School of Science & Engineering

Software Licensing Based on Linear Algebra Theories

Hayat EL ASRI – Capstone Final Report
Supervised by: Dr. Abderrazak ELBOUKILII
Supervisor’s Signature: 
Date: March 3rd, 2016
Acknowledgment

First and foremost, I would like to express my deepest appreciation to Al Akhawayn University in Ifrane and the School of Science & Engineering for providing me with a healthy work environment and entrusting me with all the necessary autonomy to allow me to work on my capstone project: ‘Software Licensing Based on Linear Algebra Theories’.

I would also like to thank Dr. Abderrazzak Elboukili, my capstone supervisor, not only for all the effort and time allocated to helping me work on this project, but also for accepting to be my supervisor in the first place, taking the time to read all my reports, and giving me valuable feedback on each one of them.

I would like to express my gratitude to Dr. Yassine Salch Alj, the capstone coordinator, for taking the time to guide, supervise, and optimize my learning experience.

I would also like to thank my friends Achraff-Nour, Anas, Jaber, Mehdi, Maroua, Nada, and Sarah for the moral support they have provided me during my journey at AUI.

Last but not least, I would like to express my deepest gratitude to my parents for their continuous support and without whom, I would probably not be here today.

4/05/2016
Abstract

In partial fulfillment of the requirements of the School of Science & Engineering at Al Akhawayn University, students must take the Capstone Design course in which they are required to show the knowledge and skills they have acquired throughout their cursus at AUI.

In recent years, license key generators have experienced significant growth. They are used in all kinds of software, from engineering to business and communication, and are becoming expensive. Our goal for this capstone project is to develop a secure and free license key generator that is based on theories of linear algebra.

Throughout this project, the license key generator is meticulously analyzed from an engineering perspective as well as from a STEEPLE perspective. This led us to the conclusion that this application is inexpensive, secure, helpful, and maintainable.

This report is going to explain the details of this project, how it was designed, and implemented.

Keywords: Software, Licenses, Key Generator, Growth.
## Contents

ACKNOWLEDGMENT ............................................................................................................. 2

ABSTRACT .......................................................................................................................... 3

1- FEASIBILITY STUDY ........................................................................................................ 7

1.1- **PRODUCT** .................................................................................................................. 7

1.2- **TECHNICAL & OPERATIONAL FEASIBILITY** ............................................................ 7

1.3- **SOCIAL & MARKET FEASIBILITY** ............................................................................. 7

1.4- **LIFE CYCLE** .............................................................................................................. 8

1.5- **SOFTWARE MODEL** .................................................................................................. 8

1.6- **CODE OF ETHICS** ....................................................................................................... 8

1.7- **PROJECT SCOPE** ....................................................................................................... 9

1.8- **TIME SCHEDULE** ..................................................................................................... 9

2- FUNCTIONAL AND NON-FUNCTIONAL REQUIREMENTS .............................. 11

2.1- **FUNCTIONAL REQUIREMENTS** ............................................................................... 11

2.2- **NON-FUNCTIONAL REQUIREMENTS** ..................................................................... 11

2.2.1- **Product requirements** .......................................................................................... 11

2.2.2- **Organizational requirements** ................................................................................ 12

2.2.3- **External requirements** .......................................................................................... 13

2.3- **USE CASE DIAGRAM** ................................................................................................ 14
1- **Feasibility Study**

Feasibility studies are very important in software engineering initial phase. Indeed, any project can be successful if the right feasibility study is conducted and followed ahead of time. This section is a feasibility study for a software that will be used by commercial software developers to generate secure license keys for their users.

1.1- **Product:**

Thousands of software and applications are developed every day. However, it is not easy to find a license for one to secure his/her project. Therefore, this project is going to benefit the entire community (AUI-ers & outsiders alike). Furthermore, this product is going to follow the software engineering requirements and is going to be maintainable, safe, reliable, efficient, and usable.

1.2- **Technical & Operational Feasibility:**

For this software, many coding languages are going to be used.

First of all, the license key generator is going to be implemented using C language. Furthermore, I will be using Java for the Graphical User Interface. Moreover, I will be working on developing advanced algorithms & methods based on asymmetric cryptographic methods.

1.3- **Social & Market Feasibility:**

Every semester, hundreds of students develop applications that they would like to sell later. After doing some research, I could not find a single free license key generator.
that students and professionals could use to secure their applications. All license key
generators are embedded within a software license that is expensive.
Therefore, this software is going to target both professionals and university students & faculty
who wish to secure their developed applications.

1.4- **Life Cycle:**

This project is going to follow the software engineering life cycle that includes:

- Identification of needs
- Design
- Implementation
- Documentation
- Testing
- Maintenance

1.5- **Software Model:**

For this project, I will be using the incremental software model that combines the
elements of both the waterfall and prototyping model and uses them in an iterative manner.
This model is very flexible. Moreover, because of the several releases, implementing, testing,
and adjusting becomes easier and time-saving.

1.6- **Code of Ethics:**

This project is going to follow the ACM, IEEE, and the British Computer Society’s code
of ethics which states the following:

- Act consistently with the public interest
- Ensure the product meets the highest professional standards
• Follow a very ethical approach to the management of the product

1.7- **Project Scope:**

Even though this capstone project is very interesting, some constraints must be taken into consideration. Actually, most software licenses require years to implement & test. Therefore, because of the time limitation, my supervisor and I decided to work on one specific part of the software licensing process which is the key generation.

1.8- **Time Schedule:**

<table>
<thead>
<tr>
<th>Weeks</th>
<th>Objectives</th>
</tr>
</thead>
<tbody>
<tr>
<td>Week 1</td>
<td>- 1\textsuperscript{st} meeting with the supervisor</td>
</tr>
<tr>
<td></td>
<td>- Agree on a project</td>
</tr>
<tr>
<td></td>
<td>- Capstone Specifications</td>
</tr>
<tr>
<td>Week 2</td>
<td>- 2\textsuperscript{nd} meeting with the supervisor</td>
</tr>
<tr>
<td></td>
<td>- Feasibility Study</td>
</tr>
<tr>
<td>Week 3, 4, &amp; 5</td>
<td>- Requirements Engineering</td>
</tr>
<tr>
<td></td>
<td>- System Functionalities</td>
</tr>
<tr>
<td></td>
<td>- System Design</td>
</tr>
<tr>
<td>Week 6 &amp; 7</td>
<td>- Literature Review</td>
</tr>
<tr>
<td></td>
<td>- RSA algorithm</td>
</tr>
<tr>
<td></td>
<td>- Algorithm I and II design</td>
</tr>
<tr>
<td>Week 8 &amp; 9</td>
<td>- Algorithm III</td>
</tr>
<tr>
<td></td>
<td>- Implementation</td>
</tr>
<tr>
<td>Week</td>
<td>Tasks</td>
</tr>
<tr>
<td>------------</td>
<td>--------------------------------------------</td>
</tr>
</tbody>
</table>
| Week 10 & 11 | - Implementation  
               |   - Final modifications                   |
| Week 12    | - Final Report                             |
| Week 13    | - Capstone Presentation                    |
2- **Functional and Non-functional Requirements**

### 2.1- Functional Requirements

1. **Generate key**

   After getting the necessary information from the user through a web interface, two keys are going to be generated following an advanced algorithm based on linear algebra theory that my supervisor and I have implemented.

2. **Check key**

   When the user enters the given key, the software will have to check it in order to determine whether or not it is authentic and unique.

   This step is called Key Verification

3. **Delete key:**

   Because we are taking several pieces of information from the user, the probability of generating the same public key for two users is nearly 0%. Still, the software must have the functionality of deleting duplicate keys in case that happens.

### 2.2 - Non-functional Requirements

#### 2.2.1- Product requirements

- **Usability requirements**

  The application is going to be entirely in English as well as user friendly.
- **Performance requirements**
  This project should have a high execution speed. Likewise, appropriate algorithms are going to be used to ensure a minimized response time.

- **Scalability requirements**
  The license key generator should be highly scalable. Indeed, the software is going to be used by many students, professors, and professionals; therefore, the number of users is expected to grow highly during the next few years.

- **Extensibility requirements**
  The license key generator should be extensible to allow adding new features in the future.

- **Security requirements**
  The system should be highly secure as it deals with private keys. Furthermore, the application should ensure the privacy and integrity of the data. Likewise, only the system administrator shall be able to view and/or modify data.

- **Maintainability requirements**
  The system must be maintainable in order to allow features as well as upgrades to be implemented in the future.

### 2.2.2 Organizational requirements

- **Implementation requirements**
  Those requirements must be identified at the earliest stage of the product life cycle. Many factors have to be taken into consideration.
  
  - C language is going to be used for implementing the key generator.
• Java is going to be used for the Graphical User Interface

• The application’s features will need a concise choice of data structures.

• Virtual environment requirements:
   Operating system: Windows 7, Windows 8, 32 or 64 bits

  o Delivery requirements

This project must be delivered in exactly twelve weeks (three months).

  o Standards requirements

Standardized IEEE format is going to be used for this project.

2.2.3- External requirements

  o Ethical requirements

  I will conform to the ACM code of ethics while working on this project which includes being honest and trustworthy, honor property rights, honor confidentiality and many others.

  o Privacy requirements

  The data I will be dealing with is highly confidential as we take personal information such as: name, address, phone number…etc. Therefore, privacy is a must.

  o Space requirements

  The application must have a suitable size since it is going to be working for a long period of time.

  o Portability requirements

  This system should operate properly on any computer regardless of both the hardware and the OS.
2.3- Use Case Diagram

Use Case 1:

Use case name: Connect

Use case ID: 001

Description: The web interface was implemented in order to allow the user to enter his personal information.

Actor: Software Developer

Main Flow: The user case starts when clicking on the submit button
**Post condition:** After submitting all the information, the user is ready to receive the private and public key.

**Use Case 2:**

**Use case name:** Generate key

**Use case ID:** 002

**Description:** After submitting all the information, the software developer generates the private and public key from the aforementioned information

**Actor:** Software developer

**Pre-condition:** Have sent the information through the web interface

**Main flow:**

- The user gets the public key
- The software developer keeps track of both the private & the public keys.

**Use Case 3:**

**Use case name:** Add software

**Use case ID:** 003

**Description:** The software developer can add a new software to secure

**Actor:** Software developer

**Pre-condition:** Have sent all the necessary information through the web interface
**Use Case 4:**

**Use case name:** Edit/Remove Software  

**Use case ID:** 004  

**Description:** The software developer should be able to modify or remove a to-be-secured software.  

**Actor:** Software Developer
2.4- Software Model

In software engineering, the incremental software model is the one that uses the constituents of the waterfall software model in an iterative manner. In fact, following this model, the requirements are initially going to be fragmented into many stand-alone modules that are processed following a priority hierarchy.

For this project, I will be using the incremental software model because of the several advantages it offers such as:

- High flexibility
- Ease of testing & debugging
- Low delivery cost
- Ease of managing risk

For my project, I have analyzed, designed, and implemented each part alone. In fact, I started by developing the private key part with all its functions and tested it to make sure it is all working perfectly. Then, I moved to working on the second part of the project: the public key generation. Finally, I worked on the key verification.
Figure 2: Incremental Model

### 3- Technology Enablers

#### 3.1- Web Application:

Concerning the web application, the following languages have been used:

<table>
<thead>
<tr>
<th>Language</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>HTML (Hyper Text Markup Language)</td>
<td>I have opted for HTML as a markup language for my web page.</td>
</tr>
<tr>
<td>CSS (Cascading Style Sheet)</td>
<td>Combining CSS with HTML will help with the design of the web page.</td>
</tr>
<tr>
<td>PHP (Hypertext Preprocessor)</td>
<td>PHP code is going to be embedded within the HTML code for scripting purposes.</td>
</tr>
<tr>
<td>ASP</td>
<td>I chose to use ASP.net to take advantage of its scripting functionalities.</td>
</tr>
</tbody>
</table>
### 3.2- Desktop Application

<table>
<thead>
<tr>
<th>C Language</th>
<th>THE C PROGRAMMING LANGUAGE</th>
<th>I opted for C language for the desktop application. As a matter of fact, C Language is a middle level language because it combines the best part of high level language with low level language. It is both user and machine oriented and provides a great number of possibilities.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Java</td>
<td>Java</td>
<td>Java was used for the graphical user interface (GUI).</td>
</tr>
</tbody>
</table>
4- License Key Algorithm for Web

After discussing the already-existing algorithms that are used by most people for key generation, my supervisor and I agreed not to use any of them! Instead, we have agreed upon implementing our own asymmetric algorithm based on linear algebra theories.

The information we are going to take from the user will be as follow:

- Last Name
- First Name
- Birth Date
- Hometown
- Address
- Phone Number
- Mac Address
- Starting Date (for the demo version)

All this information is going to be concatenated, hashed, and converted into integers using ASCII code. Then, we are going to store it all in a 25x25 matrix that we will name: A.

Matrix B will come from row operations applied to the identity matrix. Further information will be given below.

The product of A and B will generate a key of 25 digits (the public key) that we are going to send to the user under the following format: xxxxx-xxxxx-xxxxx-xxxxx-xxxxx.

Concerning the public key’s decryption, we are going to use the inverse matrix.
5- **Web Implementation**

For the web implementation, I have chosen to work on PhpStorm that is a cross-platform PHP IDE. It offers an editor for: PHP, HTML, JavaScript, and CSS that are basically all the languages I wanted to use for my web page.

![PhpStorm](image_url)

Figure 3: PhpStorm
CSS

The snapshots below show the CSS implementation.

I have opted for CSS because of the possibility to customize the design. As a matter of fact, CSS language is the perfect language to use in order to have a professional web page.

The code below shows the style design of the body, the headers, and so on.

```css
body {
  padding: 0;
  margin: 0;
  background-color: darkgreen;
  font-family: Arial, "Times New Roman", sans-serif;
}

h1, h2, h3, h4, h5, h6 {
  padding: 0;
  margin: 0;
}

#wrap {
  width: 960px;
  background: #fff;
  margin: 20px 0;
  padding: 20px;
}

#header p {
  padding: 0;
  margin: 0;
  clear: both;
}

#navigation {
  background: #000;
  color: white;
  overflow: hidden;
  clear: both;
  height: 36px;
  margin: 25px 0;
}

#navigation a {
  color: #ffffff;
}

#navigation a:hover {
  color: #000000;
}
```

Figure 4: CSS Snapshot
PHP

I have opted for PHP as a scripting language for the numerous advantages it offers. First of all, PHP has a great online documentation with a good already-built functions’ framework thanks to its huge user-base. Moreover, even though it was my first experience coding with PHP, it was fairly easy to learn and use due to the documentation and tutorials available online. Furthermore, combining PHP and HTML and CSS has become the classic combination to come up with a good web-page. Last but not least, PHP works well with ASP as they may be combined.

These two screenshots below show the header and footer of the web page.
HTML was combined with CSS.

```html
<body id="Background">
    <img src="698aaf6f-100e-49c2-bc96-cc0245f0702d.png" width="320" height="257" />
    <hr />
    <p>Registration Page</p>
    <table width="500" border="0" align="center">
        <tr>
            <td><span class="TableElement">First name</span></td>
            <td><input name="" type="text" /></td>
        </tr>
        <tr>
            <td><span class="TableElement">Last Name</span></td>
            <td><input name="" type="text" /></td>
        </tr>
        <tr>
            <td><span class="TableElement">Birth Date</span></td>
            <td><input name="" type="text" /></td>
        </tr>
        <tr>
            <td><span class="TableElement">Home Address</span></td>
            <td><input name="" type="text" /></td>
        </tr>
        <tr>
            <td><span class="TableElement">Address</span></td>
            <td><input name="" type="text" /></td>
        </tr>
        <tr>
            <td><span class="TableElement">Phone Number</span></td>
            <td><input name="" type="text" /></td>
        </tr>
    </table>
    <p>&nbsp;</p>
    <input type="button" value="Next" />
    <hr />
</body>
</html>
```
Note:

The web interface below is only a prototype. Because of the time constraint, it was not possible to design our own algorithm in addition to fully implementing both the web and desktop application.
6- **Results**

Figure 6: Web Interface 1

Figure 7: Web Interface 2
The two snapshots above show how the web interface would look like. The first page is mainly for entering personal information such as the user’s full name, address, and phone number while the second page is for more specific information such as the starting date of the demo version and the mac address of the computer where the software is going to be used.

After clicking on “Submit”, the system administrator will receive, by email, all the information entered in a text file format.

To generate a key, the system administrator will have to enter manually all the necessary information. In fact, this is the most secure way of preserving confidentiality.
7- Algorithms

7.1- Literature Review

There exist two kinds of key based algorithms: symmetric and asymmetric key algorithms. This means that one of them (symmetric) uses the same key for both encryption and decryption while the asymmetric algorithm uses two different keys.

Because asymmetric algorithms are more secure, we are going to be using it for this project.

7.1.1- Asymmetric key algorithms

In this algorithm, the encryption key is different from the decryption one and therefore cannot be calculated. Actually, the words public key and private key are often used to refer to the encryption and decryption keys respectively. Furthermore, the message can be decrypted by one person only (the one who has the private key) even though the encryption key can be used by anyone.

Several algorithms using the asymmetric key are available. Data Encryption Standard, RSA, and the Digital Signature Algorithms are one of them. After doing a lot of research, I found out that the most widely used algorithm is the RSA.

7.1.2- The Rivest-Shamir-Adleman (RSA) Algorithm

The RSA is an algorithm is one of the most popular algorithm that is used by computers nowadays in order to encrypt and decrypt messages. As stated earlier, this algorithm is based on asymmetric cryptographic algorithm that is, up to now, considered secure and hard to hack.

In fact, this algorithm is based on the assumption that it is very hard to computationally factor very large integers; which is believed by most computer scientists. Likewise, it is partly
based on Fermat's Little Theorem and the Chinese Remainder Theorem. Moreover, the keys that are used by RSA are of length of a power of two. For instance, keys can be of length: 512, 1024, and so on. However, a key of length 512 bits is no longer considered secure. A minimum of 1024 bits is required to ensure security.

This algorithm for key generation works as follow:

1. Generate two random numbers p & q. It is better to choose very large prime numbers.
2. Compute the operation: \( n = p \times q \)
3. Calculate: \( \varphi(n) = ((p-1) \times (q-1)) \)
4. Pick an integer (for the public exponent) \( e \) such that: \( 1 < e < \varphi(n) \) & \( \text{GCD}(\varphi(n), e) = 1 \)
5. For private exponent, calculate: \( d = e^{-1} \mod \varphi(n) \)

Then, the private key will depend on \([d, n]\) while the public key will depend on \([e, n]\).

Furthermore, it performs two operations: plaintext & cyphertext encryption. Concerning the encryption of plain text, the following equation is used: \( m^e \mod n \). On the other hand, to decrypt a cyphertext, the following equation is used: \( m = c^d \mod n \).

Likewise, concerning the verification process, the recipient must use the public key in order to compute: \( s^e \mod n \). The message is then extracted and the information is computed. Finally, both messages get compared to check whether they are identical or not. If they are, then the signature is considered to be valid.
Even though this algorithm seems great, we have decided to design, implement, and use our own. As a matter of fact, my supervisor and I came up with our own asymmetric algorithm that is based on linear algebra theories for both encryption and decryption.

We have decided to use two different algorithms for the public and the private key. First of all, concerning the private key, we have decided to use a hashing & encryption algorithms.

The algorithm for the public key will be fully based on linear algebra theories and mainly matrices. As a matter of fact, we are going to use two 5x5 matrices whose components come from the hashed information taken from the data file to generate the public key. To check the validity of this key, the inverse matrix is going to be used.

Furthermore, we have decided not to use a database for the moment because of a major reason: time. As a matter of fact, after reading several papers on the implementation and use of key generation, encryption, and decryption algorithms, we came to the conclusion that having a database makes the time complexity much higher.
7.2- Algorithm Overview

We are using information from the user to encrypt the license key. Therefore, we are generating a unique key for every single user. To verify the key, we should process the data given from the user and then run our algorithm in order to get the encrypted key and compare it with the first key.

- Read the data file
- Extract information from the file
- Generate two random numbers a and b (3 digits – between 100 and 999)
- Compute \( c = a \times b \)
  - These last two information are going to be stored in a file.
- Concatenate first name, last name, hometown, phone number, mac address, birth date, and c.
- Hash the entire string and get 25 digits from it.
  - This is the private key
- Store in matrix A

Matrix B, on the other hand, is going to be generated from the identity matrix:

\[
B = \begin{bmatrix}
1 & 0 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 & 0 \\
0 & 0 & 1 & 0 & 0 \\
0 & 0 & 0 & 1 & 0 \\
0 & 0 & 0 & 0 & 1
\end{bmatrix}
\]

We are going to perform some row operations on this identity matrix to increase the security of the algorithm. That way, \( B = \) row operations applied to the identity matrix.
The following row operation are going to be performed:

- Row 1 = (row 1 + row 2) *3
- Row 2 = row 5 + row 2
- Row 3 = (row 3 *4)
- Row 4 = (row 5 + row 1) * 3
- Row 5 = (row 1 + row 2 + row 3 + row 4 + row 5) *2

After multiplying matrix A by matrix B, we will perform a “modulo 10 (%10)” operation on matrix C in order to get 25 digits.

This is the public key. This later will be sent to the client under the following format:

xxxxx-xxxxx-xxxxx-xxxxx-xxxxx
7.3- Main Algorithms

7.3.1- Algorithm I:

The first algorithm used in this project was the one for the private key generation. As a matter of fact, we did not use any existing algorithm and privileged innovation to come up with our own.

After getting all the necessary information from the client, for example:

```
```

We have decided to first remove all spaces, dots, and columns. That way, we get:

```
KadiEla28111975IfraneG4A78741R50672841565
```

Then, we have decided to generate two random numbers (in the size of 3 digits), multiply them, and store them with the string. We concatenate everything.

```
KadiEla28111975IfraneG4A78741R5067284156516159195151
```

Here the hashing process begins. We have decided to generate 100 random numbers and store them in a file; that way, we will be able to keep track of them. These random numbers are going to determine the position by which every character is going to move. Moreover, it will be possible to perform the $f^{-1}(x)$ function.
Likewise, we have decided to only display the first 25 digits of the hash and store it in a file as well.

This is the private key.

7adaEla281112KaIEraneG4A7
### 7.3.2- Algorithm II:

The second main algorithm concerns the public key. After getting the private key and storing it in a file, we had to derive a public key from it.

We took that private key and converted it using ASCII code into integers. We stored the resulting integers into a 5x5 matrix (matrix A).

Following the example I worked with in the previous section, we got the following matrix:

```
55  97  100  97  69
108 97  50  56  49
 49  49  50  75  97
  73  69 114  97 110
101  71  52  65  55
```

We noticed that we have repeated patterns. To avoid that, we have decided to multiply matrix A by another matrix that is derived from the identity matrix.

Matrix B, after row operations, is as follow:

```
3 3 0 0 0
0 1 0 0 1
0 0 4 0 0
3 0 0 0 3
2 2 2 2 2
```
After multiplying the two matrices, we got a big matrix (matric C):

\[
\begin{bmatrix}
594 & 400 & 538 & 138 & 526 \\
590 & 519 & 298 & 98 & 363 \\
566 & 390 & 394 & 194 & 468 \\
730 & 508 & 676 & 220 & 580 \\
608 & 484 & 318 & 110 & 376 \\
\end{bmatrix}
\]

Because matrix C is big and we want the public key to be 25 digits only, I have decided to use \( \%10 \) (module 10) in order to have only one digit in each slot of the matrix.

After performing the module operation on the matrix, we get our public key:

\[
40886 \ 09883 \ 60448 \ 08600 \ 84806
\]
7.3.3- **Algorithm III**

The third main algorithm concerns the key validation. For this part, we had two theories, that I am going to talk about, and we ended up choosing the one that takes the less time to implements because of the capstone deadline.

**Theory 1**

The first idea we had was based on the determinant and adjoint matrix theories. In our case, we chose a matrix that contains only integers in order to have: \( \text{Det} = +/- 1 \).

Basically, our idea was to calculate the inverse matrix of \( A \) to validate the key.

\[ A^{-1} = (\text{adj A}) / (\text{detA}) \]

However, after checking, we found out that it was time consuming to calculate the adjoint matrix & that we need to come up with our own algorithm instead of using the already existing ones. Therefore, we have decided not to use it for now.

**Note:** we would like to use the Gaussian elimination to compute the inverse matrix as it is faster.

**Theory 2**

The second idea we had was simple but efficient. We decided to use it because of the lack of time. So, to verify the key, the user is going to enter the public key. Then, we are going to check in the “public key file” whether or not that key exists. If the key does not exist, we will display an error message. On the other hand, if the key does exist, we are going to check the private key file and check whether or not that public key corresponds to that private key.

From that private key, we are going to get all the information related to that user. If everything corresponds, we release the software.
7.4- Security

After reading several papers about the security of the already existing algorithms used, I came to the conclusion that there is no secure algorithm yet. Indeed, the RSA algorithm is used by most companies and individuals today; however, there are many attacks against it. Some of these attacks are as follow:

1. Forward search attack: “If message space is predictable, attacker can decrypt C simply by encrypting all possible messages until a match with C is obtained.”

2. Common modulus attack: “If everyone is given the same modulus „n” but different (e,d) pair, then under certain conditions, it is possible to decrypt the message without d.”

3. Low encryption exponents: “When encrypting with low encryption exponents (e.g., e = 3) and small values of the m, (i.e. m < n1 / e) the result of me is strictly less than the modulus n. In this case, cipher texts can be easily decrypted by taking the eth root of the cipher text over the integers.”

The algorithms we are using for key generation, encryption, and decryption are advanced. By using information from the user to generate the key, we are minimizing the risk of hacking. Moreover, we have decided to generate two numbers in order to add randomness to the key. Last but not least, we have decided to implement a hashing algorithm that is based on random numbers to be able to keep track of every character’s new position.
Besides, using linear algebra theories, and more specifically matrices, is actually new to the field. By designing this algorithm, we are hoping to create the most secure key generation algorithm it exists so far.

Also, by having two matrices with different information, we are enhancing the security of our algorithm. The public key is, therefore, going to be based not only on the information from the user, but also on all the different algorithms we are using as well as matrices operations.
8- **Implementation**

8.1- **Private Key Implementation**

Several functions have been used in order to generate the private key. Below are screenshots of the functions I have used in C language.

The initial text file looks as follow:

![Figure 9: Data File](image)

The screenshot below concerns the function that removes spaces in the text file. Similarly, I have used a couple of other functions to remove dots, columns, and lines.

```c
// to remove spaces between data in the text file
void RemoveSpaces(char* source)
{
    char* i = source;
    char* j = source;

    while(*j != 0)
    {
        *i = *j++;
        if(*i != ' ')
            i++;
        
    }
    *i = 0;
}
```

![Figure 10: Remove Space Function](image)
The screenshot below concerns the function that reads the data from the data file and stores the information in multidimensional arrays.

```c
// reads data from the file & stores them in multidimensional arrays
void update_string( FILE * fp ){
    int i=0, j;

    //open file
    fp=fopen("text.txt", "r");

    if ( fp != NULL )
    {
        while ( fgets( &string[i], sizeof( string[i] ), fp ) != NULL ) /* read a line */
        {
            strcpy(Client_Info[i], string[i]);
            //call the following functions to get rid of spaces, points, and columns
            RemoveSpaces(string[i]);
            RemovePoints(string[i]);
            RemoveColumn(string[i]);
            i++;
        }
        fclose(fp);
    }
    else
    {
        perror( "text.txt" ); /* why didn't the file open? */
    }
    clientsCount--; //updates clients' count
}
```

Figure 11: Update String Function
The screenshot below concerns two functions: concatenate and hash.

```c
// concatenate the two generated random numbers with the string
void ConcatRandom (void)
{
    int i, j, len;
    char buff[200];
    for ( i = 0; i < clienCount ; i++)
    {
        RemoveNewLine(string[i]);
        sprintf(string[i], "%s%d%d", string[i], random[i][0], random[i][1], random[i][2]);
    }

    // Hashing Function
    void Hash (int p)
    {
        int j, k, p=18;
        char buff[200];
        strcpy(curr, string[i]);
        ScanRandomForHashing();
        for(j=0; j<28; j++)
        {
            string[i][j] = buff[j];
        }
        string[i][28] = '0';
    }
```

The final result of the private key generation is stored in a data file as shown below.

![Private Keys Text File](image)

Figure 12: Concatenation & Hashing Functions

Figure 13: Private Keys Text File
8.2- Public Key Implementation

The screenshots below are for the functions that convert the private key into integers using ASCII code and storing it into a data file respectively.

```c
void FKTtoAsciiToMatrix(void){
    int i, j, k;
    for(i=0; i<clientsCount; i++){
        for(j=0; j<S; j++){
            for(k=0; k<C; k++){
                MatrixA[i][j][k] = string[i]*j*k;
            }
        }
    }
}

void storePrivateKey(FILE* ofp){
    int i;
    if ( ofp != NULL ){
        for(i=0; i<clientsCount; i++){
            fprintf(string[i], ofp);
            fprintf("\n", ofp);
        }
    }
}
```

Figure 14: Conversion Using ASCII and Storing Into Matrix Functions
The screenshot below shows the two matrices (A and B).

```c
341 void storeMatrixA(FILE * ofp2) {
342     int i, j, k;
343     if(ofp2 != NULL) {
344         for(i=0; i<clientsCount; i++) {
345             for(j=0; j<S; j++) {
346                 for(k=0; k$S; k++) {
347                     fprintf(ofp2, "%d ", MatrixA[i][j][k]);
348                 }
349             }
350             fprintf(ofp2, "\n");
351         }
352     }
353     fprintf(ofp2, "\n");
354 }
355
356 void getMatrixB(FILE *fp) {
357     for(i=0; i<S; i++) {
358         for(j=0; j=S; j++) {
359             scanf(fp, "%d", &MatrixB[i][j]);
360         }
361     }
362 }
363 ```

Figure 15: Store Matrix A into a File

After generating the public key, we store it in a data file as follow:

Figure 16: Public Key File
8.3- Public Key Verification

As explained earlier, we have opted for a search and check process instead of computing the inverse matrix.

The following screenshots show some of the functions used.

```c
//checks the information entered by the user
int CheckPublicInfo(char Key[100], int index){
    int i, j;

    /*for (i=0; i<clientsCount; i++){
        if (strcmp(Key, Client_Info[i][index], strlen(Key))==0)
            return(i);
    */

    //compares the info entered by the user with the info that matches the license key
    if(strcmp(Key, Client_Info[index], strlen(Key))==0)
        return index;

    return ('\0');
}

//checks the public key
int CheckPublicKey(char Key[50]){ 
    int i, j;

    for (i=0; i<clientsCount; i++){
        if (strcmp(Key, PublicKey[i])=='0') //checks whether or not the public key exists
            return(i);
    }

    return ('\0');
}
```

Figure 17: Verification Function
```c
// Checks if the entered key exists & matches it with the information entered later on
if (CheckPublicKey(key) != 0) {
    printf("\n **FOR TESTING** the client info is: %s\n", Client_Info[CheckPublicKey(key)]);
    printf("Please enter your first name: ");
    gets(buf);
    RemoveNewLine(buf);
    strcpy(input, " ");
    strcat(input, buf);
    printf("Please enter your last name: ");
    gets(buf);
    RemoveNewLine(buf);
    strcat(input, buf);
    printf("Please enter your birth date dd.mm.yyyy: ");
    gets(buf);
        printf("Please enter a valid birthday (dd.mm.yyyy) ");
        gets(buf);
    }
    RemoveNewLine(buf);
    strcat(input, buf);
    printf("\n");
}
```

Figure 18: Checking Information Function
8.4- Executable Screenshots

I chose not to hide the password just to show how it actually works. Both the username and password were hardcoded and set to: “admin”.

![Executable File - Password](image1.png)

Figure 19: Executable File - Password

Concerning the random numbers generation, I generated two different random numbers for every client. Each line corresponds to a specific client. For example, line 0 corresponds to client 1 in all files.
After removing all dots and columns, multiplying the two random numbers, and storing both numbers along with their product with the string, we get:

After hashing and taking the first 25 digits only, we get our private key for every client.
After converting the private key into integers using ASCII code, we get a matrix for every client:
Figure 23: Executable File – Matrix A
Matrix B is stored into a file.

After multiplying matrix A by matrix B, we get:

![Figure 24: Executable File – Matrices Product](image)

<table>
<thead>
<tr>
<th>Matrix for client number 0</th>
<th>526</th>
<th>716</th>
<th>232</th>
<th>620</th>
</tr>
</thead>
<tbody>
<tr>
<td>7777</td>
<td>550</td>
<td>526</td>
<td>138</td>
<td>607</td>
</tr>
<tr>
<td>786</td>
<td>676</td>
<td>680</td>
<td>216</td>
<td>520</td>
</tr>
<tr>
<td>809</td>
<td>601</td>
<td>470</td>
<td>194</td>
<td>634</td>
</tr>
<tr>
<td>869</td>
<td>645</td>
<td>662</td>
<td>230</td>
<td>590</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Matrix for client number 1</th>
<th>608</th>
<th>654</th>
<th>194</th>
<th>582</th>
</tr>
</thead>
<tbody>
<tr>
<td>748</td>
<td>572</td>
<td>554</td>
<td>166</td>
<td>572</td>
</tr>
<tr>
<td>798</td>
<td>644</td>
<td>590</td>
<td>202</td>
<td>548</td>
</tr>
<tr>
<td>822</td>
<td>616</td>
<td>560</td>
<td>228</td>
<td>628</td>
</tr>
<tr>
<td>839</td>
<td>580</td>
<td>610</td>
<td>206</td>
<td>631</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Matrix for client number 2</th>
<th>626</th>
<th>625</th>
<th>210</th>
<th>614</th>
</tr>
</thead>
<tbody>
<tr>
<td>749</td>
<td>616</td>
<td>630</td>
<td>194</td>
<td>519</td>
</tr>
<tr>
<td>798</td>
<td>600</td>
<td>512</td>
<td>200</td>
<td>600</td>
</tr>
<tr>
<td>861</td>
<td>626</td>
<td>610</td>
<td>222</td>
<td>610</td>
</tr>
<tr>
<td>799</td>
<td>559</td>
<td>650</td>
<td>234</td>
<td>664</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Matrix for client number 3</th>
<th>626</th>
<th>625</th>
<th>210</th>
<th>614</th>
</tr>
</thead>
<tbody>
<tr>
<td>760</td>
<td>539</td>
<td>535</td>
<td>132</td>
<td>551</td>
</tr>
<tr>
<td>704</td>
<td>607</td>
<td>614</td>
<td>194</td>
<td>493</td>
</tr>
<tr>
<td>808</td>
<td>622</td>
<td>478</td>
<td>214</td>
<td>610</td>
</tr>
<tr>
<td>859</td>
<td>595</td>
<td>602</td>
<td>214</td>
<td>601</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Matrix for client number 4</th>
<th>456</th>
<th>616</th>
<th>280</th>
<th>508</th>
</tr>
</thead>
<tbody>
<tr>
<td>721</td>
<td>525</td>
<td>513</td>
<td>130</td>
<td>534</td>
</tr>
<tr>
<td>703</td>
<td>608</td>
<td>596</td>
<td>196</td>
<td>488</td>
</tr>
<tr>
<td>785</td>
<td>591</td>
<td>460</td>
<td>200</td>
<td>594</td>
</tr>
<tr>
<td>822</td>
<td>587</td>
<td>614</td>
<td>222</td>
<td>587</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Matrix for client number 5</th>
<th>682</th>
<th>690</th>
<th>234</th>
<th>622</th>
</tr>
</thead>
<tbody>
<tr>
<td>836</td>
<td>599</td>
<td>582</td>
<td>194</td>
<td>659</td>
</tr>
<tr>
<td>771</td>
<td>577</td>
<td>606</td>
<td>130</td>
<td>526</td>
</tr>
<tr>
<td>714</td>
<td>624</td>
<td>604</td>
<td>216</td>
<td>540</td>
</tr>
<tr>
<td>869</td>
<td>642</td>
<td>470</td>
<td>194</td>
<td>615</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Matrix for client number 6</th>
<th>425</th>
<th>526</th>
<th>130</th>
<th>519</th>
</tr>
</thead>
<tbody>
<tr>
<td>784</td>
<td>608</td>
<td>606</td>
<td>216</td>
<td>512</td>
</tr>
<tr>
<td>818</td>
<td>588</td>
<td>454</td>
<td>194</td>
<td>618</td>
</tr>
<tr>
<td>692</td>
<td>466</td>
<td>546</td>
<td>110</td>
<td>466</td>
</tr>
<tr>
<td>486</td>
<td>400</td>
<td>404</td>
<td>96</td>
<td>370</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Matrix for client number 7</th>
<th>594</th>
<th>538</th>
<th>130</th>
<th>526</th>
</tr>
</thead>
<tbody>
<tr>
<td>714</td>
<td>607</td>
<td>604</td>
<td>216</td>
<td>523</td>
</tr>
<tr>
<td>818</td>
<td>591</td>
<td>470</td>
<td>194</td>
<td>615</td>
</tr>
<tr>
<td>682</td>
<td>460</td>
<td>532</td>
<td>108</td>
<td>460</td>
</tr>
<tr>
<td>169</td>
<td>427</td>
<td>500</td>
<td>112</td>
<td>370</td>
</tr>
</tbody>
</table>
In order to have 25 digits in the matrix, I used modulo 10.

Figure 25: Executable File – Product Matrix After Modulo 10
Concerning the license key validation:
Figure 28:Executable File – Information Verification 1

Figure 29:Executable File – Information Verification 2
8.5- Graphical User Interface

Figure 30: Login Window

Figure 31: Key Generation Window
9- **Future Work**

Since my work was to first come up and design efficient and new algorithms to be used in this project, I ended up with limited time for the implementation.

For future work, I would like to have a database where I could store the location of the file only. The database will be used in order to secure the location of the file, not to store information.

Furthermore, concerning the third algorithm, I would like to use the first theory which is to compute the inverse matrix rather than use the search. Even though they are both efficient, our initial idea was to be able to compute $f^{-1}(x)$ from $f(x)$. Another idea we would like to implement would be using Gauss’s elimination in order to compute the inverse matrix.

Last but not least, if the software is to be used by AUI students and faculty, we would like to integrate it to the official AUI website for everyone to benefit from.
10- **Capstone Assessment**

This capstone project was a great opportunity to use all the theories, techniques, and skills learned during these past four years. Not only did it help me sharpen my skills as a computer scientist and an engineer, but it also helped me feel ready to enter the work market.

Likewise, this project gave me the chance to use the C language that I did not use much before. Indeed, this language is needed in the job market. Unfortunately, at AUI, we only have the opportunity to work with this language in three courses: C programming, Data Structures, and Operating Systems.

Furthermore, I have not had the chance to develop algorithms based on mathematical expressions before. This has given me the chance to tackle the mathematical area of computer science and understand how to use mathematics in real life applications.
11- STEEPLE Analysis

11.1- Socio-cultural

This software might increase society’s interest in software development, as this software will help them secure their creations. This will eventually result in the development of more quality software.

11.2- Technological

Thousands of software are developed every year. Therefore, this application is going to be the first of its kind to deliver a security key to protect these software free of any charge.

11.3- Environmental

This application uses minimal computer resources and doesn’t eventually waste energy. It is environment friendly as it does not harm, in any way, the environment.

11.4- Economical

This application is going to be free for all users. Therefore, to secure their software, students & professors will not have to spend money on any other software.

11.5- Political

First of all, this application will be able to cope with all kinds of changes in laws and policies. Moreover, all users will be required to approve a set of rules as well.
11.6- **Legal**

This application is completely legal since it aims to secure students’ and professors’ applications. The use of such application agree with the regulations and laws of the country in usage.

11.7- **Ethical**

The data will be encrypted and stored in a database to insure a maximum security. Furthermore, the private key will only be sent to the concerned person.
12- **List of Figures**

- Figure 1: Use Case Diagram
- Figure 2: Software Incremental model
- Figure 3: PhpStorm
- Figure 4: CSS Snapshot
- Figure 5: Php Screenshot
- Figure 6: Web Interface I
- Figure 7: Web Interface II
- Figure 8: Encryption & Decryption Diagram
- Figure 9: Data File
- Figure 10: Remove Space Function
- Figure 11: Update String Function
- Figure 12: Concatenation & Hashing Functions
- Figure 13: Private Keys Text File
- Figure 14: Conversion Using ASCII & Storing into Matrix Functions
- Figure 15: Store Matrix A into a File Function
- Figure 16: Public Key File
- Figure 17: Verification Function
- Figure 18: Checking Information Function
Figure 19: Executable File - Password

Figure 20: Executable File – Random Numbers Generation

Figure 21: Executable File - Concatenation

Figure 22: Executable File – Private Key

Figure 23: Executable File – Matrix A

Figure 24: Executable File – Matrix Product

Figure 25: Executable File – Product Matrix After Modulo 10

Figure 26: Executable File – Public Key

Figure 27: Executable File – Key Verification

Figure 28: Executable File – Information Verification I

Figure 29: Executable File – Information Verification II

Figure 30: Login Window

Figure 31: Key Generation Window
13- References:


