Conception and Implementation of a Secure Engineering and Key Exchange Mechanism for the Open Source PLC Beremiz using a Test Driven Approach

Muhammad Arifur Rahman

Course of Study: Master of Science Information Technology

Examiner: Prof. Dr. rer. nat. Dr. h. c. Kurt Rothermel
Supervisor: Dipl.-Ing. Markus Birkhold

Commenced: July 15, 2015
Completed: February 15, 2016

CR-Classification: E.3
Abstract

Computerized control systems play a vital role in modern critical infrastructure. These systems are designed to provide better functionality and performance without concerning of security. This leaves them extremely vulnerable to cyber attacks which may lead to serious consequences. Therefore, it is of utmost importance to analyze the vulnerabilities of such system to protect them against various threats. In this thesis, several vulnerabilities of the open source automation system Beremiz were analyzed while considering several attack vectors that may affect the control system. To resolve some of the existing flaws, a secure communication protocol and an authentication system was implemented. The total development process was done using Acceptance Test Driven Development method and was tested with an automated testing framework "Twister".
# Contents

1 Introduction .................................................. 8

2 Objective .................................................. 11
   2.1 Attacker Model ........................................ 11
   2.2 Functional Requirements ............................... 12
   2.3 Non- Functional Requirements ......................... 13
   2.4 Workflow ............................................... 13

3 Key Concepts .............................................. 15
   3.1 Beremiz .................................................. 15
   3.2 PGP (Pretty good privacy) ............................. 17
   3.3 Pyro (Python remote object) ........................... 19
   3.4 Secure Socket Layer (SSL) ............................ 20
   3.5 CA Certificates ......................................... 21
   3.6 Protection against stolen Certificates ............... 23
   3.7 Encrypting algorithm .................................. 24
   3.8 PYROSSL ............................................... 25
   3.9 Securing the communication link ..................... 26
   3.10 Analyzing Network Protocol ........................... 27

4 Agile software development ............................... 29
   4.1 Methodology selection .................................. 30
   4.2 Test Automation ........................................ 31
   4.3 Selecting an Open source Test Automation Framework 33
   4.4 Twister Testing ......................................... 34

5 System Design ............................................. 37
   5.1 Current communication protocol ....................... 37
   5.2 Design Approach ....................................... 38

6 Realization and Implementation ......................... 41
   6.1 Starting the runtime server with PYROSSL ........ 41
   6.2 Enabling Certificate Authority crosscheck in the IDE 42
# List of Figures

1.1 Block diagram of a control system ................................. 8
1.2 Security Zone Concept ............................................. 9
1.3 Intruder using same Ethernet link to gain access to the control program 10
1.4 Intruder blocking the operators access and operating the control system directly .................................................. 10

2.1 Work-flow of the entire thesis ....................................... 13

3.1 Beremiz: Open source multi-platform IDE .......................... 15
3.2 Connection between IDE and runtime environment. ................. 17
3.3 Integrated Graphical Environment. ................................ 18
3.4 PGP encryption process ............................................. 18
3.5 PGP decryption process ............................................. 19
3.6 PYRO concept. ....................................................... 20
3.7 SSL Interaction between Client and Server .......................... 21
3.8 Certificate generation process. .................................... 22
3.9 Topology of Certificates. .......................................... 23
3.10 Setup for encrypted connection. ................................... 26
3.11 Connection setup where server requires the client to have authentic certificates. ........................................... 27
3.12 Connection setup where server checks the fields of client’s certificates. ............................................... 27
3.13 Wireshark interface showing captured data. .......................... 28

5.1 IP Address. .......................................................... 37
5.2 Runtime Server started. .............................................. 38
5.3 Configuring the host device. ......................................... 38

6.1 Prompt for passphrase in the terminal of runtime server. .......... 42
6.2 Runtime server started with PYROSSL ............................. 42
6.3 Shortcut button to create SSL certificates automatically. ...... 45
6.4 Notification showing that certificates already exist. ................ 45
6.5 Popup window for the user to input the certificate details. ........ 46
6.6 Notification of Certificate generated. ................................ 46
6.7 Configuration of the connection properties for SSL connection 46
<table>
<thead>
<tr>
<th>Section</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>6.8</td>
<td>Prompt for passphrase in the terminal of IDE.</td>
<td>47</td>
</tr>
<tr>
<td>6.9</td>
<td>SSL connection established.</td>
<td>47</td>
</tr>
<tr>
<td>6.10</td>
<td>Secure transmission of data.</td>
<td>47</td>
</tr>
<tr>
<td>6.11</td>
<td>Operational Flowchart.</td>
<td>48</td>
</tr>
<tr>
<td>6.12</td>
<td>Data captured and shown on screen via Wireshark.</td>
<td>49</td>
</tr>
<tr>
<td>6.13</td>
<td>Acceptance Test Driven Development (ATDD) cycle.</td>
<td>50</td>
</tr>
<tr>
<td>6.14</td>
<td>Configuration setup for the test.</td>
<td>52</td>
</tr>
<tr>
<td>6.15</td>
<td>Result of the test.</td>
<td>53</td>
</tr>
<tr>
<td>6.16</td>
<td>Detail log of the test.</td>
<td>53</td>
</tr>
</tbody>
</table>
List of Tables

4.1 Comparison of Automated Testing Tools. . . . . . . . . . . . . . . . . . . 34
1 Introduction

Computerized control systems play very important roles in modern production systems. Programmable logic controllers can be PC based or specially designed computer that are most commonly used as control system components. These controllers can be programmed using special languages under the regulation of IEC International Standard 61161-3 to perform almost any kind of tasks [1]. PLC systems can be used with motors and servo systems, providing complex multi-axis systems with advance and precise control in rotation, motion, acceleration, braking and so on. They are used in industrial electromechanical processes, for automated control of machinery in assembly lines, power substations and in almost every automation system. PLC systems are usually connectable to a computer by an Ethernet interface to interchange control and machine data between the devices. They use SCADA – Supervisory Control and Data Acquisition systems with the HMI - Human Machine Interfaces as a mean of communication between the operator and the control system. Anyone with the physical access to the HMI can operate the control system and get important feedback from the system like production parameters and rate and even with the proper control program can reconfigure the operating procedure of the control system whilst the systems are fully operational, figure 1.1.

![Figure 1.1: Block diagram of a control system](image)

A few decades ago, controls systems were safe from any external attacks because of the limited interconnection between other external systems and proprietary protocols which were only available from specific vendors and used with specified technology. They were only vulnerable to operators with limited knowledge of the system or who misused the system for personal benefit. But later control systems were introduced with Ethernet based protocol, utilizing non-proprietary technology with the capability of remote access and better connectivity, making them exposed to cyber threats [2]. Also, almost every PLCs are run by commercial operating systems like Microware OS, VxWorks which are vulnerable to bugs and security issues just as they are in other operating systems like
Microsoft Windows, Linux, Mac OS etc. Recently, several commercial PLCs were tested, which are run on these operating systems and almost all are found to be vulnerable to several bugs [3]. Bugs fixing and patching PLCs are difficult and different events like compatibility and real time operation need to be considered before updating a PLC system because of their complex natures. Thus patches for these vulnerabilities are released at a very slow rate compared to other commercial operating systems.

A recent attack by the Stuxnet worm sabotaged a uranium enrichment program which proves how severe the effects of these vulnerabilities can be. To protect a control system from unauthorized users, first the attacker's point of view should be taken into consideration. An intruder monitors the regular execution of the programs to duplicate the control program and reverse engineer the system [4]. So in order to increase the safety and security of a control system, unauthorized access to the HMI and PLC along with access to the control program must be prevented. To secure the control system, a widely used safety approach named as "Onion Approach" as shown in figure 1.2 [5], can be taken to identify the risky zone by using a security zone concept.

The control system is usually placed in a secured place typically inside a building with possibly on site guards with physical access restriction e.g. electric cabinet with lock, thus safe from unauthorized access. The external zone of the system where network access is available are more vulnerable to cyber-attacks which is the main concern of this thesis.

The main weakness of the PLCs is that they are insecure by design. Any code written by the corresponding engineering system can run on a PLC system without the legitimacy of its origin. Commonly used open source tools for example Beremiz allows the PLCs to
load and execute organization blocks without authentication. An intruder can gather the necessary knowledge from the PLC’s documentation and source code and use the same Ethernet link to access the PLC (figure 1.3) or even block the programmer’s access to the system and control the PLC directly using a completely different logic (figure 1.4) since there is no authentication or authorization, which could lead to serious consequences.

For these reasons, a built in security system is necessary in PLC, which will restrict external access to the control program and allow authentication of users and authorize access control at the very least.
2 Objective

To resolve some of the existing flaws, Institut für Steuerungstechnik der Werkzeugmaschinen und Fertigungseinrichtungen (ISW) has developed a secure access mechanism based on an asymmetric encryption method called Pretty Good Privacy (PGP) and button enhanced device authentication to secure the open source PLC Beremiz [6].

The objective of the work is to review the current security mechanism and develop an integration strategy into the PYROSSL framework thus restricting external access to the control program.

In addition, currently missing features such as authentication and secure data communication should be implemented such that ease of use and overall functionality are improved. To be able to commit the work to the open source project the implemented mechanism must be testable and modular. The PYROSSL framework allows for such implementation. In addition the whole project should be implemented with the test based development paradigm, such that outsiders can test function without the necessity to understand the underlying functions and source code. All functions, methods, etc. must be testable and have a corresponding testing function. As Beremiz is mainly implemented in python, the used language will be Python 2.7, if possible automated testing tools should be used for functional testing. For authentication and authorization PGP should be used. The engineering system must run on a Linux system in this case Ubuntu, the runtime must run on a BananaPi embedded system.

2.1 Attacker Model

The new secure communication and authorization will be basis for further development of secure access mechanisms. The PLC will run on a local system, there can be one or more engineering systems with access to the control system. There will be no direct physical access of attackers to the control system, assuming that the system is well protected with guard and secured in a safe location with lock, therefore all attack vectors based on physical access can be neglected. The only valid attack vector is via the network interface, e.g. a logical attack vector either through direct contact by compromising a maintenance worker or by establishing a direct link over the network. To explain
the valid attack vector, imagine a scenario where a technician is working from home. His/her laptop may have proper antivirus and firewall system installed, but that doesn't restrict the access to emails and web pages. Any one of those emails or webpages may contain specially written malware by an attacker who wants to sabotage a particular system the technician has access to. Now, as the technician has clicked on the malware, it will run on the background. The antivirus failed to identify the malware as a threat because it may only be a couple lines of code which was only made for this purpose and therefore no previous antivirus signature of this malware.

Now when the technician uses this compromised laptop to access the control system, the malware activates. The technician may or may not use a VPN system and proper authentication protocol. The malware will wait till the login process completes, then starts sending keystrokes and mouse movements and may be the screenshots of the active windows to the attacker. The VPN installed in the laptop may be configured to protect against such 'split tunneling', but the malware has access to the raw network, thus it can send anything it was programmed to send to the attacker[7].

Another attack vector can be explained using the recent attack on automation system by the worm "Stuxnet". This worm enters to a system using a usb device and infects all the machines running under Microsoft Windows. It contains forged certificates of some reliable company, thus avoids getting detected by automated detection systems. Then it checks whether the system is connected to the Siemens Step7 control system. If it is not the target, the worm does nothing, but if it is the targeted device, the worm connects to the internet and updates itself to the latest version. Then it monitors the control operations of the system and then uses this knowledge to take control of the centrifuges. Then it disables the feedback warning system to outside controllers and spins the centrifuges to failure[8].

This thesis will focus on the 'Man in the middle attack', not on compromised endpoints. The following subsections will focus on the functional and non-functional requirements and a top view of the entire workflow of of the thesis.

2.2 Functional Requirements

The communication between the computer and Beremiz is established via PYROSSL. The authorization of the communication has to be done by asymmetric encryption. The control program has to be encrypted and signed. The initial access can only be authorized by physical access to the control system. Access to a third party system can be granted by the means of an already authenticated system.
2.3 Non-Functional Requirements

All the functions implemented in this program have to be testable. All implemented functions should have corresponding functional tests. The access mechanism has to be modularized such that all used libraries can be exchanged. All used functions or libraries must be open source. All implemented functions must be thoroughly documented.

2.4 Workflow

The entire workflow of the thesis will be carried out according to the following flow chart (figure 2.1):

![Workflow Chart]

**Figure 2.1:** Work-flow of the entire thesis

In the first step, the open source automation software "Beremiz" and its existing unsecured communication protocol will be analyzed. At the same time, the possible ways to encrypt the communication and the appropriate testing approach will be determined. Then the embedded system, "Banana Pi" which will act as the PLC system, several open
source libraries for encryption and the testing frameworks will be analyzed. With the knowledge of these researches, the system will be designed, implemented and tested. After the implementation, the code will be optimized and checked for bugs. And then finally the project will be documented.

In the next chapter the automation software, its communication protocol, the encryption methods and the testing approach will be explained and illustrated in details.
3  Key Concepts

3.1 Beremiz

To introduce a common framework for the control system, IEC 61131-3 was introduced which defines the logical programming blocks and languages of industrial controllers. It was slowly accepted and implemented by the vendors, but due to different implementation and different file formats it was hardly possible to transfer programs between different vendors. [9]. The automation sector, mainly consists of commercial software, thus their operating safety could not be verified as their source code were closed. And also the commercial PLC programs were too expensive, so the usage was limited only to workstations and laboratory works. Beremiz was introduced with the intention to solve the above mentioned problems.

Beremiz is a multi-platform integrated development environment (IDE) providing automation, control and HMI for machine automation along with an embedded runtime environment.

Figure 3.1: Beremiz: Open source multi-platform IDE

[9]
Beremiz software. It is an open source software distributed under the GNU license, conforming to IEC-61131 among other standards like PLCOpen and CanOpen. It is independent of the targeted device and can be installed in any operating system. It relies on four major sub projects (figure 3.1):

• PLCOpen Editor,
• MatPLC’s IEC compiler,
• CanFestival and
• SVGUI.

The PLCOpen Editor is used to write PLC programs according to the IEC 61131-3 standard and the MarIEC compiler converts these into ANSI C codes. The most important feature of Beremiz in this case is the ability of plugin extensions. Because of its open source policy and plugin structure, new functions can be added to the environment easily by implementing corresponding class definitions [9].

As previously mentioned, Beremiz has:

• Integrated Development Environment
• Runtime environment

Figure 3.2 shows the connection of Beremiz with its runtime environment. The IDE runs on the host machine which is usually a conventional computer and the runtime environment which is usually run on an embedded computer and act as the PLC device. The host can communicate with the target device, regardless of the hardware or operating systems installed in them. The host can communicate and transmit data with the target using an Ethernet connection.

**Integrated Development Environment**: A development environment which is basically a graphical user interface (figure 3.3) is written in python and strongly linked with PLCOpen specification. It is run by executing Beremiz.py and can be used to create projects according to IEC 61131-3 standards program organization units (POUs). The POUs can be programmed in all the five languages defined by IEC 61131-3 standard:

• Sequential Function Chart (SFC)
• Function Block Diagram (FBD)
• Ladder Diagram (LD)
• Structured Text (ST, equivalent to C/C++)
• Instruction List (IL, equivalent to assembler)
The programs written by this GUI are stored in XML format which can be used with any other IEC 61131-3 editor that follows PLCOpen standard[9].

**Runtime environment**: A runtime environment runs on the target device by executing Beremiz_service.py. This environment can execute the code that was compiled by Beremiz.py and transmitted to it via network. When the runtime environment is running on the target device and published on the network, it can be reached by its IP address and anyone can access and operate the device by connecting to it.

### 3.2 PGP (Pretty good privacy)

PGP was introduced with the aim of confidentiality and authentication, which uses both the symmetric key and asymmetric key (also known as public key) to encrypt and decrypt something. A KEY is used by PGP to generate encrypted text which is called
3.2 PGP (Pretty good privacy)

ciphertext. The Keys are of two types: Private key and Public key. These keys are related to each other. If something is encrypted using a Public key, it can only be decrypted by the corresponding Private Key. In PGP, at first the text is encrypted using a session key. This session key uses a symmetric encryption algorithm to encrypt the plaintext producing ciphertext for faster encryption and decryption and less overhead. Once the ciphertext is created, the session key is encrypted using the recipient’s public key which is then transmitted with the ciphertext to the recipient (figure 3.4)[11].

*Figure 3.3: Integrated Graphical Environment.*

*Figure 3.4: PGP encryption process*
The decryption of the ciphertext works in the opposite way. The recipient PGP uses the private key to recover the one time usable session key, and uses it to decrypt the previously encrypted ciphertext (figure 3.5).

The combination of these encryption and decryption methods completes the PGP – Pretty Good Privacy encryption protocol [11].

This mechanism was used by Zhefeng Li on April 2015, on top of the insecure Pyro - Python RPC(Remote Procedure Call) which is a client/server intercommunication technology using the library pyro (python remote object) by which objects can talk to each other over the network, with minimal programming effort [6].

3.3 Pyro (Python remote object)

Python remote object is a library written completely in Python that enable users to build a communication protocol by which objects can communicate with each other over the network. The users only have to write the python objects in a standard way and then by adding only a few lines of codes Pyro completes the communications between the objects all over the different machines on the network. The remote object over a network can be called in the same way as the local object.

Pyro enables remotely calling an object on different servers. For example, an object "russo" registered in the pyro daemon of server B can be called from server A.

Important concepts regarding PYRO[12]

- **Pyro object**: This is a normal python object, but registered with Pyro thus can be accessed remotely. A class can also be used as a Pyro object, but then the user has to define how Pyro will create objects from them while handling remote calls.
3.4 Secure Socket Layer (SSL)

SSL is a standard encryption technology, which establishes an encrypted connection between two devices, typically a client and a server, thus ensuring security and privacy of the transmitted data.

SSL provides the following advantages[13]:

- Privacy (Encryption)
- Integrity (Data has not been altered during transmission)
- Trust (Verified User)

A SSL transaction starts with the client requesting SSL connection to the server by sending a handshake. Server responds to this request with its certificate that contains its asymmetric public key and some other information like the validity date, owner, and the domain or ip address of the server.
The client then creates a symmetric session key and encrypts it with server’s public key and sends it to Server. Server decrypts the encrypted session key using its asymmetric private key, thus both parties have the symmetric session key which they can use to encrypt and decrypt the transmitted data. The whole process is shown in figure 3.7[14].

However, it is possible that an attacker intercept this communication and provide a certificate which may look legitimate (the domain or IP address of the certificate being the same as the server) thus establishing a connection to the targeted device and eavesdrop on the communication. To prevent this, there needs to be a way to validate the trustworthiness of the certificates. A third party can be involved here, which maintains a list of valid certificates. This third party is trusted by both the client and the server, and it signs all the valid certificates with its private key. This signature proves that the certificates are not forged and the data they represent are accurate.

The public key of the Certificate Authority is distributed to both the client and the server which they can use during the connection to check for the authority’s signature. If the signature is valid, the client and the server can be confident of the identity of each other[15].

3.5 CA Certificates

As previously mentioned, Certificate Authorities/CAs issue digital certificates that contain identification data which can be used for authentication of online identity. There are two ways to obtain CA certificates. Commercial CAs are pretty expensive, but are trusted by
almost every browser and device relying on them. But a local CA can also be generated and used which is ideal for services that will not be offered to the public.

To obtain a signed digital certificate, the user/server has to generate a private/public key pair and send the public key to the Certificate Authorities. When CA receives the request, it verifies user's identity and then signs the certificate using its private key and return it to the user (figure 3.8)[16].

![Certificate generation process](image)

**Figure 3.8**: Certificate generation process.

A digital certificate signed by a Certificate Authority contains the following information:

1. Name, domain/IP and other attributes of the user
2. Public key of the user
3. Issue and expiry date of the certificate
4. Identity of the Certificate Authority issued it
5. Digital signature of the Certificate Authority

To verify the digital signature on the certificate, the user/server need the public key of the certificate authority. The public key is usually distributed inside a certificate, so it is basically a certificate issued by the Certificate Authority and signed by its private key.

The private key used by the Certificate Authority to sign certificates is also called root key. Normally Certificate Authorities are in business and they issue a lot of certificates. Thus, if they only use root certificates, it puts a lot of exposure into it. If the root key is misplaced, or gets stolen, the result can be disastrous. The thieves will be able to supplant any certificates signed by this key, or can start making their own trusted certificates and sell them to black market. To deal with this threat, the best practice is
to keep this key offline and unused as much as possible. This is where Intermediate certificates come in. The topology of the CA certificates is shown in figure 3.9[17].

Intermediate certificate is issued by the root certificate specifically to issue end user certificates on its behalf, thus reduces the risk of compromise of the root certificate. When intermediate certificate is created, a certificate chain is also created that provides a chain of trust. When an application wants to verify a certificate signed by an intermediate certificate, it also has to verify the intermediate certificate against the root certificate. This verification can be done using the certificate chain.

Every time the root certificate signs the intermediate certificate or the intermediate certificate signs an end user certificate, a certificate revocation list is generated. Whenever a certificate gets stolen, the issuer can revoke the certificate and the thief will not be able to use the certificate again.

3.6 Protection against stolen Certificates

The process of using SSL is safe as long as the certificates are not lost or stolen. If in any case the private key gets compromised, the root certificate can still be used to revoke
the certificates generated using this key. But to add an extra level of security, the keys can be protected by a passphrase using the DES3 encryption. If any keys get stolen, the thief will not be able to use it to authenticate himself because the server will ask for the password. All the key files, even the root key file can be password protected, thus if compromised, nobody will be able to use the root key to sign any other certificates without knowing the password.

3.7 Encrypting algorithm

3.7.1 Asymmetric Encryption

Encryption algorithms like RSA, ECC are used for asymmetric encryption. Usually public keys are generated and distributed because it is mathematically impossible to recreate the private key from the public key.

**RSA**: RSA is the most widely used asymmetric algorithm. It is based on the difficulty of factoring the product of two large prime numbers known as the factoring problem. A public key is created using two large prime numbers with an auxiliary value. The public key can be used to encrypt something, but to decrypt data encrypted with the public key, one has to know the prime numbers. As an example, user A can encrypt a message using user B’s public key without knowing his private key. User B can decrypt the message as only he has the private key. RSA also used to sign a message, user A can sign the message using his private key and user B can verify that message came from user A using A’s public key. But as RSA is a comparatively slow algorithm, it is not used directly to encrypt something, rather used with symmetric key encryption to optimize speed[18].

3.7.2 Symmetric Encryption

Encryption algorithm like Twofish, AES or Blowfish are used for symmetric key algorithm, but AES is the most commonly used one.

**AES**: AES is based on the combination of substitution and permutation which is known as substitution - permutation network. It is very fast in both hardware and software and uses block ciphers. It has a fixed block size of 128 bits and key size of 128, 192 or 256 bits and applies cryptographic key and algorithm to block of data as a group.
The AES algorithm starts with a random number. The data and the key are scrambled in it through four mathematical rounds: SubBytes, ShiftRows, MixColumns and AddRound-Key. In the SubBytes, a lookup table is used to determine which byte is replaced with which one. In the ShiftRows, the first row is left untouched, the second row is shifted to the left by one offset, the third row by two and the fourth row by three. But only for the case of 256bit AES, the third row is shifted by three and the fourth row is shifted by four. In the MixColoumns step, an invertible linear transformation is used to combine the four bytes in each column. In the AddRoundKey round, a round key is derived from Rijndael's key scheduler and this key is added to each byte of the state correspondingly. In the final round, all the steps except the MixColumns step are repeated again[19].

3.8 PYROSSL

To integrate SSL connection is Pyro, an extension to the normal validator is used which is Pyro.protocol.BasicSSLValidator. This validator checks whether a client has supplied a SSL certificate or not. Python normally does not support SSL, so the following libraries have to be used with python to enable this feature[12].

- M2Crypto
- OpenSSL (needed by M2Crypto)

M2Crypto: M2Crypto is a python wrapper for OpenSSL, which features RSA, DSA, DH, HMACs, message digests and symmetric ciphers (including AES). It provides SSL functionality for both client and server applications, and is thread safe.

Pyro daemon needs to be modified such way that it uses SSL instead of regular sockets. This can be done by passing a specific parameter while creating the daemon:

d = Pyro.core.Daemon(prtcol='PYROSSL')

When this daemon is modified like this, the pyro objects connected to this daemon will be registered to the nameserver using the PYROSSL protocol that will tell pyro to use SSL instead of regular sockets. A special SSL connection validator mentioned earlier can be added to the daemon which will check for client certificates. Pyro will automatically deal with the SSL if everything is set accordingly, thus the client programs can be left unchanged.[12].
3.9 Securing the communication link

There are several ways to secure a communication line. The user can choose the one which suits his/her security needs [17].

- Encryption of the communication path
- Server requires the Client to have certificates signed off by known CAs
- Server requires the Client to have certificates signed off by known CAs and checks its fields

**Encrypted connection:** This is a very basic setup which only encrypts the communication. In this setup, the server has a private key and a certificate and the client has a known CAs list. The client will only connect to the server if the server’s certificate is signed by one of its known CAs (figure 3.10) [17]. In this way the client doesn’t need to have a certificate.

![Diagram of encrypted connection](image)

**Figure 3.10:** Setup for encrypted connection.

**Server checks Client’s certificate:** This procedure is same as the above with an extra level of security. In this setup, server and client both checks each others certificates against their known list of CAs (figure 3.9). If this validates, a secure connection is established [17].

**Server checks the fields in Client’s certificate:** This procedure is same as the previous method with another level of security. In this process, server checks the fields of client’s certificates and if there is any mismatch with its configuration the connection will not be established (figure 3.12) [17].
3.10 Analyzing Network Protocol

After the implementation of SSL, the communication link must be checked to verify whether it is truly encrypted or not. There are several network analyzer which can be used to check this. For example: Wireshark, Tcpdump, Netcat, Ettercap, etc. For this thesis Wireshark will be used.

**Wireshark:** Wireshark is one of the most widely used network protocol analyzer. It can be used to capture and analyze the traffic running on a network. It has a very rich and powerful feature set, some of them are given below[20]:

- Protocols can be inspected in very deep level.
- Pakets can be captured in real time and can be analyzed later or even offline.
• Cross platform support. Works on Windows, linux, OS X, FreeBSD and almost every-others.
• User friendly Graphical User Interface which can be used to browse captured traffic.
• Contains one of the most powerful display filters.
• Supports various capture file formats.
• Can be used to decrypt protocols.
• Results can be exported in XML, CSV or in plain text.

Figure 3.13 shows the interface of Wireshark. When the capture command is given, it captures all the transmitted data and shows on the screen. Using this software it is possible to verify whether the transmitted data is actually encrypted or not.

Figure 3.13: Wireshark interface showing captured data.
4 Agile software development

Agile software development is a set of software development methodology in which the solutions and specifications evolve throughout the development cycle till the delivery of the finalized product. This methodology focuses on frequent inspection, continuous improvements and adaptation through close collaboration of the development team and the clients.

In February 2001, the Agile Manifesto was published with the aim of "Uncovering better ways of developing software by doing it and helping others do it.", which is based on twelve principles[21]:

- Provide continuous and fast delivery of the developed application to gain customer satisfaction.
- Accept change in specifications throughout the development cycle.
- Deliver functional bits of the developing application to the customer as soon as they are ready usually within weeks.
- Close collaboration between developer and the clients.
- Highly motivated individuals are charged with the development task.
- Direct communication between parties, usually face to face.
- Progress of the development is measured by developing software
- Constant pace
- Higher priority on design and technical excellence.
- Simplicity - the art of maximizing the amount of work not done -is essential.
- Self-organizing teams
- Continuous adaptation according to the changing specifications and requirements.
There are several agile development methods, all of which provides developments and adaptation throughout the development cycle with close collaboration of self-organizing teams. The most commonly used methods are TDD, ATDD and BDD\cite{22}\cite{23}\cite{24}.

**Test Driven Development (TDD):** TDD is a way of software development in which the user first writes the test file, run the test which will fail as there is nothing to test and then write the minimal code to pass the test. Then the code is refactored according to the requirements and again tested to check whether its working or not. Normally the main focus of the TDD is to test the low level functionality and unit testing, which enables easier refactoring and produces simpler code. The process is repeated until the required specifications are met.

**Acceptance Test Driven Development (ATDD):** TDD is a perfect way of testing whether the code is functioning or not, but it doesn’t check whether the written code was even required to begin with. ATDD is a process where the clients are included in the development phase before the coding starts. It is a collaboration of the developers, tester, owner and the clients, thus insures precise understanding of the requirements and the acceptance criteria so everyone knows exactly what needs to be implemented.

**Behavior Driven Development (BDD):** BDD is a combination of the principles of TDD and Domain-Driven Design. BDD focuses on the behavior of the system rather than the implementations. It guides the functionality developments according to the expected behavior. The purpose and usability of the code are described in the language of Domain-Driven Development and the native language of the developers (usually English) defined in GIVEN, WHEN and THEN (GWT) format.

### 4.1 Methodology selection

As mentioned earlier, TDD is a paradigm of writing a test first, then the code with optional refactoring if necessary. The problem with TDD is that it doesn’t guide the developer from where to begin or how it should be structured or whether the written code is even required or not.

ATDD involves clients in the beginning of the development process and focuses on the requirements to guide the development which allows a close and easier collaboration between all the involved parties.

BDD is mainly customer focused and provides a clear understanding of the system behavior to the developers from the customer points of view.
Although implementing and automating ATDD is a quite extensive work requiring high technical efforts from the developers, the benefits of applying ATDD in a project surpasses everything[22][23][24].

- ATDD focuses on client needs which is quite important in software development. Designing the functions becomes easier when the developer can see the "whole top down thing" from the beginning rather than the bottom up approach of TDD which deals with small portions of a code at one time.

- It is possible for a code to have full code coverage and pass all the unit tests with broken functionality. Or even a code can have complete functionality, but still doesn’t do what the client asked for. On the other hand, considering the acceptance tests are written accurately, if a code passes them all, it will definitely fulfill client needs.

- The clients and the developers are closely involved from the beginning of the development phase, eliminating all the possible chances of miscommunication or last minute changes because of lack of understanding of the end results.

4.2 Test Automation

4.2.1 General

In software testing, there are several special software which can be used to control and execute test cases and compare the actual results with the intended outcomes. This trend is called test automation. This procedure is used specially to automate the repetitive testing tasks in a software development process and helps with continuous testing and delivery of the developing software.

During software development, there are some testing tasks, for example: regression testing, is very time consuming to do manually, which is also not much effective to find certain defects. Test automation can be used in these cases to perform these types of testing effectively. Another advantage of using test automation is that, once the tests are automated, they can be run repeatedly and quickly, thus cost effective for testing of software with long development cycle.

Test Automation can automate the following types of tests[25]:

- Functional: Tests the functionality of the software against requirement.
4.2 Test Automation

- **Regression**: Tests whether the behavior of the system has been altered or not.
- **Stress**: Test the absolute stress capabilities of the system.
- **Exception**: Forces error conditions in the system.
- **Performance**: Tests the performance of the system to see whether it is adequate enough according to project requirements.
- **Load**: Determines the maximum load point after which the performance of the software degrades.

### 4.2.2 Test Automation Framework

A "Test Automation Framework" is an integrated system that provides an automated execution environment for the test scripts and simplifies the automation effort. Function libraries, test data sources and reusable modules are integrated in this system which act as small building blocks that are assembled as a process. The main advantages of using a testing framework are listed below[26][27][28]:

- **Reusability and Reliability**: One of the best aspects of Testing framework is that the test suites are reusable. The same test cases can be used for multiple tests and on multiple platform. In test framework, same tests perform the same operations every time they run therefore eliminates human error.

- **Full coverage**: Automated testing increases the depth and scope of the test therefore increasing the quality of the developing software. Tests can be built that cover every feature of the developing software and can be run automatically and simultaneously on multiple device with different configurations without any manual intervention. Thousands of complex test cases can be executed at once, which is not possible in manual test.

- **Cost effective**: During the development period, a software has to be tested frequently with every update and after every patch. Manually performing this repetitive tests are very costly. Test automation reduces these costs of repetitive tests as the same test can be run multiple times and simultaneously thus saving a lot of time.

- **Minimal manual intervention**: Once the test cases are written and the test suites are set up, no manual intervention is necessary to perform the test.
4.3 Selecting an Open source Test Automation Framework

- Easy reporting: In test framework, the reporting has become very easy. The reports can be generated and stored on the server or even sent to the user via email after each test. The user does not have to be on location to check the results.

4.2.3 Selecting criteria

While selecting a test automation, there are some criteria that should be kept in mind [26][27][28]:

- Cross platform support: Test Automation enables the user to perform a thorough regression test throughout all the platforms with reusable test case repositories. Thus single script can be applied against multiple devices without recreating it specifically for each platform. This saves time and money. Without cross platform support, this is not possible.

- Reporting: Ease of reporting and its delivery to the user should be considered while selecting a test automation system.

- Debugging: Different testing tools have different debugging system. The framework with better debugging and analysis system helps in the long run.

- Stability: Often longer tests need to be performed in the framework, so the stability of the framework is crucial.

- Ease of Use: Ease of use of the testing framework reduces the learning time.

- Documentation: The more documentation available for the framework, the easier to operate it.

- Extension and plugin support: The framework should support plugin extension so that user can install or create the plugins he/she needs for the testing purpose.

4.3 Selecting an Open source Test Automation Framework

There are several Open source Test Automation Frameworks in the market and to select the most favorable framework, the most rated frameworks on the market were analyzed and compared against their technical features, usability, economical efficiency, ease of use and installation difficulty and the results are shown in the table below[29][30][31].

The comparison chart shows that Twister framework is more mature than the other frameworks mentioned above and beats them in almost every category. The main
4.4 Twister Testing

Twister is an Open Source test automation framework as described earlier. Before starting with Twister, there are several things which have to be kept in mind[32]:

- Applet: This is the graphical user interface of Twister Framework. This Applet is written in JAVA and can be accessed through a browser. Any user with access to it can create projects, tests, run, pause and stop the test executions, view the reports, add plugins and can configure it according to the requirements.

- Central Engine: Central engine is just a collection of services that twister offers. It is used to send libraries to user execution engines, generate and store the logs and reports and send these to the configured email address.

- Client Machine: The execution engines are installed in this machine.

<table>
<thead>
<tr>
<th></th>
<th>Twister</th>
<th>Robot Framework</th>
<th>STAF</th>
<th>Concordion</th>
<th>Cucumber</th>
</tr>
</thead>
<tbody>
<tr>
<td>Stability</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
</tr>
<tr>
<td>Configurability</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>○</td>
</tr>
<tr>
<td>Monitorability</td>
<td>●</td>
<td>○</td>
<td>●</td>
<td>○</td>
<td>○</td>
</tr>
<tr>
<td>Intelligibility</td>
<td>●</td>
<td>●</td>
<td>○</td>
<td>●</td>
<td>●</td>
</tr>
<tr>
<td>Third Party Libraries</td>
<td>○</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
</tr>
<tr>
<td>Low Maintenance Costs</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Reporting</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
</tr>
<tr>
<td>Test Cases Creation</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>○</td>
<td>●</td>
</tr>
<tr>
<td>Documentation</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
</tr>
</tbody>
</table>

*: Better ●: Moderate ○: Poor

Table 4.1: Comparison of Automated Testing Tools.

The strength of Twister is that it is completely configurable over the internet. It also supports a large collection of add-ons which can be added to the framework very easily.

Twister Framework complies with almost all the programming languages. It is a versatile, flexible, highly configurable and the monitoring of its operations is very easy. The usability of Twister framework is better than others. It has a web interface and a Java applet for creating, editing and running the test suites. Twister is very well documented and does not require much user expertise. The reports can be customized according to user needs very easily and though the installation procedure is a bit difficult, once set up, it is easily operable[29][30][31].
• Execution Engine (EE): This engine is user specific. It waits for the start signal and sends the logs to the central engine. When start signal is generated, it launches the Runner and when the stop signal is generated it instantly kills the Runner.

• Project Files: The project files are stored in XML format. It contains the test suites and the test files and can be run on multiple test beds.

• Reporting: It is used to view the results generated by the executed tests and can be fully customized.

• Web Interface: This central Engine web interface is used for debugging. The user can view the logs and statistics through this interface and can also turn the execution engines on or off.

• Runner: Runner connects to the central engine to gain access to the libraries, checks for dependencies and waits for them to finish, skips a test if there is skip command, starts the appropriate scriptrunner for specific programs.

• Server Machine: The central engine is installed in this machine.

• User Machine: The machine which can be used by an user to access the framework.

4.4.1 Key Features

The key features of Twister are given below[32][33]:

• Multi-User architecture: Supports multiple user groups with different roles.

• Web based GUI: User friendly web based graphical user interface.

• Realtime monitoring: Realtime monitor of the test execution is possible.

• Distributed execution: Every user has their own processes.

• Parallel testing of systems: System Under Tests(SUT) can be run in parallel.

• Auto generation of reports: Reports are generated automatically after each execution and can be forwarded to the configured email address.

• Scripting languages support: Supports almost all scripting languages.

• Plugin support: Plugins can be added to the system for more functionality.

• Advance statistics: Advance statistics can be generated for each test.

• Multiple logs: Multiple logs can be generated for different levels: debug log, test log, running log.
• Customizable delay: Delays can be added between tests.
• Global variable: Support global variables.
• Panic word support: If a Panic word is assigned and found in the command line interface logs, kills every process.
5 System Design

As previously mentioned, Beremiz communicates with its runway server using PYRO. This communication is performed without any encryption and authentication. For this reason this process is completely insecure and vulnerable to man in the middle attack. Before designing an encryption system, first the current communication system will be discussed.

5.1 Current communication protocol

To establish connection in current communication protocol, the first step is to run the runtime environment on the target device. This can be obtained by entering the following code in the terminal:

```bash
$ sudo python Beremiz_service.py -i XXX.XXX.XXX.XXX
```

Here, the XXX.XXX.XXX.XXX has to be replaced by the corresponding ip address. This ip address can be viewed by typing the following code in the terminal:

```bash
$ ip add
```

The ip address is the address marked red on the figure 5.1.

![Figure 5.1: IP Address.](image-url)
Then the runtime server will start and it will wait for the host (figure 5.2). The host can connect to the target by using the "Pyro object's uri".

![Figure 5.2: Runtime Server started.](image)

After that, in the host device, a project has to be created and then in the config tab of the project file, the URI_location (figure 5.3) has to be set in exactly as the ip address and the port of the target device (figure 5.2). There is a "Build" button which can be used to build the code. Then clicking the "Connect" button will establish a connection between the host and the runtime server. Then the user can transfer the compiled code it to the target device.

![Figure 5.3: Configuring the host device.](image)

5.2 Design Approach

To enable encryption in the communication protocol Secure Socket Layer (SSL) will be used which is already explained. Thus the communication between the PC system and the embedded system will be encrypted and a hacker will not be able to eavesdrop on the communication.

Authentication will be done by using the SSL with an extra layer of security. The keys generated for SSL will be encrypted using the encryption algorithm DES3 that can only be unlocked by a passphrase. If by any chance the certificates get stolen, the thief will not be able to pose as a client because the certificates are useless without the passphrase. If the thief wants to create his own certificates, the server will decline connection to
him because his certificates will not be signed by server's known lists of Certificate Authority.

As seen from figure 5.2, Beremiz runtime server starts in PYRO, not in PYROSSL. The first task of this thesis would be to modify the Beremiz_service.py so that it starts with PYROSSL and accepts SSL connections. In Beremiz_server.py, from the 'Server' class:

```python
def Start(self):
    pyro.initServer()
    self.daemon=pyro.Daemon(host=self.ip_addr, port=self.port)
```

Here it can be seen that the server starts without the PYROSSL protocol and there is no option to check the certificates. To start runtime server with PYROSSL, these features need to be added here.

Then from the host side, from the class PYRO_connector_factory:

```python
if servicetype == "PYROS":
    schemename = "PYROLOCSSL"
    # Protect against name->IP substitution in Pyro3
    Pyro.config.PYRO_DNS_URI = True
    # Beware Pyro lib need str path, not unicode
    # don't rely on PYRO_STORAGE ! see documentation
    Pyro.config.PYROSSL_CERTDIR = os.path.abspath(str(confnodesroot.ProjectPath) + 
    '/certs')
    if not os.path.exists(Pyro.config.PYROSSL_CERTDIR):
        confnodesroot.logger.write_error('Error : the directory %s is missing for SSL certificates (certs_dir).
"Please fix it in your project.
Pyro.config.PYROSSL_CERTDIR)
        return None
    else:
        confnodesroot.logger.write(_("PYRO using certificates in '%s' 
" % (Pyro.config.PYROSSL_CERTDIR))
        Pyro.config.PYROSSL_CERT = "client.crt"
        Pyro.config.PYROSSL_KEY = "client.key"
```

PYRO_connector_factory works such way that when connecting to the runtime server, if the URI is changed to 'PYROS' from 'PYRO', it will try to establish a secure connection. It will look into the project folder for a folder named 'certs' which includes the 'client.key' and 'client.crt' files. If the directory or the certificates are missing it will show the following error:

```
Error : the directory 'certs' is missing for SSL certificates. Please fix it in your project.
```

So, the certificates have to be placed in a folder named 'certs' inside the project folder. Also, there is no option to verify the certificate chain of the SSL certificates which
is a requirement for using Certificate Authorities. This feature is also needs to be implemented.

The next task is to implement a button system by clicking which the necessary certificates will be generated automatically and will be placed in the appropriate directory.

A point to be noted that, only the root and host(client) certificates will be placed in their designated directory. The runtime server’s certificates will have to be delivered manually by email or any other storage device.

When the certificates will be created, they will also be encrypted with the DES3 encryption. Thus, whenever the server will try to use the certificates, it will ask for a 'passphrase' in the terminal. The connection cannot be established without this 'passphrase'.

If the certificates are in the corresponding directory, and the passphrase is correct, then M2Crypto will check the validity of the certificates:

```python
from M2Crypto.SSL import Connection
Connection.timeout = None
Connection.gettimeout = _gettimeout
Connection.settimeout = _settimeout
# M2Crypto.SSL.Checker.WrongHost: Peer certificate commonName does not
# match host, expected 127.0.0.1, got server
Connection.clientPostConnectionCheck = None

M2Crypto checks the validity date of the certificates and the common name with the host name. If the host name does not match with the common name, it will give the following error:

Peer certificate commonName does not match host, expected 127.0.0.1, got server

If the certificates are valid and the common name matches the host, it will establish the connection and the future communication in this session will be encrypted.
6 Realization and Implementation

According to the functional requirements and as both the development and runtime environment of Beremiz is written in Python, the choice of programming language was Python. The communication between the development environment and the runtime environment will be carried out by PYROSSL. Initial access can only be granted by the certificates generated by the asymmetric private key generated by RSA algorithm. If this key gets stolen, the thief will not be able to reproduce certificates because the key is protected by DES3 encryption. All the functions implemented in this thesis will be based on acceptance test driven development and will be tested using the automated testing framework Twister. All the libraries that will be used in this thesis will be open source.

6.1 Starting the runtime server with PYROSSL

The first task was to integrate SSL functionality in the runtime server. To do this, the server class of Beremiz_server.py was modified. The following codes were added to enable this functionality:

```python
Pyro.config.PYROSSL_CERT="server.crt"
Pyro.config.PYROSSL_KEY="server.key"
Pyro.config.PYROSSL_CA_CERT="ca.crt"
Pyro.config.WSSP_DNS_URI=True
Pyro.core.initServer()
# Construct the Pyro Daemon with our own connection validator, using SSL
self.daemon = Pyro.core.Daemon(prtcol='PYROSSL', host=self.ip_addr, port=self.port)
```

After the modification, when Beremiz_server is executed, it will prompt for a passphrase (figure 6.1).

If the passphrase is correct, the runtime server it will start with PYROSSL (figure 6.2).
6.2 Enabling Certificate Authority crosscheck in the IDE

The current configuration of the IDE side does not cross checks for a Certificate Authority. To enable this, the class PYRO_connector_factory was modified. The following line was added to the existing class:

```
Pyro.config.PYROSSL_CA_CERT="ca.crt"
```

This line of code points M2Cryto to the Root certificate with which it will compare the signed certificates. If the signature on the user end certificates matches with the root certificate, the authenticity of the user end certificates will be verified.

6.3 Creating the CA certificates

To act as a certificate authority, one has to work with the private and public certificates. The first step to create and sign certificates is to create the root pair which consists of root key and root certificate that is the identity of the CA. Then this root certificate can
be used to create and sign intermediate certificates or can be used to make end user certificates directly. In this thesis, only once set of server and client certificates had to be generated. For this reason, root certificates were used directly to create and sign user end certificates.

The first step in creating CA certificates is to create a root key. This can be done by typing the following code in the terminal:

```bash
openssl genrsa -des3 -out ca.key 1024
```

Here the key file is generated using RSA 1024 bit algorithm and encrypted with DES3 encryption algorithm.

Then the root certificate file is created using the root key file:

```bash
openssl req -new -x509 -days 3650 -key ca.key -out ca.crt
```

During this process, user has to input the necessary information (Country Name (2 letter code), State or Province Name, Locality Name, Organization Name, Organizational Unit Name, Common Name, Email Address) to the terminal. The root certificate is always self signed.

The next step is to create a server key. This can be done by typing the following code in the terminal:

```bash
openssl genrsa -des3 -out server.key 1024
```

Same as the root key, the server key is generated with RSA 1024 bit algorithm and encrypted with DES3 encryption algorithm. Then a Certificate Signing Request (CSR) is generated by using the following command:

```bash
openssl req -new -key server.key -out server.csr
```

This will be used to create the server certificate which will be signed by the root certificate. The server certificate can be generated by using the following command:

```bash
openssl x509 -req -days 3650 -in server.csr -CA ca.crt -CAkey ca.key -set_serial 01 -out server.crt
```

During this process user has to input the necessary information (Country Name (2 letter code), State or Province Name, Locality Name, Organization Name, Organizational Unit Name, Common Name, Email Address) to the terminal. Note that the common name must be the same as the domain name or IP address of the server, otherwise certificate validation will fail.

The client certificate is generated in the same way as the server certificate. First the client key is generated by RSA 1024 algorithm and then encrypted with DES3 encryption
algorithm. Then the certificate signing request is generated and using that client certificate is generated which will be signed by the root certificates.

Note that while signing a CSR for a third party, the certificate authority does not need their private key, so only providing a certificate signing request to the authority is sufficient.

To integrate this process in Beremiz, the ProjectController.py file was modified and a new class was added to the file. The basic idea behind generating the root certificates is given below:

The first step is to check whether the certificates already exist or not:

```python
# Checks whether the certificates exists or not
certs = os.path.abspath(str(confnodesroot.ProjectPath) + '/certs')
if not os.path.exists(certs):
    os.mkdir(certs)
C_F = join(certs, CERT_FILE)
K_F = join(certs, KEY_FILE)
if not exists(C_F) or not exists(K_F):
    # create a key pair
    k = crypto.PKey()
k.generate_key(crypto.TYPE_RSA, 1024)
    # create a self-signed cert
cert = crypto.X509()
cert.get_subject().C = raw_input("Country: ")
cert.get_subject().ST = raw_input("State: ")
cert.get_subject().L = raw_input("City: ")
cert.get_subject().O = raw_input("Organization: ")
cert.get_subject().OU = raw_input("Organizational Unit: ")
cert.get_subject().CN = raw_input("Certificate Authority: ")
cert.set_serial_number(1000)
cert.gmtime_adj_notBefore(0)
cert.gmtime_adj_notAfter(315360000)
cert.set_issuer(cert.get_subject())
cert.set_pubkey(k)
cert.sign(k, 'sha1')
```

This creates a key pair using OpenSSL and with RSA 1024bit algorithm. In the next step a certificate will be generated which will be self signed:
In the next step, the key file and the certificate file are opened and OpenSSL dumps the information to them:

```python
open(C_F, "wt").write(crypto.dump_certificate(crypto.FILETYPE_PEM, cert))
open(K_F, "wt").write(crypto.dump_privatekey(crypto.FILETYPE_PEM, k))
```

These codes along with a button were developed and integrated with the IDE.

### 6.4 Button development and integration

To improve the usability, a button had to be integrated in the Beremiz GUI which will be used to create the certificates automatically. The file ProjectController.py was modified to achieve this feature. The button was placed in the toolbar of Beremiz which is only available after the user creates or loads a project, figure 6.3.

![Figure 6.3: Shortcut button to create SSL certificates automatically.](image)

This button was linked with the certificate generation function. When an user presses on this button, it checks whether certificates already exist in the 'certs' directory or not. If the certificates are already there, it will show a notification that certificates already exist, figure 6.4.

![Figure 6.4: Notification showing that certificates already exist.](image)

If there is no certificate available, it will launch a popup window for the user to input the required information, figure 6.5.
6.4 Button development and integration

Figure 6.5: Popup window for the user to input the certificate details.

After giving the inputs when the user will press the submit button, there will be a notification that the certificates were generated successfully, figure 6.6.

Figure 6.6: Notification of Certificate generated.

Users can develop the project, build it and configure the connection properties. To connect to PYROSSL the URI location in the config tab has to be set to 'PYROS' and the IP address has to be exactly the same as the runtime server, figure 6.6. After this the user can try to connect to the runtime server.

Figure 6.7: Configuration of the connection properties for SSL connection

When a connection is attempted, the user will be prompted for the passphrase, figure 6.8.
6.5 Flow of Operation

If the passphrase is correct, M2Crypto will cross check the information provided in the certificates and will connect to the runtime server, figure 6.9.

After the SSL connection is established, the user can transfer the program to the runtime server safely with now fear of man in the middle attack, figure 6.10.

6.5 Flow of Operation

In the first step of the operation, the user has to create or load a project. If there is no certificate, the user has to generate certificates and transfer the server certificates along with the root certificate to the runtime server. Once the certificates are in place, the user can start the runtime server by using the passphrase. Then the user can connect to the runtime environment by entering the required passphrase in the IDE terminal. PYROSSL with the help of M2Crypto will check the certificate’s validity and the common name and the certificate authority’s signature and if everything matches will start a secure communication protocol and the future communication will be encrypted. The flow of the operation is shown in the figure 6.11.
6.6 Testing the Encryption with Wireshark

The transmitted data can be analyzed by using Wireshark. After setting up the connection, Wireshark was used to capture the data. If the communication is not encrypted, the transmitted data will be visible as plain text, but if the communication is encrypted, the data will be visible as random number. Figure 6.12 shows the screenshot of a Wireshark data capture. As seen from the screenshot, the data passing in between the development environment and runtime environment is encrypted.
6.7 Testing the code

The requirement for this thesis were to implement SSL in current communication protocol and develop a button function which will generate necessary certificates, encrypt them with a password an place them in the designated directory. The SSL encryption was tested by using Wireshark. The test functions for later part were written in Acceptance Test Driven Development (ATDD) method. This process involved creating tests before writing the code and these tests represents the expected behavior of the developing software. The tests were automated by writing the supporting code in a framework friendly format (in this case, for Twister) while implementing the feature. ATDD cycle starts with Discussing the requirements, then distilling the tests in framework friendly format, then develop the code and integrate the tests in it and then when the demo software is ready test and optimize it (figure 6.13)[35]. So to follow this cycle, the first step was to 'Discuss' the requirements.

Figure 6.12: Data captured and shown on screen via Wireshark.
The questions discussed for automatically generating the certificates can be written as:

- Where the certificates will be generated? Current directory or specific location?
- What happens if the directory is not there?
- What happens if some of the certificates are already in the directory?
- What happens if all the certificates are in the directory?

During the discussion, it was agreed that the certificates will be generated in a directory named 'certs' in the project folder. If the directory is not there, it will be created. If some certificates are already in the designated directory, the function will show a notification that those specific certificates already exist and will skip creating those and continue with the next certificate. If all the certificates are available, it will show a notification that all the certificates are available.

The next step in ATDD was to 'Distill' the tests in a Framework-Friendly format. The test cases and the actions can be listed as:

<table>
<thead>
<tr>
<th>Test Case</th>
<th>Action</th>
<th>Arguments</th>
</tr>
</thead>
<tbody>
<tr>
<td>Directory exists?</td>
<td>Go the next step</td>
<td>Yes</td>
</tr>
<tr>
<td>Create directory</td>
<td></td>
<td>No</td>
</tr>
<tr>
<td>Certificates exist?</td>
<td>Show Notification</td>
<td>Yes</td>
</tr>
<tr>
<td>Create certificates</td>
<td></td>
<td>No</td>
</tr>
</tbody>
</table>
The next step was to 'Develop' the code and integrate the tests. The tests were run first and then they were failed as expected as there was nothing to be tested. The test cases were written without writing the code and automating them. The code was developed gradually and integrated with the test function and the test was repeated with Twister. Observing the test fail for the right reason and try to make the test pass was the main way to test the tests.

To test the final code, first the framework was configured to run the test 10 times and then the report was captured. Figure 6.14 shows the test setup configurations.

Figure 6.14: Configuration setup for the test.
Figure 6.15 shows the number of file tested and the number of failed and passed tests. As seen from the test, the code has passed successfully.

![Image of test result]

**Figure 6.15:** Result of the test.

In the following figure the detail report of the test result is given:

```
===== ===== ===== ===== =====
starting suite 'l001 - thesis'
===== ===== ===== ===== =====

<<< START filename: 'l001:/home/user/twister/demo/smoke-test-twister/test01_connect.py' >>>
Generating CA Root certificates:
Generating Server certificates:
Generating client certificates:

>>> File '/home/user/twister/demo/smoke-test-twister/test01_connect.py' returned 'Pass'. <<<
Test statistics: Start time 2016-02-09 01:32:57 -- End time 2016-02-09 01:32:57 -- 0.15
<<< END filename: 'l001:/home/user/twister/demo/smoke-test-twister/test01_connect.py' >>>

=================
... All tests done ...
=================
```

**Figure 6.16:** Detail log of the test.
7 Summary and Future Works

7.1 Summary

With the help technological advancement, control systems have evolved from hardwired relays and meters to digital Programmable Logic Controller platforms with computerized Human Machine Interfaces. Integration of network components has enabled users to control a system remotely from the other side of the world. But unfortunately, this high advancement of technology and integration with network has opened the door for hackers to maliciously attack the control system which may lead to serious consequences. Almost all automation systems are designed with functionality and performance keeping in mind, without concerning for security. Thus, when added with the network, they have become the main safety and security issue of the network society. For this reason, with the rise of technological advancement, the demand for security has also increased[3][2].

In this thesis, several security risks of the control system are presented. For general consideration, the open source software Beremiz was considered. Beremiz was vulnerable to hackers because there was no authentication system in it and the data it transferred was completely unencrypted. Therefore, anyone with the access to the Ethernet network could easily hack the system and manipulate it. To resolve these issues, the objective of this thesis was to encrypt the communication path and install an authentication system so that only authorized user can access and control the system.

Based on the objective, an system was introduced that generates necessary certificates for the encryption and encrypts them with another encryption algorithm. Thus the communication path is now encrypted and the users with proper authorization only can establish the connection.

The results were verified using the packet capture software Wireshark and the total development process was done using Acceptance Test Driven Development. The developed code was tested with an automated testing framework Twister and the code passed the tests successfully.
7.2 Future Works

In the next steps, the server and the root certificates generated in the IDE have to be delivered to the runtime server by using email or some data storage device. This leaves a security issue. The attacker can block this delivery and send forged certificates to the runtime server. As without the certificates the runtime server has no other authentication method, the runtime server may get compromised and the attackers get what they want from the runtime.

To prevent this issue, and out-of-band authentication method [36] can be implemented in the future. This will make man in the middle attack extremely difficult. Out-of-band authentication uses two separate communication channel at the same time to provide authentication. This provides much better security, because if the attackers get access to the security credentials of any one channel, the authentication procedure will not be complete without the second authentication network. The authentication method is initiated in the primary channel, and then concludes by authenticating in the second channel. This second channel of authentication can be done by a QR code or maybe a button enhanced security system.
Bibliography


All links were last followed on February 12, 2016.
Declaration

I hereby declare that the work presented in this thesis is entirely my own and that I did not use any other sources and references than the listed ones. I have marked all direct or indirect statements from other sources contained therein as quotations. Neither this work nor significant parts of it were part of another examination procedure. I have not published this work in whole or in part before. The electronic copy is consistent with all submitted copies.

________________________________________________________________________

place, date, signature