytes();
SecretKey key1 = new SecretKeySpec(AikeyBytes, "HmacSHA1");
Mac m;
try {
	m = Mac.getInstance("HmacSHA1");
```

```
        m.init(key1);
        m.update(Yi.getBytes());
        mac = m.doFinal();
} catch (NoSuchAlgorithmException | InvalidKeyException |
IllegalStateException e) {
        e.printStackTrace();
}
```

We compare the MAC created with the MAC included in the encrypted log entry, after transforming them into integer format. Other formats cause problems in the comparison.

```
bi = Integer.parseInt(MACi);
if (bytesToInt(mac)==bi){  // best format to compare MACs is int
        System.out.println("MAC["+i+"] = MAC(Yi)_Ai  is
verified\n");
        MACi_verified = true;
}
```

If the MAC is not verified, then the encrypted log entry has been changed by an intruder.

Finally, we save the decrypted entries into two different files. The one called DecryptedLogDi.txt including the Di messages, which include the Mi messages. (The Mi message is the one received to Server by either the sensors or the Alarm System). The other file is called DecryptedLogMi.txt and contains only the Mi messages.

6.2.2.3 CleanFiles.java

We run this application before we start any other application, to clean the EclipseWorkspace and the RAstorage folder from the files they contain, because this will cause problems to run a new test. More specifically, we delete the acknowledgement files created from the automated and manual signing, then we delete the files created from the 1st and 2nd message exchange between US and TS. We then delete all the files in the RAstorage, i.e. all the manual signatures, the automated signatures. Then we the encrypted log files in USstorage and the corresponding decrypted log files in TSstorage. Finally, we delete the files ServerLog.txt and AlarmSystemLog.txt, which contain the messages that the ServerApp and AlarmSystem objects receives respectively.

6.2.3 Package: com.Security.AutomatedSigning

The following files are included in the package *com.Security.AutomatedSigning.* We have created the folder ASstorage to represent the local storage of the Alarm Service. This service is related to security and not to the Alarm System. That's the reason we named the package this way. In ASstorage we save some data, such as the two private-

public keys-pairs that the Alarm Service creates. The Alarm Serive also communicates with other applications through files in EclipseWorkspace. In a realistic scenario, instead of EclipseWorkspace we can use for example another server, where servers, that want to communicate, have access.

6.2.3.1 AlarmService4AutoSignRequests.java

Here we perform the "automated Signing" operation for random requests of Regulatory Authority RA, as described by Stathopoulos, Kotzanikolaou, Magkos in their security scheme. The action performed here, simulate the Alarm Service receiving and handling random requests (sent by RA) for signing the encrypted log entries currently existing in the log file. The log file it accesses is a copy of LogFile.txt, which is called LogFile3.txt.
In the main method we examine if the file Sign2Request is created by RA, to request a signature of the current log file. If this is the case, we call the method ActionOnAlarm(). In this method, we read the private key SK2, that the Alarm Service created for this specific signing purpose. We create a KeyFactory object to instantiate an RSA private key from its encoding and then in the key specification we define, that the private key was encoded according to PKCS8 standard.

```
KeyFactory keyFactory = KeyFactory.getInstance("RSA");
EncodedKeySpec privateKeySpec = new
PKCS8EncodedKeySpec(privateKeyBytes);
privateKey2 = keyFactory.generatePrivate(privateKeySpec);
```

We then call the method readFile() to read the LogFile3.txt containing a copy of the current encrypted entries and sign them, using the private key SK2.

```
signature s1 = new signature();
s1.sign(str.getBytes(), privateKey2);
byte[] realSigstr = s1.returnSign();
```

Then we run the save(byte[] Signature) method to save the signature in a file, called “Sign2_random”, and thus notify the RA that the Alarm Service created the signature it requested.
Then it waits for RA to create the file “ACK2_randomsign”, which means that it received the new signature. All the signatures sent to RA, are stored in RAstorage and are enumerated.

6.2.3.2 AlarmService4SecurityAlarms.java

This application simulates the Alarm Service which "automated signing” for security alarms it receives, as described by Stathopoulos, Kotzanikolaou, Magkos. We recognize critical events, sign the current encrypted log file and send the signatures to RA. Here as critical events we consider only the modifications on the log file. We use a copy of the main log file, called LogFile2.txt.

First, we create two private-public key pairs and store them in ASstorage. One pair SK1-PK1 will be used for “manual signing” and the other Sk2-PK2 for “automated signing”.
The application runs a TimerTask which runs every second and calls the FileWatcher function, which checks if any changes happened to the log file during that time. If the file changed, we call the ActionOnAlarm() local method. In this method, we read the entries contained in the current log file, sign them with the private key SK2 and create a file “Sign2”, which contains the signature and notifies the RA about it. Then the Alarm Service waits for RA to acknowledge the reception of the signature, by creating the file “ACK2_alarm”. All the signatures sent to RA, are stored in RAstorage and are enumerated.

6.2.3.3 FileWatcher.java

This file contains code that periodically watches a file for any change happening on it. The code was found online at [52].

6.2.4 Package: com.Security.RegulatoryAuthority

The following files are included in the package *com.Security.RegularotyAuthority.* We have created the folder RAstorage to represent the local storage of the Regulatory Authority. In RAstorage we save all the signed encrypted log entries. The RA communicates with other applications through files in EclipseWorkspace, which serves as an area where all application have access.

6.2.4.1 AutomaticSignRequest

Using this application RA generates signature requests to Alarm Service at random times. The random times are generated as a random integer between 1 and 10 and are counted in seconds:

```
Random rnd = new Random();
int r = rnd.nextInt(10);
```

The request is generated by creating the file “Sign2Request”, which the Alarm Service will read.

6.2.4.2 RA4AutoSign2Ack.java

This is an RA service for acknowledging the automated signatures created by the Alarm Service after Ra’s random requests. In the main() method the FileWatcher function is used to check every half second if the “Sign2_random” file has changed, which indicates the storage of a signature in the RAstorage. The RA reads the signature and creates the

“ACK2_randomsign” to notify the Alarm Service that the signature was received.

6.2.4.3 RA4SecurityAlarms.java

This is an RA service for acknowledging the automated signatures created by the Alarm Service after a critical event, i.e. a modification in the encrypted log file. In the main() method the FileWatcher function is used to check every half second if the “Sign2” file has changed, which indicates the storage of a signature in the RAstorage. The RA reads the signature and creates the “ACK2_alarm” to notify the Alarm Service that the signature was received.

6.2.4.4 RAService4ManualSigning.java

This is a service at RA for acknowledging receiving signatures of the log entries, signed manually by the administrator of the Untrusted Server. In the main() method the FileWatcher function is used to check every second if the “Sign1” file has changed, which indicates the storage of a signature in the RAstorage. The RA reads the signature and creates the “ACK1” to notify the Untrusted Server that the signature was received.

6.2.4.5 Start_RA_AlarmService.java

This is a class which starts all the applications of RA and Alarm Security Service, which should be running simultaneously as Threads, i.e. the java files RA4SecurityAlarms, RAService4ManualSigning, AlarmService4SecurityAlarms, AlarmService4AutoSignRequests and RA4AutoSign2Ack.

6.2.4.6 VerifySignatures.java

When logging in Untrusted Server has finished, we run manually this application to verify the signatures (of the log entries) received in RA (RAstorage).
First we read the files Signs1_at_RA.txt and Signs2_at_RA.txt, to count how many manual and automated signatures exist on RAstorage. Then it reads in ASstorage the public keys PK1 and PK2 from the key-pairs that the Alarm Service had created. After that it reads the encrypted log entries that the LogFile.txt contains after the end of the logging procedure.
Afterwards, we try to verify the manual signatures. We know the public key Pk1 and the signature, but we don’t know exactly which entries of the log file were used to create a signature. So, for every signature, we try a string text which consists of the log entries being attached gradually to the end of the

text until the combination of signatures used to create that specific signature is found and the verification returns true as a result.

```
for (int j = 0; j < logresult.length; j++){ //for every log
entry
	Log = Log + logresult[j]+"\n\n";
	ver = SignVerif(keyreceived1, signreceived,  Log);
	if (ver==true){
		System.out.println("Singature1 entry["+i+"] is
verified\n");
		ver1[i] = 1;   //true
	}
}
```

If the manual signatures are verified, then we proceed with the verification of the automated ones. The procedure followed is similar with the one used for the manual signature verification.

6.2.5 Package: com.Security.EncryptDecryptAlgorithms

The following files are included in the package *com.Security.EncryptDecryptAlgorithms.*

6.2.5.1 AES256.java

It is an algorithm described by Advanced Encryption Standard (AES). It is a symmetric-key encryption algorithm, ie. the same key is used for both encrypting and decrypting data. The algorithm was found online at [53] and was modified to serve our needs in this project. The encryption method encryptAES256() takes as input parameters the data to be encrypted and the encryption key and returns the encrypted value as a string. The decryption algorithm decryptAES256 takes as input parameters the encrypted data and the encryption key and returns the decrypted value as a string.

Both the encryption need to generate a key from the encryption key they received as input. Thus, they call the local method **generateKey**.

```
Key key = generateKey(sessionkey);
```

In this method we examine the length of the session key and add zeros in the end of it, because we need a key of length 16 for the AES-256 decryption.

After generating the key in the format that is suitable, we initialize in both encryption and decryption methods an instance of the cipher, using the AES algorithm:

```
Cipher c = Cipher.getInstance("AES");
```

And after that we perform encryption

```
c.init(Cipher.ENCRYPT_MODE, key);
byte[] encVal = c.doFinal(Data.getBytes());
```

```
String encryptedValue = new
BASE64Encoder().encode(encVal);
```
Or decryption, depending on the method
```
c.init(Cipher.DECRYPT_MODE, key);
byte[] decordedValue = new
BASE64Decoder().decodeBuffer(encryptedData);
byte[] decValue = c.doFinal(decordedValue);
```

6.2.5.2 PKE_RSA.java

This file is used for Public Key Encryption (PKE) performed with the RSA algorithm. The PKE requires a pair of keys (public-private), which are mathematically linked. The algorithm was found online at [54] and was modified to serve our needs. The method encryptPKE_RSA takes as input a public key and a plaintext and encrypts the plaintext with the key. Inside the method we initialize the cipher, using the "RSA" algorithm:
```
Cipher cipher;
cipher = Cipher.getInstance("RSA");
```
And then we encrypt the cipher with the public key:
```
cipher.init(Cipher.ENCRYPT_MODE, pk);
encryptedPKE_Key = cipher.doFinal(sessionkey.getBytes());
```
The decryptPKE_RSA method receives as input a private key and a ciphertext and decrypts the ciphertext with the key. In this method we also initialize a cipher and then perform its decryption with the private key:
```
cipher.init(Cipher.DECRYPT_MODE, priv);
byte[] decryptedMsg = cipher.doFinal(EncryptedData);
```

6.2.5.3 Key-Pair.java

This algorithm generates public and private keys (as pair). The algorithm was created after combining two online sources [55], [56].
We first create a key-pair generator object, using the RSA as a sign algorithm
```
KeyPairGenerator keyGen = KeyPairGenerator.getInstance("RSA");
```
Then we initialize the Key-Pair Generator, using theSHA1PRNG pseudo-random-number generation provided by SUN:
```
SecureRandom random = SecureRandom.getInstance("SHA1PRNG",
"SUN");
```
Finally, we generate the Pair of Keys & store them:
```
KeyPair pair = keyGen.generateKeyPair();
priv = pair.getPrivate();
pub = pair.getPublic();
```

6.2.5.4 signature.java

This file contains algorithms for signing data and for verifying signed data. The algorithms were found online and were modified.

The signature method **sign** gets as input the data to be signed in byte array forma and the private key to sign them with. The signing algorithm used is RSA.

We first create a key-pair generator object, using the RSA as a sign algorithm

```
KeyPairGenerator keyGen = KeyPairGenerator.getInstance("RSA");
```

Then we initialize the Key-Pair Generator, using theSHA1PRNG pseudo-random-number generation provided by SUN:

```
SecureRandom random = SecureRandom.getInstance("SHA1PRNG",
"SUN");
```

We then apply the a length of 1024-bit to the key

```
keyGen.initialize(1024, random);
```

Finally, we sign the data by the following procedure:

We get a signature object:

```
Signature rsa = Signature.getInstance("SHA1withRSA");
```

We then initialize the signature object with the private key:

```
rsa.initSign(priv);
```

Then we supply the signature object the data to be signed:

```
rsa.update(bytearray);
```

In the end, we can generate the signature of the data, once all of them have been supplied to the signature object

```
realSign = rsa.sign();
```

The signature verification method **SignVerif** takes as input the public key, the signed data to be verified and the unsigned data. We define the key specification: the key was encoded according to the X.509 standard for RSA by SUN

```
X509EncodedKeySpec pubKeySpec = new X509EncodedKeySpec(encKey);
```

We then create a KeyFactory object to instantiate a RSA public key from its encoding:

```
KeyFactory keyFactory = KeyFactory.getInstance("RSA");
```

We use the KeyFactory object to generate a PublicKey from the key specification:

```
PublicKey pubKey = keyFactory.generatePublic(pubKeySpec);
```

Afterwards, we initialize the signature object for verification as following:

a) we create a signature object and use the same signature algorithm as when generating the sign:
   ```
   Signature sig = Signature.getInstance("SHA1withRSA");
   ```
b) then we initialize the signature object with the public key:
   ```
   sig.initVerify(pubKey);
   ```

Then we supply the signature object with the data to be verified:

```
sig.update(Xmsg.getBytes());
```

Finally, we verify the signature of Untrusted Server:

```
verifies = sig.verify(sigToVerify);
```

The sources for this java file are [57], [58].

6.3 Application Set-Up

6.3.1 Installation

The application prototype was installed and run on a laptop computer with an 64-bit operating system Windows 7 Home Premium, Service Pack 1, processor Intel® Core™ i5-2410M CPU of frequency 2,3 GHz and RAM memory 4GB.We used JavaEE Eclipse with jre 7 and sdk 7 to develop and compile our code and Glassfish 3.1.2 as a server for configuring the JMS.
We needed to include the following libraries in our client applications, which communicated with the JMS, i.e. the sensor applications, the alarm system application and the server application, as following:

- Right click on the project name>Properties>Java Build Path>Tab: Libraries>Add Library>Server Runtime>GlassFish 3.1.2
- Right click on the project name>Properties>Java Build Path>Tab: Libraries>Add Library>JRE System Library>Workspace default JRE (jdk 1.7.0_02)
- Right click on the project name>Properties>Java Build Path>Tab: Libraries>External JARS:
  Pathname: ../glassfish3/glassfish/lib/gf-client.jar.

Finally we added in the server application also the file “openjdk-6-b14.jar” as an External JAR file, so that we can import the sun.misc.BASE64Encoder and sun.misc.BASE64Decoder for the AES-256 encryption and decryption algorithms [47].

6.3.2 JMS Configuration

We use Glassfish 3.1.2 as a server. After we have started the Glassfish server we use the Admin consol to access and define the administrative objects in the JMS Resources. We name the connection factory as:
JNDI name: “jms/GFConnectionFactory”
Physical name: “GFConnectionFactory”
And we characterize this object as an instance of “*ConnectionFactory*.*TopicConnectionFactory* “.
Then we name two destination objects:

1. JNDI name: “jms/Data”, physical name: “Data”, Type: “Topic”
2. JNDI name: “jms/Alarm”, physical name: “Alarm”, Type: “Topic”

Therefore, we have binded the administrative objects to a JNDI namespace [47].

6.3.3 Running the applications

In order to perform a demonstration of our system prototype we need to run some of the applications in the following order:

1. *com.Security.TrustedServer.CleanFiles.java,* located in the ServerApp project file, to clean any remaining files created from the previous times we run the prototype.
2. *com.Security.RegulatoryAuthority.Start_RA_AlarmService.java,* located in the ServerApp project file, to start the RA and the Alarm Security Service
3. *com.javamsgservice.ServerObject.java,* located in the ServerApp project file, to start the log-server. This takes care of the initial communication of the untrusted and the trusted server and subscribes to the "Data" and "Alarm" topics waiting for incoming messages from the JMS. We should enter the date and time that the server will timeout, i.e. the logging will stop.
4. *com.javamsgservice.AlarmObject.java,* located in the AlarmSystem project file, to start the Alarm System, waiting for sensor data.
5. *com.javamsgservice.SimpleSensor.java,* located in the SensorSimple project file and/or *com.javamsgservice.SensorwithAlarm.java,* located in the SensorWithAlarm project file and/or *com.javamsgservice.DoubleSensor.java,* located in the SensorDoubleValues project file. All these applications simulate a sensor sending measurement data.
6. *com.Security.RegulatoryAuthority.AutomaticSignRequest.java,* located in the ServerApp project file, to start RA performing automated signature requests at random times
7. com.Security.UntrustedServer.ManualSigning.java*,* located in the ServerApp project file, to perform at least one manual signing like the US administrator would do.
8. When the sensor applications have stopped publishing messages, we can stop the *AlarmObject.java* and the *ServerObject.java* applications.
9. Then we should stop the *Start_RA_AlarmService.java*
10. We run the *com.Security.TrustedServer.VerifyLog.java,* located in the ServerApp project file, to verify in TS and decrypt the log files that the US created.
11. *com.Security.RegulatoryAuthority.VerifySignatures.java,* located in the ServerApp project file, to verify the signature created by the manual and automated signing, to ensure that no one tampered with our log.

CHAPTER 7

Evaluation

In order to evaluate the performance and the effectiveness of our solution, we run some tests to examine how our algorithm behaves under different conditions. The experiments are executed on a laptop computer with an 64-bit Windows 7 Home Premium operating system, Service Pack 1, processor Intel® Core™ i5-2410M CPU of frequency 2,3 GHz and RAM memory 4GB.

7.1 Performance

In order to examine the system performance we run it with 1, 5, 10 and 14 sensors. We use the simple sensors. If we used another type of sensor, it would not make much difference in its performance. We keep stable the sensors publish frequency to 4 seconds for all our experiments.
The tests are performed in two different scenarios. In one scenario, we use temperature measurements that don't produce an alarm, therefore the Alarm System does not publish many data. In the other scenario, we use temperature measurements that reach an alarm state and thus the Alarm System publishes data together with the sensors.
The files used as input for our applications in the experiments are the following and can be found in the Input folder:

- *NormalTempValues.txt*, contains 29 temperature values, which don't exceed 23.0 Celsius
- *TestTemp.txt,* contains 9 temperature values, which don't exceed 26.0 Celsius
- *AlarmTempValues.txt* contains 20 temperature values, which start from 20.3 and increase up to 34.8 Celsius
- *sensor1.properties,* contains data about the sensor, such as id, transmission period, alarm limit etc.
- *sensorAj.properties* where j = b, c, d, …, o

Since in these experiments we want to focus on the system performance, we don't use the whole security scheme, but only part of it. So, we decide to

exclude from these tests the manual signing performed by the administrator of the Untrusted Server and the automatic requests for signing generated randomly by the Regulatory Authority. Therefore, we don't run the files *com.Security.UntrustedServer.ManualSigning.java* and *com.Security.RegulatoryAuthority.AutomaticSignRequest.java* respectively. We only use the Schneier and Kesley security scheme and the automated signing triggered by modification events on the encrypted log file. This is why we use the *com.Security.RegulatoryAuthority*.*Start_RA_AlarmService.java* file. Therefore we perform our experiments running the applications in the following order:

- ServerApp project file including:
  - com.Security.*RegulatoryAuthority*.*Start_RA_AlarmService.java*
  - *com.javamsgservice.ServerObject.java*
- AlarmSystem project file including *AlarmObject.java*
- SensorSimple project file including *SensorSimple.java*
- SensorSimplej project file including *SensorSimplej.java* where j = 2, 3, .., 15

The Server subscribes to all the messages types, but saves in the log only those which it had received before the first exit message. All the sensors read the measurement values from the same file, but we start them manually and thus they don't read the file temperature values simultaneously. An exit message is produced when a sensor finishes reading the file. We stop the logging after the first exit, because the conditions changed and we have less sensors running, so we are not interested any more in the performance. In real conditions the exit could be a message created and sent by the administrator when the drug production is finished.
After the encrypted log file is produced, we verify (and decrypt) it in the Trusted Server and we also verify the signatures created so far. This practically means running the files *com.Security.TrustedServer.VerifyLog.java* and *com.Security.RegulatoryAuthority.VerifySignatures.java* respectively.

7.1.1 Speed/ Latency

The speed at which all the sensors transmit is the same, i.e. every 4 sec period time. Since we timestamp the time the data is sent from the sensors and the time the data are logged, we are going to examine the second value to evaluate the system performance when there is more than one sensors. We are also going to observe the speed at which the data are received by the Server.

Tests with Normal Temperature Measurements

Number of Sensors: 1

One can see the decrypted log file after being verified at the Trusted Server in Appendix A.1.1. (Files used: TestTemp.txt & sensor1.properties).
We have noted with blue the time the entry is logged and with orange the time the entry was sent from the sensor. We see that the time difference (in (1/10) milliseconds) between sending and logging varies at a range [4 – 84]. More specifically the time difference values (in (1/10) milliseconds) ordered are [84, 4, 13, 4, 12, 4, 26(W), 4, 14, 4]. We notice that there is a bigger delay in the first value, but afterwards it is not so high. We presume that this happens until the new added sensor synchronizes his transmission period with JMS. We should also mention that the value 26 consists the time the data was transferred from the sensor to the alarm system, where it was recognized as warning (W), and then to the Server, where it was logged in the file. We calculate the average latency from the above ordered values, excluding the 1^{st} (8.4 milliseconds) and the warning (2.6 milliseconds). So the result is 0.73 milliseconds (or 7.3*(1/10) milliseconds).
The time difference between two continuous measurements sent from the sensor is on average 4 sec (or 4000 milliseconds). This is the case also between two continuous entries in the log. The measurement identified by the alarm system as warning, is saved twice in the server log, one time as a data sent directly from the sensor and another sent as warning by the alarm system. Therefore, the two log entries have the same sensor timestamp and very close log timestamp.

Number of Sensors: 5

Files used: NormalTempValues.txt, sensor1.properties and sensorAj.properties, where j = b, c, d, e.
We present, in the Appendix A.1.1, part of the decrypted log file after being verified at the Trusted Server. This part contains the entries logged when all 5 sensors are running simultaneously. For space reasons, we don't present the complete file log here. One can find the complete log in the file Tests/Latency/NormalValues/NoOfSensors_5/TSstorage/DecryptedLogDi.txt.
As in the case of 1 sensor, we notice that there is a bigger delay in the time difference between sensor timestamp and log timestamp the first time a sensor publishes a measurement. Here we can notice that the biggest difference is when the 5^{th} sensor (e) is added. The time differences, measured in (1/10)*milliseconds, in the part of the log we present in Appendix A.1.1, are shown in Table 7.1, where the letter in the parenthesis replaces the letter j included in the id, A11j, of the sensor, which produced the measurement. Sensor (a) id is just A11.

Sensor (a)	Sensor (b)	Sensor (c)	Sensor (d)	Sensor (e)
101	14	11	13	9526
4	6	4	4	4
4	5	4	4	4
3	4	5	4	22
4	4	3	7	4

Table 7.1: Latency between sensor timestamp and log timestamp, when 5 sensors are used and the data are directly received by the Server

We observe that the majority of the latencies have most often one of the values 0.4 or 1.4 milliseconds. However, the average latency for every sensor is presented in Table 7.2 measured in (1/10) milliseconds. For sensor (e) we exclude the value 952.5 milliseconds, because it occurs during the first transmission of the sensor and we presume it happens because the new added sensor needs some time to synchronize his transmission period with JMS.

Sensor (a)	Sensor (b)	Sensor (c)	Sensor (d)	Sensor (e)
23.2	6.6	5.4	6.4	6.8

Table 7.2: Average latency between sensor timestamp and log timestamp, when 5 sensors are used and the data are directly received by the Server

So, the average latency when we have 5 sensors is 9.68*(1/10) milliseconds, which is bigger than in the case of 1 sensor.
Moreover, we observe, in the Appendix A.1.1, that the 5th sensor (e) sends the first two measurements with time difference 4,9515 sec. until he synchronizes afterwards to sending the data every 4 sec. We presume that similar situation has occurred on some of the other sensors when they published their first measurement.
Furthermore, the time difference among sensors publishing is stable on average, but it depends when we start manually the sensor application.

Number of Sensors: 10

Files used: NormalTempValues.txt, sensor1.properties and sensorAj.properties, where j = b, c, …, j.
We present in the Appendix A.1.1 part of the decrypted log file after it has been verified by the Trusted Server. This part contains the entries logged when all 10 sensors are running simultaneously. One can find the complete log in the file Tests/Latency/NormalValues/NoOfSensors_10/TSstorage/DecryptedLogDi.txt.
The time differences between the sensor timestamp and the log timestamp for the above log part are presented in Table 7.3 measured in (1/10)

milliseconds. In the first row we have the sensors. The letter in the parenthesis replaces the letter j included in the id, A11j, of the sensor, which produced the measurement. Sensor (a) id is just A11.

(a)	(b)	(c)	(d)	(e)	(f)	(g)	(h)	(i)	(j)
4	14	4	5	4	6	5	4	4	15
12	4	12	31	56	4	53	4	15	4
3	3	4	4	4	4	33	3	4	4

Table 7.3: Latency between sensor timestamp and log timestamp, when 10 sensors are used and the data are directly received by the Server

The average latency for every sensor is presented in Table 7.4 measured in (1/10)*milliseconds.

(a)	(b)	(c)	(d)	(e)	(f)	(g)	(h)	(i)	(j)
6.3	7	6.6	13.3	21.3	4.6	30.6	3.6	7.6	7.6

Table 7.4: Average latency between sensor timestamp and log timestamp, when 10 sensors are used and the data are directly received by the Server

So, the average latency when we have 10 sensors is 10.5*(1/10) milliseconds, which is bigger than in the case of 5 and 1 sensors.
The sensors keep sending their measurements every 4 seconds. Observing the reception of measurements on the Server, we see that many data arrive with small time difference without causing any problem on the server application.

Number of Sensors: 14

Files used: NormalTempValues.txt, sensor1.properties and sensorAj.properties, where j = b, c, …, o.
We present in the Appendix A.1.1 part of the decrypted log file after being verified at the Trusted Server. This part contains the entries logged when all 13 sensors are running simultaneously. One can find the complete log in the file Tests/Latency/NormalValues/NoOfSensors_14/ TSstorage/ DecryptedLogDi.txt.
The time differences between the sensor timestamp and the log timestamp for the above log part are presented Table 7.5 measured in (1/10)*milliseconds. We can see how the sensors 12 (i), 13 (m) and 14 (n) are added.

a	b	c	d	e	f	g	h	i	j	k	l	m	n
62	31	146	18	191	27	133	139	7	14	30	36	-	-

5	7	9	6	4	10	6	19	5	7	10	8	17	-
exit	4	68	10	5	6	17	85	5	5	14	60	5	97

Table 7.5: Latency between sensor timestamp and log timestamp, when 14 sensors are used and the data are directly received by the Server.

The first measurement of 14th sensor (n) arrives in the Server when the 1st sensor (a) sends an "exit" message and logging stops. The average latency for every sensor is presented in Table 7.6 measured in (1/10) milliseconds.

a	b	c	d	e	f	g	h	i	j	k	l	m	n
33.5	14	74.3	11.3	66.6	14.3	52	81	5.6	8.6	18	34.6	11	97

Table 7.6: Average latency between sensor timestamp and log timestamp, when 14 sensors are used and the data are directly received by the Server.

We will exclude from the total average latency calculation, the latency of sensor (n), since it is its 1st measurement, we don't have any more measurements of this sensor in our log entries and it is quite different from the average latencies of the other sensors. So, the average latency when we have 13 sensors is 32.6*(1/10) milliseconds, which is bigger than in the case of 10, 5 and 1 sensors.
The sensors send their measurements every 4 seconds. The reception of measurements on the Server has small time difference between them, without causing any problem on the server application subscription.

Tests with Alarm Temperature Measurements

Number of Sensors: 1

Files used: AlarmTempValues.txt & sensor1.properties
Part of the decrypted log-file after being verified at the Trusted Server is presented in the Appendix A.1.1. The complete log-file is in the file: Tests/Latency/AlarmValues/NoOfSensors_1/TSstorage/ DecryptedLogDi.txt. We present the measurements when they start causing alarms.
When the data has been through the Alarm System before arriving to the Server we include a warning (W) or alarm (A) next to it. Usually an entry received in the Server directly by a sensor (and has the JMS topic "jms/Data") is followed by the same entry which has passed through the Alarm System before it arrived to the Server and, therefore, has been marked as alarm or warning (and has the JMS topic "jms/Alarm"). Time differences between the sensor timestamp and the log timestamp for the above log part are presented in Table 7.7. The values are presented in pairs measured in (1/10) milliseconds. Each pair is the same measurement, but on the left we have the

time difference for the entry with topic “jms/Data”, while on the right it is the difference for the same entry with topic “jms/Alarm”. We read Table 7.7 row-by-row from left to right.

0 – 47(W)	0-0(W)	0-0(W)	0-48(W)	0-0(W)	0-35(W)	0-0(A)
0-36(A)	0-0(A)	0-0(A)	0-0(A)	0-16(A)	0-9052(A)	0-0(A)

Table 7.7: Latency between sensor timestamp and log timestamp, when 1 sensor is used and the data are directly (left value) and indirectly (right value) received by the Server

The time difference cannot of course be zero, but it is very small. We see that the subscribing in the Server has changed its speed so as to capture all the incoming events. The sensor still sends its measurements every 4 sec. The average latency is 11*(1/10) milliseconds for sensor values marked as alarms or warnings and 0 for the others. So, in total, the average latency is 6.5*(1/10) milliseconds.

Number of Sensors: 5

Files used: AlarmTempValues.txt, sensor1.properties and sensorAj.properties, where j = b, c, d, e.
We present in the Appendix A.1.1 part of the decrypted log file after being verified at the Trusted Server. This part contains the entries logged when all 5 sensors are running simultaneously. The complete log-file is in the file Tests/Latency/AlarmValues/NoOfSensors_5/TSstorage/DecryptedLogDi.txt.
An entry sent directly by a sensor is followed by the same entry which has passed through the Alarm System before and therefore has been marked as alarm (A) or warning (W). Time differences between the sensor timestamp and the log timestamp for the above log part are presented in Table 7.8 measured in (1/10) milliseconds.

(a)	(b)	(c)	(d)	(e)
0 – 15 (A)	0 – 33 (W)	0 – 15 (W)	1 – 17 (W)	0 – 16 (W)
0 – 48 (A)	0 – 16 (A)	0 – 48 (W)	0 – 16 (W)	0 – 15 (W)
0 – 16 (A)	0 – 16 (A)	0 – 16 (A)	0 – 15 (W)	0 – 52 (W)

Table 7.8: Latency between sensor timestamp and log timestamp, when 5 sensors are used and the data are directly (left value) and indirectly (right value) received by the Server

From the above results we observe that the time difference between the sensor timestamp and the log timestamp is not significant when the data are sent directly to the Server. The average latency for every sensor is presented in Table 7.9 measured in (1/10) milliseconds.

(a)	(b)	(c)	(d)	(e)
0– 26.3(A)	0– 21.6(A/W)	0– 26.3(A/W)	0.3– 16(W)	0 – 27.6 (W)

Table 7.9: Average latency between sensor timestamp and log timestamp, when 5 sensors are used and the data are directly (left value) and indirectly (right value) received by the Server

The average latency, when the Alarm System first subscribes to the sensor data and then publishes them back to JMS, is 23.6*(1/10) milliseconds. So, in total, the average latency is 11.8*(1/10) milliseconds for all the sensor data until they arrive to the Server, either directly or indirectly through the Alarm System. This latency is bigger than the one for 1 sensor.

Number of Sensors: 10

Files used: AlarmTempValues.txt, sensor1.properties and sensorAj.properties, where j = b, c, …, j.
We present in the Appendix A.1.1 part of the decrypted log file after it has been verified by the Trusted Server. This part contains the entries logged when all 10 sensors are running simultaneously. One can find the complete log in the file Tests/Latency/AlarmValues/NoOfSensors_10/TSstorage/ DecryptedLogDi.txt.
An entry, sent directly by a sensor, is followed by the same entry which before had passed through the Alarm System and therefore was marked as alarm (A) or warning (W). Time differences between the sensor timestamp and the log timestamp for the above log part are presented in Table 7.10 measured in (1/10) milliseconds.

(a)	(b)	(c)	(d)	(e)	(f)	(g)	(h)	(i)	(j)
0– 48(A)	0– 15(A)	0- 16(A)	0- 16(A)	0– 16(A)	0- 16(W)	0– 16(W)	0- 16(W)	0- 16(W)	81 -
0– 16(A)	0– 16(A)	0- 16(A)	0- 15(A)	0– 16(A)	0- 16(A)	0– 15(W)	0- 16(W)	0- 15(W)	1 -

Table 7.10: Latency between sensor timestamp and log timestamp, when 10 sensors are used and the data are directly (left value) and indirectly (right value) received by the Server

From the above results we observe that the time difference between the sensor timestamp and the log timestamp is not significant when the data are sent directly to the Server. The average latency for every sensor is presented in Table 7.11 measured in (1/10) milliseconds.

(a)	(b)	(c)	(d)	(e)	(f)	(g)	(h)	(i)	(j)
0– 32(A)	0– 15.5(A)	0- 16(A)	0- 15.5(A)	0– 16(A)	0- 16(W/A)	0– 15.5(W)	0- 16(W)	0- 15.5(W)	41 -

Table 7.11: Average latency between sensor timestamp and log timestamp, when 10 sensors are used and the data are directly (left value) and indirectly (right value) received by the Server

The average latency, when the Alarm System first subscribes to the sensor data and then publishes them back to JMS, is 17.5*(1/10) milliseconds. We will exclude sensor (j) because we have only his latency when sensor data are sent directly to the Server, which is significantly different from the corresponding latencies of the rest of the sensors. So, in total, the average latency is 8.7*(1/10) milliseconds for all the sensor data until they arrive to the Server, either directly or indirectly through the Alarm System. This latency is bigger than the one for 1 sensor, but smaller than the one for 10 sensors.

Conclusions

We wanted to examine the latency between the sensor timestamp when it sends a measurement and the log timestamp when this measurement is saved in the log file. When we have in majority sensor measurements of normal temperature, then the average latency values measured in (1/10) milliseconds, are 7.3 for 1 sensor, 9.6 for 5 sensors, 10.5 for 10 sensors and 32.6 for 14 sensors running simultaneously. So, we conclude that as the number of sensors increases, the latency also increases. Moreover, we notice that, in all cases, as the number of sensor increases, latency values which are bigger (e.g. 36, 56, 100) than the average ones, appear more often.

When we have in majority sensor alarm measurements of temperature, then the latency decreases significantly for the data that are send directly from the sensors to the Server (we get values of zero, which mean that the latency is very low). It is the case now that the Server uses both topics it is subscribed to, i.e. “Data” and “Alarm” and, thus, subscribes with a higher speed to the measurement events. The latency that is presented in these experiments is mostly because the alarm system has first subscribed to the measurements and then published them back to JMS as alarm or warning events. So, the average latency values for the measurements, tagged as of “Alarm” topic and measured in (1/10) milliseconds, are 13 for 1 sensor, 23.6 for 5 sensors and 17.5 for 10 sensors. So, the average latency values for the measurements, tagged as of “Data” topic and measured in (1/10) milliseconds, are zero, which means that they are very small. Therefore, in total, the average latency measured in (1/10) milliseconds is 6.5 for 1 sensor, 11.8 for 5 sensors and 8.7 for 10 sensors, when we consider all the sensor data until they arrive to

the Server, either directly or indirectly through the Alarm System. We see the latency increasing as the number of sensors increases from 1 to 5. But this is not the case when from 5 we add up to 10 sensors. The reason for this could be related with a throughput problem that the alarm system faced in the case of 10 sensors, which we describe in the next section.
We notice that there are also here latency values which are bigger (e.g. 35, 48, 52) than the average ones. It seems that their appearance frequency is higher in the cases of 1 and 5 sensors than in the case of 10 sensors. As we already mentioned previously, the reason for this could be related with a throughput problem that the alarm system faced in the case of 10 sensors, which we describe in the next section.

7.1.2 Throughput

We notice that in all cases the server application adjusts the subscription speed and thus manages to subscribe to all the incoming data and not miss any of them.
As for the alarm system, it subscribes successfully to all the data, as well. However, we get a problem in publishing an alarm event when the sensors increase to 10. In Appendix A.1.1 we can see two of the errors we got in that specific case.
We noticed that the alarm system cannot publish all the alarm events but we get an error com.sun.messaging.jms.JMSException, followed by the message that the "*producer cannot be added to destination Alarm [Topic], limit of 100 producers would be exceeded user*". In our case we don't have 100 publishing events occurring simultaneously in the alarm system. However, the combination of the subscribe and publish functions in the same application, could be the cause that the publishing is not allowed to run more efficiently.

7.2 Security

In the Problem Analysis chapter we had identified a list of simple attacks that a malicious user having access to the Untrusted Server could perform:

1. insertion of a new entry
2. alteration of an existing entry:
    a) replacement of just a value of the entry (e.g. the measurement value, the sensor id)
    b) replacement of the whole entry by another one (of e.g. the same time)
3. reordering of entries
4. deletion of an entry
5. duplication of an entry

Here we take over the role of the malicious user and try to perform these attacks. We want to examine against which of these attacks our security scheme detects.

We use the simple sensor. We prefer to use measurements that don't produce many alarms, because we want to focus on the security scheme in these experiments. So, the files used as input for our applications in the experiments are the following:

- *TestTemp.txt*
- *Sensor1.properties*

Tests Set-Up

Since the aim of these experiments is to examine how efficient our security scheme is, we shall use the whole security scheme possibilities. This includes the Schneier and Kesley security scheme and both manual and automated signing; the manual signing is performed by the administrator of the (Untrusted) Server, the automatic requests for signing generated randomly by the Regulatory Authority and scheme and the automated signing triggered by modification events on the encrypted log file. Therefore we perform our experiments running the applications in the following order:

- ServerApp project file including:
  - *com.Security.TrustedServer.CleanFiles.java* to clean any remaining files created from the previous times we run the application
  - *com.Security.RegulatoryAuthority.Start_RA_AlarmService.java* to start the RA and the Alarm Security Service
  - *com.javamsgservice.ServerObject.java* to start the log Server which will wait for incoming messages from the JMS. We should enter the date and time that the Server will timeout, i.e. the logging will stop.
- AlarmSystem project file including *AlarmObject.java*
- SensorSimple project file including *SimpleSensor.java*
- ServerApp project file:
  - *com.Security.RegulatoryAuthority.AutomaticSignRequest.java* to start RA performing automated signature requests at random times
  - *com.Security.UntrustedServer.ManualSigning.java* to perform at least one manual signing like the US administrator would do.

The Server subscribes to all the message types and stops logging when it receives the first exit message.

After the encrypted log file is produced, we verify (and decrypt) it in the Trusted Server and, then, we verify in the Regulatory Authority RA the signatures created so far. This practically means running the files *com.Security.TrustedServer.VerifyLog.java* and *com.Security.RegulatoryAuthority.VerifySignatures.java* respectively.

Insertion of a new entry/ Duplication of an entry

In Appendix A.1.2 we can see the event messages as they arrived in the (Untrusted) Server and as they were logged as entries Li in the log file LogFile.txt. After the events are logged we copy the log entry L10 and paste it as next entry (L11). So, entry L11, becomes L12 and so on. The files in the encrypted log file after the attack are shown in Appendix A.1.2. We have marked with blue color the duplicate entry. One can also find the file at: Tests\SecurityAttacks\Insertion\USstorage\LogFile.txt. To verify the data received, we run first the log verification procedure at the Trusted Server and then the signature verification at the RA.

Log Verification at Trusted Server

We see that there was a problem in the file verification. The problem was traced in the Log entry L10 in the creation of the MAC and is related to the MAC value received from the log file. So, the Schneier and Kesley scheme detected the attack. One can find the complete verification console output in Appendix A.1.2.

```
...
L[10]:
LogEntryType
DEm++Oswr7c5wIrgExVb7K2NPrdQ3MaMQ6K4srA/TS5vaago2VZsbn0bgD6Mkd3c9r+7O
F0CRc/r
6O2s9R7R9cTcMdumAy5FuMAPLQZJevWufQ8b4mhMeb7p9r0yCKdy -830774920
1494439717
LogEntryType
Decrypted_AESmsg D[10]: LogEntryType 2013/08/02 1109450132 2013/08/02
1130000000 10 2013/08/02 1109450132 A11    Temp    23.6    Celsius
M[10]: LogEntryType 1130000000 10 2013/08/02 1109450132    A11    Temp
       23.6    Celsius
Yi: -830774920
Yi2: -830774920
Y[10] = hash(Yi-1 ,AES(Di)_Ki, Wi) is verified
mac: 1494439717
Number Format Exception! The reason is that someone tampered with
entry[10].
...
In file LogFile.txt the MACs in following entry positions were found
false
10
```

Signature Verification at RA

While the entries were being saved in the log file by the (Untrusted) Server, we ran 1 manual signing on the file and the RA did 16 random automated requests for signing the data of the file. We also had 12 automated signatures of the data, triggered by modifications on the log file. We notice that not all signatures created are verified; only 8 of the 14 random signatures and 9 out

of the 12 automatically triggered ones, are verified. So, this implies that there was an attack on the log file. One can see exactly which signatures were verified in Appendix A.1.2

Replacement of an entire entry

We have replaced the log entry L9 with another one. The new entry is marked with blue. The log file is located at Tests\SecurityAttacks\Replacing1entry\USstorage\LogFile.txt. In the Appendix A.1.2 we can see the Server Log, as it created the log entries, the log verification at the Trusted Server, which presents also the log entries after the attack, and the signatures verification in the Regulatory Authority.

Log Verification at TS

The log verification procedure finds that the log entry L9 has been changed, since it cannot decrypt the AES message, AES(D9)_K9, received from the log file, where D9 are the data encrypted with AES algorithm and K9 key. This is because this value was changed in the log file and the key K9, which TS created, cannot decrypt it. Therefore, the value Y9 = hash(Y8, AES(D9)_K9, W9), which TS generates, where W9=LogEntryType, is false and cannot be verified because it doesn't match the Y9 value of the log file. Consequently, the MAC9=MAC(Y9)_A9, where A9 is a key that TS created, is also false and cannot be verified either. Since the Y9 created is false, all the next Yi values that TS creates after this one will also be false. That's why the decryption and verification problem in log entry L9 causes verification problems in the Yi and MAC values for the entries following L9 (i.e. 10, 11, 12). The errors output on the verification of the entries L11 and L12 are similar to the one in L10 log entry presented below. One can see details in Appendix A.1.2.

```
...
L[9]:
LogEntryType
BJx1O1v2YQA8lNi32hMO5VHJo4aYi+uE7fLU8N0DQO90abol5YP5HZSpkNGOn3m3DCLad
AjtwWrs
FmTiWTvZ5Tldz+yepHDWe1gOR/pJDEMyIZF2cgQwhQnwxEQKysRP -475431498 -
456700380
AES message cannot be decrypted! Someone changed log entry [9].
Yi: 2082583920
Yi2: -475431498
L[10]:
LogEntryType
4YMw9cd6w6Deu5GixS3BWN11WB02tkPVeAll3jOlS+xUwlGpp12A/EvxH18RwiBvDKZMO
8QMDVVX
YvWDZXMSXIb8Hd6KfvWIUNkcOnx3QSiha3ZbzlLFnTE65X6h8x+v 1461733865 -
725012515
```

```
Decrypted_AESmsg D[10]: LogEntryType 2013/08/02 1738160326 2013/08/02
1759000000 10 2013/08/02 1738160322 A11    Temp   23.6   Celsius
M[10]: LogEntryType 1759000000 10 2013/08/02 1738160322   A11    Temp
       23.6   Celsius
Yi: 1088389658
Yi2: 1461733865
...
In file LogFile.txt the MACs in following entry positions were found
false
9
10
11
12
```

Signature Verification at RA

During the creation of the log entry file, we performed 1 manual signing and there were performed 21 random ones and 12 automated ones, triggered by log file modifications. Only 5 are verified from the random ones and 8 of the triggered ones, which indicates that an attack had been performed on the log.

Replacement of just a value of the entry

Here we replaced the Yi value of the entry L10 with another one, marked with blue color, i.e. Y10=hash(Y9, AES(D10)_K10, W10), where D10 is the data encrypted with AES algorithm and K10 key and W10=LogEntryType. The log file is located at Tests\SecurityAttacks\Replacing1valueof1entry\ USstorage\LogFile.txt. The Server Log, the log file after the attack, the log verification at the Trusted Server and the signatures verification in the Regulatory Authority are presented in the Appendix A.1.2.

Log Verification at TS

Since the Y10 value in the entry at the log file (marked with blue) is different from the one produced locally in the Trusted Server as Y10 = hash(Y9, AES(D9)_K9, W9), where W9=LogEntryMessage, AES(D9)_K9 are the data D9 encrypted with AES algorithm and key K9. So, the Y10 value and, thus, the MAC are not verified in the log entry L10.

```
...
L[10]:
LogEntryType
2mKlDqJCE6q/5Uals0nM+FuUb45zPI3DQ7Ex2j0w9h1rFDuEVzZ+1J7dxYYi0urrV3ux9
btBd+2f
2msM1IFU90oS/JfFkopqBie7AM5yYzDTTp5LYMLIAXm580A8nwfp 1557320632
867721926
Decrypted_AESmsg D[10]: LogEntryType 2013/08/02 1718070850 2013/08/02
1745000000 10 2013/08/02 1718070836 A11    Temp   23.6   Celsius
M[10]: LogEntryType 1745000000 10 2013/08/02 1718070836   A11    Temp
       23.6   Celsius
```

```
Yi: 1800467982
Yi2: 1557320632
...
In file LogFile.txt the MACs in following entry positions were found
false
10
```

Signature Verification at RA

During the entries logging 1 manual signing, 17 random automated signing requests and 12 automated signatures, triggered by log file modifications, have been performed. We observe that some signatures have not been verified, which indicates that an attack occurred in the log file. (Only 9 automatically triggered and 7 randomly requested signatures were verified).

Reordering of entries

The log entries L4 and L5 have been reordered. We mark them with blue color. The log file after the attack can be found at Tests\SecurityAttacks\Reordering\USstorage\LogFile.txt. The Server Log, the log verification at the Trusted Server, which presents also the log entries after the attack, and the signatures verification in the Regulatory Authority are presented in the Appendix A.1.2.

Log Verification at TS

The log verification procedure finds that the log entries L4 and L5 have been changed, since it cannot decrypt their AES messages, i.e. values AES(D4)_K4 and AES(D5)_K5, where Di are the data encrypted with AES algorithm and Ki key (here i=4,5). The reason for this is that the attacker has changed the encrypted AES messages in the log file and the keys K4 and K5, which TS created for each value respectively, cannot decrypt them. As a result, the value Y4 = hash(Y3, AES(D4)_K4, W4), where W4=LogEntryType, is false and cannot be verified. Consequently, the MAC4=MAC(Y4)_A4, where A4 is a key that TS created, is also false and cannot be verified either. Similar problem faces log entry L5. Since the Y5 created is false, all the next Yi values created after it will also be false. That's why the decryption and verification problem in log entry L5 causes verification problems in the Yi and MAC values for the entries following L5 (i.e. L6, …, L12). The errors output on the verification of the entries L7 to L12 are similar to the one in L6 log entry presented below. One can see details in Appendix A.1.2.

```
...
L[4]:
LogEntryType
X5yZhmdZoiJs6UouYhNqjqolzP3KYTVI7ld0ymJev5hBbrST+N6vFRJuqzFx3xR35zcQx
gutCX0C
```

```
lssbAcrp1LFgyWG8GrURStMVrUxoYMPKuQNqp2K05PS3LYTxjpyc 1763019560 -
1774162090
AES message cannot be decrypted! Someone changed log entry [4].
Yi: 1190319881
Yi2: 1763019560
L[5]:
LogEntryType
mqvgvLAWMntj9weq3+bMjhdH3g9rLqbaHsIAOIzgXJAzaVelquBhmtIVMmVjGkCrO67Dj
j/mgqCD
TcBUXMf0HPmPWJ+51lJyj5atjDAvelgHOzwjngr1JAiDlsL1Cdfh -1178753234
1092979152
AES message cannot be decrypted! Someone changed log entry [5].
Yi: 875957742
Yi2: -1178753234
L[6]:
LogEntryType
qiaccfor4ZornhF6KmFSyqoyKKcfbMCEYXkBkYD7q1cj61KnBerM0ee/pEpD0ca/MLAK/
+pBlxcj
/Lck7+aVHz7U3s0rCycL2i46q7dwfmd5vCmqAOEJwgQd7dNbN2Xb -953235483
1556122196
Decrypted_AESmsg D[6]: LogEntryType 2013/08/08 1547220174 2013/08/08
1700000000 6 2013/08/08 1547220174   A11     Temp    24.0    Celsius
M[6]: LogEntryType 1700000000 6 2013/08/08 1547220174     A11     Temp
       24.0    Celsius
Yi: -1579941509
Yi2: -953235483
...
In file LogFile.txt the MACs in following entry positions were found
false
4
5
6
7
8
9
10
11
12
```

Signature Verification at RA

While saving the log entries 1 manual, 30 random automated signatures and 12 automated ones, triggered by log file modifications, were created. However, only the manual, 5 random and 4 automatically triggered signatures were finally verified, which implies an attack on the log entries.

Deletion of an entry

We delete the last 3 entries, i.e. L10, L11, L12. The Server Log, the log verification at the Trusted Server and the signatures verification in the Regulatory Authority are presented in the Appendix A.1.2. The log file after

the attack can be found at Tests\SecurityAttacks\Deleting\USstorage\ LogFile.txt

Log Verification at TS

As expected, the deletion attack is not detected by the Schneier and Kesley security scheme.

Signatures Verification at RA

While saving the log entries, 1 manual signing, 16 random automated signature requests and 12 automated ones, triggered by log file modifications, were performed. We see that only 6 of the automatically created signatures were verified and only 9 of the triggered ones, therefore, we detect that an attack has occurred on the log file.

Conclusions

To sum up, all the types of simple manual attacks which were performed on the log file were detected by the Stathopoulos et al. security scheme, while the deletion attack was not detected by the Schneier and Kesley, as expected. However, the second one gives us a better indication on what entries have been tampered by the attacker, compared to the first one. The combination of these two schemes is sufficient for detecting an attack. Here the attacks were detected on a txt, but this gives us an indication that the security scheme should also detect the alterations occurring in a database.

CHAPTER 8

Conclusions

The problem we are dealing with in this thesis concerns the difficulties the pharmaceutical companies face, when they need to extend the system they use for producing pharmaceutical products. These systems are usually wired, time-triggered, soft real-time systems and need extended re-validation when we want to change them by adding or removing sensors from them. The reason is to ensure whether the new system is compliant with FDA regulations as a whole, because its components are coupled. This procedure is costly, as it requires a lot of time and paperwork.

The solution we have proposed in this thesis is a decoupled system network that facilitates the communication dependencies between the sensors, which send the data, and the log Server, which receives it. This way we can extend the system network by adding or removing sensors, without affecting the rest of the components of the system. Therefore, the FDA validation of the new system, after scaling will concern only the new sensors and not the other components.
We decided to design a prototype that will show how the new system network would behave. We assume that we have a distributed system of wireless sensors in a clean room area. We also assume that the sensors measure the environmental conditions of the room and contain a clock to timestamp the data they collect. There is also an alarm and a warning (flashlight) notification provided by an Alarm System. These are also time stamped. And finally there is a log Server where all the data are gathered and saved. We want to investigate which would be the ideal message exchange method in order to develop a decoupled components network.

To validate the proposed solution we have carried out a literature study of the FDA regulations to verify what requirements they impose on our system. Most of them concerned the safety of the electronic records before and after they are logged in the log Server. We also studied an example of the most

common user requirements of a Facility Monitoring System and the sensor requirements given by a known sensor vendor. Then we investigated the possible data transferring methods and we chose the publish-subscribe paradigm, as it is decoupled in space, time and synchronization, which implies scalability for the system. Moreover, its topic-based implementation ensures that a message is delivered to more than one receiver. Then we investigated the possible implementations of the publish-subscribe paradigm and chose the Java Messaging Service, which is responsible for creating, reading, sending and receiving messages among the components which communicate with it. We chose this approach because the publish and subscribe methods can be called from different threads and, thus, different components.
Furthermore, to ensure the safety of the data, after they had been sent to the Server, we have investigated several security schemes for logging the data in an encrypted way in the database. After identifying the possible attacks on the database, and vulnerabilities and limitations of each security scheme, we chose the one suggested by Stathopoulos et al., which includes the scheme of Schneier and Kesley, but extends it so that it overcome its vulnerabilities. The reason we chose this scheme is because it can work well with databases and it has very low intrusion probabilities.

From the results of this thesis we've learned that it is possible to decouple the existing communication system between sensors and log Server. This way the FDA validation is easier and quicker and our system scales faster.
We examined the latency between the sensor timestamp when it sends a measurement and the log timestamp when this measurement is saved in the log file. When the majority of our measurements include normal values, which don't produce warnings or alarms, then the average latency values are increasing as the number of sensors we run simultaneously increases.
When the majority of our measurements includes values which produce warnings or alarms, then the latency time decreases significantly for the data that are sent directly from the sensors (with topic "Data") to the Server. And this is reasonable, since the rest of the data passes through the alarm system first and, if it exceeds their thresholds, they it published again to the JMS with a new topic ("Alarm"). So, the average latency values for the data with topic "Alarm" increases as the number of sensors increases. The same happens with the frequency that values, deviating from the average ones appear. In all cases, the Server adjusts the subscription speed and thus manages to subscribe to all the incoming data and not miss any of it. However, the alarm system subscribes successfully to all the data, but cannot publish all the alarm events, most likely because the combination of the subscribe and publish functions in the same application, decreases the efficiency of the

second one. So, we should not include both functionalities on the same application.

Finally, we examined the effectiveness of Stathopoulos et al. security scheme against the attacks we identified for our log Server. The manual and automated signing method they suggest detects all the attacks we performed, but cannot inform us exactly which entries were compromised. The Schneier and Kesley scheme included in the above scheme cannot detect the deletion attacks, as its authors inform us, but gives a good indication which entries have been tampered by the intruder. So, the scheme of Stathopoulos et al., which extends the one of Schneir and Kesley, is sufficient for detecting an attack. The efficiency of the scheme on a text file gives us an indication that the security scheme can also detect security attacks on the entries when they are saved in a database.

CHAPTER 9

Future Work

In this chapter we consider aspects and methods we can improve and add in our system.

We should consider the way the encrypted logging must proceed after the system recovers from a sudden shut-down. Moreover, we should consider how to face situations when communication is lost temporarily between a component and the JMS. A watchdog timer [51] could be used, in this case, to detect when a computer application malfunctions and to notify the system administrator. We should also include more of the critical events for the Alarm Security Service. According to Stathopoulos et al. these are system restart and modifications of users and their privileges. For this purpose, more than one Alarm Service applications could be used.

We could also connect the Untrusted Server to a database, instead of the text file we currently use to store the encrypted log entries. Using a database application must also imply a management of access rights for the applications which are needed to access the encrypted log entries. These are the Untrusted Server, the Regulatory Authority and the Alarm Security Service.

As concerns the problem that the Alarm System faces when it has to both subscribe and publish data, it could be handled by using mostly sensors that evaluate their own data and raise alarms. These data will still be logged on the Server. The Alarm Server can be used only for subscribing to sensor measurements, evaluating and raising alarms and warnings, sending sms or email notifications to the user responsible for controlling the environmental conditions in the clean room. Although, we have already mentioned that sensors will raise their own alarms, it is good to have a system more reliable and secure than a sensor to detect for alarm events. However, we should avoid using the publish action, too.

In case we want to save the user actions, upon acknowledging an alarm, on the Server, we can publish them from another component (e.g. Control System) under a new topic “UserActions”, to which the Server will be

subscribed. These actions can be logged either together with the encrypted log entries, or in a separate log.

Finally, implementing our solution in an entire distributed system is also part of future work, as we should examine what hardware is needed (sensors, servers), how it is connected and design a network architecture (e.g. where to place the servers, the routers and other hardware components).

Abbreviations

FDA - Food and Drug Administration
FMS – Facility Monitoring System
GAMP - Good Automated Manufacturing Practice
GMP – Good Manufacturing Practice
GUM - Guide to the Expression of Uncertainty in Measurement
GWT - Google Web Toolkit
JMS – Java Messaging Service
JSF – Java Server Faces
PLC – Programmable Logic Controller
SCADA - Supervisory Control And Data Acquisition

Bibliography

[1] Food and Drug Administration (FDA), 21 CFR Part 11 – Electronic Records; Electronic Signatures, Ch. 1, U.S.A., pages 110-113, (4–1–2006).
[2] Food and Drug Administration (FDA), 21 CFR Part 820 – Quality Systems Regulation, Ch. 1, U.S.A., pages 138-151, (4–1–2006).
[3] ISPE, GAMP 5, A Risk-Based Approach to Compliant GxP Computerized Systems, (2008).
[4] http://eventbus.org/usage.html, last accessed: 28-04-2013.
[5] http://eventbus.org/api/org/bushe/swing/event/package-summary.html, last accessed: 28-04-2013.
[6] http://eventbus.org/api/index.html, last accessed: 28-04-2013.
[7] http://docs.oracle.com/javaee/6/tutorial/doc/bncdr.html, last-accessed: 27-05-2013.
[8] http://docs.oracle.com/javaee/6/tutorial/doc/bncdx.html, last-accessed: 27-05-2013.
[9] http://docs.oracle.com/javaee/6/tutorial/doc/bnceh.html. last-accessed: 28-05-2013.
[10] Food and Drug Administration (FDA), 21 CFR Part 210 – Current Good Manufacturing Practice in Manufacturing, Processing, Packing or Holding of Drugs; General, Ch. 1, U.S.A., pages 132-134, (4–1–2006).
[11] http://eventbus.org/, last accessed: 28-04-2013.
[12] http://codeblock.engio.net/?p=37, last accessed: 28-04-2013.
[13] https://code.google.com/p/simpleeventbus/, last accessed: 28-04-2013.
[14] http://spin.atomicobject.com/2012/01/13/the-guava-eventbus-on-guice/, last accessed: 28-04-2013.
[15] https://code.google.com/p/guava-libraries/wiki/EventBusExplained, last accessed: 28-04-2013.
[16] http://alextretyakov.blogspot.dk/2011/11/gwt-event-bus-basics.html, last accessed: 28-04-2013.
[17] http://tv.jetbrains.net/videocontent/gwt-event-bus-basics, last accessed: 28-04-2013.
[18] https://github.com/bennidi/mbassador#readme, last accessed: 28-04-2013.
[19] http://docs.codehaus.org/display/GUMTREE/How+to+use+the+GumTree+Event+Bus, last accessed: 28-04-2013.
[20] http://jumping-duck.com/tutorial/jquery-as-a-global-event-bus/, last accessed: 28-04-2013.
[21] http://www.dossier-andreas.net/software_architecture/eventbus.html, last accessed: 28-04-2013.

[22] http://northconcepts.com/blog/2011/07/05/use-dynamic-proxies-to-create-a-simple-powerful-event-bus-part-1/, last accessed: 28-04-2013.
[23] http://mate.asfusion.com/, last accessed: 28-04-2013.
[24] http://www.tutorialspoint.com/jsf/jsf_event_handling.htm, last accessed: 28-04-2013.
[25] http://www.onjava.com/pub/a/onjava/excerpt/JSF_chap8/index.html?page=1, last accessed: 28-04-2013.
[26] Food and Drug Administration (FDA), 21 CFR Part 211 – Current Good Manufacturing Practice for Finished Pharmaceuticals, Ch. 1, U.S.A., pag. 134-155, (4–1–2006).
[27] Food and Drug Administration (FDA), 21 CFR Part 58 – Good Laboratory Practice for Nonclinical Laboratory Studies, Ch. 1, U.S.A., pag. 301-315, (4–1–2006).
[28] GAMP5, "The Good Automated Manufacturing Practice (GAMP) Guide for Validation of Automated Systems in Pharmaceutical Manufacture", ISPE (International Society for Pharmaceutical Engineering), 2008.
[29] Bellare M., Yee B.S., Forward Integrity For Secure Audit Logs, Dept. of Computer Science & Engineering, University of California, San Diego, USA, pages 1-16, (23-11-1997).
[30] http://perso.ens-lyon.fr/laurent.lefevre/DOSMOS/DSM.html, last accessed: 17-06-2013.
[31] B. Wilkinson & M. Allen, Slides for *Parallel Programming Techniques & Applications Using Networked Workstations & Parallel Computers 2nd ed.*, Pearson Education Inc, slides 1-51, 2004.
[32] PACS Training Group, Introduction to MPI, Board of Trustees of the University of Illinois, NCSA Access, pages 1-28, 2001.
[33] http://searchsoa.techtarget.com/definition/Remote-Method-Invocation, last accessed: 17-06-2-13.
[34] http://careerride.com/RMI-Defined.aspx, last-accessed: 17-06-2013.
[35] http://stackoverflow.com/questions/2339853/benefits-and-disadvantages-of-using-java-rmi, last-accessed: 17-06-2013.
[36] http://www.codeproject.com/Articles/34333/Topic-based-publish-subscribe-design-pattern-imple, last-accessed: 18-06-2013.
[37]http://docs.oracle.com/cd/A97630_01/appdev.920/a96590/adg15pub.htm, last-accessed: 18-06-2013.
[38] http://c2.com/cgi/wiki?PublishSubscribeModel, last-accessed: 18-06-2013.
[39] http://www.slideshare.net/ishraqabd/publish-subscribe-model-overview-13368808, last-accessed: 18-06-2013.
[40] Schneier B., Kelsey J., Secure Audit Logs to Support Computer Forensics, ACM Transactions on Information and System Security, Vol. 2, No. 2, Pages 159–176, May 1999.
[41] http://www.metricstream.com/insights/sys_validation.htm, last-accessed: 10-07-2013.
[41] Ma D, Tsudik G., A New Approach to Secure Logging, ACM Transactions of Storage (TOS), volume 5, issue 1, article no.2, New York, USA, March 2009.

[42] Testo, 6351 Ethernet - Differential Pressure Transmitter, P2A software - Parameterizing, Adjusting and Analyzing Software, Instruction Manual.
[43] Ma D., Tsudik G., Forward-Secure Sequential Aggregate Authentication, IEEE Symposium on Security and Privacy, 2007 (SP '07), University of California, Irvine, pag. 1-10, 2007.
[44] Ma D., Practical Forward Secure Sequential Aggregate Signatures, in the Proceedings of the 2008 ACM symposium on Information, computer and communications security (ASIACSS '08) in Tokyo, Japan, University of California, Irvine, pag. 341- 350, 2008.
[45] Stathopoulos V., Kotzanikolaou P., Magkos E., A Framework for Secure and Verifiable Logging in Public Communication Networks, Springer-Verlag, LNCS 4347, Berlin, Heidelberg, pag.273-284, 2006.
[46] Kopetz H., Real-Time Systems, Design Principles for Distributed Embedded Applications, Springer, 2nd ed., XVIII, 378 p, 2011.
[47] http://www.youtube.com/watch?v=H7ecAAVUUZg, last accessed: 08-08-2013.
[48] http://www.youtube.com/watch?v=KUFn4wjFUb0, last accessed: 08-08-2013.
[49] http://www.hydronix.com/products/usb_module.php, last accessed: 08-08-2013.
[50] https://www.mfg-labs.com/mfg-labs/MetalMAX/sensor_interface_module.htm, last accessed: 08-08-2013.
[51]http://www.pcmag.com/encyclopedia/term/54224/watchdog-timer, last accessed: 08-08-2013.
[52] http://www.rgagnon.com/javadetails/java-0490.html, last accessed: 20-07-2013.
[53]http://www.digizol.com/2009/10/java-encrypt-decrypt-jce-salt.html, last accessed: 20-07-2013.
[54]http://liguoliang.com/2013/java-rsa-key-generationencryptiondecryption-rsa/, last accessed: 20-07-2013.
[55] http://docs.oracle.com/javase/tutorial/security/apisign/step2.html, last accessed: 20-07-2013.
[56]http://stackoverflow.com/questions/1709441/generate-rsa-key-pair-and-encode-private-as-string, last accessed: 20-07-2013.
[57] http://docs.oracle.com/javase/tutorial/security/apisign/, last accessed: 20-07-2013.
[58] http://docs.oracle.com/javase/tutorial/security/apisign/versig.html, last accessed: 20-07-2013.

APPENDIX A

Appendix

A.1 Output of the Evaluation Tests

A.1.1 Performance

Tests with Normal Temperature Measurements

The sensor timestamp is marked with blue color, while the log timestamp with orange.

Number of Sensors: 1

```
LogfileInitializationType 2013/08/02 1809580223 2013/08/02 1830000000 0 0
4444   ÅŠ2c,T;P\ÅÉW5x'ß`¼ÄÃtdùd±VÜ?£i:*Tf¥H?¢»+-??œ"”xáæöå¸Ýõ²L±:Òq-
ÉKì6-JÞè|â#ÒŸ‰JðO”Ÿ´—Sìs´yI°àþò—-´————————————————————————ÕD7A{ä]DY
š=lypÛ{ˆ¯c

6Xja49WYWo1cHq0AJy47hkynzWkK/lmtSFTZDpnuWA1AhWVkrOdew/7DuHqYrOOVRyi8S/oDO+OG
Jv30PCQpTvxKR1QlO4c7upIuto4UCl9VyUcZ9YpP/dqonn0OtUifFv2f85kxoBittnLXSCJ7qI3u
4umTiE+fdusGJsYw1x0wVTIgzK+tply/5dorSn+MkoNESgCGzIDDtLSxeAY2DlNKlRldTRHGUQLm
JPb7na8=
ResponseMessageType 2013/08/02 1809580244 2013/08/02 1830000000 1 1   1234
lKý¾’•P0————————————————————————————àÅñbC (!ê´ÖÖÇ¿»UC]™x¤˜™§],—
˜‡9³?Y#ŽÌÊÿ
p·¯ÈÀ»cÓˆh?’¤Xð/{        ª?@b’äçR-nãá†ÀM¬Œ¶í±){$ô…æ{|þô‰úUäÈ
Ëí.1Õ,C.inYF
Uu8glnmhbKNExbgqR18RWtImIinuIQvQUKR/2DpMgpY+Svmb6HFW1q4hjVGVLuxHMNF7pLzeQanq
fp4I/b9Oj4p/iKn9uFrLlm77xlf9DIwqnoNwBy7BsnbCrXPkifDSO/yRAAQBqnvfZeZY2/bWGmnX
D4u2tL5BadJwF7M4bf1Up7NVWc4+t1Egr5JQMO7vBWjHGuv65UZkYurAut+AHQ==
LogEntryType 2013/08/02 1810250249 2013/08/02 1830000000 2 2013/08/02
1810250165       A11      Temp     20.0     Celsius
LogEntryType 2013/08/02 1810290209 2013/08/02 1830000000 3 2013/08/02
1810290205       A11      Temp     20.4     Celsius
LogEntryType 2013/08/02 1810330219 2013/08/02 1830000000 4 2013/08/02
1810330206       A11      Temp     21.0     Celsius
LogEntryType 2013/08/02 1810370214 2013/08/02 1830000000 5 2013/08/02
1810370210       A11      Temp     22.0     Celsius
LogEntryType 2013/08/02 1810410223 2013/08/02 1830000000 6 2013/08/02
1810410211       A11      Temp     24.0     Celsius
LogEntryType 2013/08/02 1810450217 2013/08/02 1830000000 7 2013/08/02
1810450213       A11      Temp     25.4     Celsius
LogEntryType 2013/08/02 1810450239 2013/08/02 1830000000 8 AlarmSystem:
```

```
WARNING: 2013/08/02 1810450213   A11     Temp    25.4    Celsius
LogEntryType 2013/08/02 1810490218 2013/08/02 1830000000 9 2013/08/02
1810490214      A11     Temp    24.0    Celsius
LogEntryType 2013/08/02 1810530230 2013/08/02 1830000000 10 2013/08/02
1810530216      A11     Temp    23.6    Celsius
LogEntryType 2013/08/02 1810570223 2013/08/02 1830000000 11 2013/08/02
1810570219      A11     Temp    22.2    Celsius
NormalCloseMessage 2013/08/02 1811010239 2013/08/02 1830000000 12
```

Number of Sensors: 5

```
LogEntryType  2013/08/02  1840230326  2013/08/02  1859000000  23  2013/08/02
1840220800      A11e    Temp    20.0    Celsius
LogEntryType  2013/08/02  1840230522  2013/08/02  1859000000  24  2013/08/02
1840230508      A11b    Temp    21.2    Celsius
LogEntryType  2013/08/02  1840230653  2013/08/02  1859000000  25  2013/08/02
1840230642      A11c    Temp    20.9    Celsius
LogEntryType  2013/08/02  1840240227  2013/08/02  1859000000  26  2013/08/02
1840240214      A11d    Temp    20.6    Celsius
LogEntryType  2013/08/02  1840260502  2013/08/02  1859000000  27  2013/08/02
1840260401      A11     Temp    22.4    Celsius
LogEntryType  2013/08/02  1840270319  2013/08/02  1859000000  28  2013/08/02
1840270315      A11e    Temp    20.3    Celsius
LogEntryType  2013/08/02  1840270517  2013/08/02  1859000000  29  2013/08/02
1840270511      A11b    Temp    21.6    Celsius
LogEntryType  2013/08/02  1840270649  2013/08/02  1859000000  30  2013/08/02
1840270645      A11c    Temp    20.9    Celsius
LogEntryType  2013/08/02  1840280221  2013/08/02  1859000000  31  2013/08/02
1840280217      A11d    Temp    20.8    Celsius
LogEntryType  2013/08/02  1840300406  2013/08/02  1859000000  32  2013/08/02
1840300402      A11     Temp    22.8    Celsius
LogEntryType  2013/08/02  1840310320  2013/08/02  1859000000  33  2013/08/02
1840310316      A11e    Temp    20.6    Celsius
LogEntryType  2013/08/02  1840310519  2013/08/02  1859000000  34  2013/08/02
1840310514      A11b    Temp    22.0    Celsius
LogEntryType  2013/08/02  1840310650  2013/08/02  1859000000  35  2013/08/02
1840310646      A11c    Temp    21.2    Celsius
LogEntryType  2013/08/02  1840320232  2013/08/02  1859000000  36  2013/08/02
1840320218      A11d    Temp    20.9    Celsius
LogEntryType  2013/08/02  1840340407  2013/08/02  1859000000  37  2013/08/02
1840340403      A11     Temp    21.1    Celsius
LogEntryType  2013/08/02  1840350339  2013/08/02  1859000000  38  2013/08/02
1840350317      A11e    Temp    20.8    Celsius
LogEntryType  2013/08/02  1840350520  2013/08/02  1859000000  39  2013/08/02
1840350516      A11b    Temp    22.4    Celsius
LogEntryType  2013/08/02  1840350652  2013/08/02  1859000000  40  2013/08/02
1840350647      A11c    Temp    21.6    Celsius
LogEntryType  2013/08/02  1840360225  2013/08/02  1859000000  41  2013/08/02
1840360221      A11d    Temp    20.9    Celsius
LogEntryType  2013/08/02  1840380408  2013/08/02  1859000000  42  2013/08/02
1840380405      A11     Temp    21.5    Celsius
LogEntryType  2013/08/02  1840390323  2013/08/02  1859000000  43  2013/08/02
1840390319      A11e    Temp    20.9    Celsius
LogEntryType  2013/08/02  1840390522  2013/08/02  1859000000  44  2013/08/02
1840390518      A11b    Temp    22.8    Celsius
LogEntryType  2013/08/02  1840390652  2013/08/02  1859000000  45  2013/08/02
1840390649      A11c    Temp    22.0    Celsius
LogEntryType  2013/08/02  1840400229  2013/08/02  1859000000  46  2013/08/02
1840400222      A11d    Temp    21.2    Celsius
```

LogEntryType 2013/08/02 1840420410 2013/08/02 1859000000 47 2013/08/02
1840420406 A11 Temp 21.2 Celsius

Number of Sensors: 10

LogEntryType 2013/08/02 1925070195 2013/08/02 1945000000 92 2013/08/02
1925070180 A11j Temp 20.0 Celsius
LogEntryType 2013/08/02 1925070368 2013/08/02 1945000000 93 2013/08/02
1925070354 A11b Temp 21.0 Celsius
LogEntryType 2013/08/02 1925080482 2013/08/02 1945000000 94 2013/08/02
1925080478 A11h Temp 20.9 Celsius
LogEntryType 2013/08/02 1925080564 2013/08/02 1945000000 95 2013/08/02
1925080558 A11f Temp 22.0 Celsius
LogEntryType 2013/08/02 1925090287 2013/08/02 1945000000 96 2013/08/02
1925090283 A11i Temp 20.8 Celsius
LogEntryType 2013/08/02 1925090302 2013/08/02 1945000000 97 2013/08/02
1925090297 A11g Temp 21.2 Celsius
LogEntryType 2013/08/02 1925090539 2013/08/02 1945000000 98 2013/08/02
1925090535 A11 Temp 22.0 Celsius
LogEntryType 2013/08/02 1925090586 2013/08/02 1945000000 99 2013/08/02
1925090582 A11e Temp 22.8 Celsius
LogEntryType 2013/08/02 1925100182 2013/08/02 1945000000 100 2013/08/02
1925100178 A11d Temp 21.5 Celsius
LogEntryType 2013/08/02 1925100422 2013/08/02 1945000000 101 2013/08/02
1925100418 A11c Temp 20.7 Celsius
LogEntryType 2013/08/02 1925110197 2013/08/02 1945000000 102 2013/08/02
1925110193 A11j Temp 20.3 Celsius
LogEntryType 2013/08/02 1925110370 2013/08/02 1945000000 103 2013/08/02
1925110366 A11b Temp 21.2 Celsius
LogEntryType 2013/08/02 1925120484 2013/08/02 1945000000 104 2013/08/02
1925120480 A11h Temp 20.9 Celsius
LogEntryType 2013/08/02 1925120564 2013/08/02 1945000000 105 2013/08/02
1925120560 A11f Temp 22.4 Celsius
LogEntryType 2013/08/02 1925130300 2013/08/02 1945000000 106 2013/08/02
1925130285 A11i Temp 20.9 Celsius
LogEntryType 2013/08/02 1925130354 2013/08/02 1945000000 107 2013/08/02
1925130301 A11g Temp 21.6 Celsius
LogEntryType 2013/08/02 1925130550 2013/08/02 1945000000 108 2013/08/02
1925130538 A11 Temp 22.0 Celsius
LogEntryType 2013/08/02 1925130640 2013/08/02 1945000000 109 2013/08/02
1925130584 A11e Temp 21.1 Celsius
LogEntryType 2013/08/02 1925140213 2013/08/02 1945000000 110 2013/08/02
1925140182 A11d Temp 21.2 Celsius
LogEntryType 2013/08/02 1925140432 2013/08/02 1945000000 111 2013/08/02
1925140420 A11c Temp 21.0 Celsius
LogEntryType 2013/08/02 1925150198 2013/08/02 1945000000 112 2013/08/02
1925150194 A11j Temp 20.6 Celsius
LogEntryType 2013/08/02 1925150370 2013/08/02 1945000000 113 2013/08/02
1925150367 A11b Temp 21.6 Celsius
LogEntryType 2013/08/02 1925160484 2013/08/02 1945000000 114 2013/08/02
1925160481 A11h Temp 21.2 Celsius
LogEntryType 2013/08/02 1925160565 2013/08/02 1945000000 115 2013/08/02
1925160561 A11f Temp 22.8 Celsius
LogEntryType 2013/08/02 1925170292 2013/08/02 1945000000 116 2013/08/02
1925170288 A11i Temp 20.9 Celsius
LogEntryType 2013/08/02 1925170336 2013/08/02 1945000000 117 2013/08/02
1925170303 A11g Temp 22.0 Celsius
LogEntryType 2013/08/02 1925170548 2013/08/02 1945000000 118 2013/08/02
1925170545 A11 Temp 21.7 Celsius
LogEntryType 2013/08/02 1925170595 2013/08/02 1945000000 119 2013/08/02

```
1925170591      A11e    Temp    21.5    Celsius
LogEntryType  2013/08/02  1925180201  2013/08/02  1945000000  120  2013/08/02
1925180197      A11d    Temp    20.7    Celsius
LogEntryType  2013/08/02  1925180427  2013/08/02  1945000000  121  2013/08/02
1925180423      A11c    Temp    21.2    Celsius
```

Number of Sensors: 14

```
LogEntryType  2013/08/02  1955000794  2013/08/02  2015000000  163  2013/08/02
1955000787      A11i    Temp    21.6    Celsius
LogEntryType  2013/08/02  1955010111  2013/08/02  2015000000  164  2013/08/02
1955010084      A11f    Temp    20.7    Celsius
LogEntryType  2013/08/02  1955010280  2013/08/02  2015000000  165  2013/08/02
1955010089      A11e    Temp    21.2    Celsius
LogEntryType  2013/08/02  1955010369  2013/08/02  2015000000  166  2013/08/02
1955010338      A11b    Temp    22.4    Celsius
LogEntryType  2013/08/02  1955010685  2013/08/02  2015000000  167  2013/08/02
1955010671      A11j    Temp    20.9    Celsius
LogEntryType  2013/08/02  1955020174  2013/08/02  2015000000  168  2013/08/02
1955020156      A11d    Temp    22.0    Celsius
LogEntryType  2013/08/02  1955020322  2013/08/02  2015000000  169  2013/08/02
1955020189      A11g    Temp    21.5    Celsius
LogEntryType  2013/08/02  1955020782  2013/08/02  2015000000  170  2013/08/02
1955020752      A11k    Temp    20.8    Celsius
LogEntryType  2013/08/02  1955030282  2013/08/02  2015000000  171  2013/08/02
1955030246      A11l    Temp    20.6    Celsius
LogEntryType  2013/08/02  1955030452  2013/08/02  2015000000  172  2013/08/02
1955030306      A11c    Temp    21.9    Celsius
LogEntryType  2013/08/02  1955030501  2013/08/02  2015000000  173  2013/08/02
1955030439      A11     Temp    21.9    Celsius
LogEntryType  2013/08/02  1955030585  2013/08/02  2015000000  174  2013/08/02
1955030446      A11h    Temp    22.8    Celsius
LogEntryType  2013/08/02  1955040059  2013/08/02  2015000000  175  2013/08/02
1955040042      A11m    Temp    20.0    Celsius
LogEntryType  2013/08/02  1955040794  2013/08/02  2015000000  176  2013/08/02
1955040789      A11i    Temp    22.0    Celsius
LogEntryType  2013/08/02  1955050114  2013/08/02  2015000000  177  2013/08/02
1955050110      A11e    Temp    21.6    Celsius
LogEntryType  2013/08/02  1955050131  2013/08/02  2015000000  178  2013/08/02
1955050121      A11f    Temp    21.0    Celsius
LogEntryType  2013/08/02  1955050348  2013/08/02  2015000000  179  2013/08/02
1955050341      A11b    Temp    22.5    Celsius
LogEntryType  2013/08/02  1955050681  2013/08/02  2015000000  180  2013/08/02
1955050674      A11j    Temp    21.2    Celsius
LogEntryType  2013/08/02  1955060167  2013/08/02  2015000000  181  2013/08/02
1955060161      A11d    Temp    21.7    Celsius
LogEntryType  2013/08/02  1955060197  2013/08/02  2015000000  182  2013/08/02
1955060191      A11g    Temp    21.2    Celsius
LogEntryType  2013/08/02  1955060782  2013/08/02  2015000000  183  2013/08/02
1955060772      A11k    Temp    20.9    Celsius
LogEntryType  2013/08/02  1955070264  2013/08/02  2015000000  184  2013/08/02
1955070256      A11l    Temp    20.8    Celsius
LogEntryType  2013/08/02  1955070318  2013/08/02  2015000000  185  2013/08/02
1955070309      A11c    Temp    22.2    Celsius
LogEntryType  2013/08/02  1955070449  2013/08/02  2015000000  186  2013/08/02
1955070444      A11     Temp    21.8    Celsius
LogEntryType  2013/08/02  1955070471  2013/08/02  2015000000  187  2013/08/02
1955070452      A11h    Temp    21.1    Celsius
LogEntryType  2013/08/02  1955080059  2013/08/02  2015000000  188  2013/08/02
1955080054      A11m    Temp    20.3    Celsius
```

```
LogEntryType  2013/08/02  1955080795  2013/08/02  2015000000  189  2013/08/02
1955080790      A11i    Temp    22.4    Celsius
LogEntryType  2013/08/02  1955090116  2013/08/02  2015000000  190  2013/08/02
1955090111      A11e    Temp    22.0    Celsius
LogEntryType  2013/08/02  1955090132  2013/08/02  2015000000  191  2013/08/02
1955090126      A11f    Temp    21.2    Celsius
LogEntryType  2013/08/02  1955090346  2013/08/02  2015000000  192  2013/08/02
1955090342      A11b    Temp    22.2    Celsius
LogEntryType  2013/08/02  1955090682  2013/08/02  2015000000  193  2013/08/02
1955090677      A11j    Temp    21.6    Celsius
LogEntryType  2013/08/02  1955100172  2013/08/02  2015000000  194  2013/08/02
1955100162      A11d    Temp    21.8    Celsius
LogEntryType  2013/08/02  1955100209  2013/08/02  2015000000  195  2013/08/02
1955100192      A11g    Temp    20.7    Celsius
LogEntryType  2013/08/02  1955100789  2013/08/02  2015000000  196  2013/08/02
1955100775      A11k    Temp    20.9    Celsius
LogEntryType  2013/08/02  1955110263  2013/08/02  2015000000  197  2013/08/02
1955110166      A11n    Temp    20.0    Celsius
LogEntryType  2013/08/02  1955110317  2013/08/02  2015000000  198  2013/08/02
1955110257      A11l    Temp    20.9    Celsius
LogEntryType  2013/08/02  1955110408  2013/08/02  2015000000  199  2013/08/02
1955110340      A11c    Temp    22.4    Celsius
LogEntryType  2013/08/02  1955110539  2013/08/02  2015000000  200  2013/08/02
1955110454      A11h    Temp    21.5    Celsius
NormalCloseMessage 2013/08/02 1955110657 2013/08/02 2015000000 201
```

Tests with Alarm Temperature Measurements

Number of Sensors: 1

```
LogEntryType  2013/08/02  2005180832  2013/08/02  2030000000  8  2013/08/02
2005180832      A11     Temp    24.5    Celsius
LogEntryType  2013/08/02  2005180879  2013/08/02  2030000000  9  AlarmSystem:
WARNING: 2013/08/02 2005180832  A11     Temp    24.5    Celsius
LogEntryType  2013/08/02  2005220843  2013/08/02  2030000000  10  2013/08/02
2005220843      A11     Temp    25.2    Celsius
LogEntryType  2013/08/02  2005220843  2013/08/02  2030000000  11  AlarmSystem:
WARNING: 2013/08/02 2005220843  A11     Temp    25.2    Celsius
LogEntryType  2013/08/02  2005260854  2013/08/02  2030000000  12  2013/08/02
2005260854      A11     Temp    25.7    Celsius
LogEntryType  2013/08/02  2005260854  2013/08/02  2030000000  13  AlarmSystem:
WARNING: 2013/08/02 2005260854  A11     Temp    25.7    Celsius
LogEntryType  2013/08/02  2005300865  2013/08/02  2030000000  14  2013/08/02
2005300865      A11     Temp    27.3    Celsius
LogEntryType  2013/08/02  2005300913  2013/08/02  2030000000  15  AlarmSystem:
WARNING: 2013/08/02 2005300865  A11     Temp    27.3    Celsius
LogEntryType  2013/08/02  2005340877  2013/08/02  2030000000  16  2013/08/02
2005340877      A11     Temp    28.2    Celsius
LogEntryType  2013/08/02  2005340877  2013/08/02  2030000000  17  AlarmSystem:
WARNING: 2013/08/02 2005340877  A11     Temp    28.2    Celsius
LogEntryType  2013/08/02  2005380889  2013/08/02  2030000000  18  2013/08/02
2005380889      A11     Temp    29.1    Celsius
LogEntryType  2013/08/02  2005380924  2013/08/02  2030000000  19  AlarmSystem:
WARNING: 2013/08/02 2005380889  A11     Temp    29.1    Celsius
LogEntryType  2013/08/02  2005420904  2013/08/02  2030000000  20  2013/08/02
2005420904      A11     Temp    30.0    Celsius
LogEntryType  2013/08/02  2005420904  2013/08/02  2030000000  21  AlarmSystem:
ALARM: 2013/08/02 2005420904    A11     Temp    30.0    Celsius
LogEntryType  2013/08/02  2005460918  2013/08/02  2030000000  22  2013/08/02
```

```
2005460918      A11     Temp    30.6    Celsius
LogEntryType  2013/08/02  2005460954  2013/08/02  2030000000  23  AlarmSystem:
ALARM: 2013/08/02 2005460918     A11     Temp    30.6    Celsius
LogEntryType  2013/08/02  2005500933  2013/08/02  2030000000  24  2013/08/02
2005500933      A11     Temp    31.2    Celsius
LogEntryType  2013/08/02  2005500933  2013/08/02  2030000000  25  AlarmSystem:
ALARM: 2013/08/02 2005500933     A11     Temp    31.2    Celsius
LogEntryType  2013/08/02  2005540946  2013/08/02  2030000000  26  2013/08/02
2005540946      A11     Temp    32.4    Celsius
LogEntryType  2013/08/02  2005540946  2013/08/02  2030000000  27  AlarmSystem:
ALARM: 2013/08/02 2005540946     A11     Temp    32.4    Celsius
LogEntryType  2013/08/02  2005580957  2013/08/02  2030000000  28  2013/08/02
2005580957      A11     Temp    33.5    Celsius
LogEntryType  2013/08/02  2005580957  2013/08/02  2030000000  29  AlarmSystem:
ALARM: 2013/08/02 2005580957     A11     Temp    33.5    Celsius
LogEntryType  2013/08/02  2006020968  2013/08/02  2030000000  30  2013/08/02
2006020968      A11     Temp    33.8    Celsius
LogEntryType  2013/08/02  2006020984  2013/08/02  2030000000  31  AlarmSystem:
ALARM: 2013/08/02 2006020968     A11     Temp    33.8    Celsius
LogEntryType  2013/08/02  2006060980  2013/08/02  2030000000  32  2013/08/02
2006060980      A11     Temp    34.3    Celsius
LogEntryType  2013/08/02  2006070032  2013/08/02  2030000000  33  AlarmSystem:
ALARM: 2013/08/02 2006060980     A11     Temp    34.3    Celsius
LogEntryType  2013/08/02  2006100995  2013/08/02  2030000000  34  2013/08/02
2006100995      A11     Temp    34.8    Celsius
LogEntryType  2013/08/02  2006100995  2013/08/02  2030000000  35  AlarmSystem:
ALARM: 2013/08/02 2006100995     A11     Temp    34.8    Celsius
NormalCloseMessage 2013/08/02 2006150007 2013/08/02 2030000000 36
```

Number of Sensors: 5

```
LogEntryType  2013/08/02  2018380032  2013/08/02  2045000000  70  2013/08/02
2018380032      A11e    Temp    24.5    Celsius
LogEntryType  2013/08/02  2018380048  2013/08/02  2045000000  71  AlarmSystem:
WARNING: 2013/08/02 2018380032   A11e    Temp    24.5    Celsius
LogEntryType  2013/08/02  2018380236  2013/08/02  2045000000  72  2013/08/02
2018380236      A11b    Temp    29.1    Celsius
LogEntryType  2013/08/02  2018380269  2013/08/02  2045000000  73  AlarmSystem:
WARNING: 2013/08/02 2018380236   A11b    Temp    29.1    Celsius
LogEntryType  2013/08/02  2018380878  2013/08/02  2045000000  74  2013/08/02
2018380877      A11d    Temp    25.7    Celsius
LogEntryType  2013/08/02  2018380894  2013/08/02  2045000000  75  AlarmSystem:
WARNING: 2013/08/02 2018380877   A11d    Temp    25.7    Celsius
LogEntryType  2013/08/02  2018400114  2013/08/02  2045000000  76  2013/08/02
2018400114      A11c    Temp    28.2    Celsius
LogEntryType  2013/08/02  2018400129  2013/08/02  2045000000  77  AlarmSystem:
WARNING: 2013/08/02 2018400114   A11c    Temp    28.2    Celsius
LogEntryType  2013/08/02  2018410130  2013/08/02  2045000000  78  2013/08/02
2018410130      A11     Temp    31.2    Celsius
LogEntryType  2013/08/02  2018410145  2013/08/02  2045000000  79  AlarmSystem:
ALARM: 2013/08/02 2018410130     A11     Temp    31.2    Celsius
LogEntryType  2013/08/02  2018420036  2013/08/02  2045000000  80  2013/08/02
2018420036      A11e    Temp    25.2    Celsius
LogEntryType  2013/08/02  2018420051  2013/08/02  2045000000  81  AlarmSystem:
WARNING: 2013/08/02 2018420036   A11e    Temp    25.2    Celsius
LogEntryType  2013/08/02  2018420239  2013/08/02  2045000000  82  2013/08/02
2018420239      A11b    Temp    30.0    Celsius
LogEntryType  2013/08/02  2018420255  2013/08/02  2045000000  83  AlarmSystem:
ALARM: 2013/08/02 2018420239     A11b    Temp    30.0    Celsius
LogEntryType  2013/08/02  2018420881  2013/08/02  2045000000  84  2013/08/02
```

2018420881 A11d Temp 27.3 Celsius
LogEntryType 2013/08/02 2018420897 2013/08/02 2045000000 85 AlarmSystem:
WARNING: 2013/08/02 2018420881 A11d Temp 27.3 Celsius
LogEntryType 2013/08/02 2018440116 2013/08/02 2045000000 86 2013/08/02
2018440116 A11c Temp 29.1 Celsius
LogEntryType 2013/08/02 2018440164 2013/08/02 2045000000 87 AlarmSystem:
WARNING: 2013/08/02 2018440116 A11c Temp 29.1 Celsius
LogEntryType 2013/08/02 2018450132 2013/08/02 2045000000 88 2013/08/02
2018450132 A11 Temp 32.4 Celsius
LogEntryType 2013/08/02 2018450180 2013/08/02 2045000000 89 AlarmSystem:
ALARM: 2013/08/02 2018450132 A11 Temp 32.4 Celsius
LogEntryType 2013/08/02 2018460039 2013/08/02 2045000000 90 2013/08/02
2018460039 A11e Temp 25.7 Celsius
LogEntryType 2013/08/02 2018460091 2013/08/02 2045000000 91 AlarmSystem:
WARNING: 2013/08/02 2018460039 A11e Temp 25.7 Celsius
LogEntryType 2013/08/02 2018460247 2013/08/02 2045000000 92 2013/08/02
2018460247 A11b Temp 30.6 Celsius
LogEntryType 2013/08/02 2018460263 2013/08/02 2045000000 93 AlarmSystem:
ALARM: 2013/08/02 2018460247 A11b Temp 30.6 Celsius
LogEntryType 2013/08/02 2018460888 2013/08/02 2045000000 94 2013/08/02
2018460888 A11d Temp 28.2 Celsius
LogEntryType 2013/08/02 2018460903 2013/08/02 2045000000 95 AlarmSystem:
WARNING: 2013/08/02 2018460888 A11d Temp 28.2 Celsius
LogEntryType 2013/08/02 2018480122 2013/08/02 2045000000 96 2013/08/02
2018480122 A11c Temp 30.0 Celsius
LogEntryType 2013/08/02 2018480138 2013/08/02 2045000000 97 AlarmSystem:
ALARM: 2013/08/02 2018480122 A11c Temp 30.0 Celsius
LogEntryType 2013/08/02 2018490140 2013/08/02 2045000000 98 2013/08/02
2018490140 A11 Temp 33.5 Celsius
LogEntryType 2013/08/02 2018490156 2013/08/02 2045000000 99 AlarmSystem:
ALARM: 2013/08/02 2018490140 A11 Temp 33.5 Celsius

Number of Sensors: 10

LogEntryType 2013/08/02 2042360571 2013/08/02 2059000000 173 2013/08/02
2042360490 A11j Temp 22.0 Celsius
LogEntryType 2013/08/02 2042360758 2013/08/02 2059000000 174 2013/08/02
2042360758 A11 Temp 34.3 Celsius
LogEntryType 2013/08/02 2042360806 2013/08/02 2059000000 175 AlarmSystem:
ALARM: 2013/08/02 2042360758 A11 Temp 34.3 Celsius
LogEntryType 2013/08/02 2042370008 2013/08/02 2059000000 176 2013/08/02
2042370008 A11e Temp 30.0 Celsius
LogEntryType 2013/08/02 2042370024 2013/08/02 2059000000 177 AlarmSystem:
ALARM: 2013/08/02 2042370008 A11e Temp 30.0 Celsius
LogEntryType 2013/08/02 2042370338 2013/08/02 2059000000 178 2013/08/02
2042370338 A11g Temp 27.3 Celsius
LogEntryType 2013/08/02 2042370354 2013/08/02 2059000000 179 AlarmSystem:
WARNING: 2013/08/02 2042370338 A11g Temp 27.3 Celsius
LogEntryType 2013/08/02 2042380447 2013/08/02 2059000000 180 2013/08/02
2042380447 A11b Temp 33.8 Celsius
LogEntryType 2013/08/02 2042380462 2013/08/02 2059000000 181 AlarmSystem:
ALARM: 2013/08/02 2042380447 A11b Temp 33.8 Celsius
LogEntryType 2013/08/02 2042390166 2013/08/02 2059000000 182 2013/08/02
2042390166 A11i Temp 24.5 Celsius
LogEntryType 2013/08/02 2042390182 2013/08/02 2059000000 183 AlarmSystem:
WARNING: 2013/08/02 2042390166 A11i Temp 24.5 Celsius
LogEntryType 2013/08/02 2042390432 2013/08/02 2059000000 184 2013/08/02
2042390432 A11f Temp 29.1 Celsius
LogEntryType 2013/08/02 2042390448 2013/08/02 2059000000 185 AlarmSystem:
WARNING: 2013/08/02 2042390432 A11f Temp 29.1 Celsius

```
LogEntryType  2013/08/02  2042390794  2013/08/02  2059000000  186  2013/08/02
2042390794       A11d     Temp     31.2     Celsius
LogEntryType  2013/08/02  2042390810  2013/08/02  2059000000  187  AlarmSystem:
ALARM: 2013/08/02 2042390794     A11d     Temp     31.2     Celsius
LogEntryType  2013/08/02  2042400294  2013/08/02  2059000000  188  2013/08/02
2042400294       A11h     Temp     25.7     Celsius
LogEntryType  2013/08/02  2042400310  2013/08/02  2059000000  189  AlarmSystem:
WARNING: 2013/08/02 2042400294   A11h     Temp     25.7     Celsius
LogEntryType  2013/08/02  2042400483  2013/08/02  2059000000  190  2013/08/02
2042400483       A11c     Temp     33.5     Celsius
LogEntryType  2013/08/02  2042400498  2013/08/02  2059000000  191  AlarmSystem:
ALARM: 2013/08/02 2042400483     A11c     Temp     33.5     Celsius
LogEntryType  2013/08/02  2042400515  2013/08/02  2059000000  192  2013/08/02
2042400514       A11j     Temp     22.5     Celsius
LogEntryType  2013/08/02  2042400765  2013/08/02  2059000000  193  2013/08/02
2042400765       A11      Temp     34.8     Celsius
LogEntryType  2013/08/02  2042400781  2013/08/02  2059000000  194  AlarmSystem:
ALARM: 2013/08/02 2042400765     A11      Temp     34.8     Celsius
LogEntryType  2013/08/02  2042410016  2013/08/02  2059000000  195  2013/08/02
2042410016       A11e     Temp     30.6     Celsius
LogEntryType  2013/08/02  2042410032  2013/08/02  2059000000  196  AlarmSystem:
ALARM: 2013/08/02 2042410016     A11e     Temp     30.6     Celsius
LogEntryType  2013/08/02  2042410345  2013/08/02  2059000000  197  2013/08/02
2042410345       A11g     Temp     28.2     Celsius
LogEntryType  2013/08/02  2042410360  2013/08/02  2059000000  198  AlarmSystem:
WARNING: 2013/08/02 2042410345   A11g     Temp     28.2     Celsius
LogEntryType  2013/08/02  2042420454  2013/08/02  2059000000  199  2013/08/02
2042420454       A11b     Temp     34.3     Celsius
LogEntryType  2013/08/02  2042420470  2013/08/02  2059000000  200  AlarmSystem:
ALARM: 2013/08/02 2042420454     A11b     Temp     34.3     Celsius
LogEntryType  2013/08/02  2042430174  2013/08/02  2059000000  201  2013/08/02
2042430174       A11i     Temp     25.2     Celsius
LogEntryType  2013/08/02  2042430189  2013/08/02  2059000000  202  AlarmSystem:
WARNING: 2013/08/02 2042430174   A11i     Temp     25.2     Celsius
LogEntryType  2013/08/02  2042430440  2013/08/02  2059000000  203  2013/08/02
2042430440       A11f     Temp     30.0     Celsius
LogEntryType  2013/08/02  2042430456  2013/08/02  2059000000  204  AlarmSystem:
ALARM: 2013/08/02 2042430440     A11f     Temp     30.0     Celsius
LogEntryType  2013/08/02  2042430803  2013/08/02  2059000000  205  2013/08/02
2042430803       A11d     Temp     32.4     Celsius
LogEntryType  2013/08/02  2042430818  2013/08/02  2059000000  206  AlarmSystem:
ALARM: 2013/08/02 2042430803     A11d     Temp     32.4     Celsius
LogEntryType  2013/08/02  2042440303  2013/08/02  2059000000  207  2013/08/02
2042440303       A11h     Temp     27.3     Celsius
LogEntryType  2013/08/02  2042440319  2013/08/02  2059000000  208  AlarmSystem:
WARNING: 2013/08/02 2042440303   A11h     Temp     27.3     Celsius
LogEntryType  2013/08/02  2042440491  2013/08/02  2059000000  209  2013/08/02
2042440491       A11c     Temp     33.8     Celsius
LogEntryType  2013/08/02  2042440507  2013/08/02  2059000000  210  AlarmSystem:
ALARM: 2013/08/02 2042440491     A11c     Temp     33.8     Celsius
LogEntryType 2013/08/02 2042440523 2013/08/02 2059000000 211 2013/08/02
2042440522       A11j     Temp     23.7     Celsius
```

Throughput

Part of the Alarm System Console Output

```
Sender: A11d
```

```
Subscribe to Message: 2013/08/02 2042590822      A11d     Temp     34.8
        Celsius
Publish: AlarmSystem: ALARM: 2013/08/02 2042590822       A11d     Temp     34.8
        Celsius

Sender: A11e
Subscribe to Message: 2013/08/02 2043010057      A11e     Temp     34.3
        Celsius
Aug  02,  2013  8:43:01  PM  com.sun.messaging.jmq.jmsclient.ExceptionHandler
logCaughtException
WARNING:  [I500]:  Caught  JVM  Exception:  com.sun.messaging.jms.JMSException:
[ADD_PRODUCER_REPLY(19)] [C4036]: A broker error occurred. :[409] [B4183]:
Producer can not be added to destination Alarm [Topic], limit of 100 producers
would be exceeded user=guest, broker=localhost:7676(49652)
com.sun.messaging.jms.JMSException: [ADD_PRODUCER_REPLY(19)] [C4036]: A broker
error occurred. :[409] [B4183]: Producer can not be added to destination Alarm
[Topic],  limit  of  100  producers  would  be  exceeded  user=guest,
broker=localhost:7676(49652)                                              at
com.sun.messaging.jmq.jmsclient.ProtocolHandler.throwServerErrorException(Prot
ocolHandler.java:4103)   at
com.sun.messaging.jmq.jmsclient.ProtocolHandler.createMessageProducer(Protocol
Handler.java:1353)       at
com.sun.messaging.jmq.jmsclient.ProtocolHandler.createMessageProducer(Protocol
Handler.java:1247)       at
com.sun.messaging.jmq.jmsclient.ProtocolHandler.createMessageProducer(Protocol
Handler.java:1241)       at
com.sun.messaging.jmq.jmsclient.MessageProducerImpl.<init>(MessageProducerImpl
.java:121)       at
com.sun.messaging.jmq.jmsclient.TopicPublisherImpl.<init>(TopicPublisherImpl.j
ava:73) at
com.sun.messaging.jmq.jmsclient.UnifiedSessionImpl.createPublisher(UnifiedSess
ionImpl.java:430)        at
com.sun.messaging.jms.ra.SessionAdapter.createPublisher(SessionAdapter.java:34
7)       at com.javamsgservice.AlarmObject.publish(AlarmObject.java:124)
        at com.javamsgservice.AlarmObject.onMessage(AlarmObject.java:86)  at
com.sun.messaging.jmq.jmsclient.MessageConsumerImpl.deliverAndAcknowledge(Mess
ageConsumerImpl.java:358)        at
com.sun.messaging.jmq.jmsclient.MessageConsumerImpl.onMessage(MessageConsumerI
mpl.java:287)    at
com.sun.messaging.jmq.jmsclient.SessionReader.deliver(SessionReader.java:119)
        at
com.sun.messaging.jmq.jmsclient.ConsumerReader.run(ConsumerReader.java:192)
        at java.lang.Thread.run(Unknown Source)
Caused    by:    com.sun.messaging.jms.JMSException:    [ADD_PRODUCER_REPLY(19)]
[C4036]: A broker error occurred. :[409] [B4183]: Producer can not be added to
destination  Alarm  [Topic],  limit  of  100  producers  would  be  exceeded
user=guest, broker=localhost:7676(49652) at
com.sun.messaging.jmq.jmsclient.ProtocolHandler.throwServerErrorException(Prot
ocolHandler.java:4088)
        ... 14 more

Sender: A11f
Subscribe to Message: 2013/08/02 2043030476      A11f     Temp     33.8
        Celsius
Aug  02,  2013  8:43:03  PM  com.sun.messaging.jmq.jmsclient.ExceptionHandler
logCaughtException
WARNING:  [I500]:  Caught  JVM  Exception:  com.sun.messaging.jms.JMSException:
[ADD_PRODUCER_REPLY(19)] [C4036]: A broker error occurred. :[409] [B4183]:
Producer can not be added to destination Alarm [Topic], limit of 100 producers
would be exceeded user=guest, broker=localhost:7676(49652)
```

```
com.sun.messaging.jms.JMSException: [ADD_PRODUCER_REPLY(19)] [C4036]: A broker
error occurred. :[409] [B4183]: Producer can not be added to destination Alarm
[Topic], limit of 100 producers would be exceeded user=guest,
broker=localhost:7676(49652)                                          at
com.sun.messaging.jmq.jmsclient.ProtocolHandler.throwServerErrorException(Prot
ocolHandler.java:4103)  at
com.sun.messaging.jmq.jmsclient.ProtocolHandler.createMessageProducer(Protocol
Handler.java:1353)      at
com.sun.messaging.jmq.jmsclient.ProtocolHandler.createMessageProducer(Protocol
Handler.java:1247)      at
com.sun.messaging.jmq.jmsclient.ProtocolHandler.createMessageProducer(Protocol
Handler.java:1241)      at
com.sun.messaging.jmq.jmsclient.MessageProducerImpl.<init>(MessageProducerImpl
.java:121)      at
com.sun.messaging.jmq.jmsclient.TopicPublisherImpl.<init>(TopicPublisherImpl.j
ava:73) at
com.sun.messaging.jmq.jmsclient.UnifiedSessionImpl.createPublisher(UnifiedSess
ionImpl.java:430)       at
com.sun.messaging.jms.ra.SessionAdapter.createPublisher(SessionAdapter.java:34
7)      at com.javamsgservice.AlarmObject.publish(AlarmObject.java:124)
        at com.javamsgservice.AlarmObject.onMessage(AlarmObject.java:86)  at
com.sun.messaging.jmq.jmsclient.MessageConsumerImpl.deliverAndAcknowledge(Mess
ageConsumerImpl.java:358)       at
com.sun.messaging.jmq.jmsclient.MessageConsumerImpl.onMessage(MessageConsumerI
mpl.java:287)   at
com.sun.messaging.jmq.jmsclient.SessionReader.deliver(SessionReader.java:119)
        at
com.sun.messaging.jmq.jmsclient.ConsumerReader.run(ConsumerReader.java:192)
        at java.lang.Thread.run(Unknown Source)
Caused   by:   com.sun.messaging.jms.JMSException:   [ADD_PRODUCER_REPLY(19)]
[C4036]: A broker error occurred. :[409] [B4183]: Producer can not be added to
destination Alarm [Topic], limit of 100 producers would be exceeded
user=guest, broker=localhost:7676(49652) at
com.sun.messaging.jmq.jmsclient.ProtocolHandler.throwServerErrorException(Prot
ocolHandler.java:4088)
        ... 14 more

Sender: A11d
Subscribe to Message: exit
```

A.1.2 Security Attacks

Insertion of a new entry/Duplication of an entry

Server

Here we see the data being logged as entries Li by the Untrusted Server in a log file.

```
L0: LogfileInitializationType
UGVr2k9Tuq/n2UNSsebnvUhKS5J9x0MiyrE72+AsJsEJSW5//E+P5o23MvPCzT7d1nphNmSRzR78
EIAKp2FZNbZ211HocZAKoJsYC+Dpf6pb0Bk/pyt6vaQhmjC2VrD6NXq8kIxLEzzTU+7xLcyWAUIY
r9aSxcKd0EkHtLeYEM04cX6dDRHFbr+MyIrm5iAqSoFZTa9dblMU/CgfSk8z2juDxqAepn9AhrwN
BbEckA8draKbMYrNvSf1TS7rP7dD4oxWatjiSPuvA1yvZ0vQQFq4NHmcNl60OoQu1eNgA4BNJct1
fwRHlxUof2CqhGcvKUaHIb2Jm3h+AnkJlBikvMJjAQ3DMVm+KqWt9HiPLzYPeKeZqNFa2n2X/fTp
EwXaV5p3fLB1NLG+IuljQ9Yc+hI/+2Vvldv3jZcWXoKrdpKO+h5zUsfkdAKbSej79qvd01ZO3BSh
/Qltbt1vZzy8QbC2iF4gmy4o8L2b2JWArH12ZtMR6Iqf2ZgbBiD8y6wW1IaohczeB4W87C6PEPVO
```

```
COtlYSgztDE5VMEIYEfC+dyTKmF/4WO34udRgJ2UmXiq   -2131737990   -1788860721
L1: ResponseMessageType
gUywYu3ZjPTvlsYDsJIu5W+R3wG2pmazNmYXsCSGQTHL83HdStEqSiMb+G1exHsI36Y+TM8jHzwz
r9qBzKyVMRhZqjSfqay9XnfHEwrZBKUGTM2Q58lwczLzzB8MaEJhUYYlf7T7AXsR1a54oPJ6dcwB
zLecKhAhlj6vGn5P76RhYDsvVh5PxfwFNn6VzfylA2q3Fh+A/VO1ucBlnYR7qedj0Ct/j0teUQ9u
bA8nI8+zfM9jdYpCt95isTJod7suTn8Joz/THwsZNs9amfe4LStQdnjOGoQt1UFs5FCiB6jpKqcS
Qha8rsYkeWmsfDiusAFayvL99zUFpD8qfTlSqbWyhzoOZpPobGuF9DZgy0TIw4KdJoYiMepJgz14
9tHcekYi3V3H+Vmv42C1MCU7TcivMCVEEVijEna8it/pVOIDCHgiX/ZemkYEjGiMgAIx+9U6zhyr
QxB7vVWWQLseq2CFMBRu9d1qKbtiWJuLTtiB8GX/WrAktt5DVSRtmq1z3QZdc+Qwgbt9uKucjpBy
WUByX+UaHdXzGXR2mz7pHuM=   -673921588   -1097811294
Sender: A11
Message: 2013/08/02 1109170037   A11     Temp    20.0    Celsius
L2: LogEntryType
G693AlsnQhBh/BVYr7pM95LK68SXCqRjgWQ7vDozihQ8goOLKU1zw0mUETHHUoaCPveptdw6oYfD
Ri7ZP14YWRD94+2GRZGugD0A2lvrYelXpIPG4DRtuFVoh2x/pB5F   -1798661150   19329142
Sender: A11
Message: 2013/08/02 1109210067   A11     Temp    20.4    Celsius
L3: LogEntryType
IJ6z0QgONqWemDWSqObwtvZ/DB0D6iFGVf3imoIp6i7AONa8IuJzP1o5wkJEBNgtsXPpmrVjSw+2
AQT6nC09CuMwezCy15yafdzfX+IoY+5XbsT96EKbLxqDqnMcloWb   -834139319   1263724178
Sender: A11
Message: 2013/08/02 1109250079   A11     Temp    21.0    Celsius
L4: LogEntryType
l4rAv3YGIDH6wpmf4+sN8RIcxdR3KxXJeW/z6b/P0ItlsyQK/fdOfB822wPfzCyiPkWK4C6bs+UR
rxX0bcvKxGkNJOusJyijphpxSm6V8SEV+PF+3ndwfACX/nO052Zx   -2096999132   -
2078260828
Sender: A11
Message: 2013/08/02 1109290090   A11     Temp    22.0    Celsius
L5: LogEntryType
kDUiQUlJAeGH1v9dC/UnIG3dK4GBVEztaRDdTfxTJKhkfWi+uOlPeUUOtAryiXpbVpJZrh1K1y9S
zS5493giqthlO0Q2QefU36aGxNwK+gP5OBopCiOrdPk7h9SfuZqT   989650785   1385486530
Sender: A11
Message: 2013/08/02 1109330092   A11     Temp    24.0    Celsius
L6: LogEntryType
5DcZft5wiQt8W90pC2vWETCDg0/kjIefs1qwM4El+r1xYVz3Qyk1IcztLyEveyNHTL3QoRxFsadF
E02w9MGTRAP9oiPc9uHksv6KbB4sf8w2sSLPk/eTePAME3RW4n0/   -816886799   2080965872
Sender: A11
Message: 2013/08/02 1109370105   A11     Temp    25.4    Celsius
L7: LogEntryType
Fc7bF6KLQqSAg5llsGDcn7OWLJ8A5+cQSOGRMRKsCCorKdEjUIuUg2PqfMQ2cY1uW+hs/KxUKvlM
oq2v3dEp+ULODkZDpKahPOR6COvNk9Vy3ihHCQG9mxKISEqzH3KU   -745156940   678947021
Sender: A11
Message: AlarmSystem: WARNING: 2013/08/02 1109370105     A11     Temp    25.4
        Celsius
L8: LogEntryType
4fJmgbU8G7aCCNErd5rWFJLYg8CDdR2lZpmMJEPwEtbrT+L5thSS+XFtItxCzvXNqCif9o7YiKNk
HAW2RXU4JbvyQNkOerLue1Y6rYqJDr12O+wE1xzGfgphAoCjIb+vSpDjy43Q+AvmdL7fAoV7KpWT
P70sSwtSAYYm9jE3aao=   -1856995104   1778848661
Sender: A11
Message: 2013/08/02 1109410119   A11     Temp    24.0    Celsius
L9: LogEntryType
Q1R2vWoIXG7N9aSxQdJ+hzZMZLcYoPoj2SEdAQoYBTv2b+M6LOWqWSgREBOKOuOZ+C41pO6Xo5lJ
+KPzaOM9+o5zG43M11GU8krWKzth8dlDfqSlLx6xUZF/ByfofnR4   -289471064   1402134062
Sender: A11
Message: 2013/08/02 1109450132   A11     Temp    23.6    Celsius
L10: LogEntryType
DEm++Oswr7c5wIrgExVb7K2NPrdQ3MaMQ6K4srA/TS5vaago2VZsbn0bgD6Mkd3c9r+7OF0CRc/r
6O2s9R7R9cTcMdumAy5FuMAPLQZJevWufQ8b4mhMeb7p9r0yCKdy   -830774920   1494439717
Sender: A11
Message: 2013/08/02 1109490142   A11     Temp    22.2    Celsius
```

```
L11: LogEntryType
NtlWKXcAHU2zo+GDc8vGcdl+Y3ilJ8mLtHkcVxDQlq8zE8R3fhDdzTd9ZE0uSgGZLJX1VbAgv4oO
iVlXsKR80NMQW73qb2z2PkJR1IlgODWgtdMncV2wvehempTleT2c   1041793453   -589096981
Sender: A11
Message: exit
L12: NormalCloseMessage
a/SO8PCchzIIPMOW7cW4FWuFePmMmIYkfBvV4eejDJHnCeIYYza1DH0mRqWnyNsi   862931347
-1702989854
```

Log Verification at Trusted Server

We mark with red color the problems that occurred in the verification and with blue color the duplicate entry.

```
Verification of : LogFile.txt
L[0]:
LogfileInitializationType
UGVr2k9Tuq/n2UNSsebnvUhKS5J9x0MiyrE72+AsJsEJSW5//E+P5o23MvPCzT7d1nphNmSRzR78
EIAKp2FZNbZ211HocZAKoJsYC+Dpf6pb0Bk/pyt6vaQhmjC2VrD6NXq8kIxLEzzTU+7xLcyWAUIY
r9aSxcKd0EkHtLeYEM04cX6dDRHFbr+MyIrm5iAqSoFZTa9dblMU/CgfSk8z2juDxqAepn9AhrwN
BbEckA8draKbMYrNvSf1TS7rP7dD4oxWatjiSPuvA1yvZ0vQQFq4NHmcNl60OoQu1eNgA4BNJct1
fwRHlxUof2CqhGcvKUaHIb2Jm3h+AnkJlBikvMJjAQ3DMVm+KqWt9HiPLzYPeKeZqNFa2n2X/fTp
EwXaV5p3fLB1NLG+IuljQ9Yc+hI/+2Vvldv3jZcWXoKrdpKO+h5zUsfkdAKbSej79qvd01ZO3BSh
/Qltbt1vZzy8QbC2iF4gmy4o8L2b2JWArH12ZtMR6Iqf2ZgbBiD8y6wW1IaohczeB4W87C6PEPVO
COtlYSgztDE5VMEIYEfC+dyTKmF/4WO34udRgJ2UmXiq -2131737990 -1788860721
Decrypted_AESmsg D[0]: LogfileInitializationType 2013/08/02 1108530188
2013/08/02 1130000000 0 0   4444
¶_!Ñ,_›ÚØQg¿ÁÀbØ³žz?_‹Œ}ž+_«Á_LSà›Ì_âãa„¥ë°_˜«ø_ŸÙ#Ë¸_"wÚÕ¤ºó_ëÛ»½¥ÇŸ_¸~êÊNeLM
n°Ë¢A_,‡B–Ôž,8éì’Ž«__B¡Q}_J
Œhe_éÒÜŠ?ÚA1révo_¿Ln^_
2pbMnzc7vTq9kGz2sWxNDqxY+kEQNjzMMKCAoqmWnHDn8wWSS0HwUPnQYsyToH7uIVjzhioQ8DWz
GEF58AZnhbA9X8c86ScunClLGKo8+tmXfHMMonnYME6CMDQYaAVq+RDMzo7BqWNweOPLuQjf+yGl
i3JWvo39Q/Lb+4RNiV+AvOTmA8Bnsjnyv/OYvnNCNCgBkjyt7fBTs/pt4W0sHiyKy1LUQJmDIYOn
q4+2tbA=
M[0]: LogfileInitializationType 1130000000 0 0   4444
¶_!Ñ,_›ÚØQg¿ÁÀbØ³žz?_‹Œ}ž+_«Á_LSà›Ì_âãa„¥ë°_˜«ø_ŸÙ#Ë¸_"wÚÕ¤ºó_ëÛ»½¥ÇŸ_¸~êÊNeLM
n°Ë¢A_,‡B–Ôž,8éì’Ž«__B¡Q}_J
Œhe_éÒÜŠ?ÚA1révo_¿Ln^_
2pbMnzc7vTq9kGz2sWxNDqxY+kEQNjzMMKCAoqmWnHDn8wWSS0HwUPnQYsyToH7uIVjzhioQ8DWz
GEF58AZnhbA9X8c86ScunClLGKo8+tmXfHMMonnYME6CMDQYaAVq+RDMzo7BqWNweOPLuQjf+yGl
i3JWvo39Q/Lb+4RNiV+AvOTmA8Bnsjnyv/OYvnNCNCgBkjyt7fBTs/pt4W0sHiyKy1LUQJmDIYOn
q4+2tbA=
Yi: -2131737990
Yi2: -2131737990
Y[0] = hash(Yi-1 ,AES(Di)_Ki, Wi) is verified
mac: -1788860721
mac2: -1788860721
MAC[0] = MAC(Yi)_Ai  is verified

L[1]:
ResponseMessageType
gUywYu3ZjPTvlsYDsJIu5W+R3wG2pmazNmYXsCSGQTHL83HdStEqSiMb+G1exHsI36Y+TM8jHzwz
r9qBzKyVMRhZqjSfqay9XnfHEwrZBKUGTM2Q58lwczLzzB8MaEJhUYYlf7T7AXsR1a54oPJ6dcwB
zLecKhAhlj6vGn5P76RhYDsvVh5PxfwFNn6VzfylA2q3Fh+A/VO1ucBlnYR7qedj0Ct/j0teUQ9u
bA8nI8+zfM9jdYpCt95isTJod7suTn8Joz/THwsZNs9amfe4LStQdnjOGoQt1UFs5FCiB6jpKqcS
Qha8rsYkeWmsfDiusAFayvL99zUFpD8qfTlSqbWyhzoOZpPobGuF9DZgy0TIw4KdJoYiMepJgz14
9tHcekYi3V3H+Vmv42C1MCU7TcivMCVEEVijEna8it/pVOIDCHgiX/ZemkYEjGiMgAIx+9U6zhyr
QxB7vVWWQLseq2CFMBRu9d1qKbtiWJuLTtiB8GX/WrAktt5DVSRtmq1z3QZdc+Qwgbt9uKucjpBy
WUByX+UaHdXzGXR2mz7pHuM= -673921588 -1097811294
```

```
Decrypted_AESmsg D[1]: ResponseMessageType 2013/08/02 1108530204 2013/08/02
1130000000 1 1   1234   _Sá€•Ø”xœ)ˆ0_‡_¡_’È«Q__–KA_+(LßÁ_`HÞ{Q*Už¡÷J–
‰&D“Ê:Æ÷_?=Š3ß!œ`yY_9ÄÐ®^_?él2Ï?_öÀ_Pñ4ÇfgËZ«Z\¤¹˜Ìž‰€cH?T_¬ršs5
        _Eú¨~x+Õµœ wRf¥,ÆZ
4wNQ+odd7avRi2ygBqqDvDoePKPvF+3gT7ixN5GqIwTflljr0veCnG6voaTEZF5KlDJ/q9fUUMCs
n546cMo7IXZmp/T51X6rLhnnOhjsoz3dGKdEPaQpcFVUVufeN2HeN/C28Mzx2JEnfzym3U8Y4Ok8
AsYTDjdLxP1AqB3nMfGXWLH7Gqzf8T4/ueAs1xqjuT/14f9Pp/vgHAk4gET4AQ==
M[1]: ResponseMessageType 1130000000 1 1   1234   _Sá€•Ø”xœ)ˆ0_‡_¡_’È«Q__–
KA_+(LßÁ_`HÞ{Q*Už¡÷J–
‰&D“Ê:Æ÷_?=Š3ß!œ`yY_9ÄÐ®^_?él2Ï?_öÀ_Pñ4ÇfgËZ«Z\¤¹˜Ìž‰€cH?T_¬ršs5
        _Eú¨~x+Õµœ wRf¥,ÆZ
4wNQ+odd7avRi2ygBqqDvDoePKPvF+3gT7ixN5GqIwTflljr0veCnG6voaTEZF5KlDJ/q9fUUMCs
n546cMo7IXZmp/T51X6rLhnnOhjsoz3dGKdEPaQpcFVUVufeN2HeN/C28Mzx2JEnfzym3U8Y4Ok8
AsYTDjdLxP1AqB3nMfGXWLH7Gqzf8T4/ueAs1xqjuT/14f9Pp/vgHAk4gET4AQ==
Yi: -673921588
Yi2: -673921588
Y[1] = hash(Yi-1 ,AES(Di)_Ki, Wi) is verified
mac: -1097811294
mac2: -1097811294
MAC[1] = MAC(Yi)_Ai  is verified

L[2]:
LogEntryType
G693AlsnQhBh/BVYr7pM95LK68SXCqRjgWQ7vDozihQ8goOLKU1zw0mUETHHUoaCPveptdw6oYfD
Ri7ZP14YWRD94+2GRZGugD0A2lvrYelXpIPG4DRtuFVoh2x/pB5F -1798661150 19329142
Decrypted_AESmsg D[2]: LogEntryType 2013/08/02 1109170052 2013/08/02
1130000000 2 2013/08/02 1109170037       A11     Temp    20.0    Celsius
M[2]: LogEntryType 1130000000 2 2013/08/02 1109170037    A11     Temp    20.0
        Celsius
Yi: -1798661150
Yi2: -1798661150
Y[2] = hash(Yi-1 ,AES(Di)_Ki, Wi) is verified
mac: 19329142
mac2: 19329142
MAC[2] = MAC(Yi)_Ai  is verified

L[3]:
LogEntryType
IJ6z0QgONqWemDWSqObwtvZ/DB0D6iFGVf3imoIp6i7AONa8IuJzP1o5wkJEBNgtsXPpmrVjSw+2
AQT6nC09CuMwezCy15yafdzfX+IoY+5XbsT96EKbLxqDqnMcloWb -834139319 1263724178
Decrypted_AESmsg D[3]: LogEntryType 2013/08/02 1109210067 2013/08/02
1130000000 3 2013/08/02 1109210067       A11     Temp    20.4    Celsius
M[3]: LogEntryType 1130000000 3 2013/08/02 1109210067    A11     Temp    20.4
        Celsius
Yi: -834139319
Yi2: -834139319
Y[3] = hash(Yi-1 ,AES(Di)_Ki, Wi) is verified
mac: 1263724178
mac2: 1263724178
MAC[3] = MAC(Yi)_Ai  is verified

L[4]:
LogEntryType
l4rAv3YGIDH6wpmf4+sN8RIcxdR3KxXJeW/z6b/P0ItlsyQK/fdOfB822wPfzCyiPkWK4C6bs+UR
rxX0bcvKxGkNJOusJyijphpxSm6V8SEV+PF+3ndwfACX/nO052Zx -2096999132 -2078260828
Decrypted_AESmsg D[4]: LogEntryType 2013/08/02 1109250079 2013/08/02
1130000000 4 2013/08/02 1109250079       A11     Temp    21.0    Celsius
M[4]: LogEntryType 1130000000 4 2013/08/02 1109250079    A11     Temp    21.0
        Celsius
Yi: -2096999132
```

```
Yi2: -2096999132
Y[4] = hash(Yi-1 ,AES(Di)_Ki, Wi) is verified
mac: -2078260828
mac2: -2078260828
MAC[4] = MAC(Yi)_Ai  is verified

L[5]:
LogEntryType
kDUiQUlJAeGH1v9dC/UnIG3dK4GBVEztaRDdTfxTJKhkfWi+uOlPeUUOtAryiXpbVpJZrh1K1y9S
zS5493giqthlO0Q2QefU36aGxNwK+gP5OBopCiOrdPk7h9SfuZqT 989650785 1385486530
Decrypted_AESmsg D[5]: LogEntryType 2013/08/02 1109290090 2013/08/02
1130000000 5 2013/08/02 1109290090       A11     Temp    22.0    Celsius
M[5]: LogEntryType 1130000000 5 2013/08/02 1109290090    A11     Temp    22.0
        Celsius
Yi: 989650785
Yi2: 989650785
Y[5] = hash(Yi-1 ,AES(Di)_Ki, Wi) is verified
mac: 1385486530
mac2: 1385486530
MAC[5] = MAC(Yi)_Ai  is verified

L[6]:
LogEntryType
5DcZft5wiQt8W90pC2vWETCDg0/kjIefs1qwM4El+r1xYVz3Qyk1IcztLyEveyNHTL3QoRxFsadF
E02w9MGTRAP9oiPc9uHksv6KbB4sf8w2sSLPk/eTePAME3RW4n0/ -816886799 2080965872
Decrypted_AESmsg D[6]: LogEntryType 2013/08/02 1109330092 2013/08/02
1130000000 6 2013/08/02 1109330092       A11     Temp    24.0    Celsius
M[6]: LogEntryType 1130000000 6 2013/08/02 1109330092    A11     Temp    24.0
        Celsius
Yi: -816886799
Yi2: -816886799
Y[6] = hash(Yi-1 ,AES(Di)_Ki, Wi) is verified
mac: 2080965872
mac2: 2080965872
MAC[6] = MAC(Yi)_Ai  is verified

L[7]:
LogEntryType
Fc7bF6KLQqSAg5llsGDcn7OWLJ8A5+cQSOGRMRKsCCorKdEjUIuUg2PqfMQ2cY1uW+hs/KxUKvlM
oq2v3dEp+ULODkZDpKahPOR6COvNk9Vy3ihHCQG9mxKISEqzH3KU -745156940 678947021
Decrypted_AESmsg D[7]: LogEntryType 2013/08/02 1109370105 2013/08/02
1130000000 7 2013/08/02 1109370105       A11     Temp    25.4    Celsius
M[7]: LogEntryType 1130000000 7 2013/08/02 1109370105    A11     Temp    25.4
        Celsius
Yi: -745156940
Yi2: -745156940
Y[7] = hash(Yi-1 ,AES(Di)_Ki, Wi) is verified
mac: 678947021
mac2: 678947021
MAC[7] = MAC(Yi)_Ai  is verified

L[8]:
LogEntryType
4fJmgbU8G7aCCNErd5rWFJLYg8CDdR2lZpmMJEPwEtbrT+L5thSS+XFtItxCzvXNqCif9o7YiKNk
HAW2RXU4JbvyQNkOerLue1Y6rYqJDr12O+wE1xzGfgphAoCjIb+vSpDjy43Q+AvmdL7fAoV7KpWT
P70sSwtSAYYm9jE3aao= -1856995104 1778848661
Decrypted_AESmsg D[8]: LogEntryType 2013/08/02 1109370153 2013/08/02
1130000000 8 AlarmSystem: WARNING: 2013/08/02 1109370105 A11     Temp    25.4
        Celsius
```

```
M[8]: LogEntryType 1130000000 8 AlarmSystem: WARNING: 2013/08/02 1109370105
        A11     Temp    25.4    Celsius
Yi: -1856995104
Yi2: -1856995104
Y[8] = hash(Yi-1 ,AES(Di)_Ki, Wi) is verified
mac: 1778848661
mac2: 1778848661
MAC[8] = MAC(Yi)_Ai  is verified

L[9]:
LogEntryType
Q1R2vWoIXG7N9aSxQdJ+hzZMZLcYoPoj2SEdAQoYBTv2b+M6LOWqWSgREBOKOuOZ+C41pO6Xo5lJ
+KPzaOM9+o5zG43M11GU8krWKzth8dlDfqSlLx6xUZF/ByfofnR4 -289471064 1402134062
Decrypted_AESmsg D[9]: LogEntryType 2013/08/02 1109410122 2013/08/02
1130000000 9 2013/08/02 1109410119       A11     Temp    24.0    Celsius
M[9]: LogEntryType 1130000000 9 2013/08/02 1109410119    A11     Temp    24.0
        Celsius
Yi: -289471064
Yi2: -289471064
Y[9] = hash(Yi-1 ,AES(Di)_Ki, Wi) is verified
mac: 1402134062
mac2: 1402134062
MAC[9] = MAC(Yi)_Ai  is verified

L[10]:
LogEntryType
DEm++Oswr7c5wIrgExVb7K2NPrdQ3MaMQ6K4srA/TS5vaago2VZsbn0bgD6Mkd3c9r+7OF0CRc/r
6O2s9R7R9cTcMdumAy5FuMAPLQZJevWufQ8b4mhMeb7p9r0yCKdy -830774920 1494439717
LogEntryType
Decrypted_AESmsg D[10]: LogEntryType 2013/08/02 1109450132 2013/08/02
1130000000 10 2013/08/02 1109450132      A11     Temp    23.6    Celsius
M[10]: LogEntryType 1130000000 10 2013/08/02 1109450132  A11     Temp    23.6
        Celsius
Yi: -830774920
Yi2: -830774920
Y[10] = hash(Yi-1 ,AES(Di)_Ki, Wi) is verified
mac: 1494439717
Number Format Exception! The reason is that someone tampered with entry[10].

L[11]:
LogEntryType
NtlWKXcAHU2zo+GDc8vGcdl+Y3ilJ8mLtHkcVxDQlq8zE8R3fhDdzTd9ZE0uSgGZLJX1VbAgv4oO
iVlXsKR80NMQW73qb2z2PkJR1IlgODWgtdMncV2wvehempTleT2c 1041793453 -589096981
Decrypted_AESmsg D[11]: LogEntryType 2013/08/02 1109490142 2013/08/02
1130000000 11 2013/08/02 1109490142      A11     Temp    22.2    Celsius
M[11]: LogEntryType 1130000000 11 2013/08/02 1109490142  A11     Temp    22.2
        Celsius
Yi: 1041793453
Yi2: 1041793453
Y[11] = hash(Yi-1 ,AES(Di)_Ki, Wi) is verified
mac: -589096981
mac2: -589096981
MAC[11] = MAC(Yi)_Ai  is verified

L[12]:
NormalCloseMessage
a/SO8PCchzIIPMOW7cW4FWuFePmMmIYkfBvV4eejDJHnCeIYYza1DH0mRqWnyNsi 862931347 -
1702989854
Decrypted_AESmsg D[12]: NormalCloseMessage 2013/08/02 1109530154 2013/08/02
1130000000 12
```

```
M[12]: NormalCloseMessage 1130000000 12
Yi: 862931347
Yi2: 862931347
Y[12] = hash(Yi-1 ,AES(Di)_Ki, Wi) is verified
mac: -1702989854
mac2: -1702989854
MAC[12] = MAC(Yi)_Ai  is verified

In file LogFile.txt the MACs in following entry positions were found false
10
```

Signature Verification at RA

```
Singature1 entry[0] is verified
Manual Signatures verification: true
Singature2 entry[0] is verified
Singature2 entry[1] is verified
Singature2 entry[2] is verified
Singature2 entry[3] is verified
Singature2 entry[4] is verified
Singature2 entry[5] is verified
Singature2 entry[6] is verified
Singature2 entry[7] is verified
Singature2 entry[11] is verified
Singature2_random entry[12] is verified
Singature2_random entry[13] is verified
Singature2_random entry[14] is verified
Singature2_random entry[15] is verified
Singature2_random entry[16] is verified
Singature2_random entry[23] is verified
Singature2_random entry[24] is verified
Singature2_random entry[25] is verified
```

Replacement of an entire entry

Server

```
L0: LogfileInitializationType
h7MemoeQfSqnOXQsd1GItR6s84Bk2/3S91VCHn2Y1qG3NDa9qL1MpvMCkdOzLtQ9wWI3cWP80bMs
JrNOAWshdrSqcAnC/VjYhSdRfNYjs3YyaBH/QcZeSrJ2ewC/5Scx30aWK1448k53XbkfdkcOebWf
bOcBSjG5v6igWlSI8c4P4AaTUVtsO8IzftG12UkdZAOgGp4h5/40DSDZAbRX0NDL/yrrzWyw2F3S
2h48wNyhJWs0CysfSs8AhT8FdXmvDwCDe2I/cRRXd0xB2OEmJ3WyQu3k49xFUp80JI0hnpCItrEx
+fh+mOCuPpMG+K5J7f6zYhtAlEXXQCSOdoQkP0fNvlGoAO+Hn6ZzIrGbEpc5XhonmmQwnIOYz7L2
uRZ/2VHsMi/c5ariOyLVjsqhPIARBfQvNEuOZfZe8BVL7uevZFPNQcLLn/iYF9jh3aM1NxQnxLrR
VttGlzq24GC9mvzSwOJ6rkG1gcmTf0dYixHrGvYdMMx4f98TH3W+yiQfIW3s8VySdnwxu0DAru64
uIxWyBUxMcScuw0f9PZlZ2y2cYja2FBmt72PlF09/Q49   -1685051933   -1917753776
L1: ResponseMessageType
2bTduCdyxfQUcmOa61aOz2QbpmQa5aq2RXXVtXUp3bxqfIrwtp5ZDFt01QfKP67AQ+hp7clcuz/5
5GXHe64oJDEq7sfUKRqRQsj7MPERsM3dMou9YSMUnL+5hk/Gx+3llW4jewmwXk+bWuhqBA/s7deq
NruakPgOFCNI4BDJ9SQqonlrqZnDlpHsUAw7yqHA5sQA5VssczUYyq9tFyGpCBCu5IVhD+hOh6cn
qqnsxKO7X1Toa6Fn91i+JwR7NzjGHph1C+p3B5Er+fCv8frypY3748Pm4DtvDsYXqhOHIDnT3gwU
mz9JgYqzP46O98sDyIdIaW20n8Vop7YDnSowPSKX7qalVM2tWC7OE6aC1a16P+3ASis7XdGkDc+Y
d1qDiQXrjfzuTpm18XPDSrnUlULKri7YzTvzJi3t1pNdLPALt16gKSQXYZ9Hewf2GPWc5VbYnLs6
0vTjmTwzy5q8OwVJ2Aem0Q0vLCsRckp/P8bVuIEapzoYJSMx0xtMZSNzWj9SDbSXopEn9YNVeOcI
+GWBWAniLa91t3+Npj4i52U=   -1230973750   -1595473668
Sender: A11
Message: 2013/08/02 1737480301   A11     Temp    20.0    Celsius
L2: LogEntryType
NXSYGdXlAhV7KTgp2Hk9KDXzWM5/Q6Aihe4NCxY97qHnBIVnlO7PKOJx3V2yMsRpOxCDv9LRVkBd
kCfcB+T6HMvr8U/kzN79kA2YPFngV9RfqmrPup2lFdRBqJidr3Kr   276872671   1950180975
```

```
Sender: A11
Message: 2013/08/02 1737520313   A11     Temp    20.4    Celsius
L3: LogEntryType
7PrnrJ0KGffrFA8RSvDY4cn5h6DYNhBKJMI8zoT+/1fPaIlq4CaFAimErmVde0zEQcs8cUN2rqYT
WuBzCkVX8neTS8H8ubX/OLFYEcfjz05Bn3scQ112v8mRNxBhyf6a   1390648148   -339556863
Sender: A11
Message: 2013/08/02 1737560314   A11     Temp    21.0    Celsius
L4: LogEntryType
C4Bq6+bzLtvAwm3qlXbm+/gL/Mapq15pECcVJEUEF9+JMmI5AHk/DyU2AqRthzWl9Bm/aCAgL14s
gSwzAfjoAt9XLhtAWuwJr7e0nzap9p5hfBf4fIWE5H2P8UFr/B87   -2090064901
1426447832
Sender: A11
Message: 2013/08/02 1738000316   A11     Temp    22.0    Celsius
L5: LogEntryType
jYGod8gAZwxu6bbHGaS59uD/xyVO6ysuIuV/LPqf9F975vaPB08mgyxJgzUNhP4St0VHkSSynnH9
oSpugqjM2how/mItuiNx3BGQN+nznrJG7f2z0duQP0DY8Qram9xG   -1550202999   122174429
Sender: A11
Message: 2013/08/02 1738040318   A11     Temp    24.0    Celsius
L6: LogEntryType
dxBrtRZfC6NzCEtJoTUpbjKnZsp6GMDyRBxnBK3vsDavDeYDddUTMM7ARHxKXCvus5f8nBGhYW9P
2lv1SsRMpkk/SS2RrimtZBKo5oVdYNBvXnmRd+hVwY6I+ylQilAQ   1745297055   2052256232
Sender: A11
Message: 2013/08/02 1738080319   A11     Temp    25.4    Celsius
L7: LogEntryType
rK2d3vFuXhxJLnieUQDH2zisNuwgiPq88gHzrbsNXyr59qi4mNbdECu04bK1rd9PLhQr7EQyuLAy
fl3wdcrUwLXWZMRRQkRWlBhA/V3Nx0C+CPWNDNJeCBA0wtwd1X7J   77055538   1221522561
Sender: A11
Message: AlarmSystem: WARNING: 2013/08/02 1738080319     A11     Temp    25.4
        Celsius
L8: LogEntryType
luDIaX93Ax0Lmj6Y9j9ws77zpx01RVFmmvz+ec/f/V+oMmOZai+ISS/3SuBV5y+MEOvXouctifPR
jxjGyWxVGkLsDP/p63tt99dowjfpxBFZSMeeG6UPDBfzKpRap++7xi+lvO/g3/aip95Qve2t23UM
siKE9dXWZkhX4btZ5Nw=   158211288   -1975095810
Sender: A11
Message: 2013/08/02 1738120320   A11     Temp    24.0    Celsius
L9: LogEntryType
p7aK087UHOWuu4sUMSfB9sKqLPT5ph+DJxQdAADVtBQSIVBKrvrpgqY0by5eqkZR6gFM+5dljGCG
/TqHPMAP7BSAMZzti9Ba+Ks3mpo9IHRvGkIXWUUaiDZW10q1a7f6   147449289   1786230764
Sender: A11
Message: 2013/08/02 1738160322   A11     Temp    23.6    Celsius
L10: LogEntryType
4YMw9cd6w6Deu5GixS3BWN11WB02tkPVeAll3jOlS+xUwlGpp12A/EvxH18RwiBvDKZMO8QMDVVX
YvWDZXMSXIb8Hd6KfvWIUNkcOnx3QSiha3ZbzlLFnTE65X6h8x+v   1461733865   -725012515
Sender: A11
Message: 2013/08/02 1738200323   A11     Temp    22.2    Celsius
L11: LogEntryType
5RNIPpFlp7C8qnEHvYdSKPUBPxHFpoGxI8P3MnmPTjgGESk0Ao8C9uzdnbKXH9gds58Hc8FhGwsW
WBWjjEWdeeCFF+8xOKzbSt5CTujjemeSEu5rZdxAT3dbvWGisFfF   1150163232   -
1863768431
Sender: A11
Message: exit
L12: NormalCloseMessage
WWa1XwKhl0FfX8zQPC84mHvPjCBn/hjJcic5yQd1XHG05a4HN0EV6hZVRJ39xgAo   -1294127047
1542216267
```

Log Verification at TS

The attack performed on the log file was replacing the log entry L9 with another one. The new entry is marked with blue.

```
Verification of : LogFile.txt
L[0]:
LogfileInitializationType
h7MemoeQfSqnOXQsd1GItR6s84Bk2/3S91VCHn2Y1qG3NDa9qL1MpvMCkdOzLtQ9wWI3cWP80bMs
JrNOAWshdrSqcAnC/VjYhSdRfNYjs3YyaBH/QcZeSrJ2ewC/5Scx30aWK1448k53XbkfdkcOebWf
bOcBSjG5v6igWlSI8c4P4AaTUVtsO8IzftG12UkdZAOgGp4h5/40DSDZAbRX0NDL/yrrzWyw2F3S
2h48wNyhJWs0CysfSs8AhT8FdXmvDwCDe2I/cRRXd0xB2OEmJ3WyQu3k49xFUp80JI0hnpCItrEx
+fh+mOCuPpMG+K5J7f6zYhtAlEXXQCSOdoQkP0fNvlGoAO+Hn6ZzIrGbEpc5XhonmmQwnIOYz7L2
uRZ/2VHsMi/c5ariOyLVjsqhPIARBfQvNEuOZfZe8BVL7uevZFPNQcLLn/iYF9jh3aM1NxQnxLrR
VttGlzq24GC9mvzSwOJ6rkG1gcmTf0dYixHrGvYdMMx4f98TH3W+yiQfIW3s8VySdnwxu0DARu64
uIxWyBUxMcScuw0f9PZlZ2y2cYja2FBmt72P1F09/Q49 -1685051933 -1917753776
Decrypted_AESmsg D[0]: LogfileInitializationType 2013/08/02 1737240829
2013/08/02 1759000000 0 0 4444 •³º®_í_ð4/oºvT
÷V?ô?EŸÑýÖñS¡ÃÙc8_Pº_3•_ø%Ë´0È-e,7Þ,0v_w_O!Î™í ª *Á_(ü7Žl2(þ?U:óÑ)Æ_=ÔK?PZÝƒ
ü«ç"f^L_gAFÖ|@[±.0¢ZÎ~ÿ_X–Q„‡L_[y/
yRljp32M2qx5ukdawoi4nnLywhPmnFnhGOgJXMshhB/2swzTcZ5T6TaRen7wUkezh1f5RvGRudAn
+2ygGGRd2zmcNI0vw07SXAMuR5w2Ye1k6hzkNNcbxU3UIrXA1mQEEcLP+oDzZg9ciN89ch5/bGAP
JHyotKqPsWAn1qfppEkBUpXkaLGRp94oHNjAD8LfTGyEstWQDvgrSREy120mFYaQ6FxLYxOaSqth
dH9y8cw=
M[0]: LogfileInitializationType 1759000000 0 0 4444 •³º®_í_ð4/oºvT
÷V?ô?EŸÑýÖñS¡ÃÙc8_Pº_3•_ø%Ë´0È-e,7Þ,0v_w_O!Î™í ª *Á_(ü7Žl2(þ?U:óÑ)Æ_=ÔK?PZÝƒ
ü«ç"f^L_gAFÖ|@[±.0¢ZÎ~ÿ_X–Q„‡L_[y/
yRljp32M2qx5ukdawoi4nnLywhPmnFnhGOgJXMshhB/2swzTcZ5T6TaRen7wUkezh1f5RvGRudAn
+2ygGGRd2zmcNI0vw07SXAMuR5w2Ye1k6hzkNNcbxU3UIrXA1mQEEcLP+oDzZg9ciN89ch5/bGAP
JHyotKqPsWAn1qfppEkBUpXkaLGRp94oHNjAD8LfTGyEstWQDvgrSREy120mFYaQ6FxLYxOaSqth
dH9y8cw=
Yi: -1685051933
Yi2: -1685051933
Y[0] = hash(Yi-1 ,AES(Di)_Ki, Wi) is verified
mac: -1917753776
mac2: -1917753776
MAC[0] = MAC(Yi)_Ai is verified

L[1]:
ResponseMessageType
2bTduCdyxfQUcmOa61aOz2QbpmQa5aq2RXXVtXUp3bxqfIrwtp5ZDFt01QfKP67AQ+hp7clcuz/5
5GXHe64oJDEq7sfUKRqRQsj7MPERsM3dMou9YSMUnL+5hk/Gx+3llW4jewmwXk+bWuhqBA/s7deq
NruakPgOFCNI4BDJ9SQqonlrqZnDlpHsUAw7yqHA5sQA5VssczUYyq9tFyGpCBCu5IVhD+hOh6cn
qqnsxKO7X1Toa6Fn91i+JwR7NzjGHph1C+p3B5Er+fCv8frypY3748Pm4DtvDsYXqhOHIDnT3gwU
mz9JgYqzP46O98sDyIdIaW20n8Vop7YDnSowPSKX7qalVM2tWC7OE6aC1a16P+3ASis7XdGkDc+Y
d1qDiQXrjfzuTpm18XPDSrnUlULKri7YzTvzJi3t1pNdLPALt16gKSQXYZ9Hewf2GPWc5VbYnLs6
0vTjmTwzy5q8OwVJ2Aem0Q0vLCsRckp/P8bVuIEapzoYJSMx0xtMZSNzWj9SDbSXopEn9YNVeOcI
+GWBWAniLa91t3+Npj4i52U= -1230973750 -1595473668
Decrypted_AESmsg D[1]: ResponseMessageType 2013/08/02 1737240856 2013/08/02
1759000000 1 1 1234 ¤Y¢_CŽÃÕš__õä>kê†Ê8Oã,ÑÃì(
óæ¿¿å?ø|N?Jmî+’_¶[_´j¬_«©ÝíBJOk£±_Œ_yn~zÓ_nËòŠºÑæû0²¤*ô~½ý-r”¿[á
_öÆ__DŠ½¸‰H^~O ö]_öÞVïçÃÜv¥S#÷@ 8S.n
ty9vTk3LcbCf33KJVntfnj7xrVAo6UyEULWxIvhBHr9YNlKt4/Kp7gWVxjUiwjPiSplF3kKfkPb0
9hlSV8F5tOlXineC0frmFZLRfgfW11H8uCGDwTcdG5vQjzG6fuBQsh/4k3gqqht1vRxxPr+3ldNx
zEBA3PvSbWyQC1aqPfiRVVwpcSOFAuxItEigJOrnpPXBORdL79ZujTjvtkpptw==
M[1]: ResponseMessageType 1759000000 1 1 1234 ¤Y¢_CŽÃÕš__õä>kê†Ê8Oã,ÑÃì(
óæ¿¿å?ø|N?Jmî+’_¶[_´j¬_«©ÝíBJOk£±_Œ_yn~zÓ_nËòŠºÑæû0²¤*ô~½ý-r”¿[á
_öÆ__DŠ½¸‰H^~O ö]_öÞVïçÃÜv¥S#÷@ 8S.n
ty9vTk3LcbCf33KJVntfnj7xrVAo6UyEULWxIvhBHr9YNlKt4/Kp7gWVxjUiwjPiSplF3kKfkPb0
9hlSV8F5tOlXineC0frmFZLRfgfW11H8uCGDwTcdG5vQjzG6fuBQsh/4k3gqqht1vRxxPr+3ldNx
zEBA3PvSbWyQC1aqPfiRVVwpcSOFAuxItEigJOrnpPXBORdL79ZujTjvtkpptw==
Yi: -1230973750
Yi2: -1230973750
Y[1] = hash(Yi-1 ,AES(Di)_Ki, Wi) is verified
mac: -1595473668
```

```
mac2: -1595473668
MAC[1] = MAC(Yi)_Ai  is verified

L[2]:
LogEntryType
NXSYGdXlAhV7KTgp2Hk9KDXzWM5/Q6Aihe4NCxY97qHnBIVnlO7PKOJx3V2yMsRpOxCDv9LRVkBd
kCfcB+T6HMvr8U/kzN79kA2YPFngV9RfqmrPup2lFdRBqJidr3Kr 276872671 1950180975
Decrypted_AESmsg   D[2]:   LogEntryType   2013/08/02   1737480327   2013/08/02
1759000000 2 2013/08/02 1737480301       A11     Temp    20.0    Celsius
M[2]: LogEntryType 1759000000 2 2013/08/02 1737480301    A11     Temp    20.0
        Celsius
Yi: 276872671
Yi2: 276872671
Y[2] = hash(Yi-1 ,AES(Di)_Ki, Wi) is verified
mac: 1950180975
mac2: 1950180975
MAC[2] = MAC(Yi)_Ai  is verified

L[3]:
LogEntryType
7PrnrJ0KGffrFA8RSvDY4cn5h6DYNhBKJMI8zoT+/1fPaIlq4CaFAimErmVde0zEQcs8cUN2rqYT
WuBzCkVX8neTS8H8ubX/OLFYEcfjz05Bn3scQ112v8mRNxBhyf6a 1390648148 -339556863
Decrypted_AESmsg   D[3]:   LogEntryType   2013/08/02   1737520317   2013/08/02
1759000000 3 2013/08/02 1737520313       A11     Temp    20.4    Celsius
M[3]: LogEntryType 1759000000 3 2013/08/02 1737520313    A11     Temp    20.4
        Celsius
Yi: 1390648148
Yi2: 1390648148
Y[3] = hash(Yi-1 ,AES(Di)_Ki, Wi) is verified
mac: -339556863
mac2: -339556863
MAC[3] = MAC(Yi)_Ai  is verified

L[4]:
LogEntryType
C4Bq6+bzLtvAwm3qlXbm+/gL/Mapq15pECcVJEUEF9+JMmI5AHk/DyU2AqRthzWl9Bm/aCAgL14s
gSwzAfjoAt9XLhtAWuwJr7e0nzap9p5hfBf4fIWE5H2P8UFr/B87 -2090064901 1426447832
Decrypted_AESmsg   D[4]:   LogEntryType   2013/08/02   1737560318   2013/08/02
1759000000 4 2013/08/02 1737560314       A11     Temp    21.0    Celsius
M[4]: LogEntryType 1759000000 4 2013/08/02 1737560314    A11     Temp    21.0
        Celsius
Yi: -2090064901
Yi2: -2090064901
Y[4] = hash(Yi-1 ,AES(Di)_Ki, Wi) is verified
mac: 1426447832
mac2: 1426447832
MAC[4] = MAC(Yi)_Ai  is verified

L[5]:
LogEntryType
jYGod8gAZwxu6bbHGaS59uD/xyVO6ysuIuV/LPqf9F975vaPB08mgyxJgzUNhP4St0VHkSSynnH9
oSpugqjM2how/mItuiNx3BGQN+nznrJG7f2z0duQP0DY8Qram9xG -1550202999 122174429
Decrypted_AESmsg   D[5]:   LogEntryType   2013/08/02   1738000328   2013/08/02
1759000000 5 2013/08/02 1738000316       A11     Temp    22.0    Celsius
M[5]: LogEntryType 1759000000 5 2013/08/02 1738000316    A11     Temp    22.0
        Celsius
Yi: -1550202999
Yi2: -1550202999
Y[5] = hash(Yi-1 ,AES(Di)_Ki, Wi) is verified
mac: 122174429
```

```
mac2: 122174429
MAC[5] = MAC(Yi)_Ai  is verified

L[6]:
LogEntryType
dxBrtRZfC6NzCEtJoTUpbjKnZsp6GMDyRBxnBK3vsDavDeYDddUTMM7ARHxKXCvus5f8nBGhYW9P
2lv1SsRMpkk/SS2RrimtZBKo5oVdYNBvXnmRd+hVwY6I+ylQilAQ 1745297055 2052256232
Decrypted_AESmsg   D[6]:   LogEntryType   2013/08/02   1738040322   2013/08/02
1759000000 6 2013/08/02 1738040318       A11     Temp    24.0    Celsius
M[6]: LogEntryType 1759000000 6 2013/08/02 1738040318    A11     Temp    24.0
        Celsius
Yi: 1745297055
Yi2: 1745297055
Y[6] = hash(Yi-1 ,AES(Di)_Ki, Wi) is verified
mac: 2052256232
mac2: 2052256232
MAC[6] = MAC(Yi)_Ai  is verified

L[7]:
LogEntryType
rK2d3vFuXhxJLnieUQDH2zisNuwgiPq88gHzrbsNXyr59qi4mNbdECu04bK1rd9PLhQr7EQyuLAy
fl3wdcrUwLXWZMRRQkRWlBhA/V3Nx0C+CPWNDNJeCBA0wtwd1X7J 77055538 1221522561
Decrypted_AESmsg   D[7]:   LogEntryType   2013/08/02   1738080323   2013/08/02
1759000000 7 2013/08/02 1738080319       A11     Temp    25.4    Celsius
M[7]: LogEntryType 1759000000 7 2013/08/02 1738080319    A11     Temp    25.4
        Celsius
Yi: 77055538
Yi2: 77055538
Y[7] = hash(Yi-1 ,AES(Di)_Ki, Wi) is verified
mac: 1221522561
mac2: 1221522561
MAC[7] = MAC(Yi)_Ai  is verified

L[8]:
LogEntryType
luDIaX93Ax0Lmj6Y9j9ws77zpx01RVFmmvz+ec/f/V+oMmOZai+ISS/3SuBV5y+MEOvXouctifPR
jxjGyWxVGkLsDP/p63tt99dowjfpxBFZSMeeG6UPDBfzKpRap++7xi+lvO/g3/aip95Qve2t23UM
siKE9dXWZkhX4btZ5Nw= 158211288 -1975095810
Decrypted_AESmsg   D[8]:   LogEntryType   2013/08/02   1738080365   2013/08/02
1759000000 8 AlarmSystem: WARNING: 2013/08/02 1738080319 A11     Temp    25.4
        Celsius
M[8]: LogEntryType 1759000000 8 AlarmSystem: WARNING: 2013/08/02 1738080319
        A11     Temp    25.4    Celsius
Yi: 158211288
Yi2: 158211288
Y[8] = hash(Yi-1 ,AES(Di)_Ki, Wi) is verified
mac: -1975095810
mac2: -1975095810
MAC[8] = MAC(Yi)_Ai  is verified

L[9]:
LogEntryType
BJx1O1v2YQA8lNi32hMO5VHJo4aYi+uE7fLU8N0DQO90abol5YP5HZSpkNGOn3m3DCLadAjtwWrs
FmTiWTvZ5Tldz+yepHDWe1gOR/pJDEMyIZF2cgQwhQnwxEQKysRP -475431498 -456700380
AES message cannot be decrypted! Someone changed log entry [9].
Yi: 2082583920
Yi2: -475431498

L[10]:
```

```
LogEntryType
4YMw9cd6w6Deu5GixS3BWN11WB02tkPVeAll3jOlS+xUwlGpp12A/EvxH18RwiBvDKZMO8QMDVVX
YvWDZXMSXIb8Hd6KfvWIUNkcOnx3QSiha3ZbzlLFnTE65X6h8x+v 1461733865 -725012515
Decrypted_AESmsg   D[10]:   LogEntryType   2013/08/02   1738160326   2013/08/02
1759000000 10 2013/08/02 1738160322      A11      Temp     23.6     Celsius
M[10]: LogEntryType 1759000000 10 2013/08/02 1738160322  A11      Temp     23.6
        Celsius
Yi: 1088389658
Yi2: 1461733865

L[11]:
LogEntryType
5RNIPpFlp7C8qnEHvYdSKPUBPxHFpoGxI8P3MnmPTjgGESk0Ao8C9uzdnbKXH9gds58Hc8FhGwsW
WBWjjEWdeeCFF+8xOKzbSt5CTujjemeSEu5rZdxAT3dbvWGisFfF 1150163232 -1863768431
Decrypted_AESmsg   D[11]:   LogEntryType   2013/08/02   1738200355   2013/08/02
1759000000 11 2013/08/02 1738200323      A11      Temp     22.2     Celsius
M[11]: LogEntryType 1759000000 11 2013/08/02 1738200323  A11      Temp     22.2
        Celsius
Yi: 1625993030
Yi2: 1150163232

L[12]:
NormalCloseMessage
WWa1XwKhl0FfX8zQPC84mHvPjCBn/hjJcic5yQd1XHG05a4HN0EV6hZVRJ39xgAo    -1294127047
1542216267
Decrypted_AESmsg  D[12]:  NormalCloseMessage  2013/08/02  1738240330  2013/08/02
1759000000 12
M[12]: NormalCloseMessage 1759000000 12
Yi: 1045309149
Yi2: -1294127047

In file LogFile.txt the MACs in following entry positions were found false
9
10
11
12
```

Signature Verification at RA

```
Singature1 entry[0] is verified
Manual Signatures verification: true
Singature2 entry[0] is verified
Singature2 entry[1] is verified
Singature2 entry[2] is verified
Singature2 entry[3] is verified
Singature2 entry[4] is verified
Singature2 entry[5] is verified
Singature2 entry[6] is verified
Singature2 entry[11] is verified
Singature2_random entry[12] is verified
Singature2_random entry[13] is verified
Singature2_random entry[14] is verified
Singature2_random entry[15] is verified
Singature2_random entry[32] is verified
```

Replacement of just a value of the entry

Server

```
L0: LogfileInitializationType
26/7L70JfmWl+kkNmKDaDNZ35rEC5GbHz+KgWWJHdc+rEMffBwc+eJmEx1GTcnjrSV5uHhcVMsfw
fZG50P3hD0VQkqsPikGhZ7t8zAbZmA+cUuYezEuNk2alHTfdqlV0QiBYeTZuEEGFmEolt9vss1FM
7+zvUVYb/lt+o/84RCIJri3xf2OgKG51lGlmeP95mobX9IypJojouZY1yvOauZM3EjjT36dts3rD
wKemkHi9rdjq8J8wPWoMBZ59fXN/UuuDVvxDvVKQ+CXZC8dU00UUJ7pMlY8hoxScnvt77Nhg6Van
AT9T+kT/8mnRApna3NDCJ/rF3omlnfancH3AdrYAphMdoYHyBT8B6zkB3+F+RpeSyo3pnfYruKSZ
SxoUKRSSH9/t8Xhs+E4V9OWuAcW0n+V2Pu1SJIrqJDAg+cvGeu94rypiBQ5mMbr+LkjghHvxxAoP
yjaSaarNzfZrFJMMzeVbEuzSHUgLL+ztIOTc/ik1kM7LULWZ+hQhJHhxdWFf7yXy3SjfwIzBM+nn
AdfOfbZyyAbMwQx9gxFQZ+E+2rUTXXv6LQ8Zfb3+cqNu   -2028916008   -1748646915
L1: ResponseMessageType
yth9+uYyf3eXeOuTKetkruDz1hIPRQxmpgozBEJeC8b8XeAsbs/inwILycOU0futWIXC0BM2VBTk
XQ0nDkJ4aOS7nmGc72DWiB7Z/ab1RaXqf7WzJ2JSifxOPJyhWiTx7DWteNi4U3SDYW8W76IuVyTw
UaosbOQKqxiztGDpdDhvEcgTafrbEMabaQCqBbYaEcqPosuRy282iSCnTNv2G86Wu01YjUCHWQyH
i8BQrdX4PHkuatOXbcYn7VjgGPZDLlRc53Zhq6oTk4/k41+ZtSLaqDYrFsf7xX3shU/K83ej0eJ1
68DrtebZ34X0vubwRn6AGvAc9+Zc7SuOvrEoLDgc5XTOn0PoYpon0HCSeqF7YEW3pYg7cfr/VchE
O4x5DrCzgPngCznWtCob4eQSpQkWrMi9cRXp95fLfSXlgykThg3MI+Gu0LeRu5RzJgkC4J/xACta
FGUl2zh3JdnUZvxDj0ZiWoBFerty1Yp2I/X6UxXCg3UCDk3bg8fplThKz+EZ01OneFDAArTTZGIJ
evhONCOs6BlIeSkhgZ5KkUs=   1668274918   1316783358
Sender: A11
Message: 2013/08/02 1717390813   A11     Temp    20.0    Celsius
L2: LogEntryType
gGfHqpMFiL3Eu9iHneAh2SXcSdlJNIjCpswJLd9N/LTMotWmiLOX68Y8RdQsW8LfFYy2JkUSgeEq
2IyHwcA/TYJ0E7BymwIPcCjv/2kCK4msiKh4Iqf2GoxcyAgKItrh   -289087486   -736976821
Sender: A11
Message: 2013/08/02 1717430825   A11     Temp    20.4    Celsius
L3: LogEntryType
tjRsJTiSKf/dDtl77s/6mSngCpy20nJjm+VbTRNjTNG68K/AhqXW5+/AzGc3JWQgiM3oxAf3PbxR
R+dHlQuLgHkZdmKWt81stzM9H4YlyyG/DuH5kTm9+5jiRZq94YYz   -1473285691   491059733
Sender: A11
Message: 2013/08/02 1717470826   A11     Temp    21.0    Celsius
L4: LogEntryType
VZxtozjm8r6ZDgT8CIh+o1/Q3/UPJ3DfeiR0cY6KaSIabuhhwWWI0ckBB0GUBGS0cIMXZlTocx+V
r4D2HlvYVQUpp5/GM/foT91BDEImx9TFbjI6bcaxD8j6dqn6Uh4g   1825419175   1985086190
Sender: A11
Message: 2013/08/02 1717510828   A11     Temp    22.0    Celsius
L5: LogEntryType
l6DS3l3Csob6+uIUQj3I+GClaf5/RZn/CsL+nH5T0L/7xqG/lbz/UKNbXH94+sXAn5LLCEBHMBSz
AJ+V7e+epISPsIzpUbaqYFMd0qr1UWhEw2S8Yi9hug0BAOQWU/ym   1631633144   2125718670
Sender: A11
Message: 2013/08/02 1717550831   A11     Temp    24.0    Celsius
L6: LogEntryType
O8CsUgaYIQVBCMk/8xKsfcFPelS1AVEyUJXyrtRGsAfDjDGZLl6A77PxUKKTd/f53/C4+AX3oPCX
CDXKGejjVNE5MmcYoiv3TMZlRf7vyiPLgrHTHI2SrhK29cWsN9wJ   1193776054   -
1473755810
Sender: A11
Message: 2013/08/02 1717590832   A11     Temp    25.4    Celsius
L7: LogEntryType
iO+Rb0gHhmbV8HEEmaD1jLjzV0bWGqRW5DBLPaxQZs0S7SYHh2ts0mEm9AdhnC75fWAFgR/KC6g5
IRQU3Pp4czkWoehA5opCuNzLEclhU+Og5WpgJCBEnzooXyLXEzBb   914446864   -1726915664
Sender: A11
Message: AlarmSystem: WARNING: 2013/08/02 1717590832     A11     Temp    25.4
        Celsius
L8: LogEntryType
tn4uUUoCYFGt0wezpME4hcCLZ3spSqdczOaJEqe6VXDK0CQy8lUoAqbSEYbBUlotfRuKjoZIHKFU
lTdUsk1TF8DNfXvmmYNJEc09KjjIruOoqeIi88fnaD6/qyZ7T4hdQhHgaA49NlEOqZLZiJPFsz97
dSS74dzl4v/ScndqYmY=   836145177   931247676
Sender: A11
Message: 2013/08/02 1718030835   A11     Temp    24.0    Celsius
L9: LogEntryType
BJx1O1v2YQA8lNi32hMO5VHJo4aYi+uE7fLU8N0DQO90abol5YP5HZSpkNGOn3m3DCLadAjtwWrs
```

```
FmTiWTvZ5Tldz+yepHDWe1gOR/pJDEMyIZF2cgQwhQnwxEQKysRP   -475431498   -456700380
Sender: A11
Message: 2013/08/02 1718070836   A11     Temp    23.6    Celsius
L10: LogEntryType
2mKlDqJCE6q/5Uals0nM+FuUb45zPI3DQ7Ex2j0w9h1rFDuEVzZ+1J7dxYYi0urrV3ux9btBd+2f
2msM1IFU90oS/JfFkopqBie7AM5yYzDTTp5LYMLIAXm580A8nwfp   1800467982   867721926
Sender: A11
Message: 2013/08/02 1718110840   A11     Temp    22.2    Celsius
L11: LogEntryType
pqNHI1/Rwkgl+STs74cZXL+wGjlcqD4u1stJRZ1R90KHmBO9V6Atxldc2qOqv6CKPUfERet9VubL
xpFKRlT3WeWVOnJV6dxDWh3sV6crz/4wfho6TTR/0c4V1PI8ELfM   -963140695   858046017
Sender: A11
Message: exit
L12: NormalCloseMessage
XkWfyJ6VuCqUhXZSOhpdbZUhwurrDjEK8ikNbsIiYr5hZ4kVVaTtwOstGoXvG5q8   -2096724193
-1026114158
```

Log Verification at TS

The attack performed at the log file was replacing the Yi value of the entry L10 with another one, marked with blue color.

```
Verification of : LogFile.txt
L[0]:
LogfileInitializationType
26/7L70JfmWl+kkNmKDaDNZ35rEC5GbHz+KgWWJHdc+rEMffBwc+eJmEx1GTcnjrSV5uHhcVMsfw
fZG50P3hD0VQkqsPikGhZ7t8zAbZmA+cUuYezEuNk2alHTfdqlV0QiBYeTZuEEGFmEolt9vss1FM
7+zvUVYb/lt+o/84RCIJri3xf2OgKG51lGlmeP95mobX9IypJojouZY1yvOauZM3EjjT36dts3rD
wKemkHi9rdjq8J8wPWoMBZ59fXN/UuuDVvxDvVKQ+CXZC8dU00UUJ7pMlY8hoxScnvt77Nhg6Van
AT9T+kT/8mnRApna3NDCJ/rF3omlnfancH3AdrYAphMdoYHyBT8B6zkB3+F+RpeSyo3pnfYruKSZ
SxoUKRSSH9/t8Xhs+E4V9OWuAcW0n+V2Pu1SJIrqJDAg+cvGeu94rypiBQ5mMbr+LkjghHvxxAoP
yjaSaarNzfZrFJMMzeVbEuzSHUgLL+ztIOTc/ik1kM7LULWZ+hQhJHhxdWFf7yXy3SjfwIzBM+nn
AdfOfbZyyAbMwQx9gxFQZ+E+2rUTXXv6LQ8Zfb3+cqNu -2028916008 -1748646915
Decrypted_AESmsg D[0]: LogfileInitializationType 2013/08/02 1717160020
2013/08/02 1745000000 0 0   4444   ŒÇ0ßùÚ)œ«7’qd)ƒ¤«ªs0ÁÖ÷#†›‰·LÅY_k§_–
o?8g]îÅ‰\wT½Ê&œ¢„2w¨ú§Ñ_@¯_¦!_}qªZûÐ__¡µn·x½[]9;_—Ü´4_5_…_Nu”²j_7ÖÀã_¢€hÂ_…8…W
…lYt¶—_-ã{ëC
k3b1IDN6mCDKAwYDuWm/BKMhRalFpNrVjphnj7OwSBdFDpO+UYXwS2iRg562o5gMdoJvUhXMSsFS
2wN/mzCVTYc+3FuZB6QLTttwBZS7NgUgq4FDcgm5gvNRXuIq24aZToNwG+jdPBFvrM/+c4BbNx5w
jlfyOxfF2MyRxLER0KX63+lP5XH8ZArWATZbU01VBqg3lTzSlvhn0l0IsEmwcLhanMxj83QVjnNx
3TvnLm0=
M[0]: LogfileInitializationType 1745000000 0 0   4444
ŒÇ0ßùÚ)œ«7’qd)ƒ¤«ªs0ÁÖ÷#†›‰·LÅY_k§_–
o?8g]îÅ‰\wT½Ê&œ¢„2w¨ú§Ñ_@¯_¦!_}qªZûÐ__¡µn·x½[]9;_—Ü´4_5_…_Nu”²j_7ÖÀã_¢€hÂ_…8…W
…lYt¶—_-ã{ëC
k3b1IDN6mCDKAwYDuWm/BKMhRalFpNrVjphnj7OwSBdFDpO+UYXwS2iRg562o5gMdoJvUhXMSsFS
2wN/mzCVTYc+3FuZB6QLTttwBZS7NgUgq4FDcgm5gvNRXuIq24aZToNwG+jdPBFvrM/+c4BbNx5w
jlfyOxfF2MyRxLER0KX63+lP5XH8ZArWATZbU01VBqg3lTzSlvhn0l0IsEmwcLhanMxj83QVjnNx
3TvnLm0=
Yi: -2028916008
Yi2: -2028916008
Y[0] = hash(Yi-1 ,AES(Di)_Ki, Wi) is verified
mac: -1748646915
mac2: -1748646915
MAC[0] = MAC(Yi)_Ai  is verified

L[1]:
ResponseMessageType
yth9+uYyf3eXeOuTKetkruDz1hIPRQxmpgozBEJeC8b8XeAsbs/inwILycOU0futWIXC0BM2VBTk
XQ0nDkJ4aOS7nmGc72DWiB7Z/ab1RaXqf7WzJ2JSifxOPJyhWiTx7DWteNi4U3SDYW8W76IuVyTw
```

```
UaosbOQKqxiztGDpdDhvEcgTafrbEMabaQCqBbYaEcqPosuRy282iSCnTNv2G86Wu01YjUCHWQyH
i8BQrdX4PHkuatOXbcYn7VjgGPZDL1Rc53Zhq6oTk4/k41+ZtSLaqDYrFsf7xX3shU/K83ej0eJ1
68DrtebZ34X0vubwRn6AGvAc9+Zc7SuOvrEoLDgc5XTOn0PoYpon0HCSeqF7YEW3pYg7cfr/VchE
O4x5DrCzgPngCznWtCob4eQSpQkWrMi9cRXp95fLfSXlgykThg3MI+Gu0LeRu5RzJgkC4J/xACta
FGUl2zh3JdnUZvxDj0ZiWoBFerty1Yp2I/X6UxXCg3UCDk3bg8fplThKz+EZ01OneFDAArTTZGIJ
evhONCOs6BlIeSkhgZ5KkUs= 1668274918 1316783358
Decrypted_AESmsg D[1]: ResponseMessageType 2013/08/02 1717160029 2013/08/02
1745000000 1 1   1234   ió¯={k_Â_-)F]MFáWËÃ_–
uU}á_Ìö!^ö?÷Ï£zzg_a_5Ñ_€€ÍÞÁX_Ió¼cÜof\1éêÎÌ²ëy‘[_²O³á__Ÿ_æw_žù) _Õ
‹V~õ_Añ ]û0È(åX_ÄÓ$›Q_£¹_Úœ«°Ù÷lÉø°
?_Ÿ>
E8hDITp4okjPr1ghbAHrwGxuw8WGMZF1XVjTa/mO/KD30TZ5K8Dew315OAoWMOrrQGAbsHg9XB0p
VkrPLujcWVrEeOmIVX22shovc3aZg/T4czJrqO3JaOn71CZXpxUkFUrM3vESAkI4Hz0IOeKIwa6H
lDjN+CBiV43vD4V9tw5g/MUk0/RCEkOvjRoPp4BFUSMDrRhNNTp2jkZL5yR5dA==
M[1]: ResponseMessageType 1745000000 1 1   1234   ió¯={k_Â_-)F]MFáWËÃ_–
uU}á_Ìö!^ö?÷Ï£zzg_a_5Ñ_€€ÍÞÁX_Ió¼cÜof\1éêÎÌ²ëy‘[_²O³á__Ÿ_æw_žù) _Õ
‹V~õ_Añ ]û0È(åX_ÄÓ$›Q_£¹_Úœ«°Ù÷lÉø°
?_Ÿ>
E8hDITp4okjPr1ghbAHrwGxuw8WGMZF1XVjTa/mO/KD30TZ5K8Dew315OAoWMOrrQGAbsHg9XB0p
VkrPLujcWVrEeOmIVX22shovc3aZg/T4czJrqO3JaOn71CZXpxUkFUrM3vESAkI4Hz0IOeKIwa6H
lDjN+CBiV43vD4V9tw5g/MUk0/RCEkOvjRoPp4BFUSMDrRhNNTp2jkZL5yR5dA==
Yi: 1668274918
Yi2: 1668274918
Y[1] = hash(Yi-1 ,AES(Di)_Ki, Wi) is verified
mac: 1316783358
mac2: 1316783358
MAC[1] = MAC(Yi)_Ai  is verified

L[2]:
LogEntryType
gGfHqpMFiL3Eu9iHneAh2SXcSdlJNIjCpswJLd9N/LTMotWmiLOX68Y8RdQsW8LfFYy2JkUSgeEq
2IyHwcA/TYJ0E7BymwIPcCjv/2kCK4msiKh4Iqf2GoxcyAgKItrh -289087486 -736976821
Decrypted_AESmsg D[2]: LogEntryType 2013/08/02 1717390838 2013/08/02
1745000000 2 2013/08/02 1717390813       A11     Temp    20.0    Celsius
M[2]: LogEntryType 1745000000 2 2013/08/02 1717390813    A11     Temp    20.0
        Celsius
Yi: -289087486
Yi2: -289087486
Y[2] = hash(Yi-1 ,AES(Di)_Ki, Wi) is verified
mac: -736976821
mac2: -736976821
MAC[2] = MAC(Yi)_Ai  is verified

L[3]:
LogEntryType
tjRsJTiSKf/dDtl77s/6mSngCpy20nJjm+VbTRNjTNG68K/AhqXW5+/AzGc3JWQgiM3oxAf3PbxR
R+dHlQuLgHkZdmKWt81stzM9H4YlyyG/DuH5kTm9+5jiRZq94YYz -1473285691 491059733
Decrypted_AESmsg D[3]: LogEntryType 2013/08/02 1717430829 2013/08/02
1745000000 3 2013/08/02 1717430825       A11     Temp    20.4    Celsius
M[3]: LogEntryType 1745000000 3 2013/08/02 1717430825    A11     Temp    20.4
        Celsius
Yi: -1473285691
Yi2: -1473285691
Y[3] = hash(Yi-1 ,AES(Di)_Ki, Wi) is verified
mac: 491059733
mac2: 491059733
MAC[3] = MAC(Yi)_Ai  is verified

L[4]:
```

```
LogEntryType
VZxtozjm8r6ZDgT8CIh+o1/Q3/UPJ3DfeiR0cY6KaSIabuhhwWWI0ckBB0GUBGS0cIMXZlTocx+V
r4D2HlvYVQUpp5/GM/foT91BDEImx9TFbjI6bcaxD8j6dqn6Uh4g 1825419175 1985086190
Decrypted_AESmsg D[4]: LogEntryType 2013/08/02 1717470830 2013/08/02
1745000000 4 2013/08/02 1717470826       A11     Temp    21.0    Celsius
M[4]: LogEntryType 1745000000 4 2013/08/02 1717470826    A11     Temp    21.0
        Celsius
Yi: 1825419175
Yi2: 1825419175
Y[4] = hash(Yi-1 ,AES(Di)_Ki, Wi) is verified
mac: 1985086190
mac2: 1985086190
MAC[4] = MAC(Yi)_Ai  is verified

L[5]:
LogEntryType
l6DS3l3Csob6+uIUQj3I+GClaf5/RZn/CsL+nH5T0L/7xqG/lbz/UKNbXH94+sXAn5LLCEBHMBSz
AJ+V7e+epISPsIzpUbaqYFMd0qr1UWhEw2S8Yi9hug0BAOQWU/ym 1631633144 2125718670
Decrypted_AESmsg D[5]: LogEntryType 2013/08/02 1717510841 2013/08/02
1745000000 5 2013/08/02 1717510828       A11     Temp    22.0    Celsius
M[5]: LogEntryType 1745000000 5 2013/08/02 1717510828    A11     Temp    22.0
        Celsius
Yi: 1631633144
Yi2: 1631633144
Y[5] = hash(Yi-1 ,AES(Di)_Ki, Wi) is verified
mac: 2125718670
mac2: 2125718670
MAC[5] = MAC(Yi)_Ai  is verified

L[6]:
LogEntryType
O8CsUgaYIQVBCMk/8xKsfcFPelS1AVEyUJXyrtRGsAfDjDGZLl6A77PxUKKTd/f53/C4+AX3oPCX
CDXKGejjVNE5MmcYoiv3TMZlRf7vyiPLgrHTHI2SrhK29cWsN9wJ 1193776054 -1473755810
Decrypted_AESmsg D[6]: LogEntryType 2013/08/02 1717550835 2013/08/02
1745000000 6 2013/08/02 1717550831       A11     Temp    24.0    Celsius
M[6]: LogEntryType 1745000000 6 2013/08/02 1717550831    A11     Temp    24.0
        Celsius
Yi: 1193776054
Yi2: 1193776054
Y[6] = hash(Yi-1 ,AES(Di)_Ki, Wi) is verified
mac: -1473755810
mac2: -1473755810
MAC[6] = MAC(Yi)_Ai  is verified

L[7]:
LogEntryType
iO+Rb0gHhmbV8HEEmaD1jLjzV0bWGqRW5DBLPaxQZs0S7SYHh2ts0mEm9AdhnC75fWAFgR/KC6g5
IRQU3Pp4czkWoehA5opCuNzLEclhU+Og5WpgJCBEnzooXyLXEzBb 914446864 -1726915664
Decrypted_AESmsg D[7]: LogEntryType 2013/08/02 1717590844 2013/08/02
1745000000 7 2013/08/02 1717590832       A11     Temp    25.4    Celsius
M[7]: LogEntryType 1745000000 7 2013/08/02 1717590832    A11     Temp    25.4
        Celsius
Yi: 914446864
Yi2: 914446864
Y[7] = hash(Yi-1 ,AES(Di)_Ki, Wi) is verified
mac: -1726915664
mac2: -1726915664
MAC[7] = MAC(Yi)_Ai  is verified

L[8]:
```

```
LogEntryType
tn4uUUoCYFGt0wezpME4hcCLZ3spSqdczOaJEqe6VXDK0CQy8lUoAqbSEYbBUlotfRuKjoZIHKFU
lTdUsk1TF8DNfXvmmYNJEc09KjjIruOoqeIi88fnaD6/qyZ7T4hdQhHgaA49NlEOqZLZiJPFsz97
dSS74dzl4v/ScndqYmY= 836145177 931247676
Decrypted_AESmsg D[8]: LogEntryType 2013/08/02 1717590891 2013/08/02
1745000000 8 AlarmSystem: WARNING: 2013/08/02 1717590832 A11     Temp    25.4
        Celsius
M[8]: LogEntryType 1745000000 8 AlarmSystem: WARNING: 2013/08/02 1717590832
        A11     Temp    25.4    Celsius
Yi: 836145177
Yi2: 836145177
Y[8] = hash(Yi-1 ,AES(Di)_Ki, Wi) is verified
mac: 931247676
mac2: 931247676
MAC[8] = MAC(Yi)_Ai  is verified

L[9]:
LogEntryType
BJx1O1v2YQA8lNi32hMO5VHJo4aYi+uE7fLU8N0DQO90abol5YP5HZSpkNGOn3m3DCLadAjtwWrs
FmTiWTvZ5Tldz+yepHDWe1gOR/pJDEMyIZF2cgQwhQnwxEQKysRP -475431498 -456700380
Decrypted_AESmsg D[9]: LogEntryType 2013/08/02 1718030839 2013/08/02
1745000000 9 2013/08/02 1718030835       A11     Temp    24.0    Celsius
M[9]: LogEntryType 1745000000 9 2013/08/02 1718030835    A11     Temp    24.0
        Celsius
Yi: -475431498
Yi2: -475431498
Y[9] = hash(Yi-1 ,AES(Di)_Ki, Wi) is verified
mac: -456700380
mac2: -456700380
MAC[9] = MAC(Yi)_Ai  is verified

L[10]:
LogEntryType
2mKlDqJCE6q/5Uals0nM+FuUb45zPI3DQ7Ex2j0w9h1rFDuEVzZ+1J7dxYYi0urrV3ux9btBd+2f
2msM1IFU90oS/JfFkopqBie7AM5yYzDTTp5LYMLIAXm580A8nwfp 1557320632 867721926
Decrypted_AESmsg D[10]: LogEntryType 2013/08/02 1718070850 2013/08/02
1745000000 10 2013/08/02 1718070836      A11     Temp    23.6    Celsius
M[10]: LogEntryType 1745000000 10 2013/08/02 1718070836  A11     Temp    23.6
        Celsius
Yi: 1800467982
Yi2: 1557320632

L[11]:
LogEntryType
pqNHI1/Rwkgl+STs74cZXL+wGjlcqD4u1stJRZ1R90KHmBO9V6Atxldc2qOqv6CKPUfERet9VubL
xpFKRlT3WeWVOnJV6dxDWh3sV6crz/4wfho6TTR/0c4V1PI8ELfM -963140695 858046017
Decrypted_AESmsg D[11]: LogEntryType 2013/08/02 1718110844 2013/08/02
1745000000 11 2013/08/02 1718110840      A11     Temp    22.2    Celsius
M[11]: LogEntryType 1745000000 11 2013/08/02 1718110840  A11     Temp    22.2
        Celsius
Yi: -963140695
Yi2: -963140695
Y[11] = hash(Yi-1 ,AES(Di)_Ki, Wi) is verified
mac: 858046017
mac2: 858046017
MAC[11] = MAC(Yi)_Ai  is verified

L[12]:
```

```
NormalCloseMessage
XkWfyJ6VuCqUhXZSOhpdbZUhwurrDjEK8ikNbsIiYr5hZ4kVVaTtwOstGoXvG5q8 -2096724193 -
1026114158
Decrypted_AESmsg D[12]: NormalCloseMessage 2013/08/02 1718150846 2013/08/02
1745000000 12
M[12]: NormalCloseMessage 1745000000 12
Yi: -2096724193
Yi2: -2096724193
Y[12] = hash(Yi-1 ,AES(Di)_Ki, Wi) is verified
mac: -1026114158
mac2: -1026114158
MAC[12] = MAC(Yi)_Ai  is verified

In file LogFile.txt the MACs in following entry positions were found false
10
```

Signature Verification at RA

```
Singature1 entry[0] is verified
Manual Signatures verification: true
Singature2 entry[0] is verified
Singature2 entry[1] is verified
Singature2 entry[2] is verified
Singature2 entry[3] is verified
Singature2 entry[4] is verified
Singature2 entry[5] is verified
Singature2 entry[6] is verified
Singature2 entry[7] is verified
Singature2 entry[11] is verified
Singature2_random entry[12] is verified
Singature2_random entry[13] is verified
Singature2_random entry[14] is verified
Singature2_random entry[15] is verified
Singature2_random entry[16] is verified
Singature2_random entry[27] is verified
Singature2_random entry[28] is verified
```

Reordering of entries

Server

```
L0: LogfileInitializationType
KwlqUB6JTFfWO9XhyzkmEmnjhVQoUZK4mlBS532z1a4RP0775ctQe5PM2pIgV5+GU4k0GxW6Dh4t
mg5dSis/YTdzmxXNmfp3nBlcvMgXVcU3DuBWNo7WTsUNGX41a8jZ6xUXlP2Y1R0+oq/GEPV7WzHT
AXP+RwQgrmKmYVF6uc/BSIneaMgjqx3a/3wohcFDgshvJviLqnF2Yd/bTab188wCNj6Wu0XH3FUH
MQssLZv23rrYqDNaI5auRMk63ZmWO3BuKFw0K0G5RaK4jAObXAITTPanhWR9YYxLS5Yb0tttcB+S
n6RDNlFB7oo/DUYLMR7lekOiusIievAyzGOUgtqO/6X50P9OJQChTbjDrMu6hEM7p/OqadL6Exhp
B5xxQlj06MrB1SMemkHmB3iFePNASD2IRzgcJs4cDAbh4ro9j1k24uXbkszcSS9VLA3tvuHdPntX
W4w41oZpWLcKm7ag6XpTF09ijdDOiVSCfbc03V0xWCKE1E2EVJz5lCME+O9d9uLGlijcemZmix82
KQtTb/adEzPsjHvPRFT+Sx483uch5XTass8uEitiNv7S   1065272843   -1689420931
L1: ResponseMessageType
ge/k4e+xKmNsMLusRcVkREx51OpAi1SytDvt9xXUEZDnKxiNPzvBdRJ9pWwbzFwDXYj4oRQBb2/A
Q7yq3WqlorrGZe1j6a+hw3ce0OD9cq7lHP9FwmQJ9MhITpwkuFUkAruDeAleopzaUD4Ov7/vbDEC
hr7TcQ/NTmTaXBXtQ9NrbFKOj/IgizRGOHLMv5tF81yFhrO+GPXudROL94zVvMSXaVFMD+EKuTZt
MTy9g/WX16GbYkrtr2RZTY9LEUM94v1EwjA9BTF4vOA5JiKwpS2cSq5eLti3qJIrdIAeN84v63er
ubvk50twRzikAQCGRDs673uya7dwOrK2WmhTzcypvuN5qPbUT+3DJnlB5TXYXbSgcIZEKmy1C3uC
JuvZNtK5/TomHMWg0QDcdJr1W82CfLHpvYXsub3MvzB7sROU04X6YBf1sGEfacyohbrI4q1V/VwR
MWjrGe5by9HT6n0OrtLlMKRpKYuNXZvk8OV/rW5XHJiIhTM7FMc1z1jF8x78gWlhCV334kXyXwrQ
GGJ7dYNmkmfkCtwWrAiIOdE=   -243663288   -359316197
Sender: A11
```

```
Message: 2013/08/08 1547060078   A11     Temp    20.0    Celsius
L2: LogEntryType
LJG+wXHBEOQzlobuXXQ6uDEElIJ0KlmIoIcdRFSjBzKm4XE9vCfb/a7nnCG+cmhD7qWmkHRvQ3HM
BP+9V8sRuG5aHBfMFWQquFsGSIGb5Rv16/05jpRlJ5UbW2ov7Igk   -1858318454   -
1617516675
Sender: A11
Message: 2013/08/08 1547100141   A11     Temp    20.4    Celsius
L3: LogEntryType
W8EiswqHWxWKfsq3+/UhWC6Aesuz0cMEaSpbwNlVvEpr72KP5ZqMM7e2fuMTFAHDEAWzNn3O38tv
ekSVkZAb62jTtnHP5DsaX15hdClnB5hMrKcuyxrinYVcDfOggx1d   -419961772   -
1836305766
Sender: A11
Message: 2013/08/08 1547140146   A11     Temp    21.0    Celsius
L4: LogEntryType
mqvgvLAWMntj9weq3+bMjhdH3g9rLqbaHsIAOIzgXJAzaVelquBhmtIVMmVjGkCrO67Djj/mgqCD
TcBUXMf0HPmPWJ+51lJyj5atjDAvelgHOzwjngr1JAiDlsL1Cdfh   -1178753234
1092979152
Sender: A11
Message: 2013/08/08 1547180160   A11     Temp    22.0    Celsius
L5: LogEntryType
X5yZhmdZoiJs6UouYhNqjqolzP3KYTVI7ld0ymJev5hBbrST+N6vFRJuqzFx3xR35zcQxgutCX0C
lssbAcrp1LFgyWG8GrURStMVrUxoYMPKuQNqp2K05PS3LYTxjpyc   1763019560   -
1774162090
Sender: A11
Message: 2013/08/08 1547220174   A11     Temp    24.0    Celsius
L6: LogEntryType
qiaccfor4ZornhF6KmFSyqoyKKcfbMCEYXkBkYD7q1cj61KnBerM0ee/pEpD0ca/MLAK/+pBlxcj
/Lck7+aVHz7U3s0rCycL2i46q7dwfmd5vCmqAOEJwgQd7dNbN2Xb   -953235483   1556122196
Sender: A11
Message: 2013/08/08 1547260187   A11     Temp    25.4    Celsius
L7: LogEntryType
VE5AVbaMM2ogXijrDoULVvhJifbPMTLDK2RReImBSBrAEMYO//vONfNslnU18cOnNyhuuzy//u4S
5Zd1AqMyrMInqpILsyV1c9CaqqInzVx7JNhVrec0NYqvqY6p9Ye+   -1166300104
1698578123
Sender: A11
Message: AlarmSystem: WARNING: 2013/08/08 1547260187     A11     Temp    25.4
        Celsius
L8: LogEntryType
tmQwlsok2WICUB9HWa2DPkCv2zGfUezM4e2DyuUQvZY8VqBAgeq115wdWCj7vdT1b7K1kuM82piD
fyL1YRJ1yyEgwMU7Ym/lCqxAL42aU2QRzbUy+aVQv06NFul/3VOFTOJXASTdLAHNMbCce8Hj3Mhp
/a6eszSU2sRdemwu68s=   1351742253   1386614301
Sender: A11
Message: 2013/08/08 1547300198   A11     Temp    24.0    Celsius
L9: LogEntryType
5f+kDdTn3qZKg5ml434ZDOvFeoVQdWqU+Uc8gFowcny1K41lvpoNtK3xP57A1+X8dQyvihnxAuvC
E6icrprhoZ/j5a0rDivlpCnEA9ssGKZQ5cUioY5sDf038BjwgsKG   893092105   1322186465
Sender: A11
Message: 2013/08/08 1547340212   A11     Temp    23.6    Celsius
L10: LogEntryType
JSLPuEE2QNuw6mtHOuc6qo+eJuhrpbjswpwGOD6SqYGW69Thfcg+LJ65zfSG4Y0CM3sHCwN27W38
iMJIg6UDCMJRJMqAg2L2KPQtbA0cDstjvxDxDC5z6ck8YkX20Ekj   -729911469   812367851
Sender: A11
Message: 2013/08/08 1547380224   A11     Temp    22.2    Celsius
L11: LogEntryType
/3km+OZAZcJhw2pW3dxpoge2CPdNJFNhOKZ/cgrcf4OZImrxKTQIOJFIEGiXvwLY3dpbVfBiBGcO
Tk7FLoktU2t/3rKpCBvCNMuio0GcB3m19CaD/CFlnAXI2+wsKH7   1977346094   1057321247
Sender: A11
Message: exit
```

```
L12: NormalCloseMessage
hB3sYH8L8l4CLGvaffTpj/scFqC8NC3b3uw7NwQgS1S42yOQiDU7sKZiUf45H5X7   1886200846
1127868429
```

Log Verification at TS

The log entries L4 and L5 have been reordered at the log file. We mark them with blue color here.

```
Verification of : LogFile.txt
L[0]:
LogfileInitializationType
KwlqUB6JTFfWO9XhyzkmEmnjhVQoUZK4mlBS532z1a4RP0775ctQe5PM2pIgV5+GU4k0GxW6Dh4t
mg5dSis/YTdzmxXNmfp3nBlcvMgXVcU3DuBWNo7WTsUNGX41a8jZ6xUXlP2Y1R0+oq/GEPV7WzHT
AXP+RwQgrmKmYVF6uc/BSIneaMgjqx3a/3wohcFDgshvJviLqnF2Yd/bTab188wCNj6Wu0XH3FUH
MQssLZv23rrYqDNaI5auRMk63ZmWO3BuKFw0K0G5RaK4jAObXAITTPanhWR9YYxLS5Yb0tttcB+S
n6RDNlFB7oo/DUYLMR7lekOiusIievAyzGOUgtqO/6X50P9OJQChTbjDrMu6hEM7p/OqadL6Exhp
B5xxQlj06MrB1SMemkHmB3iFePNASD2IRzgcJs4cDAbh4ro9j1k24uXbkszcSS9VLA3tvuHdPntX
W4w41oZpWLcKm7ag6XpTF09ijdDOiVSCfbc03V0xWCKE1E2EVJz5lCME+O9d9uLGlijcemZmix82
KQtTb/adEzPsjHvPRFT+Sx483uch5XTass8uEitiNv7S 1065272843 -1689420931
Decrypted_AESmsg D[0]: LogfileInitializationType 2013/08/08 1546260079
2013/08/08 1700000000 0 0   4444
¿Nš²N¾¹°°LâkDÞßá®ÆÐÿ_<ÞÉ¶ê]‡”÷ƒ{"_¸*ª/_Ï?_àE?BŒT“hÆ_¼?æ?›ë¶äºÐcÂâ²è_6|m
"‹8uß_¥¸Š€c_)n¬Sá___»n¬³?LˆN{Ÿœ V&__ú_’?D³Ö¢ú*nYŒ)´Æ¿í]?
pJx1wpW2ixbz5gSvYPUxpKYbRaNGDcvcHOB1afB8kkQygbrhfLEUBoCeRBBLn6URniM/7rzHnPxj
5CjiEaIdO3DfCmTnCYwCtJAY6OK8ZB/uR1AmRFqdXrI6hV+ODohbZpyyYmXKicGQkwJ9P+pzOBdZ
swxR/PDoIpAiLQTHL564H6fJTUrsBsEEjoh2ovyf78bnxLP4bvAY5qEgYFLBlxDN5BUlNz+ozFh3
FIKbLOQ=
M[0]: LogfileInitializationType 1700000000 0 0   4444
¿Nš²N¾¹°°LâkDÞßá®ÆÐÿ_<ÞÉ¶ê]‡”÷ƒ{"_¸*ª/_Ï?_àE?BŒT“hÆ_¼?æ?›ë¶äºÐcÂâ²è_6|m
"‹8uß_¥¸Š€c_)n¬Sá___»n¬³?LˆN{Ÿœ V&__ú_’?D³Ö¢ú*nYŒ)´Æ¿í]?
pJx1wpW2ixbz5gSvYPUxpKYbRaNGDcvcHOB1afB8kkQygbrhfLEUBoCeRBBLn6URniM/7rzHnPxj
5CjiEaIdO3DfCmTnCYwCtJAY6OK8ZB/uR1AmRFqdXrI6hV+ODohbZpyyYmXKicGQkwJ9P+pzOBdZ
swxR/PDoIpAiLQTHL564H6fJTUrsBsEEjoh2ovyf78bnxLP4bvAY5qEgYFLBlxDN5BUlNz+ozFh3
FIKbLOQ=
Yi: 1065272843
Yi2: 1065272843
Y[0] = hash(Yi-1 ,AES(Di)_Ki, Wi) is verified
mac: -1689420931
mac2: -1689420931
MAC[0] = MAC(Yi)_Ai  is verified

L[1]:
ResponseMessageType
ge/k4e+xKmNsMLusRcVkREx51OpAi1SytDvt9xXUEZDnKxiNPzvBdRJ9pWwbzFwDXYj4oRQBb2/A
Q7yq3WqlorrGZe1j6a+hw3ce0OD9cq7lHP9FwmQJ9MhITpwkuFUkAruDeAleopzaUD4Ov7/vbDEC
hr7TcQ/NTmTaXBXtQ9NrbFKOj/IgizRGOHLMv5tF81yFhrO+GPXudROL94zVvMSXaVFMD+EKuTZt
MTy9g/WX16GbYkrtr2RZTY9LEUM94v1EwjA9BTF4vOA5JiKwpS2cSq5eLti3qJIrdIAeN84v63er
ubvk50twRzikAQCGRDs673uya7dwOrK2WmhTzcypvuN5qPbUT+3DJnlB5TXYXbSgcIZEKmy1C3uC
JuvZNtK5/TomHMWg0QDcdJr1W82CfLHpvYXsub3MvzB7sROU04X6YBf1sGEfacyohbrI4q1V/VwR
MWjrGe5by9HT6n0OrtLlMKRpKYuNXZvk8OV/rW5XHJiIhTM7FMc1z1jF8x78gWlhCV334kXyXwrQ
GGJ7dYNmkmfkCtwWrAiIOdE= -243663288 -359316197
Decrypted_AESmsg D[1]: ResponseMessageType 2013/08/08 1546260079 2013/08/08
1700000000 1 1   1234   _?H#‹:?z?ÑÞ_v¬'øVâÒeÿ_º’¶Ø/ê,*a\üCø$4š_?—
¼¢»õ_a__ë6qÈ_ÆÏ§¼ÇË}_r,N_MÊµŒ¼å÷dv:@BUÆcÂ…ú÷ýd›ðÝ„¦
        `/_D¡’¾Î|;YUû_xàoƒ›rƒ÷vËi_78?ù¼]_m©
tMpT2d4GMGV8+/gab8InLjrv486ay+wzHFcGApwDyWBAcv0kAeVDERZtcuW6OOfSrhWXaPSqFbU9
aQgmFPw92HKOPegA3AkxPx/AwPp8t5BLaXBzSYtVwbj9ChvJ5WaO22u1PjxebOVp0bADI2AqPN9x
58MsJ9/B2T+13dplLjHlGCuxexowKjYfUdyDyjWBTZRtwxven61MOpYEU/e9Zw==
```

```
M[1]: ResponseMessageType 1700000000 1 1   1234
_?H#‹:?z?ÑÞ_v¬'øVâÒeÿ_º’¶Ø/ê,*a\üCø$4š_?—
¼¢»õ_a__ë6qÊ_ÆÏ§¼ÇË}_r,N_MÊµŒ¼å÷dv:@BUÆcÂ…ú÷ýd>ðÝ„¦
        `/_D¡’¾Î|;YUû_xàoƒ›rƒ÷vËi_78?û¼]_m©
tMpT2d4GMGV8+/gab8InLjrv486ay+wzHFcGApwDyWBAcv0kAeVDERZtcuW6OOfSrhWXaPSqFbU9
aQgmFPw92HKOPegA3AkxPx/AwPp8t5BLaXBzSYtVwbj9ChvJ5WaO22u1PjxebOVp0bADI2AqPN9x
58MsJ9/B2T+13dplLjHlGCuxexowKjYfUdyDyjWBTZRtwxven61MOpYEU/e9Zw==
Yi: -243663288
Yi2: -243663288
Y[1] = hash(Yi-1 ,AES(Di)_Ki, Wi) is verified
mac: -359316197
mac2: -359316197
MAC[1] = MAC(Yi)_Ai  is verified

L[2]:
LogEntryType
LJG+wXHBEOQzlobuXXQ6uDEElIJ0KlmIoIcdRFSjBzKm4XE9vCfb/a7nnCG+cmhD7qWmkHRvQ3HM
BP+9V8sRuG5aHBfMFWQquFsGSIGb5Rv16/05jpRlJ5UbW2ov7Igk -1858318454 -1617516675
Decrypted_AESmsg D[2]: LogEntryType 2013/08/08 1547060157 2013/08/08
1700000000 2 2013/08/08 1547060078       A11     Temp    20.0    Celsius

M[2]: LogEntryType 1700000000 2 2013/08/08 1547060078    A11     Temp    20.0
        Celsius

Yi: -1858318454
Yi2: -1858318454
Y[2] = hash(Yi-1 ,AES(Di)_Ki, Wi) is verified
mac: -1617516675
mac2: -1617516675
MAC[2] = MAC(Yi)_Ai  is verified

L[3]:
LogEntryType
W8EiswqHWxWKfsq3+/UhWC6Aesuz0cMEaSpbwNlVvEpr72KP5ZqMM7e2fuMTFAHDEAWzNn3O38tv
ekSVkZAb62jTtnHP5DsaX15hdClnB5hMrKcuyxrinYVcDfOggx1d -419961772 -1836305766
Decrypted_AESmsg D[3]: LogEntryType 2013/08/08 1547100174 2013/08/08
1700000000 3 2013/08/08 1547100141       A11     Temp    20.4    Celsius

M[3]: LogEntryType 1700000000 3 2013/08/08 1547100141    A11     Temp    20.4
        Celsius

Yi: -419961772
Yi2: -419961772
Y[3] = hash(Yi-1 ,AES(Di)_Ki, Wi) is verified
mac: -1836305766
mac2: -1836305766
MAC[3] = MAC(Yi)_Ai  is verified

L[4]:
LogEntryType
X5yZhmdZoiJs6UouYhNqjqolzP3KYTVI7ld0ymJev5hBbrST+N6vFRJuqzFx3xR35zcQxgutCX0C
lssbAcrp1LFgyWG8GrURStMVrUxoYMPKuQNqp2K05PS3LYTxjpyc 1763019560 -1774162090
AES message cannot be decrypted! Someone changed log entry [4].
Yi: 1190319881
Yi2: 1763019560

L[5]:
LogEntryType
mqvgvLAWMntj9weq3+bMjhdH3g9rLqbaHsIAOIzgXJAzaVelquBhmtIVMmVjGkCrO67Djj/mgqCD
TcBUXMf0HPmPWJ+51lJyj5atjDAvelgHOzwjngr1JAiDlsL1Cdfh -1178753234 1092979152
```

```
AES message cannot be decrypted! Someone changed log entry [5].
Yi: 875957742
Yi2: -1178753234

L[6]:
LogEntryType
qiaccfor4ZornhF6KmFSyqoyKKcfbMCEYXkBkYD7q1cj61KnBerM0ee/pEpD0ca/MLAK/+pBlxcj
/Lck7+aVHz7U3s0rCycL2i46q7dwfmd5vCmqAOEJwgQd7dNbN2Xb -953235483 1556122196
Decrypted_AESmsg D[6]: LogEntryType 2013/08/08 1547220174 2013/08/08
1700000000 6 2013/08/08 1547220174       A11     Temp    24.0    Celsius
M[6]: LogEntryType 1700000000 6 2013/08/08 1547220174    A11     Temp    24.0
        Celsius
Yi: -1579941509
Yi2: -953235483

L[7]:
LogEntryType
VE5AVbaMM2ogXijrDoULVvhJifbPMTLDK2RReImBSBrAEMYO//vONfNslnU18cOnNyhuuzy//u4S
5Zd1AqMyrMInqpILsyV1c9CaqqInzVx7JNhVrec0NYqvqY6p9Ye+ -1166300104 1698578123
Decrypted_AESmsg D[7]: LogEntryType 2013/08/08 1547260187 2013/08/08
1700000000 7 2013/08/08 1547260187       A11     Temp    25.4    Celsius
M[7]: LogEntryType 1700000000 7 2013/08/08 1547260187    A11     Temp    25.4
        Celsius
Yi: -94812402
Yi2: -1166300104

L[8]:
LogEntryType
tmQwlsok2WICUB9HWa2DPkCv2zGfUezM4e2DyuUQvZY8VqBAgeq115wdWCj7vdT1b7K1kuM82piD
fyL1YRJ1yyEgwMU7Ym/lCqxAL42aU2QRzbUy+aVQv06NFul/3VOFTOJXASTdLAHNMbCce8Hj3Mhp
/a6eszSU2sRdemwu68s= 1351742253 1386614301
Decrypted_AESmsg D[8]: LogEntryType 2013/08/08 1547260202 2013/08/08
1700000000 8 AlarmSystem: WARNING: 2013/08/08 1547260187 A11     Temp    25.4
        Celsius
M[8]: LogEntryType 1700000000 8 AlarmSystem: WARNING: 2013/08/08 1547260187
        A11     Temp    25.4    Celsius
Yi: -1095276413
Yi2: 1351742253

L[9]:
LogEntryType
5f+kDdTn3qZKg5ml434ZDOvFeoVQdWqU+Uc8gFowcny1K41lvpoNtK3xP57A1+X8dQyvihnxAuvC
E6icrprhoZ/j5a0rDivlpCnEA9ssGKZQ5cUioY5sDf038BjwgsKG 893092105 1322186465
Decrypted_AESmsg D[9]: LogEntryType 2013/08/08 1547300198 2013/08/08
1700000000 9 2013/08/08 1547300198       A11     Temp    24.0    Celsius
M[9]: LogEntryType 1700000000 9 2013/08/08 1547300198    A11     Temp    24.0
        Celsius
Yi: 426818653
Yi2: 893092105

L[10]:
LogEntryType
JSLPuEE2QNuw6mtHOuc6qo+eJuhrpbjswpwGOD6SqYGW69Thfcg+LJ65zfSG4Y0CM3sHCwN27W38
iMJIg6UDCMJRJMqAg2L2KPQtbA0cDstjvxDxDC5z6ck8YkX20Ekj -729911469 812367851
Decrypted_AESmsg D[10]: LogEntryType 2013/08/08 1547340212 2013/08/08
1700000000 10 2013/08/08 1547340212      A11     Temp    23.6    Celsius
M[10]: LogEntryType 1700000000 10 2013/08/08 1547340212  A11     Temp    23.6
        Celsius
Yi: -197065343
Yi2: -729911469
```

```
L[11]:
LogEntryType
/3km+OZAZcJhw2pW3dxpoge2CPdNJFNhOKZ/cgrcf4OZImrxKTQIOJFIEGiXvwLY3dpbVfBiBGcO
Tk7FLokntU2t/3rKpCBvCNMuio0GcB3m19CaD/CFlnAXI2+wsKH7 1977346094 1057321247
Decrypted_AESmsg D[11]: LogEntryType 2013/08/08 1547380224 2013/08/08
1700000000 11 2013/08/08 1547380224      A11      Temp     22.2     Celsius
M[11]: LogEntryType 1700000000 11 2013/08/08 1547380224  A11      Temp     22.2
        Celsius
Yi: -104617180
Yi2: 1977346094

L[12]:
NormalCloseMessage
hB3sYH8L8l4CLGvaffTpj/scFqC8NC3b3uw7NwQgS1S42yOQiDU7sKZiUf45H5X7 1886200846
1127868429
Decrypted_AESmsg D[12]: NormalCloseMessage 2013/08/08 1547420236 2013/08/08
1700000000 12
M[12]: NormalCloseMessage 1700000000 12
Yi: 1338924421
Yi2: 1886200846

In file LogFile.txt the MACs in following entry positions were found false
4
5
6
7
8
9
10
11
12
```

Signature Verification at RA

```
Singature1 entry[0] is verified
Manual Signatures verification: true
Singature2 entry[0] is verified
Singature2 entry[1] is verified
Singature2 entry[2] is verified
Singature2 entry[11] is verified
Singature2_random entry[12] is verified
Singature2_random entry[13] is verified
Singature2_random entry[39] is verified
Singature2_random entry[40] is verified
Singature2_random entry[41] is verified
```

Deletion of an entry

Server

```
L0: LogfileInitializationType
+ukhe3ocWUwZ5KyeEXFPeOiPS3A334mK1ihubDkdamkMjgPI8MUay6q5aHn+LHViz/pCzuamg7Mf
OBNR85Na6uIvFNsgb78MeyFqBZ94GcfWiR4iEi0tD37rQYZLx48ObPbPI+DU4uuXILd64/0N5Gdp
dQnupD/LfdldL423EtOaN6R/Vw5gh18O4grJJCMhcWX8hzUm768ArGZw/oftr1yLp/L6wScWOXWI
tve8hCUQf8+9LeAXh8bNIWpD/WVeSxBitrI5afVXHoTLF6Yg725ZWbnY9P3kP+C6MeQVw6n5UVbX
+69zkGX1MlRb9ZWjEQDGwyUMuv8lkPV4pePzP3EcEBQ0iZkaqkwv58R5GbULM0OVNlFBGc13gxAa
BSxpEvRrr1Y5Y4rYhI/rrw8hl96akx9wesjboA84l/xTx+efahM1B+a+Ob66AbAWloMoqzwXZFFk
mv3r+J/e1Y4Jn2TmhkYMuPtEmDxv2qdqQUVfPfnfBkhYtOtIZXObI9hUjybT6uWcf5TJ6e4zqUEh
UaCQl7PEIaNOUXJ2xoSlT6/Doc2IN3iuosm4xdvYkxRO    -831566549    604785157
```

```
L1: ResponseMessageType
bJzxDJZqto455UtT/0TZ4q1rQSUjaFTm8Bp5KzVDQ1ErbZurZKKYd3FDqBG6lJaeciEnLBJHtfxj
LBP5Sj5qWh8r5vkwpj2I72Hxk24v8+k60JCLUaMYqfneBNjirfvM91CAKdJXbgZDcL0fUgKWc/qX
zEX1w4mq4Fgs4rH9ZBf+XZqE1voirBWprRhaFHxg1Xl0b1oFdulbKs0FHW9Em++e49grM1fQEY/m
bh6WszvuWD73mFk0n/jBX2k2+SMgJqSgLaowNQe5EAAA2qpyBPPrMFEl3KDMSUJ0jMdZVCbS3nRI
M/tX0V8BbjIr2Fv/HTcyeeLZz/VuHUhPXq9xaeE/xQPkHpsginwkunhR98NkMrfBqgxVYqEQf5XA
LqiG6nxXoAtg5cC3UlGSSIwu3pIsVeUs9jk7CwXDhYjFsKY+Uw/M2wY829+X/SWFwKoWlm/1a6S3
B/C96AA5SJyrHv3c8PIHjfo5kIJViWEv4q8VRYTnLZxlclSajSdaCDVPt3GraS0R+PByGtSsuFWe
QDKply6Nehkg6A1DcvxWF2c=   -1930519867   -400813681
Sender: A11
Message: 2013/08/02 1257530815   A11     Temp    20.0    Celsius
L2: LogEntryType
9nSXW92jbZUBK9bDKdw31Ef2xKpc48rx4pcSuA4CP5cPWib430CW2l6enYYXvDCxHD3CyBqn+ENI
iv4XD/h57M4T53wlMMjcGCmlSyChApT7E1L8UDbzYXFLtaD7Xvn9   -1671185904
1902346479
Sender: A11
Message: 2013/08/02 1257570828   A11     Temp    20.4    Celsius
L3: LogEntryType
BW23Fto4md697G2onEoK0ZjNwRiGmi6jy2sK3fFStyxNKriRdvBtM6tpzsS0Gw9L76eEaarQQRUQ
7JNgxCAq0WobjtAUa129PQGRk/cgRdJoeRqDtXa1SwSm7X1lc71t   1374022861   -
2010687340
Sender: A11
Message: 2013/08/02 1258010839   A11     Temp    21.0    Celsius
L4: LogEntryType
ccleLTCC8yJuq5RIF+IOjdjKIWfdUDNw/eIWNt3jCqYXus/MWvNHTMFOE/UIMng0SRabLu0IL73I
1KlzGedPdiVYeH07Yyy4vxNSL/ntmVRgVBhAB6RQZTOnzdpYGIv1   11442941   -820787939
Sender: A11
Message: 2013/08/02 1258050853   A11     Temp    22.0    Celsius
L5: LogEntryType
LlEWZFdoMrkp88d5OIIR2SxZp6fTCGFO6wu6TWr8Qbi7GdvzokUYNXkKIRCpM5AS9ko8Y8LJ5fBJ
h4xa4v+/2HMcxHE1zqvP3iMpzcS1jAt96tNxyUa/HkdKjEs95yFc   -1940249984   -
304847547
Sender: A11
Message: 2013/08/02 1258090864   A11     Temp    24.0    Celsius
L6: LogEntryType
0TcHf50+eMAwVl1j0u12xaLisIIcDYQUkJtS0eJfYgdqXjdjxxpHqfRBeyhajt+eTYHEQNg0xuaX
+xkjzdYeuIuZa4ea8/yHCXAa/5vB9NkANbyoZcuGcne0ieaFV3nG   -2001192187   -
745038720
Sender: A11
Message: 2013/08/02 1258130877   A11     Temp    25.4    Celsius
L7: LogEntryType
Xe6ykapmHGzb4FEdcRUklv44JOkTw7PRxwyMREoTQ1DU1mNpaRPZVP2SjNwpEf87OevdDr3O0VXX
k+h9xrK67myJbSSTUeyA8j4AeMiYD0jb9Wvl7roycsno4srAfSzR   1637110083   -414528027
Sender: A11
Message: AlarmSystem: WARNING: 2013/08/02 1258130877     A11     Temp    25.4
        Celsius
L8: LogEntryType
vkpEnnrl5n4KxdPW9vvnM+jhzRVRGmDCioqpASFAR+mUWjszio2miec30AfcbkLRnUcxlCd3XtW2
eLNaEqChlIRxHPYWo6imwh13CtFesYt6+9apM4gq1tUJK5vMEMNp4MpaoR80rUSfXY1XMzJDD7GF
JBfr9Rl/HmSXzXgc/Ds=   1188695447   216820858
Sender: A11
Message: 2013/08/02 1258170890   A11     Temp    24.0    Celsius
L9: LogEntryType
T4S41aum+oLO1suZ5G7YeMDd59fAoL0JrResukyyf2yjbqN265zeCtkQq0dYnDyh67xngnps530t
gytnYHsJjgE4l6HNTIr7GIKZPSiitPpeu1uKZE22nw1U0xeTQ+4e   1876567101   1103409306
Sender: A11
Message: 2013/08/02 1258210903   A11     Temp    23.6    Celsius
L10: LogEntryType
qvEqamktwH+Nb14ifwU+xP+kSnGRLmQo+CRAWu/43ZORfnycCM1tSoclJh8lMHa8PPQUWFbYUHxx
```

```
foEIYTN3+GnVpTIZbPQMwpDmDa1XVZ2cAx6D1m6xFk1MriYuK3I8   -1479660103   -
1610056578
Sender: A11
Message: 2013/08/02 1258250914   A11     Temp    22.2    Celsius
L11: LogEntryType
48ncJi8fWIgrvECWrUWBUEIMmrqTfdHIal5oTRGiXbmLbpC7bWobnnEGce8ri1fDe3W4xWIKNR5I
xEzXwDosMaDqgKr7c8aD4pY45nXipdI0Qgdn148asq/htmYlY0WD   -1746053275   569341126
Sender: A11
Message: exit
L12: NormalCloseMessage
LH2gGm7jOeHMFgQLjl7L+dfTDYAQ1tr76Yqbe3NVnAfMLrK/ZWPyZedvGEQs13VS   -1712145935
1034890612
```

Log Verification at TS

The attack performed at the log file was deleting the last 3 entries, i.e. L10, L11, L12.

```
Verification of : LogFile.txt
L[0]:
LogfileInitializationType
+ukhe3ocWUwZ5KyeEXFPeOiPS3A334mK1ihubDkdamkMjgPI8MUay6q5aHn+LHViz/pCzuamg7Mf
OBNR85Na6uIvFNsgb78MeyFqBZ94GcfWiR4iEi0tD37rQYZLx48ObPbPI+DU4uuXILd64/0N5Gdp
dQnupD/LfdldL423EtOaN6R/Vw5gh18O4grJJCMhcWX8hzUm768ArGZw/oftr1yLp/L6wScWOXWI
tve8hCUQf8+9LeAXh8bNIWpD/WVeSxBitrI5afVXHoTLF6Yg725ZWbnY9P3kP+C6MeQVw6n5UVbX
+69zkGX1MlRb9ZWjEQDGwyUMuv8lkPV4pePzP3EcEBQ0iZkaqkwv58R5GbULM0OVNlFBGc13gxAa
BSxpEvRrr1Y5Y4rYhI/rrw8hl96akx9wesjboA84l/xTx+efahM1B+a+Ob66AbAWloMoqzwXZFFk
mv3r+J/e1Y4Jn2TmhkYMuPtEmDxv2qdqQUVfPfnfBkhYtOtIZXObI9hUjybT6uWcf5TJ6e4zqUEh
UaCQl7PEIaNOUXJ2xoSlT6/Doc2IN3iuosm4xdvYkxRO -831566549 604785157
Decrypted_AESmsg   D[0]:   LogfileInitializationType   2013/08/02   1257270689
2013/08/02    1320000000    0    0           4444          k_—GB³_Øp´ÑÙ#__=[n
‰_«Ô_”:óà?7Õ5R_zìš¼;_@_~_í´e
vÄØbO§d_ú_š<Í&é€|%5<¡Ö;®a_Ÿ__µÄu_@’_ò<•_˜Ì@@Éç_Ž?Û__æ…w÷_4²ÿÌè³W
_¡__5bVof‘•×¡H_
X3WUiYq8PiiPLLzJuuuoKgaqWlym7S21LI5rjFy9beTLth0Uxf6I68ZsoYVwz6lXSXLEzZCTC4BL
2/CeA2HeSnGitC+Nk8Fau3Lw2p9VUUYtxwLjNRCUT48UwMf5FyFGwA993sakWJrSfXZ3MTXwmqKz
KisJEubjtrKxhYDA6KbL7JMwVqPerFCbDWNckMFPTTcGT4/7513hcC+0L9Zd3fTvBLsllwWgB68w
XDY5yt4=
M[0]: LogfileInitializationType 1320000000 0 0    4444    k_—GB³_Øp´ÑÙ#__=[n
‰_«Ô_”:óà?7Õ5R_zìš¼;_@_~_í´e
vÄØbO§d_ú_š<Í&é€|%5<¡Ö;®a_Ÿ__µÄu_@’_ò<•_˜Ì@@Éç_Ž?Û__æ…w÷_4²ÿÌè³W
_¡__5bVof‘•×¡H_
X3WUiYq8PiiPLLzJuuuoKgaqWlym7S21LI5rjFy9beTLth0Uxf6I68ZsoYVwz6lXSXLEzZCTC4BL
2/CeA2HeSnGitC+Nk8Fau3Lw2p9VUUYtxwLjNRCUT48UwMf5FyFGwA993sakWJrSfXZ3MTXwmqKz
KisJEubjtrKxhYDA6KbL7JMwVqPerFCbDWNckMFPTTcGT4/7513hcC+0L9Zd3fTvBLsllwWgB68w
XDY5yt4=
Yi: -831566549
Yi2: -831566549
Y[0] = hash(Yi-1 ,AES(Di)_Ki, Wi) is verified
mac: 604785157
mac2: 604785157
MAC[0] = MAC(Yi)_Ai  is verified

L[1]:
ResponseMessageType
bJzxDJZqto455UtT/0TZ4q1rQSUjaFTm8Bp5KzVDQ1ErbZurZKKYd3FDqBG6lJaeciEnLBJHtfxj
LBP5Sj5qWh8r5vkwpj2I72Hxk24v8+k60JCLUaMYqfneBNjirfvM91CAKdJXbgZDcL0fUgKWc/qX
zEX1w4mq4Fgs4rH9ZBf+XZqE1voirBWprRhaFHxg1Xl0b1oFdulbKs0FHW9Em++e49grM1fQEY/m
bh6WszvuWD73mFk0n/jBX2k2+SMgJqSgLaowNQe5EAAA2qpyBPPrMFEl3KDMSUJ0jMdZVCbS3nRI
M/tX0V8BbjIr2Fv/HTcyeeLZz/VuHUhPXq9xaeE/xQPkHpsginwkunhR98NkMrfBqgxVYqEQf5XA
```

```
LqiG6nxXoAtg5cC3UlGSSIwu3pIsVeUs9jk7CwXDhYjFsKY+Uw/M2wY829+X/SWFwKoWlm/1a6S3
B/C96AA5SJyrHv3c8PIHjfo5kIJViWEv4q8VRYTnLZxlclSajSdaCDVPt3GraS0R+PByGtSsuFWe
QDKply6Nehkg6A1DcvxWF2c= -1930519867 -400813681
Decrypted_AESmsg D[1]: ResponseMessageType 2013/08/02 1257270705 2013/08/02
1320000000 1 1 1234
:[Š¹Ã_Ö°JõµêU=_kž!nÔ_:U__]_âÓØ]niþ&¾1£Žë9èwÊ†^_Hë__§ã?tòÐÔv\!œPž¼àtv_*U|Ò6_•F;
Ë@/ã™É__ptP_Q(á{¹ûú¸üã ªb<ÜðRéÝaX_zbŸr|¼Ó¸/úÎ_YÇ³Çz
X9BrOi1y+dobkbqTCJbR9XFPZlfowhujkdDl1k138Qvwm/tFLCa/FmHt4oh6OS8urUcHZdST3wKZ
jUuuwLb1GxCE5pR5xufTY4osFvrBssFQn8jHcvsTDGJwIpa81GMRj3fztfOIlyYrUA460dviwaG6
OlICVs+9NbsExj7hxRQmszSkT8NDIsX0UJeluVdkxKvH5U/3u7aIgNYA0EAeNg==
M[1]: ResponseMessageType 1320000000 1 1 1234
:[Š¹Ã_Ö°JõµêU=_kž!nÔ_:U__]_âÓØ]niþ&¾1£Žë9èwÊ†^_Hë__§ã?tòÐÔv\!œPž¼àtv_*U|Ò6_•F;
Ë@/ã™É__ptP_Q(á{¹ûú¸üã ªb<ÜðRéÝaX_zbŸr|¼Ó¸/úÎ_YÇ³Çz
X9BrOi1y+dobkbqTCJbR9XFPZlfowhujkdDl1k138Qvwm/tFLCa/FmHt4oh6OS8urUcHZdST3wKZ
jUuuwLb1GxCE5pR5xufTY4osFvrBssFQn8jHcvsTDGJwIpa81GMRj3fztfOIlyYrUA460dviwaG6
OlICVs+9NbsExj7hxRQmszSkT8NDIsX0UJeluVdkxKvH5U/3u7aIgNYA0EAeNg==
Yi: -1930519867
Yi2: -1930519867
Y[1] = hash(Yi-1 ,AES(Di)_Ki, Wi) is verified
mac: -400813681
mac2: -400813681
MAC[1] = MAC(Yi)_Ai is verified

L[2]:
LogEntryType
9nSXW92jbZUBK9bDKdw31Ef2xKpc48rx4pcSuA4CP5cPWib430CW2l6enYYXvDCxHD3CyBqn+ENI
iv4XD/h57M4T53wlMMjcGCmlSyChApT7E1L8UDbzYXFLtaD7Xvn9 -1671185904 1902346479
Decrypted_AESmsg D[2]: LogEntryType 2013/08/02 1257530830 2013/08/02
1320000000 2 2013/08/02 1257530815 A11 Temp 20.0 Celsius
M[2]: LogEntryType 1320000000 2 2013/08/02 1257530815 A11 Temp 20.0
Celsius
Yi: -1671185904
Yi2: -1671185904
Y[2] = hash(Yi-1 ,AES(Di)_Ki, Wi) is verified
mac: 1902346479
mac2: 1902346479
MAC[2] = MAC(Yi)_Ai is verified

L[3]:
LogEntryType
BW23Fto4md697G2onEoK0ZjNwRiGmi6jy2sK3fFStyxNKriRdvBtM6tpzsS0Gw9L76eEaarQQRUQ
7JNgxCAq0WobjtAUa129PQGRk/cgRdJoeRqDtXa1SwSm7X1lc71t 1374022861 -2010687340
Decrypted_AESmsg D[3]: LogEntryType 2013/08/02 1257570828 2013/08/02
1320000000 3 2013/08/02 1257570828 A11 Temp 20.4 Celsius
M[3]: LogEntryType 1320000000 3 2013/08/02 1257570828 A11 Temp 20.4
Celsius
Yi: 1374022861
Yi2: 1374022861
Y[3] = hash(Yi-1 ,AES(Di)_Ki, Wi) is verified
mac: -2010687340
mac2: -2010687340
MAC[3] = MAC(Yi)_Ai is verified

L[4]:
LogEntryType
ccleLTCC8yJuq5RIF+IOjdjKIWfdUDNw/eIWNt3jCqYXus/MWvNHTMFOE/UIMng0SRabLu0IL73I
1KlzGedPdiVYeH07Yyy4vxNSL/ntmVRgVBhAB6RQZTOnzdpYGIv1 11442941 -820787939
Decrypted_AESmsg D[4]: LogEntryType 2013/08/02 1258010839 2013/08/02
1320000000 4 2013/08/02 1258010839 A11 Temp 21.0 Celsius
```

```
M[4]: LogEntryType 1320000000 4 2013/08/02 1258010839    A11     Temp    21.0
        Celsius
Yi: 11442941
Yi2: 11442941
Y[4] = hash(Yi-1 ,AES(Di)_Ki, Wi) is verified
mac: -820787939
mac2: -820787939
MAC[4] = MAC(Yi)_Ai  is verified

L[5]:
LogEntryType
LlEWZFdoMrkp88d5OIIR2SxZp6fTCGFO6wu6TWr8Qbi7GdvzokUYNXkKIRCpM5AS9ko8Y8LJ5fBJ
h4xa4v+/2HMcxHE1zqvP3iMpzcS1jAt96tNxyUa/HkdKjEs95yFc -1940249984 -304847547
Decrypted_AESmsg   D[5]:   LogEntryType   2013/08/02   1258050853   2013/08/02
1320000000 5 2013/08/02 1258050853       A11     Temp    22.0    Celsius
M[5]: LogEntryType 1320000000 5 2013/08/02 1258050853    A11     Temp    22.0
        Celsius
Yi: -1940249984
Yi2: -1940249984
Y[5] = hash(Yi-1 ,AES(Di)_Ki, Wi) is verified
mac: -304847547
mac2: -304847547
MAC[5] = MAC(Yi)_Ai  is verified

L[6]:
LogEntryType
0TcHf50+eMAwVl1j0u12xaLisIIcDYQUkJtS0eJfYgdqXjdjxxpHqfRBeyhajt+eTYHEQNg0xuaX
+xkjzdYeuIuZa4ea8/yHCXAa/5vB9NkANbyoZcuGcne0ieaFV3nG -2001192187 -745038720
Decrypted_AESmsg   D[6]:   LogEntryType   2013/08/02   1258090864   2013/08/02
1320000000 6 2013/08/02 1258090864       A11     Temp    24.0    Celsius
M[6]: LogEntryType 1320000000 6 2013/08/02 1258090864    A11     Temp    24.0
        Celsius
Yi: -2001192187
Yi2: -2001192187
Y[6] = hash(Yi-1 ,AES(Di)_Ki, Wi) is verified
mac: -745038720
mac2: -745038720
MAC[6] = MAC(Yi)_Ai  is verified

L[7]:
LogEntryType
Xe6ykapmHGzb4FEdcRUklv44JOkTw7PRxwyMREoTQ1DU1mNpaRPZVP2SjNwpEf87OevdDr3O0VXX
k+h9xrK67myJbSSTUeyA8j4AeMiYD0jb9Wvl7roycsno4srAfSzR 1637110083 -414528027
Decrypted_AESmsg   D[7]:   LogEntryType   2013/08/02   1258130877   2013/08/02
1320000000 7 2013/08/02 1258130877       A11     Temp    25.4    Celsius
M[7]: LogEntryType 1320000000 7 2013/08/02 1258130877    A11     Temp    25.4
        Celsius
Yi: 1637110083
Yi2: 1637110083
Y[7] = hash(Yi-1 ,AES(Di)_Ki, Wi) is verified
mac: -414528027
mac2: -414528027
MAC[7] = MAC(Yi)_Ai  is verified

L[8]:
LogEntryType
vkpEnnrl5n4KxdPW9vvnM+jhzRVRGmDCioqpASFAR+mUWjszio2miec30AfcbkLRnUcxlCd3XtW2
eLNaEqChlIRxHPYWo6imwh13CtFesYt6+9apM4gq1tUJK5vMEMNp4MpaoR80rUSfXY1XMzJDD7GF
JBfr9Rl/HmSXzXgc/Ds= 1188695447 216820858
```

```
Decrypted_AESmsg   D[8]:   LogEntryType   2013/08/02   1258130892   2013/08/02
1320000000 8 AlarmSystem: WARNING: 2013/08/02 1258130877 A11     Temp    25.4
        Celsius
M[8]: LogEntryType 1320000000 8 AlarmSystem: WARNING: 2013/08/02 1258130877
        A11     Temp    25.4    Celsius
Yi: 1188695447
Yi2: 1188695447
Y[8] = hash(Yi-1 ,AES(Di)_Ki, Wi) is verified
mac: 216820858
mac2: 216820858
MAC[8] = MAC(Yi)_Ai  is verified

L[9]:
LogEntryType
T4S41aum+oLO1suZ5G7YeMDd59fAoL0JrResukyyf2yjbqN265zeCtkQq0dYnDyh67xngnps530t
gytnYHsJjgE4l6HNTIr7GIKZPSiitPpeu1uKZE22nw1U0xeTQ+4e 1876567101 1103409306
Decrypted_AESmsg   D[9]:   LogEntryType   2013/08/02   1258170890   2013/08/02
1320000000 9 2013/08/02 1258170890       A11     Temp    24.0    Celsius
M[9]: LogEntryType 1320000000 9 2013/08/02 1258170890    A11     Temp    24.0
        Celsius
Yi: 1876567101
Yi2: 1876567101
Y[9] = hash(Yi-1 ,AES(Di)_Ki, Wi) is verified
mac: 1103409306
mac2: 1103409306
MAC[9] = MAC(Yi)_Ai  is verified

In file LogFile.txt the MACs in following entry positions were found false
```

Signatures Verification at RA

```
Singature1 entry[0] is verified
Manual Signatures verification: true
Singature2 entry[0] is verified
Singature2 entry[1] is verified
Singature2 entry[2] is verified
Singature2 entry[3] is verified
Singature2 entry[4] is verified
Singature2 entry[5] is verified
Singature2 entry[6] is verified
Singature2 entry[7] is verified
Singature2 entry[11] is verified
Singature2_random entry[12] is verified
Singature2_random entry[13] is verified
Singature2_random entry[14] is verified
Singature2_random entry[15] is verified
Singature2_random entry[26] is verified
Singature2_random entry[27] is verified
```