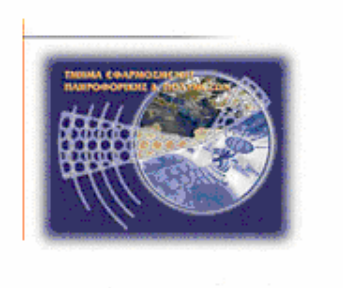
HELLENIC MEDITERRANEAN UNIVERSITY
SCHOOL OF ENGINEERING
DEPARTMENT OF ELECTRICAL AND COMPUTER ENGINEERING
Programme of Studies: Informatics Engineering T.E.



Thesis Title:

Privacy on large files using optimized cryptography software

Paris Chatzigeorgiou

Advisor: Papadourakis George

HERAKLION 2020

ABSTRACT

The purpose of this thesis is the designing and development of a cryptography software for Universal Windows Platform (UPW) on Windows 10 operating system that utilizes well known robust cryptographic algorithms and techniques wrapped in a user-friendly interface, users can securely exchange files of any type and storing encrypted personal files.

This software offers two major features:

- Files can be encrypted and stored using AES combined with SHA256 and Base64 encoding by providing passwords.
- Files can be exchanged securely between users by using RSA combined AES, SHA256 and Base64. The software provides a very simple way to extract keys for exchange.

This software is designed to be balanced between performance and security while it focusses on the ease of use.

ΣΥΝΟΨΗ

Ο σκοπός αυτής της πτυχιακής είναι ο σχεδιασμός και η ανάπτυξη ενός λογισμικού κρυπτογράφησης πάνω στην Universal Windows Platform (UWP) για το λειτουργικό σύστημα Windows 10. Το λογισμικό αυτό χρησιμοποιεί πολύ γνωστούς και πανίσχυρους αλγόριθμους και μεθόδους κρυπτογράφησης όλα μαζί να συνδυάζονται σε ένα φιλικό για τον χρήστη περιβάλλον.

Οι χρήστες μπορούν να ανταλλάζουν με ασφάλεια αρχεία και να αποθηκεύουν κρυπτογραφημένα προσωπικά αρχεία.

Αυτό το λογισμικό περιλαμβάνει δύο κύρια χαρακτηριστικά:

- Τα αρχεία μπορούν να κρυπτογραφηθούν και να αποθηκευτούν με τη χρήση του AES σε συνδυασμό με SHA256 και Base64 κωδικοποίηση με τη χρήση κωδικών χρήστη.
- Τα αρχεία μπορούν να διαμοιραστούν με ασφάλεια μεταξύ χρηστών με τη χρήση RSA σε συνδυασμό με AES, SHA256 και Base64 κωδικοποίηση. Το πρόγραμμα προσφέρει ένα εξαιρετικά απλό τρόπο να παράγει κλειδιά για διαμοιρασμό.

Το λογισμικό αυτό έχει σχεδιαστεί ώστε να είναι γρήγορο και να αξιοποιεί την επεξεργαστική ισχύ ενώ ταυτόχρονα παρέχει τη μέγιστη δυνατή ασφάλεια χωρίς να απαιτεί εξιδεικευμένες γνώσεις από τον χρήστη.

Acknowledgments

I would like to thank our professor Dr. George Papadourakis for the opportunities, knowledge and the inspiration he gave me during my studies.

Special thanks to Ioannis Deligiannis that as a mentor and a friend supported me, shared his knowledge and gave me priceless guidance during my studies and this thesis.

Table of Contents

CHAPTER 1: INTRODUCTION

1.1 Summary

This thesis is developed with a modern approach to software engineering and development utilizing the universal platform of Microsoft. That means any device on Windows 10 can run this application with full compatibility. Assemblies are compiled for multiple processor architectonics, ARM and x86/x64 are compatible. This feature of universal Windows platform is very important for software that mends to be used on a variety of different devices other than classic personal computers.

In general, its philosophy and the same techniques can be implemented on any platform and programming language with compatibility between them. Such platforms could support Linux OS, Mac OS, Android and more. This thesis restricts the implementation of Microsoft's technologies since the universal platform proves this concept for the purposes of this thesis.

For the materialization of the application, Visual Studio 2017 IDE, C#, .NET Framework, Universal Windows Platform SDK, Microsoft's Cryptography API for RSA, AES and SHA256, XML serialization and LINQ were used.

The project is hosted on GitLab using GIT and SourceTree as a client for source and version control.

1.2 Motive

Modern society suffers concerns about their personal data exposure, this concern applies to companies and individual persons. Most of the cryptographic solutions are either complicated for the average user, having performance issues or even security issues. While threats make privacy more vulnerable day by day, there are robust techniques to offer decent protection and keep it simple to the user. This thesis tries to achieve a balance between these requirements and make military-grade encryption available to the average user.

CHAPTER 2: FUNDAMENTALS

2.1 Analysis and Development Methods

2.1.1 GitLab[1]

Analysis and development of a software is a methodological procedure where the total project breaks down to smaller pieces followed by tasks for each feature. This will drive to the better overall design and it will make the development easier not only to materialize the project but to maintain it as well, Additional, this can be done much easier with source controlling. Source control not only gives the basic needs of multiple copies and making changes without destroying previous versions of our project but has made coding a social endeavor as well. From big projects to small, we can ensure that all code is easily maintained and managed.

CHAPTER 3: WORK PLAN

3.1 State of the Art

Modern society and business models have a heavy dependency on information exchange, but the world of information hides a lot of malicious intentions. Personal data are stolen and used as leverage for blackmailing while companies might lose years of work and profit by stolen information. This thesis purpose is to make a statement about how important is to keep what's private safe from unwanted eyes and prevent falling in pitfalls in a passive way.

By providing the best practices of military grade encryption on the tip of the finger this software will protect the data even when these data falls to the wrong hands. Practically the protected files are useful only to the rightful owners.

The application will protect any type of file of any size, it will encrypt it and decrypt as fast it can the hardware to do so without using few megabytes of RAM and a couple of clicks, and it makes information exchange secure, RSA ensures that only the receiver can access the encrypted file while AES-256 with large random generated keys and hashing combined with 2048 bit RSA keys, ensures military grade security and speed on the process.

Performance in not only about algorithms, programming techniques and the best possible practices in data manipulation is the core of a good performance. All data are streamed through the process, this stream will be changing and manipulation as it loads and written on output without any delays or large RAM consumptions.

These features are the state of art in security applications, they lack from most of the applications and even well-known applications until recently had large resources consumption. Moreover, they are complicated for the average user since they provide options that confuse the user, this compromise security. The application is a black box of security, it provides the best possible encryption today without demanding from the user knowledge or effort.

3.2 Technologies

The technologies used in this software are exclusively from Microsoft's stuck. Microsoft is providing their entire platform for free to educational institutes, development tools and operating systems such as Visual Studio and Windows can be downloaded for free using academical credentials.
Microsoft is a leading platform to enterprises development, not many students choose to use their platform since is not open source, however Microsoft's entire framework is moving to cross platform development and its open source under the name .NET Core. Microsoft's framework is being under continuous development for more than 20 years, it is very robust and extremely powerful since it is a modern OOP language but at the same time it can go low enough to manipulate pointers such as C and C++.

The .NET framework provides everything, all possible API's needed to develop any application are provided by Microsoft, they are tested by the company, so the developer doesn't have to worry about bugs by third parties, this way .NET framework encourages productivity.

Of course API's provided by Microsoft for the given fields are covering a very generic implementation so it can be flexible enough to be used by the developer by any way is necessary. There are third party API's and they are called nuget packages, those API's are given through Microsoft's platform but they are developed by other programmers, some of those require a payment and it is not guaranteed if they will be supported forever.

In this thesis the Windows Universal Platform is chosen since it may support any device that runs Windows 10.

3.2.1 Windows Universal Platform

Software engineering and development is a challenging and complex task for every platform and programming language. Scientific knowledge and creativity is a combination for every successful application and the result can apply for every different development platform. From a classic desktop application on Windows and Linux to modern mobile applications.

Here is where the true challenge comes, every platform has its own implementation styles, frameworks, programming languages and limitations. Developing an idea as a software in multiple platforms may change the initial requirements and something like that makes shifts the original software to something different.

Windows Universal Platform overcomes this issue to a point, the same code can be compiled for every device that runs Windows 10 and run on it. Also, there is on works a project that will bring Windows universal application to Linux and Mac systems.

Since the availability of a software is a major requirement windows universal platform seems very promising.

3.2.2 .NET Framework [2]

According to Wikipedia, .NET Framework is Microsoft's framework that runs mostly on Windows machines. It supports language interoperability (each language can use code written in other languages) with other programming languages and has a large class library called Framework Class Library (FCL). Software written for .NET Framework running in a software environment (not in a hardware environment) named Common Language Runtime (CLR). By using on the running machine an application virtual machine which includes services like security, memory management, and exception handling. This feature is called managed code. FCL and CLR compose the .NET Framework.

FCL provides user interface, data access, database connectivity, cryptography, web application development, numeric algorithms, and network communications. Developers creating software using .Net framework other libraries and their code. The framework is designed to be used by new applications for Windows. Microsoft provides an IDE for .NET software called Visual Studio that supports other languages as well..

.NET Framework resulted to a set of .NET platforms that targeting mobile computing, embedded devices, alternative operating systems, and web browser plug-ins. Mono is available for multiple operating systems and is customized for smartphone operating systems (Android and iOS) and game engines like Unity 3D. .NET Core targets cross-platform and cloud computing workloads.

3.2.3 C# programming language [3]

C# is a very interesting, robust and powerful OOP language, a general object-oriented programming (OOP) language for networking and Web development. C# is specified as a common language infrastructure (CLI) language.

In 1999, Anders Hejlsberg and his team started to develop C# on account to Microsoft's NET framework. Initially, C# was developed as C-Like Object Oriented Language (Cool), C# upgraded many C and C ++ features

3.2.4 Design Patterns [4]

Design patterns are general reusable solutions on commonly occurring problem in software design. They can be described as formal best practices that can be used from a programmer on designing an application or system to solve problems or to prevent them. Design patterns gained popularity in computer science by the book “Design Patterns: Elements of Reusable Object-Oriented Software” that published in 1994 by the “Gang of 4”.

Design patterns can help to speed up the development process and make the code more readable to those familiar with them. To use them requires consideration of issues that might occur later, so by using them it will help to prevent the problems. On every application, a design pattern must be programmed from start. Patterns usually show relations and interactions between classes or objects, without specifying the final application classes. They often described using UMLs so the relations and interactions can be easily understandable by developers.

Design patterns were originally grouped into three categories: creational patterns, behavioral patterns and structural patterns. They were 23 in number, but new patterns are presented by the years, having a result of new categories such as the architectural design pattern that may be applied at the architecture level of the software such as the Model–View–Controller pattern.

3.3 Cryptography

The main theme of this thesis is cryptography. Protection of user data is done with some of the most robust and trusted techniques utilizing algorithms that have never been ‘cracked’. The main algorithms are AES and RSA combined with SHA256 and Base64 encoding offering high level of encryption.

3.3.1 AES (Advanced Encryption Standard) [5]

The AES is a specification for the encryption of digital data established be the U.S National Institute of Standards and Technology (NIST) in 2001.
AES is a subset of Rijndael algorithm that has been selected from NIST as candidate for AES among fifteen submissions.
Rijndael algorithm designed by tow Belgian cryptologists, Vincent Rijmen and Joan Daemen, the

algorithm is named after their surnames.
The Rijndael algorithm is a new generation symmetric block cipher that supports key sizes of 128,192 and 256 bits and data handled in 128 bits blocks, Rijndael uses a variable number of rounds, depending on key sizes. A key of size 128 bits has 9 rounds, 11 rounds for 192 bits key and 13 rounds if the key size is 256 bits.
Rijndael is linear transformation cipher and it does not require Feistel network. It uses triple discreet invertible uniform transformations. These are Linear Mix transform, Non-linear Transform and Key Addition Transform.
Before the first round, a simple key addition layer is performed and that adds to security, then X-1 rounds and finally an extra round. Transformations forms a State when started but before the entire process completes.

The shift row transformation sees the State shifted over variable offsets. The shift offset values are dependent on the block length of the State. The mix column transformation sees the State columns take on polynomial characteristics over a Galois Field values (28), multiplied x4 + 1 (modulo) with a fixed polynomial. Finally, the round key transform is XORed to the key schedule helps the cipher key determine the round keys through key expansion and round selection.
The structure of Rijndael displays a high degree of modular design , which should make modification to counter any attack developed in the future much simpler than with past algorithm designs.
The difference between Rijndael is that AES is using a fixed block size of 128 bits and a key size of 128 bits, 192 bits or 256 bits while Rijndael can be specified with block and key sizes in any multiple of 32 bits, with a minimum of 128 bits and a maximum of 256 bits. The reason of this because of NIST minimum acceptability requirements.
For a block length above 128 bits, it takes 3 rounds to realize full diffusion, i.e., the diffusion power of a round, relative to the block length, diminishes with the block length.
Larger block length causes the range of possible patterns that can be applied at the input/output of a sequence of rounds to increase. This added flexibility may allow to extend attacks by one or more rounds.
Furthermore, the authors recommend a number of round (Nr) of

Nr=max(Nk,Nb)+6,

where Nk is the key size and Nb the block size (both in 32-bit blocks).
This means that Rijndael-128-256 will be slower than Rijndael-128-128 (AES-128), and that Rijndael-256-256 may have a lower security margin that Rijndael-256-128.

3.3.2 Military grade encryption

Military grade encryption refers to an AES encryption that utilizes a 256-bit length key (AES-256) it was the first publicly accessible and open cipher approved by the National Security Agency (NSA) to protect information of high sensitivity level. It is widely accepted as the strongest

encryption today and is used by governments, militaries, banks and other organizations worldwide to protect sensitive data.
This thesis utilizes military grade encryption enhanced with Base64 encoding, the algorithm divides the data into blocks of 128 bits each as AES suggests, then with a 256-bit key is used to scramble the blocks for fourteen (14) different rounds of encryption. The different rounds of encryption ensure the security attacks that utilizing patterns to guess the key.

3.3.3 AES-256 vulnerabilities

Today there is no successful attempt to crack AES-256, all attacks against the algorithm had no practical result, furthermore, a key with size of 256-bit has 1.1×10^{77} possible combinations, the fastest supercomputer in the world has the processing power of 10.51 Penta-flops according to and using brute force attack it could take 3.31×10^{56} years to crack the key.
However there is a type of attack called 'Side channel attack', these types of attack take advantage of the information leakage from the machine that executes the algorithm.
While a machine does calculations to perform the algorithm the attackers might take information about the key from factors like the energy consumption, the electromagnetic leaks, or even sound that can provide extra source information which can be exploited. Microsoft's researchers and from Indiana university found that many side attacks are based on statistical means.

Some attacks require knowledge of the systems hardware and software operation, while others through differential power analysis they are called black-box attacks.

3.3.4 Countermeasures

The countermeasures against side channel attacks is to reduce or eliminate the release of such information and eliminate the relationship between the leaked information and the secret data. This can be done by applying the factor of randomization of the cipher that transforms the data in a way that can be undone after the cryptographic operation (decryption). This countermeasure can be very effective and enhance the security against every type of attack.
For attacks utilizing the electromagnetic emissions there are available physical enclosures that will reduce the risk of surreptitious installation of microphones and other micro-monitoring devices.
One partial countermeasure against simple power attacks, but not differential power-analysis attacks, is to design the software so that it is "PC-secure" in the "program counter security model". In a PC-secure program, the execution path does not depend on secret values. In other words, all conditional branches depend only on public information. (This is a more restrictive condition than isochronous code, but a less restrictive condition than branch-free code.) Even though multiply

operations draw more power than NOP on practically all CPUs, using a constant execution path prevents such operation-dependent power differences (differences in power from choosing one branch over another) from leaking any secret information.[#] On architectures where the instruction execution time is not data-dependent, a PC-secure program is also immune to timing attacks.[7][8]

3.3.5 Base64 Encoding

According to Wikipedia [#] Base64 is a group of similar binary-to-text encoding schemes that represent binary data in an ASCII string format by translating it into a radix-64 representation. The term Base64 originates from a specific MIME (Multipurpose Internet Mail Extensions) content transfer encoding. Each Base64 digit represents exactly 6 bits of data. Three 8-bit bytes can therefore be represented by 6-bit Base64 digits.
There are many different encodings that they use specific bytes as special symbols with a special meaning for each encoding, transmitting data in binary format through different systems might end up corrupted. Base64 was established to solve this problem, it will convert bytes to ASCII characters that they don't contain any symbol with special meaning to other encodings. This comes with a cost since every 3 bytes of data is encoded to 4 ASCII characters increasing the size of the final product. Moreover, one ASCII character can be represented as a 7-bit value, the standard for modern systems is 8-bits equals 1 byte hence one ASCII character is equal to 1byte in size. That means, for every 3 bytes of raw data after Base64 encoding they will be need 4 bytes of space. In general, for N bytes of data after Base64 encoding the amount of space for the same data will be 4 x (N/3) bytes.
For example, 1kb of data would need 4 x (1024/3) = 1365 bytes or 1.365 KB
Countermeasures to this incretion of size is to compress with a proper compression algorithm the encoded data, this procedure will size down the data close enough to its origin size.

3.3.6 RSA (Rivest Shamir Adleman) [9]

The RSA algorithm was first published in 1978 by Ron Rivest, Adi Shamir and Leonard Adleman of the Massachusetts Institute of Technology. Known as public key and as asymmetric cryptography, utilizes two different but mathematically linked keys, one public and one private. The public key can be displayed in plain sight, known to everyone while the private key must be kept secretly and secure.
In RSA cryptography, both the public and the private keys can encrypt a message, the opposite key from the one used to encrypt a message is used to decrypt it. This attribute is one reason why RSA has become the most widely used asymmetric algorithm: It provides a method of assuring the confidentiality, integrity, authenticity and non-reputability of electronic communications and data storage.
Users of RSA create and shares the public key , this key is made by using two large prime numbers

and an auxiliary value. The prime numbers must be safe. Anyone can encrypt a message with the public key, but only with the current methods and the public key must be large enough, only the person that has the prime numbers can decrypt the message.

RSA is a slow encrypting method and is not used to encrypt data directly. RSA is used to encrypt symmetric keys that protect the target data. Symmetric algorithms like AES are way faster. This way we have utilize the features of asymmetric cryptography and the speed of symmetric cryptography.

A basic principle behind RSA is the observation that it is practical to find three very large positive integers e, d and n such that with modular exponentiation for all integer m (with $0 \leq m < n$):

$$(m^e)^d \equiv m \pmod{n}$$

and that even knowing e and n or even m it can be extremely difficult to find d. In addition, for some operations it is convenient that the order of the two exponentiations can be changed and that this relation also implies:

$$(m^d)^e \equiv m \pmod{n}$$

RSA involves a public key and a private key. The public key can be known by everyone, and it is used for encrypting messages. The intention is that messages encrypted with the public key can only be decrypted in a reasonable amount of time by using the private key. The public key is represented by the integers n and e; and, the private key, by the integer d (although n is also used during the decryption process. Thus, it might be considered to be a part of the private key, too). m represents the message (previously prepared with a certain technique explained below).

Key generation

The keys for the RSA algorithm are generated the following way:

1. Choose two distinct prime numbers **p** and **q**.
- For security purposes, the integers p and q should be chosen at random and should be similar in magnitude but differ in length by a few digits to make factoring harder. Prime integers can be efficiently found using a primality test.
2. Compute $\mathbf{n = p \cdot q}$.
- n is used as the modulus for both the public and private keys. Its length, usually expressed in bits, is the key length.
3. Compute $\mathbf{\lambda(n) = lcm(\lambda(p), \lambda(q)) = lcm\,(p-1, q-1)}$, where λ is Carmichael's totient function. This value is kept private.
4. Choose an integer e such that $\mathbf{1 < e < \lambda(n)}$ and $\mathbf{gcd\,(e, \lambda(n)) = 1}$; i.e., e and $\lambda(n)$ are coprime.

5. Determine **d** as **$d \equiv e^{-1} \pmod{\lambda(n)}$**; i.e., d is the modular multiplicative inverse of e (modulo λ(n)).
- This is more clearly stated as: solve for **d** given **$d \cdot e \equiv 1 \pmod{\lambda(n)}$.**
- **e** having a short bit-length and small Hamming weight results in more efficient encryption – most commonly e = 216 + 1 = 65,537. However, much smaller values of e (such as 3) have been shown to be less secure in some settings.
- **e** is released as the public key exponent.
- **d** is kept as the private key exponent.

The public key consists of the modulus n and the public (or encryption) exponent e. The private key consists of the private (or decryption) exponent d, which must be kept secret. p, q, and λ(n) must also be kept secret because they can be used to calculate d.

3.3.7 Example

Here is an example of RSA encryption and decryption. The parameters used here are artificially small, but one can also use OpenSSL to generate and examine a real keypair.

Calculation of Modulus and Totient

Lets choose two primes: p=11p=11 and q=13q=13. Hence the modulus is n=p×q=143n=p×q=143. The totient of n $\phi(n)=(p-1)\cdot(q-1)=120$$\phi(n)=(p-1)\cdot(q-1)=120$.

Key Generation

For the public key, a random prime number that has a greatest common divisor (gcd) of 1 with $\phi(n)$$\phi(n)$ and is less than $\phi(n)$$\phi(n)$ is chosen. Let's choose 77 (note: both 33 and 55 do not have a gcd of 1 with $\phi(n)$$\phi(n)$. So, e=7, and to determine d, the secret key, we need to find the inverse of 77 with $\phi(n)$$\phi(n)$. This can be done very easily and quickly with the *Extended Euclidean Algorithm*, and hence d=103d=103. This can be easily verified:
$e \cdot d=1 \bmod \phi(n)$ and $7\cdot 103=721=1 \bmod 120$.

Encryption/Decryption

Let’s choose our plaintext message, m to be 9:

Encryption:
$m^e \bmod n = 9^7 \bmod 143 = 48 = c$

Decryption:

$c^d \bmod n = 48^{103} \bmod 143 = 9 = m$

3.3.8 Padding schemes [6]

To avoid these problems, practical RSA implementations typically embed some form of structured, randomized padding into the value m before encrypting it. This padding ensures that m does not fall into the range of insecure plaintexts, and that a given message, once padded, will encrypt to one of a large number of different possible ciphertexts.

Standards such as PKCS#1 have been carefully designed to securely pad messages prior to RSA encryption. Because these schemes pad the plaintext m with some number of additional bits, the size of the un-padded message M must be somewhat smaller. RSA padding schemes must be carefully designed so as to prevent sophisticated attacks which may be facilitated by a predictable message structure. Early versions of the PKCS#1 standard (up to version 1.5) used a construction that appears to make RSA semantically secure. However, at Crypto 1998, Bleichenbacher showed that this version is vulnerable to a practical adaptive chosen ciphertext attack. Furthermore, at Eurocrypt 2000, Coron et al. showed that for some types of messages, this padding does not provide a high enough level of security. Later versions of the standard include Optimal Asymmetric Encryption Padding (OAEP), which prevents these attacks. As such, OAEP should be used in any new application, and PKCS#1 v1.5 padding should be replaced wherever possible. The PKCS#1 standard also incorporates processing schemes designed to provide additional security for RSA signatures, e.g. the Probabilistic Signature Scheme for RSA (RSA-PSS). Secure padding schemes such as RSA-PSS are as essential for the security of message signing as they are for message encryption. Two US patents on PSS were granted (USPTO 6266771 and USPTO 70360140); however, these patents expired on 24 July 2009 and 25 April 2010, respectively. Use of PSS no longer seems to be encumbered by patents. Note that using different RSA key-pairs for encryption and signing is potentially more secure.

CHAPTER 4: MAIN PART

4.1 Problem Analysis

There are many encryption software that are able to protect personal data however this thesis tries to showcase a more ‘modern’ approach on the matter by implementing a modern user interface and features that not only allow to keep any type of file safe but to exchange files with others knowing that only this contact cant access this file.
For many people having the responsibility of access is more important than trusting this responsibility to third parties, since personal information leaks has happened to large companies before. These types of users would prefer to hold their ‘key’ on their hands knowing that if they lose it they will lose their data. This might seem dangerous but on the other hand it gives the reassurance that they have all the control over their data.
This way users can store their data anywhere and yet keep the control over them. For example, a user that encrypts a file and he is the only one knows the key of encryption, can store it to the cloud knowing that even if it leaks no one can access this file.
This thesis respects this approach and keeps everything offline, offering just an interface and the encryption mechanisms that are required to achieve that.

4.2 Problem description

Keeping personal data safe is one thing and exchange them in privacy is another, users might have common passwords among many files and accounts so sharing an encrypted file using this key would be a bad idea. If both users, sender and receiver had agreed to a common password would solve the problem of private passwords but yet other threats underlying over this since a third party might discover the password while they agree to it. Using symmetric cryptography only on exchanging files is not the best practice.
To solve this problem this thesis is using RSA, this way each user just uses the public key of the receiver to make sure that only the receiver can access the file. Since RSA is really slow on encryption creates a big performance issue, the final approach in this thesis is using both algorithms. By doing this we have the performance of AES and the benefits of RSA.
All those features should be wrapped in a simple and easy to use interface. The end user should be able to use it with minimum required actions on their behalf.

4.2.1 System Requirements

The project was divided in three major parts, the user interface, the model, and the control (MVC pattern). The interface contains classes only with code related to the interface controls like the drag ‘n drop area, contacts, text encryption and settings.
The model contains classes with code related to the behavior of the core functionalities like encryption decryption key generation and manipulation of the byte arrays which implying the binary file during encryption and decryption.
The control contains classes that handles how the files are going to be stored or loaded on encryption and decryption, how the contacts are stored and other data related to the functionality of the software but need to stay as a from of a database.
Since the dependencies of the software should be restricted to the UWP API an SQL database or equivalent was not an option, this could create confusion for average users as they should install it separately from the software considering that this software is distributed by Windows store. On this matter using serialization and the already existing cryptography mechanisms it was possible to create a form of database to keep data like contacts and keys safely stored locally. The user can access them and backup those files knowing that are encrypted, in case that needs to restore them.
After research about the type and the size of files to be kept encrypted at the majority of use cases are documents, pictures videos that are few megabytes each taking in mind this to favor speed there should be large buffers that keeps in memory the data until the procedure finish. This is justified technique considering that the majority of desktops of the last 5 years have at least 4 GB RAM, however there must be a limitation of the input files at 2GB.
This could be avoid by reading and writing the data to be handled in smaller chunks this would mean no limitation and no issue with the RAM in lower end systems but something like this compromises speed in great manner for some cases.

The above problem will not be a concern since all the files are handled by streams eliminating the constrains of size and speed. Streams are reading continuously data in small chunks and each chunk is being processed and disposed while the next chunk is loading. Handling files with streams is fast as a processor and a hard drive allows to be while the RAM usage is kept at minimum. Of course, the larger chunk will curry more amount of data to process at once utilizing more RAM, this will make someone to assume that the entire progress will be faster but that depends on the processor power and the hard drive speed. In all tests the difference was insignificant and it really depends one each machine’s hardware components.

Using streams properly in the application allows the user to import any amount of files of any size, even entire hard drives without the need of much RAM.

For the purpose of the thesis dummy contacts and random files where used to showcase and test the software, the following table are use cases showing the achieved goals.

USE CASE 1	User has one file that wants to encrypt and keep it safe.	User can drag and drop the file into the respective drag ‘n drop area, the software will ask to enter a password that will be used to encrypt the file, then a directory chooser will appear for the user to select where the file will be exported.
USE CASE 2	User has multiple files that wants to encrypt them and keep them safe.	User can drag and drop multiple files by selecting them as a group or by dragging and dropping a folder that contains the files. The software will ask from the user to enter a password that will encrypt all the files, a directory chooser will appear for the user to select the export path.
USE CASE 3	User has one encrypted file that needs to access it.	User can double click an encrypted file and a respective dialog will popup asking for the password that protects the file. Alternative, the user can drag and drop the file to the respective area in the software, the same popup dialog will appear for the user to enter the password. A directory chooser will appear allowing the user to choose where the plain file will be exported.
USE CASE 4	User has multiple encrypted files that needs to access them.	User can double click one by one the encrypted files and a respective dialog will popup asking for the password that protects the file. Alternative, the user can drag and drop all the files to the respective area in the software with the condition that the file group is protected by the same password, the same popup dialog will appear for the user to enter the password. A directory chooser will appear allowing the user to choose where the plain files will be exported.
USE CASE 5	User has multiple encrypted files that needs to access some of them.	User can select the desired encrypted files if they are encrypted using different passwords the user must double click or drag n drop to the respective area the files individually, if they are protected by the same password the user can drag and drop the folder that contains the files or the selected filegroup to the

		respective area. User will be asked to enter the password and to choose the directory to be exported.
USE CASE 6	User has requested to send a file encrypted to a second user.	User “A” has requested from user “B” a file. User “B” sends once his public key to user “A” and imports it to the respective contact in the contact view. Once the contact contains the respective public key, user “A” clicks on the contact, a file chooser appears and selects the desired file. The file will be encrypted and exported to the desired file path. User “A” send the file to contacts mail or with any other media. User “B” receives the file and has access to it with the private key.
USE CASE 7	User has requested to send multiple files encrypted to a second user.	User “A” has requested from user “B” multiple files. User “B” sends once his public key to user “A” and imports it to the respective contact in the contact view. Once the contact contains the respective public key, user “A” clicks on the contact, a file chooser appears and selects the multiple files. The files will be encrypted and exported to the desired file path. User “A” send the files to contacts mail or with any other media. User “B” receives the file and has access to it with the private key.
USE CASE 8	User has an encrypted file send by another user and need to access it.	User “B” has the encrypted file with the public key, double clicks on the file or drags ‘n drop it to the respective area, a directory chooser will appear to select the export path of the plain file. User will be asked before decryption the password that protects his private key.
USE CASE 9	User has multiple files send by another user and needs to access them.	User “B” drags ‘n drops the encrypted files to the respective area or double clicks every file, a folder chooser will appear to select the export path of the plain files. User will be asked before decryption the password that protects his private key.
USE CASE 10	User wants to share public key.	User can generate a pair of keys from the settings area. On generation of kays the software will ask a password that will protect the private key and an export path of the public key. The private key is stored within the

		default application folder that is reserved on installation from the UWP container. The public key is safe to be stored plain anywhere the user wishes. User can share this file as public key to any contact that wants to communicate.
USE CASE 11	User wants to change keys for symmetric encryption.	User has to decrypt the file using his old password and encrypt it using a new password.
USE CASE 12	User wants to change keys for asymmetric encryption.	User can generate a pair of keys from the settings area. By doing so if there is already a pair of keys the former private key will be overridden from the new one and the new public key will be exported to the desired path. A new password will be asked from the user to protect the new private key.
USE CASE 13	User wants to add a contact	User can navigate to the contacts view click the button add, the people's app will open allowing to select multiple contacts. After selection the contacts will appear in the view and they will be stored in the database.
USE CASE 14	User wants to remove a contact.	User can navigate to the contacts view select a contact and click the remove button. The contact will be removed from the view and the database.
USE CASE 15	User wants to add a public key to a contact.	User can click on a new contact, if that contact doesn't have a public key a popup dialog will prompt asking the user to browse for a specific type of file that is the public key of the contact. The key will be imported to the database in the respective contact record.
USE CASE 16	User wants to remove a key from a contact.	User must remove the contact. This action will remove the public key as well from the database.
USE CASE 17	User tries to encrypt requested file for a contact	User will be asked to import a public key in order to encrypt a file for this contact.

	that doesn't have a public key.	
USE CASE 18	User wants to restore content of the software on a new installation.	User can navigate to the settings view and click the import keys button, a file chooser will be prompt allowing the selection of a specific type of file that is the pair of keys. After this action the user can use these keys to encrypt and decrypt files.
USE CASE 19	User wants to backup the content of the software.	User must navigate to the software's installation folder and copy the pair keys file to any desired location. The file is encrypted by the user and its safe to store it anywhere in case that a restoration is needed.

4.3 Implementation

4.3.1 Project Implementation

The project separated to two smaller projects, they are the main application project and the cryptographic library. Each one of them is separated to even smaller tasks, for the main application was the front end/ UI, the control of actions and the data manipulation. The cryptographic library is separated to symmetric, asymmetric cryptography and common utilities as SHA computing.
In the main application there is a directory that includes the pages (views) of the application and the related controls, there is a second directory that hosts the classes of control and behaviors, some of those classes are cryptography which acts as a wrapper to the cryptographic library, the XML serialization/deserialization classes that contain methods to process the serialized objects or to be serialized objects, the files format controller which manipulates files as binaries before and after encryption providing a custom format during encryption and decryption and enhance security and a class that loads the license status of the application.
The pages directory hosts for each view the XAML class that are the views, they are composed by partial classes, the XAML class contains code which renders the interface wile the C# class implements the code to be executed bonded with the UI controls.
The Assets directory holds the icons of the application or any other possible asset for the application to use.
The entire project development was separated to tasks using GitHub's forms and hosted in the same service for source control. The branches are separated to master and develop. For each new

implementation or bug fix there should be a push to repository along with good description for better control over the source code.

4.4 Detailed UWP Implementation

4.4.1 User Stories

In order to materialize the project for each step there was a user story concerning the application usage is described below along with explanation of techniques that used in code.

UWP user story 1

As user, I want to encrypt files	
1. Encrypt files using the application 2. UI for the encryption	A new Page created. A new DragDropArea created. A new List<StorageFile > created

UWP user story 2

As user, I want to provide files for encryption	
1. Select the plain files to be encrypted 2. Drag n drop them on the UI 3. Enter password 4. Select export file path	The DragDropArea's event handlers get triggered from the drag n drop acrion. CalledFunction(IReadOnlyList<IStorageItem> items) captures the elements that are dropped in the area. The List<StrorageItem> updates it's collection. A new Password view object is created and asks to enter a password. A file browser will ask for the export file path.

	A foreach loop goes through the collection of plain files checking their extension and if they are plain. Then encrypts each file and extract it to the selected file path.

UWP user story 3

As user, I want to provide files for decryption	
1. Select the encrypted files to be encrypted 2. Drag n drop them on the UI 3. Enter password 4. Select export file path	The DragDropArea's event handlers get triggered from the drag n drop acrion. CalledFunction(IReadOnlyList<IStorageItem> items) captures the elements that are dropped in the area. The List<StrorageItem> updates it's collection. A new Password view object is created and asks to enter a password that protects the files. A file browser will ask for the export file path. A foreach loop goes through the collection of plain files checking their extension and if they are encrypted. Then decrypts each file and extract it to the selected file path.

UWP user story 4

As user, I provide mixed files on drag n drop area	
1. Select mixed files 2. Drag n drop them on the UI 3. Enter password 4. Select export file path	The DragDropArea's event handlers get triggered from the drag n drop acrion. CalledFunction(IReadOnlyList<IStorageItem> items) captures the elements that are dropped in the area.

5. Select action encryption/decryption	The List<StrorageItem> updates it's collection. A new Password view object is created and asks to enter a password. A file browser will ask for the export file path. A foreach loop goes through the collection of the files checking their extension and if there are mixed files a pop-up dialog will ask the user if the intent is to encrypt or decrypt the files. Depending on the choice the respected files will remain in the collection while the others will be removed from the list. Then each file will go through the respected method (Encrypt/Decrypt) and the result will exported to the selected file path.

UWP user story 5

As user, I want to Decrypt one file saved in a file path without using the drag n drop area	
1. Double click the encrypted file 2. Provide password 3. Select export file path.	The application register's file types with extensions ".jlk", ".rlk" as properties during installation. Double click of a file of thins type will trigger the application to start and ask for the password. The user provides the password and the export file path. The file is exported as plain with its former extension.

UWP user story 6

As user, I want to add a contact

1. Import contact into the application 2. Contacts UI	A new Contacts Page is created. Searches the serialized file for existing contacts, if any they are displayed in the UI by association of the windows build in Peoples app. If the contact exist in peoples the photo , the unique id and the email will be loaded in the view. User clicks to the Add button at the bottom of the view, the Peoples app is opening in selection mode, selects the desired contact and returns the property object to the application. The application will read the needed properties of the object and it will serialize the contact using the XML serialization. The serialized entry will imported to the existing list in the respected file on the hard drive.

UWP user story 7

As user, I want add a public key to a contact	
1. Select contact 2. Import public key	Selecting a contact will trigger the app to look up for the public key of the selected contact if there is not any a pop-up dialog will ask to import a public key. Clicking on import will open a file browser the user will select the public key file and the app will add it to the contact's property. The XML serializer will serialize the contact properties and will update the respectful entry in the storage file.

UWP user story 8

As user, I want add a public key to a contact	
1. Select contact 2. Import public key	Selecting a contact will trigger the app to look up for the public key of the selected contact if there is not any a pop-up dialog will ask to import a public key. Clicking on import will open a file browser the user will select the public key file and the app will add it to the contact's property. The XML serializer will serialize the contact properties and will update the respectful entry in the storage file.

4.5 Software design

This software includes some key areas of code, the core of the project, those key parts will be analyzed in this section.

4.5.1 Input / Output mechanisms

This application provides three ways to insert data for process and one to store them.
For input the user can use the drag and drop area, double clicking an encrypted file or select files using the build in file picker.

After the process of data the result will be stored using a directory selector that will appear on the initialization of the process.

This section will focus on the mechanism and the code used for input output mechanisms rather the user interface.

4.5.2 Input / Output mechanisms implementation

The challenges when handling input data are the performance and recourses, utilizing more RAM will increase speed but compromises resources which can lead in performance issues if is not done correct.

Files can be inserted into the application in two ways, as a byte array containing the whole file or as a stream which means reading the file continuously in small chunks that are getting processed one by one.

Both methods have benefits and disadvantages, reading a file and storing it into a byte array will make the process faster since the file is accessible through the RAM which is considerably faster than the hard drive, but it will consume RAM equal to its size.
Streams will consume RAM equal to the buffer size, a small byte array that is filled with parts of the file at each circle of the procedure. For example, if the buffer size is 16KB the stream will consume continuously only 16KB from the RAM until the end of the process, but it will utilize the hard disk every time it reads the next chunk, slowing the procedure.

In general steaming data into the algorithms is considering as the best option since it can handle any size of file without crushing the system due the lack of available RAM.

The same logic applies for the output mechanisms, RAM consumption should stay to the minimum using streams to be written in the hard disk.

```
216         using (FileStream fileStream = new FileStream(filePathEncryptionBuild,FileMode.Open))
217         {
218             StorageFile newFile = await folder.CreateFileAsync($"{file.DisplayName}.rlk",
219                 CreationCollisionOption.GenerateUniqueName);
220
221             aesEncryptedFilePath =
222                 await AES.Encrypt(fileStream, Base.RandomString(25), newFile);
223             File.Delete(filePathEncryptionBuild);
224         }
```

On the above block is displayed a use case of steaming data.
The FileStream class inherits from the abstract class Stream of .NET framework and is used to read or write from and to the hard disk, the file stream is passed as argument to the encryption method and it will be processed while small chunks are loaded continuously.
The key word “using(…)” ensures that all objects that are initialized within this block will be disposed as soon as the running thread leaves this scope, this way all resources used within this scope will be freed.

4.6 Encryption

Encryption is divided in two separated parts, symmetric and asymmetric, they are independently from each other so they can be used in many different scenarios.

Before encryption the data need to be processed in a way that the encrypted file will keep protected information regarding the original data, like the original file name, file extension and the version of the software that encrypted this file.
It is important to keep these information safe within the encrypted file, this way when the file is decrypted it will be restored with the original name and file extension (for example personalDocument.pdf) while the version will ensure backwards compatibility for the future version of the software.

This section will focus on the encryption procedure, the pre – encryption process will be analyzed on the respective section.

4.6.1 Symmetric encryption

Symmetric encryption is utilizing the AES algorithm, what it makes the algorithm AES-256 is the way the key is provided into the algorithm.

Bellow is the block that materialize the AES-256 encryption.

```
22      public static byte[] Encrypt(string password, byte[] input)
23      {
24          IBuffer key = Convert.FromBase64String(SHA256.GetHashed(password)).AsBuffer();
25          IBuffer m_iv = Convert.FromBase64String("15CV1/ZOnVI3rY4wk4INBg==").AsBuffer();
26
27          SymmetricKeyAlgorithmProvider provider = SymmetricKeyAlgorithmProvider.OpenAlgorithm(SymmetricAlgorithmNames.AesCbcPkcs7);
28          CryptographicKey m_key = provider.CreateSymmetricKey(key);
29
30          AesCryptographicKey = key;
31
32          IBuffer bufferMsg = CryptographicBuffer.CreateFromByteArray(input);
33          IBuffer bufferEncrypt = CryptographicEngine.Encrypt(m_key, bufferMsg, m_iv);
34          return bufferEncrypt.ToArray();
35      }
```

AES-256 encryption that gets the input data as a byte array.

On the line 26 the IBuffer value with name key, is going to be used as the key for the encryption.

1. Convert.FromBase64String method will take as arguments a string type value in base 64 format and it will convert it into a byte array.

2. SHA256.GetHashed method will take as argument a string type value and it will calculate its hashed value using the SHA256 algorithm, the return value is a string in base 64 format.
3. Finally the return value of Convert.FromBase64String will be casted to IBuffer type by the extension method AsBuffer, in order to be assigned to the key variable.

Following these steps, is a way to convert a password provided by a user in a 256 bit key ready to be used as an encryption key.

On the line 25 the IBuffer type named m_iv will be assigned with a value that it will be the initialization vector and its provided a fixed Base64 string.

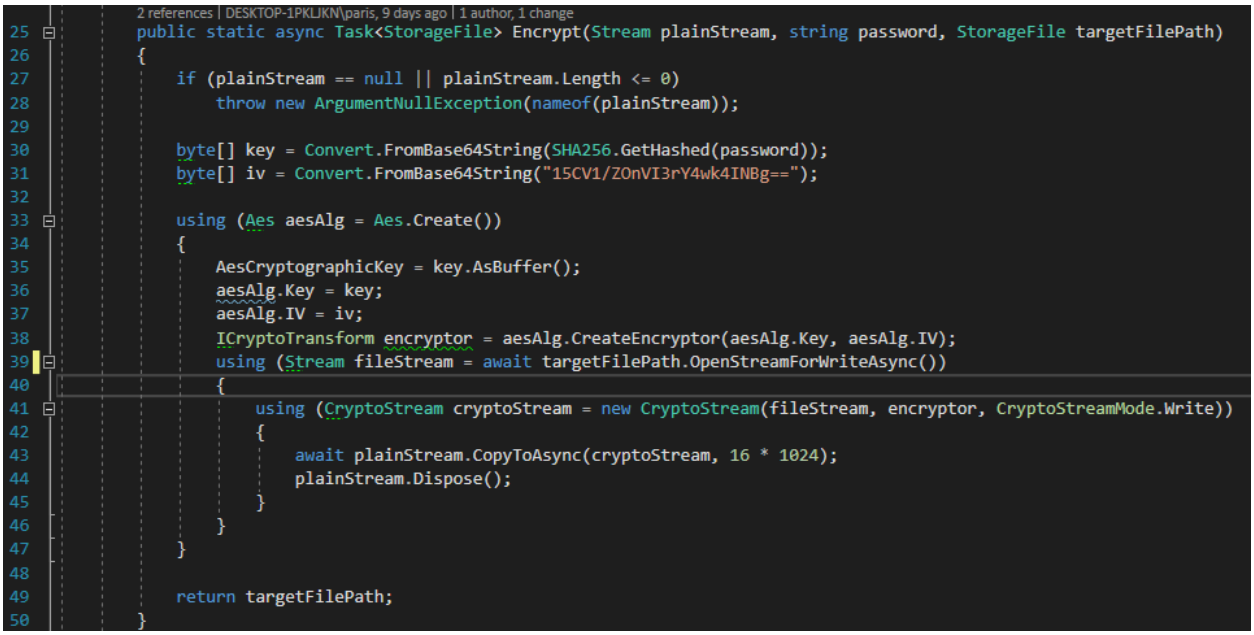
On line 27 by using the provider from the framework API we can select a cryptographic algorithm by name, in this case is AES with CBC and PKCS7

Then the actual key for the encryption will be created by the provider, the method CreateSymmetricKey will return a value of CryptograficKey type. This method will use the IBuffer key values as a material to create the actual key.

The next step is to prepare the input data to be encrypted by transforming them to the proper format, more specifically the input byte array must be casted to IBuffer type, this cab be done by the method CryptographicBuffer.CreateFromByteArry.

Finally the encrypted buffer will be created by Cryptographic engine, as arguments the method Encrypt needs the finalized key , the IV and the plain value.
The returned value is the encrypted value of the user input.

For cases that the input data is too large to be kept in memory there is an overload of this method that is taking that data as a stream instead of a byte array. This way the data are provided to the algorithm as they are loading from the hard drive in small pieces, the buffer size is only 16KB, this reduces RAM consumption to the minimum.
This method allows the encryption of files with any size.



```
2 references | DESKTOP-1PKLJKN\paris, 9 days ago | 1 author, 1 change
public static async Task<StorageFile> Encrypt(Stream plainStream, string password, StorageFile targetFilePath)
{
    if (plainStream == null || plainStream.Length <= 0)
        throw new ArgumentNullException(nameof(plainStream));

    byte[] key = Convert.FromBase64String(SHA256.GetHashed(password));
    byte[] iv = Convert.FromBase64String("15CV1/ZOnVI3rY4wk4INBg==");

    using (Aes aesAlg = Aes.Create())
    {
        AesCryptographicKey = key.AsBuffer();
        aesAlg.Key = key;
        aesAlg.IV = iv;
        ICryptoTransform encryptor = aesAlg.CreateEncryptor(aesAlg.Key, aesAlg.IV);
        using (Stream fileStream = await targetFilePath.OpenStreamForWriteAsync())
        {
            using (CryptoStream cryptoStream = new CryptoStream(fileStream, encryptor, CryptoStreamMode.Write))
            {
                await plainStream.CopyToAsync(cryptoStream, 16 * 1024);
                plainStream.Dispose();
            }
        }
    }

    return targetFilePath;
}
```

AES-256 encryption as an async task that streams the file due encryption procedure.

More over this method is more advance since it is utilizing Task<Type> which allows asynchronous call, it will be executed on a different thread allowing the main thread free. For faster performance when there is a que of many files to be encrypted they can be encrypted in parallel threads or if the file is too large, for example over 2GB it can be divided into parts being encrypted in parallel in differed threads, one per part, after the parts are ready they can be united into one file again.

4.6.2 Asymmetric encryption

The asymmetric encryption utilizes the RSA algorithm, using a pair of keys, a public and a private key. However, the RSA and asymmetric cryptography in general is not a good option when the concern is performance, encryption of large data using RSA is considering as a bad practice.

RSA should always used in combination of other algorithms like AES-256, RSA will be used to encrypt only the symmetric key of AES-256.

1. The first step is the generation of keys pair, to do so an asymmetric key algorithm provider should be initialized by specifying the type of algorithm and its padding.
2. Next the keys are extracted by specifying their length, a key pair with length of 2048 bits increases the complexity of encryption but will decrease performance against a pair with length 1024 bits.

3. When the keys are ready is safe to export the public key and distribute it, while the private key must be stored secured. In this application the keys are stored as a pair encrypted with AES-256 with a master password provided by the user. Each time the user uses one of the keys is decrypted on the fly.
4. The private key is alive as plain for a fraction of a second in memory while the public key can be distributed as plain to the target contact.
5. Finally the EncryptSessionKey method will use a public key (the key of the receptor) to encrypt a random generated AES-256 symmetric key, used previously to encrypt the data to be send.

The RSA encrypted AES-256 key with the AES-256 encrypted data will be send as one file in order the receptor can decrypt with the RSA private key the random generated AES-256 key and use it to decrypt the data.

RSA keys are stored secured by a master password for as long the user wishes, however, if user resets or loose this pair, encrypted messages with the public key will no longer be able to be decrypted.

After a pair is regenerated the public key must distributed again to the contacts of the user that wishes to keep communicating with.

Bellow is the block for RSA key generator and encryption:

```
16      public  IBuffer AsymmetricKeysInitialization(UInt32 keyLength)
17      {
18          AsymmetricKeyAlgorithmProvider algorithmProvider = AsymmetricKeyAlgorithmProvider.OpenAlgorithm(AsymmetricAlgorithmNames.RsaPkcs1);
19          CryptographicKey keyPair = algorithmProvider.CreateKeyPair(keyLength);
20          _buffPublicKey = keyPair.ExportPublicKey();
21
22          return keyPair.Export();
23      }
```

RSA keys generator

```
23      public static IBuffer EncryptSessionKey(IBuffer buffSessionKeyToEncrypt, IBuffer buffPublickey)
24      {
25          AsymmetricKeyAlgorithmProvider algorithmProvider = AsymmetricKeyAlgorithmProvider.OpenAlgorithm(AsymmetricAlgorithmNames.RsaPkcs1);
26          CryptographicKey publicKey = algorithmProvider.ImportPublicKey(buffPublickey);
27
28          return CryptographicEngine.Encrypt(publicKey, buffSessionKeyToEncrypt, null);
29      }
```

RSA encryption

As mentioned before RSA should not be used to encrypt large files, since is used to encrypt only keys of an AES-256 algorithm it will always encrypt data with size of 32bytes which takes fractions of a second. There is no need for overload utilizing asynchronous calls. However, it can

be called within a different thread that it will already exist. Minimizing the number of different threads running at the same time is a good practice since they are expensive regarding CPU consumption.

4.7 Decryption

Decryption is divided in two separated parts, symmetric and asymmetric, they are independently from each other so they can be used in many different scenarios.

After decryption the plain data are not yet in their original form, the decrypted file contains information about the original file like the original file name and extension, along with the version of the software that it used to be encrypted.

In this section is analyzed the decryption procedure, while the post – decryption process will be analyzed in the respective section.

4.7.1 Symmetric decryption

Decryption is the opposite procedure of encryption using the AES-256 algorithm, the encrypted file must be provided to the algorithm along with the same key that is used to encrypt the file. All the data manipulation regarding encodings should be used in the opposite order.

Below is a block that decrypts a file.

```
3 references | DESKTOP-1PKLJKN\paris, 12 days ago | 1 author, 3 changes
94  public static byte[] Decrypt(string password, byte[] input)
95  {
96      try
97      {
98          IBuffer key = Convert.FromBase64String(SHA256.GetHashed(password)).AsBuffer();
99          IBuffer m_iv = Convert.FromBase64String("15CV1/ZOnVI3rY4wk4INBg==").AsBuffer();
100
101         SymmetricKeyAlgorithmProvider provider = SymmetricKeyAlgorithmProvider.OpenAlgorithm(SymmetricAlgorithmNames.AesCbcPkcs7);
102         CryptographicKey m_key = provider.CreateSymmetricKey(key);
103
104         IBuffer bufferDecrypt = CryptographicEngine.Decrypt(m_key, input.AsBuffer(), m_iv);
105         return bufferDecrypt.ToArray();
106     }
107     catch (Exception ex)
108     {
109         Debug.WriteLine("AES--->" + ex.Message);
110     }
111     return null;
112 }
113
```

AES-256 decryption that gets the input data as a byte array.

The main difference for the decryption method against the encryption is the call of the Decrypt method provided by the cryptographic engine.

The key and the IV has to be reproduced with the same way during encryption, when the buffer is filled with the plain data the method will return them as a byte array.

Like in the encryption methods there is an overload that utilizes asynchronous thread and streams to minimize the RAM consumption and leave the main thread free.
This method uses a different cryptographic API, it utilizes the same algorithms but provides the CryptoStream class that allows to work with streams of data flowless.

```
1 reference | DESKTOP-1PKLJKN\paris, 12 days ago | 1 author, 1 change
public static async Task<StorageFile> Decrypt(Stream cipherStream, string password, StorageFolder folder)
{
    if (cipherStream == null || cipherStream.Length <= 0)
        throw new ArgumentNullException(nameof(cipherStream));

    StorageFile targetFile = await folder.CreateFileAsync("AESDecrypted");

    byte[] key = Convert.FromBase64String(SHA256.GetHashed(password));
    byte[] iv = Convert.FromBase64String("15CV1/ZOnVI3rY4wk4INBg==");
    using (Aes aesAlg = Aes.Create())
    {
        aesAlg.Key = key;
        aesAlg.IV = iv;
        ICryptoTransform decryptor = aesAlg.CreateDecryptor(aesAlg.Key, aesAlg.IV);
        using (Stream fileStream = await targetFile.OpenStreamForWriteAsync())
        {
            using (CryptoStream cryptoStream = new CryptoStream(fileStream, decryptor, CryptoStreamMode.Write))
            {
                cipherStream.CopyTo(cryptoStream, 16 * 1024);
                cipherStream.Dispose();
            }
        }
    }

    return targetFile;
}
```

AES-256 encryption as an async task that streams the file due encryption procedure.

Decryption work flow:

1. Validation of the input stream (cipher stream), if the stream is null or empty an exception will be thrown that has to be handled by the caller.
2. Preparation of a StorageFile object that will create a temporary file to keep the plain data for further process.
3. Preparation of the key and IV, material that is used to create the decryption procedure.
4. Within a ‘’using’’ key word the AES algorithm is initialized, ‘’using’’ key word ensures that all values will be disposed.
5. Initialization of key and IV for the AES algorithm.
6. Initialization of the decryptor through the relative interface by providing the key and IV of the AES algorithm.
7. Initialization of a file stream that is going to write the plain data to the target file path.

8. Finally, the initialization of the CryptoStream that decrypts the cipherStream, the CryptoStream using the decryptor, will continuously write data to the file stream while the file stream will write them to the hard drive.
   On this part of the code the work flow goes from the bottom to the top, the cipher stream while loads from the hard drive gets copied to the cryptographic stream that decrypts every chunk (buffer with size of 16KB) and feeds every chunk to the file stream that continuously writes the plain data to the hard drive.
9. The method will return the temporary plain file for further process.

While there are more overloads for the decryption method the asynchronous method that utilizes streams is the best option for all cases, moreover there are overloads for different cases of usage, for example when is combined with asymmetric cryptography the key is a random generated and its passed as a byte array to the method while when its used just for symmetric cryptography, instead a byte array key, it takes as argument a password string.

4.7.2 Asymmetric decryption

Asymmetric decryption utilizes like in encryption the RSA algorithm, like mentioned in the encryption section RSA should used to encrypt / decrypt small amount of data since it is very slow against AES.

The below block is the code for RSA decryption:

```
1 reference | DESKTOP-1PKLJKN\paris, 316 days ago | 1 author, 2 changes
public static IBuffer DecryptSessionKey(IBuffer buffKeyPair, IBuffer buffEncryptedSessionKey)
{
    try
    {
        AsymmetricKeyAlgorithmProvider algorithmProvider = AsymmetricKeyAlgorithmProvider.OpenAlgorithm(AsymmetricAlgorithmNames.RsaPkcs1);
        CryptographicKey keyPair = algorithmProvider.ImportKeyPair(buffKeyPair);
        IBuffer reuslt = CryptographicEngine.Decrypt(keyPair, buffEncryptedSessionKey, null);
        return reuslt;
    }
    catch(Exception ex)
    {
        Debug.WriteLine(ex.Message);
    }
    return null;
}
```

For an RSA protected file the following steps are required:

1. When the user receives a file that is encrypted with the public key, the software will load the private key.
2. The user will be asked for the master password that protects the private key, it will decrypt it and it will feed it to the specified method as an IBuffer object.
3. That part of the file contains the RSA encrypted key of the actual file that is encrypted with AES-256 algorithm.

4. In the method is initialized an asymmetric key RSA – pkcs1 algorithm and the private key of the user is imported to the specified algorithm
5. Using the cryptographic engine from Microsoft's API the symmetric key is being decrypted and returns it to the caller code block.

From this point the AES-256 key is used to decrypt the actual file like explained in the section for symmetric decryption.

As mentioned before RSA should be used to encrypt/decrypt small amounts of data in this case the size of the AES-256 key will always be only 32bytes.

4.8 Pre – Encryption and Post – Decryption data process

One of the main goals of this software is to make available advance cryptography easy to use for the end user, it should not require any technical knowledge to be used and more importantly it should not create confusion.

For that purpose, this software implements a custom format of files, every file that is encrypted by this software contains information about the original file that are encrypted as well, and they reveled only when the file has been decrypted.

That information is:

1. File name, the original name of the file so the user will not have to remember the name of the file after the process the software will save the file with the original name after decryption.
2. File extension, the original extension of the file. The encrypted file extensions are jlk and rlk for symmetric and asymmetric encrypted files respectively. After decryption the file will have its original file extension, so the user will not have to remember or change the type of the file.
   For example, a PDF AES-256 encrypted file will have as an extension jlk but when its decrypted it will be a PDF file again.
3. The version of the software, this will help in backwards compatibility on future versions. This way the only part of the code that needs to be the same is how those information will be obtained allowing the developers to change the encryption mechanisms and keep compatibility with older files.

4.8.1 Pre – Encryption data manipulation

Before the encryption of a file starts there is process of the plain file that is taking place. Of course, this is done on a copy of the file to avoid the corruption of the original file if any unexpected event like a shutdown by power failure happens.

There are two types of pre – encryption data manipulation systems, one for AES-256 files and one for RSA encrypted files.

AES-256 Pre – Encryption process

This system will process the file while is streamed to be written as a copied file, the copied file will already have the extra information as plain data using a formula that it will be analyzed later.

One important part of this process is the time and the recourses that are required and in this software they are minimized in favor of RAM, storage and speed.

When the target file is selected by the user and the password is provided the software will initialize the prefix of the file and it will attach in at the begging of the stream while the streaming file is written to the hard drive as a custom file.

After this process the custom file will be streamed once again into the AES-256 algorithm and while encrypting the file it will be written once again into the hard drive, as soon as the last step is completed the custom plain file will be deleted to free the occupied space. This way the user will still have the original plain file and a copy of it as a custom file with the prefix that is encrypted.

The prefix data for the AES-256 files is created by the following formula.

[JLK][v]<Software version>[/v][f]<FE>[/f][250 * '\0'][/JLK]

[JLK] : The symbol that indicates the start of the customized file.

[v]: The symbol that indicates the start of the software version number.

[/v]: The symbol that indicates the end of the software version number.

[f]: The symbol that indicates the start of the original file name and extension (F and E respectively) that are a string without spaces.

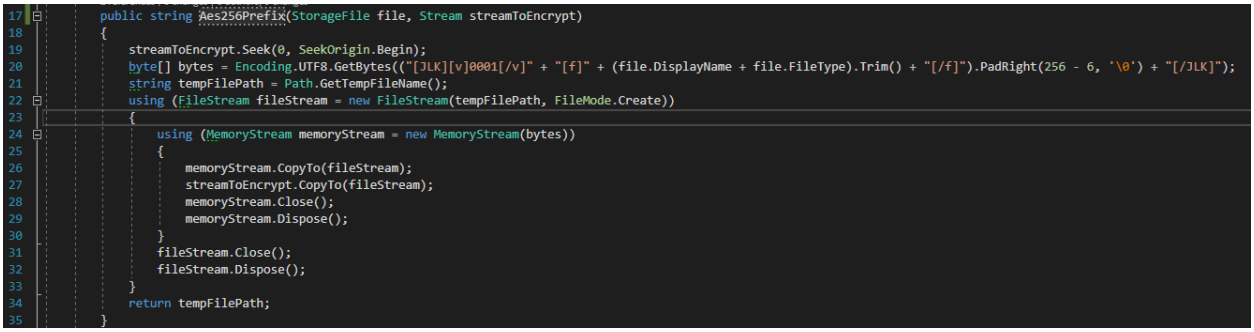
[/f]: The symbol that indicates the end of the original file name and extension.

[250 * '\0']: After the symbol [/f] 250 spaces occupied using the character ‘\0’ to increase the complexity for security attacks.

[/JLK]: The symbol that indicates the end of the AES-256 attack.

The format of the prefix is inspired from XML syntax, usage of XML is not efficient for this case since it would require to convert the string to a Byte64 encoding that requires much more space for the same data.

Below is the block that will create the custom file by adding the prefix:



```
public string Aes256Prefix(StorageFile file, Stream streamToEncrypt)
{
    streamToEncrypt.Seek(0, SeekOrigin.Begin);
    byte[] bytes = Encoding.UTF8.GetBytes(("[JLK][v]0001[/v]" + "[f]" + (file.DisplayName + file.FileType).Trim() + "[/f]").PadRight(256 - 6, '\0') + "[/JLK]");
    string tempFilePath = Path.GetTempFileName();
    using (FileStream fileStream = new FileStream(tempFilePath, FileMode.Create))
    {
        using (MemoryStream memoryStream = new MemoryStream(bytes))
        {
            memoryStream.CopyTo(fileStream);
            streamToEncrypt.CopyTo(fileStream);
            memoryStream.Close();
            memoryStream.Dispose();
        }
        fileStream.Close();
        fileStream.Dispose();
    }
    return tempFilePath;
}
```

This method is designed to create the custom file as the file is loading and written from and to the hard drive, when the process is complete the method will return the file path of the custom file.

1. The stream to be encrypted (the original plain file) position will be restored to its beginning.
2. The prefix will be initialized and it will be converted to a byte array, since its size its only few bytes the RAM consumption is not a concern.
3. A temporary full file path will be created at the application temporary folder that exists for sort living files.
4. Within the ‘’using’’ key word a file stream will create the temporary file and it will start to write data in it.
5. A memory stream will be initialized with the prefix byte array.
6. Using the extension of streams CopyTo(Stream stream) will copy at the beginning of the file stream to be written the prefix from the memory stream.
7. Using the same extension on the original plain file’s stream, it will copy the plain data of the actual file starting at the position where the prefix ends.
8. While the steps 6 and 7 are happening the file stream writes the data to the hard disk
9. The memory stream is closing and disposing any data and resources as soon as it reach the end of the prefix.
10. The file stream is closing and disposing any data and resources as soon as the custom file is written to the hard drive.
11. Finally the method will return to the callred code block the temporary file path of the custom file to encrypt it.
12. After encryption this file must be deleted, if not the operation system will automatically delete it at some point in the future.

RSA pre – Encryption process

The RSA custom file system is used on top of the AES-256 system, it simply adds a layer in front of the AES-256 prefix.

The formula is described below.

<RSA encrypted AES-256 key bytes>[RLK][JLK][v]<Software version>[/v][f]<FE>[/f][250 * '\0'][/JLK]

The file always will have in its beginning the RSA encrypted AES-256 key bytes

[RLK]: The symbol that indicated that the RSA encrypted AES-256 key byte array ends.

Below is the block that creates the custom file.

```
1 reference | 0 changes | 0 authors, 0 changes
public async Task<StorageFile> RsaPrefix(StorageFolder storageFolder, StorageFile aesEncryptedFilePath, byte[] encryptedAesKeyBytes)
{
    byte[] bytes = Encoding.UTF8.GetBytes($"{Convert.ToBase64String(encryptedAesKeyBytes)}[RLK]");

    StorageFile newFile = await storageFolder.CreateFileAsync("temp",CreationCollisionOption.GenerateUniqueName);
    using (Stream fileStream = await storageFolder.OpenStreamForWriteAsync("temp",CreationCollisionOption.ReplaceExisting))
    {
        using (MemoryStream memoryStream = new MemoryStream(bytes))
        {
            using (Stream readerFileStream = await aesEncryptedFilePath.OpenStreamForReadAsync())
            {
                memoryStream.CopyTo(fileStream);
                readerFileStream.CopyTo(fileStream);
            }
        }
    }

    await aesEncryptedFilePath.DeleteAsync();
    return newFile;
}
```

1. The prefix up to the symbol [RLK] will be initialized and it will be converted as a byte array.
2. A file will be created by the name ‘temp’ this will have the data of the customized format.
3. Within ‘using’ key word the file will be opened as a stream for writing data.
4. A memory stream will stream the prefix.
5. A file stream will start reading the AES-256 encrypted file from the temporary location.
6. The memory stream will copy the prefix at the beginning of the file stream to be written.
7. The reader stream will copy the AES-256 encrypted file to the file stream to be written where the RSA prefix ends.
8. When the process ends the AES-256 temporary file will be deleted to free the occupied space.
9. Finally, the method will return the ‘temp’ file details.

After this process the file is ready to be renamed by the users provided details.

Once again the process is happening on a copy of the AES-256 encrypted file to avoid file corruption in the case of an unexpected event like power failure.

This custom format for RSA protected files allows to utilize RSA and AES-256 at the same time, by doing so we have the advantages from both algorithms. AES-256 provide speed and strong security for large files while RSA will make sure that only the receiver will be able to decrypt the AES-256 key and therefore the entire file.

A file after it has been encrypted using asymmetric cryptography will have the following format:

<RSA encrypted AES-256 key bytes>[RLK][JLK][v]<Software version>[/v][f]<FileName.Extention>[/f][250 * '\0'][/JLK]<AES-256 encrypted file bytes>

4.8.2 Post – Decryption data manipulation and data restoration to the original format

Decrypting a file in this software is not enough to bring it into its original state. The files contain more information regarding the file as it has been explained to the last section. The plain file need further process in order to ger the original file's information and finally the actual plain file.

AES – 256 Post decryption process

Decrypting a password protected file will expose the custom formatted file. To get the original file the it needs a process to read the bytes of the prefix as described in the last section.
After having the plain custom file the software will read the bytes up to the point of the [\JLK] symbol, those bytes will be converted into a utf-8 string and the original file information will be extracted.

Below is the block of code that extracts these information:

```
public async Task StoreDecryptedFile(StorageFile tempFileName, StorageFolder folderLocation)
{
    byte[] storedInfoBytes = new byte[256];
    using (Stream fileStream = await tempFileName.OpenStreamForReadAsync())
    {
        fileStream.Read(storedInfoBytes, 0, storedInfoBytes.Length);
        string storedInfo = Base.enc8.GetString(storedInfoBytes.SubArray(0, 256));
        string fileName = storedInfo.Substring(storedInfo.IndexOf("[f]", StringComparison.OrdinalIgnoreCase) + 3,
            storedInfo.IndexOf("[/f]", StringComparison.OrdinalIgnoreCase) - (storedInfo.IndexOf("[f]", StringComparison.OrdinalIgnoreCase) + 3));
        string version = storedInfo.Substring(storedInfo.IndexOf("[v]", StringComparison.OrdinalIgnoreCase) + 3,
            storedInfo.IndexOf("[/v]", StringComparison.OrdinalIgnoreCase) - (storedInfo.IndexOf("[v]", StringComparison.OrdinalIgnoreCase) + 3));
        StorageFile newFile = await folderLocation.CreateFileAsync(fileName, CreationCollisionOption.GenerateUniqueName);

        fileStream.Seek(256, SeekOrigin.Begin);
        using (Stream storingFileStream = await newFile.OpenStreamForWriteAsync())
        {
            await fileStream.CopyToAsync(storingFileStream);
        }
    }
}
```

Once again the process will take place on copy of the file, of course the copied file will be already plain and stored in the special temporary folder that will automatically delete the files when the operating systems decides to. Working on a copy of the original encrypted file will ensure that the file will remain intact.

Steps that are taking place in this method:

1. A stream will be initialized to open and read the temporary plain file.
2. The prefix has the size of 256bytes, these bytes are extracted from the stream, converted to a string with UTF -8 encoding and assigned into a variable of type string. This string contains the entire prefix.
3. Using substring we can specify the range of information we want to extract from the prefix by using the symbols as marking points, an important step is to use the string comparison enum to disable the character case sensitivity. On future versions there might be more symbols and more complex, this way we avoid mistakes from the side of the developer, like adding a symbol with capital and low case characters. Without the string comparison option the software would not recognize the symbols.
4. Initialization of string variables that will be assigned with the properties of the prefix. Those are the file name with the extension and the version of the software.
5. Initialization of a new file with the file name and extension that have been extracted from the prefix. This will create an empty file that it will have the name and extension of the original file. The usage of the enum CreationCollisionOption with the option GenerateUniqueName it will ensure that if already a file with the same name and extension exists in the same location it will not override it, instead it will have a number in a parenthesis at the end of the file name, where the number is the counter of the files with the same name. For example file(1).txt if there is a file.txt
6. Next the file stream has to be reset its position where the bytes of the original file begin, in our custom file that will be at the position of 256 bytes with a reference at the beginning of the stream.
7. Finally, a new file stream is initialized to write the rest of the data into the new file, this file will be the original plain file with the original name and extension exactly as it was before the encryption.

Note that the temporary file will have to be deleted to avoid unnecessary space consumption on the hard drive, however for this case this should be done on the caller's side, the future developers might want to process the file differently.

RSA Post decryption process

STATE 1

As mentioned before the RSA prefix is on top of the AES-256 prefix, there fore it requires more steps to retrieve the original file after encryption. In this case there are two states of decryption, first the RSA decryption and then the AES-256 decryption.

Once the receiver of an RSA protected file wants to decrypt the file with the unique private key, the software will extract the section of the file that contains the RSA encrypted value, this value in plain state is an AES-256 key that was randomly generated before. As soon as the key is plain the software will use it to decrypt the rest of the file which contains the AES-256 prefix and the actual file.

Below is the code block that extracts the RSA encrypted, AES-256 key.

```
1 reference | DESKTOP-1PKLJKN\paris, 27 days ago | 1 author, 3 changes
283 private async Task<IBuffer> GetAesDecryptedKey(StorageFile file, string password)
284 {
285     string symbol = "[RLK]";
286     int foundIndex = -1;
287     string value = string.Empty;
288     StringBuilder builder = new StringBuilder();
289     Stream stream = await file.OpenStreamForReadAsync();
290     foreach (var chars in ReadCharsByChunks(16, stream))
291     {
292         builder.Append(chars);
293
294         var existing = builder.ToString();
295
296         if ((foundIndex = existing.IndexOf(symbol, StringComparison.OrdinalIgnoreCase)) >= 0)
297         {
298             builder.Remove(foundIndex, symbol.Length+3);
299             value = builder.ToString();
300             break;
301         }
302     }
303     return await GetAesKeyFromRsa(password, Convert.FromBase64String(value));
304 }
```

Because the size of the random generated password for the AES-256 encryption is unknown, we need an algorithm that it will detect the [RLK] symbol within the stream.
In this case the stream that reads the file will be fed into a method named ReadCharsByChunks, this method will read the stream in chunks of 16bytes and will return them up to the point where the [RLK] symbol exists.

Below is the block of the ReadCharsByChunks method:

```
2 references | DESKTOP-1PKLJKN\paris, 27 days ago | 1 author, 1 change
306  private IEnumerable<string> ReadCharsByChunks(int bufferSize, Stream filePath)
307  {
308
309      byte[] buffer = new byte[bufferSize];
310      int currentRead;
311      while ((currentRead = filePath.Read(buffer, 0, bufferSize)) > 0)
312      {
313          yield return Encoding.UTF8.GetString(buffer, 0, currentRead);
314      }
315  }
```

The most important part in this block is the return type of the method, IEnumerable is an interface that every collection in .NET implements, using this type enables the key word ‘’yield’’ in frond of the return key word. This practically means that while the stream reads chunks of bytes the method will return this chunk of bytes to the caller until the stream ends or until the caller decides to stop. This technique is named as ‘’lazy loading’’, in this case a collection is filled with data when is needed for as long is needed.

Moving on to the previews method there is a loop that will keep asking for the next item as soon it finish with the last item in the collection, the loop (caller) will get one chunk per iteration each time it starts over until it pass the condition and breaks out of the loop.

In other words the software doesn’t need to get through the entire file to detect the symbol but only until to the point that it will meet it in the stream as it reads the file.

All this would not make sense if the index of the stream that fount the symbol is not stored in a variable to continue the second state of the process.

The value that is extracted from this process is the RSA prefix, as soon the software obtains the prefix it will decrypt it using the private key.

Below is the block that retrieves the plain random generated AES-256 password:

```
1 reference | DESKTOP-1PKLJKN\paris, 27 days ago | 1 author, 3 changes
273  private async Task<IBuffer> GetAesKeyFromRsa(string password, byte[] encryptedAesKey)
274  {
275      StorageFolder folder = ApplicationData.Current.LocalFolder;
276      StorageFile privateKeyFile = await folder.GetFileAsync("JLockParameters.pkl");
277      string privateKey = await FileIO.ReadTextAsync(privateKeyFile);
278      byte[] plainPrivateEkyBytes = AES.Decrypt(password, Convert.FromBase64String(privateKey));
279
280      return J_Lock.Security.Cryptography.Asymmetric.RSA.DecryptSessionKey(plainPrivateEkyBytes.AsBuffer(), encryptedAesKey.AsBuffer());
281  }
```

The argument named password is the password for the RSA private key, remember the private key is protected as well with AES-256 encryption and the user’s master password, while the byte array is the RSA encrypted password for the actual file.

The first state is completed when the RSA value is decrypted, and the method returns the plain random generated password that is used to encrypt the actual file.

STATE 2

Using the plain password, the software will decrypt the rest of the file as it appears in the image bellow:

```
382         if (isRsa)
383         {
384             IBuffer key = await GetAesDecryptedKey(file, password);
385             StorageFile outputFile = await GetAesEncryptedFile(file, folder);
386             tempFileName = await AES.Decrypt(await outputFile.OpenStreamForReadAsync(), key.ToArray(), folder);
387             await outputFile.DeleteAsync();
```

The plain key will be used to the AES.Decrypt method that will store the file in the special temporary folder as explained to the respective section.

All the steps that taking place in the post -decryption process for AES-256 encrypted files will take place here as well. The respective prefix will be extracted, and the information will be used to build the original plain file. As soon as this process ends the temporary file will be deleted.

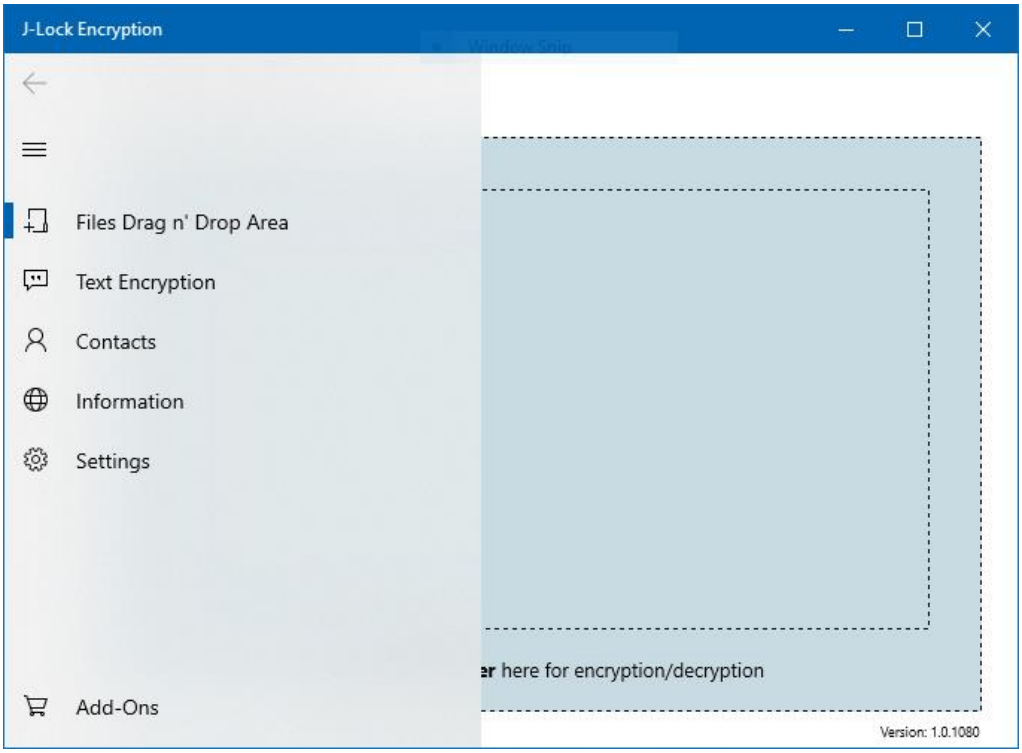
4.9 User interface and code behind

Windows universal platform utilizes XAML to describe in code the graphical user interface, it is a very powerful engine. The true potential of XAML in Windows universal platform and WPF is when is used with the MVVM (model – view – view model) design pattern and bindings that separates entirely the user interface from the code, however this is out of the scope of this thesis. Instead in this project the entire logic is implemented in the code behind.

In the following sections will be analyzed the user interface along with the software logic since there are tight together as user interface and code behind.

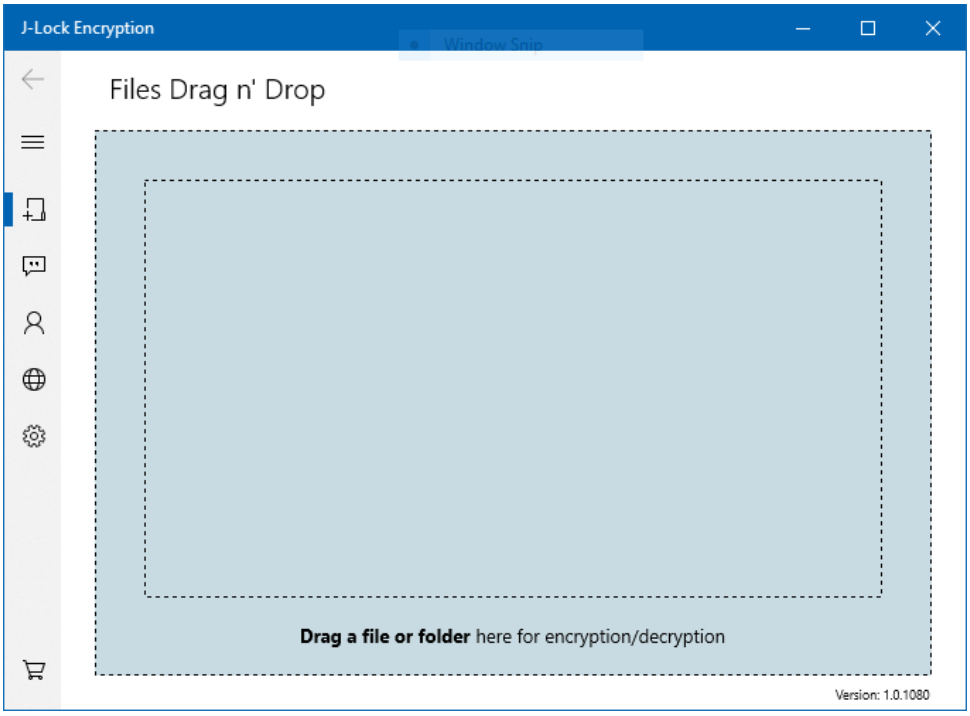
4.9.1 Hamburger menu

The application has a modern design approach, the main menu design is called hamburger menu it minimizes the space that occupies on the window while the user can expand it to view more details with text for each choice.



Sandwich menu expanded

Users can click on the three lines symbols to expand the menu, by clicking the arrow on top it will minimize as appears in the image below.



Hamburger menu minimized

Users can still navigate through the software by clicking on the menu items

The hamburger menu is written in XAML and placed in the Grid context of the main page and its parted from two different components.

1. Within the grid the component NavigationView has to be initialized in XAML code.

```
<NavigationView x:Name="NavView" HorizontalAlignment="Stretch" Margin="0,0,0,0"
                VerticalAlignment="Stretch" Loaded="NavView_Loaded"
                SelectionChanged="NavView_SelectionChanged" ItemInvoked="NavView_ItemInvoked">
```

2. Within the initialization tag and the finalization tag are placed the rest of the components, the next component is the icons list items. Those items are the ‘’buttons’’ that users will see and select to navigate.

```
<NavigationView.MenuItems>
    <NavigationViewItem x:Uid="AppsNavItem" Content="Files Drag n' Drop Area" Tag="files">
        <NavigationViewItem.Icon>
            <SymbolIcon Symbol="NewFolder"/>
        </NavigationViewItem.Icon>
    </NavigationViewItem>
    <NavigationViewItem x:Uid="ContactsItem" Content="Contacts" Tag="contacts">
        <NavigationViewItem.Icon>
            <SymbolIcon Symbol="Contact"/>
        </NavigationViewItem.Icon>
    </NavigationViewItem>
    <NavigationViewItem x:Uid="GamesNavItem" Content="Text Encryption" Tag="texts">
        <NavigationViewItem.Icon>
            <SymbolIcon Symbol="Comment"/>
        </NavigationViewItem.Icon>
    </NavigationViewItem>
    <NavigationViewItem x:Uid="AppsNavItem3"  Content="Information" Tag="about">
        <NavigationViewItem.Icon>
            <SymbolIcon Symbol="Globe"/>
        </NavigationViewItem.Icon>
    </NavigationViewItem>
    <NavigationViewItem x:Uid="SettingsItem" Content="Settings" Tag="settings">
        <NavigationViewItem.Icon>
            <SymbolIcon Symbol="Setting"/>
        </NavigationViewItem.Icon>
    </NavigationViewItem>
</NavigationView.MenuItems>
<NavigationView.HeaderTemplate>
```

3. Microsoft’s framework will generate the relation between the view components and the code behind, this generated code is nothing more than an event handler that will get triggered each time the user selects an item in the sandwich menu.

```
((global::Windows.UI.Xaml.Controls.NavigationView)this.NavView).Loaded += this.NavView_Loaded;
((global::Windows.UI.Xaml.Controls.NavigationView)this.NavView).SelectionChanged += this.NavView_SelectionChanged;
```

   Each time the user changes the selection it will invoke a method named NavView_SelectionChanged, this method contains developers code that describes what view will be initialized.

4. The invoked method contains a simple switch to determinate where the user wants to navigate within the software.
   In order to have the utilities views (Drag n Drop area, settings, contacts, etc.) separated from other views like Addons (features that will be available in the future) the invoked method has a simple if statement.

```
1 reference | 0 changes | 0 authors, 0 changes
75  private void NavView_SelectionChanged(NavigationView sender, NavigationViewSelectionChangedEventArgs args)
76  {
77      if (args.IsSettingsSelected)
78      {
79          NavView.Header = "Purchase Add-Ons";
80          ContentFrame.Navigate(sourcePageType: typeof(Pages.AddOns));
81      }
82      else
83      {
84          NavigationViewItem item = args.SelectedItem as NavigationViewItem;
85          NavView_Navigate(item);
86      }
87  }
```

In code's scope the addons view is considering as settings view, however is decided that the application settings should be a selection along with the rest of the utilities and that the addons should appear at the bottom. If user selects the bottom icon (addons) the property IsSettingsSelected is true and using the content frame we call the method navigate with the type of Pages.Addons as argument. It will initialize an instance of the page and it will render it over the main frame.

5. If the selection is another button the selected item will be casted as NavigationViewItem, it is the type that we used to initialize the items in the hamburger menu. Then the item is passed as an argument in a private method named NavView(NavigetionViewItem item).
6. Each item in the XAML code has a tag name, this name is a property of the NavigationViewItem object, this property is used in a switch.

```
2 references | DESKTOP-1PKLJKN\paris, 324 days ago | 1 author, 3 changes
89  private void NavView_Navigate(NavigationViewItem item)
90  {
91
92      switch (item.Tag)
93      {
94          case "files":
95
96              NavView.Header = "Files Drag n' Drop";
97              ContentFrame.Navigate(sourcePageType: typeof(Pages.MainFiles));
98              Pages.MainFiles var = (Pages.MainFiles)ContentFrame.Content;
99              var.SetRedirectPage(Redirect);
100
101             break;
102
103         case "texts":
104             NavView.Header = "Text Cryptography";
105             ContentFrame.Navigate(sourcePageType: typeof(Pages.MainText));
106             break;
107
108         case "contacts":
109             NavView.Header = "Contacts";
110             ContentFrame.Navigate(sourcePageType: typeof(Pages.Contacts));
111             break;
112
113         case "settings":
114             NavView.Header = "Settings";
115             ContentFrame.Navigate(sourcePageType: typeof(Pages.Settings));
116             break;
117
118         case "about":
119             NavView.Header = "Information";
120             ContentFrame.Navigate(sourcePageType: typeof(Pages.About));
121             break;
122     }
123 }
```

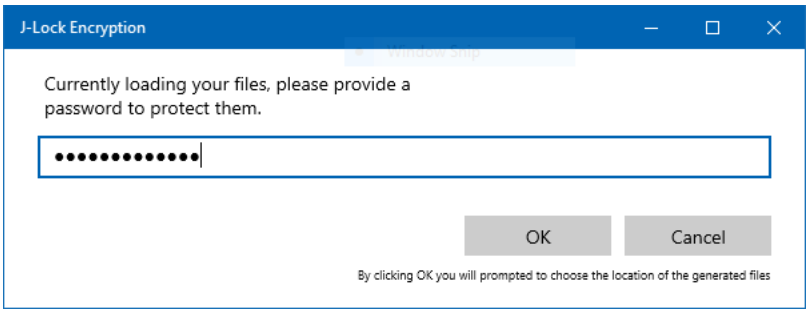
Depending on the value the content frame will initialize the respective page.

4.9.2 Drag n drop view

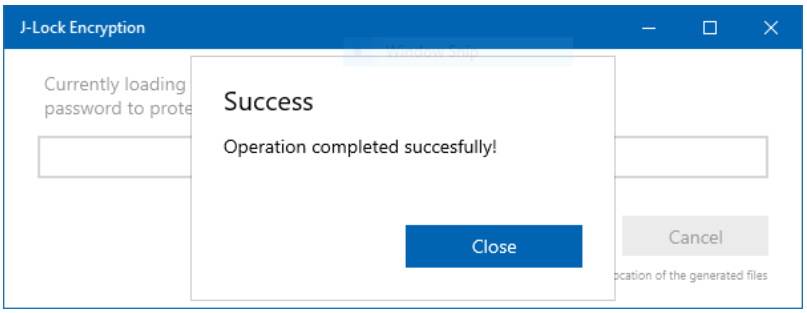
Drag n drop view provides a large area that users can just drag an folder that contain files or multiple files and those files will be encrypted using the AES-256, this action will detect the file paths for all files in order to read them and encrypt them. By doing this action the application will ask for a password that will protect those files, this way the user can use one password for multiple files at once.

In the image above is appears that the users has dragged three PDF files, the drag n drop area it detects the action and will get the links (file paths in this case) of the respective files.

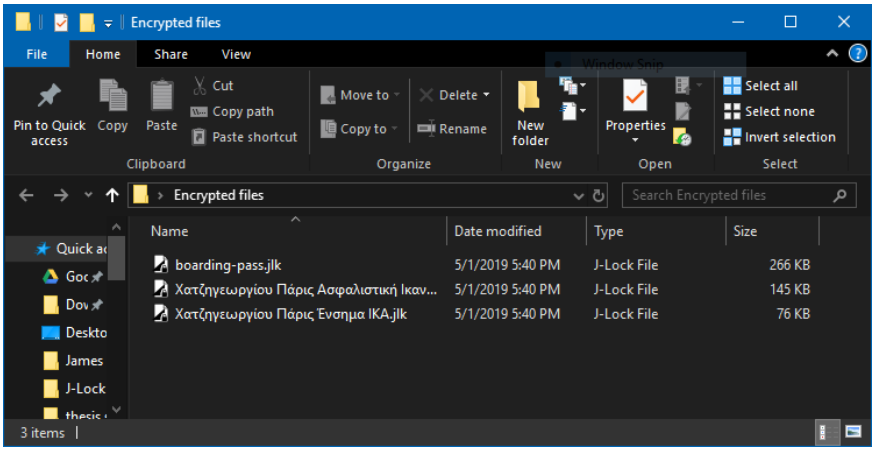
Then it will ask from the user a password to protect those files.



When user click OK a file explorer will be prompted to select the location that the encrypted files will be stored. Once the operation is complete a respective message will be displayed to notify the user.

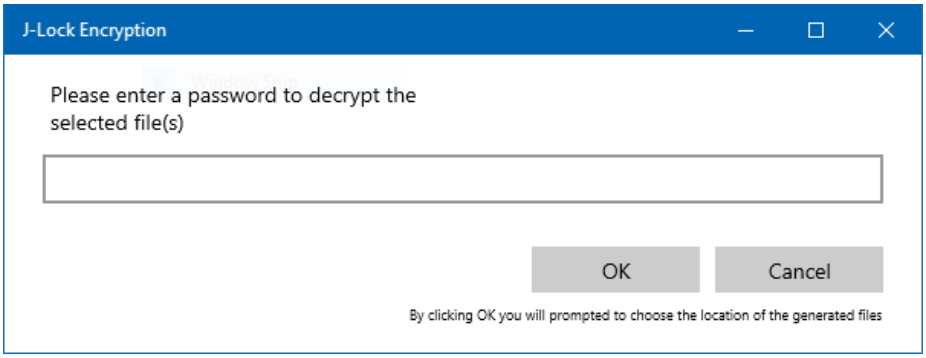


The encrypted files will be available at the selected location with the original name but different icon and file extension.



The original files are intact in their location, it is better to let the user decide what to do with the plain files, in other words these files are encrypted copies.

There are two ways to decrypt those files, drag n drop area is one of them, it can detect if a file is plain or if it is already encrypted and it will operate accordingly. If the user drags n drop those files the application will ask for the password that protects them, if the password is correct the decryption will operate as expected, but if the password is wrong a respective message will be displayed to the user.



The application declares the file types it produces to the system when is installed, this allows the developer to decide what will happen if the user open a JLK or RLK file. Doing so it will run the software calling the code that initialize the decryption procedure. Double clicking an encrypted file will bring the same view.

The drag n drop area is a component provided by Microsoft's framework, it redirects the file paths that is reading to the code and is triggered by the drop event, thus when the user releases the mouse button.

```
1 reference | 0 changes | 0 authors, 0 changes
47  private async void Grid_Drop(object sender, DragEventArgs e)
48  {
49      ((Grid)sender).Background = new SolidColorBrush(Color.FromArgb(a: 255, r: 201, g: 219, b: 226));
50
51      if (e.DataView.Contains(StandardDataFormats.StorageItems))
52      {
53          var items = await e.DataView.GetStorageItemsAsync();
54
55          if (items.Count == 0)
56              return;
57
58          if(function!=null)
59              await function(items);
60
61          redirect?.Invoke();
62      }
63  }
```

The event argument has a property named DataView that can be used to check the items from the event, in other words on line 51 the items that are dropped are casted to the selected type that in this case is StorageItems, a class that contains information about a file that is stores on the machine. Then we can get those items and process them, a validation ensure there are files.

Both redirect and function has been declared as delicate, short of like using a pointer to a specific location of the memory that is used by those objects but also ensures that is thread safe.

```
17  public delegate Task<bool> CalledFunction(IReadOnlyList<IStorageItem> items);
18  public delegate void Redirect();
```

The CalledFunction is a type that can be assigned with any method and it can be used as a Task, when is called it will start a new thread that will execute the assigned method and that method returns a Boolean value (the developer can declare any return type for a Task).

Within its arguments we declare a collection that is only for read and the items in it are IStorageItem , this is an interface that the StorageItem implements. The method function(items) at line 59 of the first image will fill the collection with StorageItem objects, those objects as said before have all the information we need for the files the user dropped in the area.

By calling the redirect?.invoke(); another object from a different scope will be called, in our case this is the Password base that holds the code to encrypt or decrypt the files in the collection.

```
6 references | 0 changes | 0 authors, 0 changes
25  public sealed partial class MainFiles : Page
26  {
27      public static Redirect redirect = null;
28
        0 references | 0 changes | 0 authors, 0 changes
29      public MainFiles()
30      {
31          this.InitializeComponent();
32          this.dropbox.function = Base.Callback;
33
34      }
35
        3 references | 0 changes | 0 authors, 0 changes
36      protected override void OnNavigatedTo(NavigationEventArgs e)
37      {
38          Base.files = new List<StorageFile>();
39      }
40
        1 reference | 0 changes | 0 authors, 0 changes
41      public void SetRedirectPage(Redirect red)
42      {
43          redirect = red;
44          this.dropbox.redirect = redirect;
45      }
46  }
```

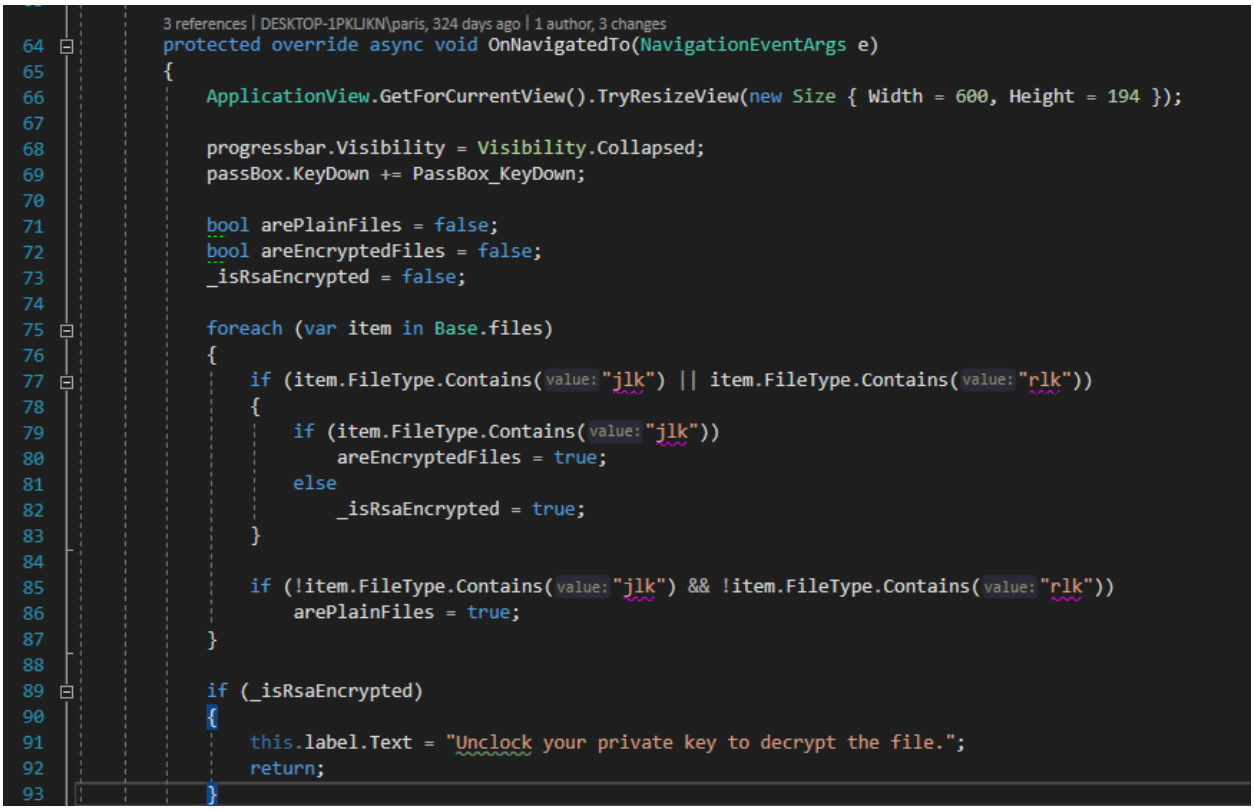
In the Drag n Drop area class. The filed function will be initialized by getting assigned a method named CallBack.

```
2 references | 0 changes | 0 authors, 0 changes
public static async Task<bool> Callback(IReadOnlyList<IStorageItem> items)
{
    foreach (var item in items)
    {
        if (item.IsOfType(StorageItemTypes.File))
        {
            Base.files.Add(item as StorageFile);
        }
        else if (item.IsOfType(StorageItemTypes.Folder))
        {
            var actual_files = await (item as StorageFolder).GetFilesAsync();
            foreach (var item1 in actual_files)
            {
                Base.files.Add(item1 as StorageFile);
            }
        }
    }

    return true;
}
```

This method will check if the dropped item is a folder or just files, in both cases it will read all the items that are redirected from the drag n drop area and it will fill a collection that is used later for encryption or decryption.

The redirection as said before is towards the password view, a page that is initialized as soon as the CalledFunction task is completed, in other words when the drag n drop area get the information about the dropped files, then the Password page is initialized as a child of the drag n drop area. This new instance of the Password object has access to the collection that the CalledFunction filled.

The Password view is the dialog that asks to provide a password either for encryption or decryption each specific instance knows which case is since initialization.



```
3 references | DESKTOP-1PKLJKN\paris, 324 days ago | 1 author, 3 changes
protected override async void OnNavigatedTo(NavigationEventArgs e)
{
    ApplicationView.GetForCurrentView().TryResizeView(new Size { Width = 600, Height = 194 });

    progressbar.Visibility = Visibility.Collapsed;
    passBox.KeyDown += PassBox_KeyDown;

    bool arePlainFiles = false;
    bool areEncryptedFiles = false;
    _isRsaEncrypted = false;

    foreach (var item in Base.files)
    {
        if (item.FileType.Contains(value: "jlk") || item.FileType.Contains(value: "rlk"))
        {
            if (item.FileType.Contains(value: "jlk"))
                areEncryptedFiles = true;
            else
                _isRsaEncrypted = true;
        }

        if (!item.FileType.Contains(value: "jlk") && !item.FileType.Contains(value: "rlk"))
            arePlainFiles = true;
    }

    if (_isRsaEncrypted)
    {
        this.label.Text = "Unclock your private key to decrypt the file.";
        return;
    }
```

OnNavigateTo method is called when the instance is created after a navigation or redirect call, up to the lice 93 is the code that checks what type of files are in the collection. If there are mixed type of files (plain and encrypted) the application will ask to chose how to proceed. User will have the option to decrypt the encrypted files or encrypt the plain files.

4.9.3 Contacts view

Contacts view will display contacts that the user has imported from Microsoft's People application then the user can import for this contact the public key that the specific contact has provided.

By selecting a contact, the application will detect if there is an imported public key, if not it will ask to provide one.

Provided that the selected contact has a public key, it will call the RSA algorithm to encrypt a random generated password that it will encrypt the file using AES-256 encryption the final file will contain the RSA encrypted key along with the original file.

The graphical interface is composed by a few controls, there is a list view tha will hold instances of a custom control to display a contact item, a process bar that displays the process of encryption and three simple buttons to manage the contacts (add, delete and import a public key).

Below is the XAML code that compose the contacts view.

```
<Grid Background="{ThemeResource ApplicationPageBackgroundThemeBrush}" HorizontalAlignment="Stretch" VerticalAlignment="Stretch"  Margin="0,0,0,0">
    <ProgressBar x:Name="progressBar" Height="24" VerticalAlignment="Center" Width="1480" RenderTransformOrigin="0.5,0.5" Visibility="Collapsed" Margin="0,0,0,0">
        <ProgressBar.RenderTransform>
            <CompositeTransform ScaleY="-1"/>
        </ProgressBar.RenderTransform>
    </ProgressBar>
    <TextBlock x:Name="waitText" Text="Please wait while the files are encrypted..."
               TextWrapping="Wrap" Width="261" RenderTransformOrigin="0.507,0.579" Margin="0,0,0,0" Visibility="Collapsed" FontWeight="Bold" FontSize="20"/>
    <ListView x:Name="contactsListView" HorizontalAlignment="Stretch" VerticalAlignment="Stretch" Margin="0,0,0,0">
    </ListView>
    <StackPanel HorizontalAlignment="Stretch" Height="auto" VerticalAlignment="Bottom"
                Width="auto" Orientation="Horizontal" Margin="0,0,0,0" FlowDirection="RightToLeft">
        <Button Content="Remove" HorizontalAlignment="Stretch" VerticalAlignment="Stretch" Margin="0,0,025,0" Padding="30,4,30,4" Click="Button_Click_1" />
        <Button x:Name="importKeyButton" Content="Import key" HorizontalAlignment="Stretch"
                VerticalAlignment="Stretch" Margin="0,0,025,0" Padding="30,4,30,4" Click="Button_Click_2"  />
        <Button Content="Add" HorizontalAlignment="Stretch" VerticalAlignment="Stretch" Margin="0,0,25,0" Padding="30,4,30,4" Click="Button_Click" />
    </StackPanel>
</Grid>
```

Below is tha XAML code that compose the contact item.

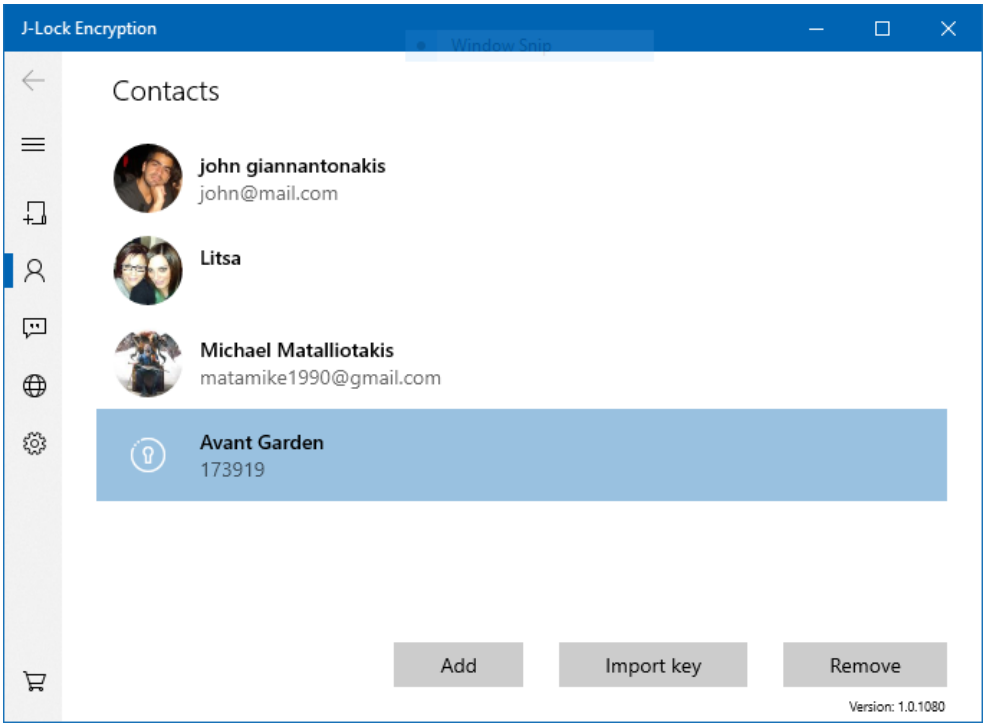
```
<Grid>
    <StackPanel Orientation="Horizontal" Height="64" AutomationProperties.Name="">
        <Ellipse Height="48" Width="48" VerticalAlignment="Center" Margin="0,8,0,8">
            <Ellipse.Fill>
                <ImageBrush x:Name="image" ImageSource="ms-appx:///Assets/SmallTile.scale-125.png" />
            </Ellipse.Fill>
        </Ellipse>
        <StackPanel Orientation="Vertical" VerticalAlignment="Center" Margin="12,0,0,0">
            <TextBlock x:Name="name" Text=""  Style="{ThemeResource BaseTextBlockStyle}" Foreground="{ThemeResource SystemControlPageTextBaseHighBrush}" />
            <TextBlock x:Name="info1" Text="" Style="{ThemeResource BodyTextBlockStyle}" Foreground="{ThemeResource SystemControlPageTextBaseMediumBrush}"/>
        </StackPanel>
    </StackPanel>
</Grid>
```

The contact items are a stack panel with text and an ellipse shape that can hold an image, the idea is that each instance will have the full name of the contact, the email and a photo. All this information and the image will be obtained from the People app in windows.

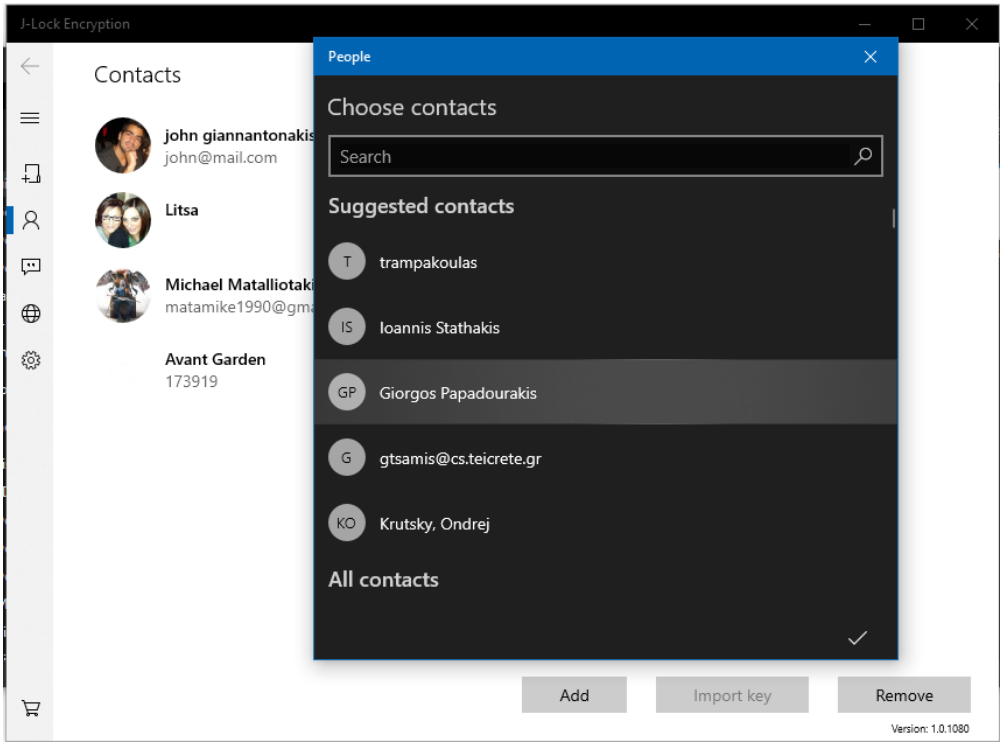
Moreover, to minimize the storage requirements the app itself holds stored only the unique ID of a People app contact and its public key, this part will be explained later on the code behind.

The visual result of the contacts view tries to adapt a modern approach inspired by modern apps contacts view like the Messenger of Facebook.

Bellow the image displays the contact view, if the list is longer that the window size it will allow to scroll through all the contacts, the buttons and the list view are not in the same container so the button will stay at the same place as the user scrolls through the contacts. Another detail is that if the app does not find any photo of a contact it will add the app’s icon as a default image like in the last contact item in the image bellow.



When a user clicks on the Add button the People app will open a contact selector window that lays over the contacts view as a child window. Users can select multiple contacts to add at once.



The code behind is responsible for these actions, when the view is initialized, as soon as the constructor is called it will check if there is any imported contact. If there is it will find the actual contacts by their unique ID in Windows People app, it will read their info and image and it will initialize for each one a contact view item to add it in the list. If there is not any imported contact the view will be empty.

```
0 references | DESKTOP-1PKLJKN\paris, 62 days ago | 1 author, 5 changes
public Contacts()
{
    this.InitializeComponent();
    importKeyButton.IsEnabled = false;
    NavigationCacheMode = NavigationCacheMode.Required;
    InitializeContactsAsync();
    if (Base.ContactByListViewItem.Keys.ToList().Count > 0 && contactsListView.Items.Count == 0)
    {
        foreach (ListViewItem item in Base.ContactByListViewItem.Keys.ToList())
        {
            contactsListView.Items.Add(item);
        }
    }
    this.Unloaded += Contacts_Unloaded;
}
```

The method InitializeContactsAsync() contains the code that will prepare a Dictionary (ContactByListViewItem) object that will contains the unique ID of a contact as keys and a ListViewItem object as value.

To do so on start up the software loads a serialized list that contain the unique ID and the public key for each contact.

```
private async void InitializeContactsAsync()
{
    StorageFolder folder = ApplicationData.Current.LocalFolder;
    if (!File.Exists(Path.Combine(folder.Path, "Contacts.xml")))
        return;

    XmlReader.ReadContactsXml();
```

This serialized object has an XML format, a static class named XmlReader contains a method to read the file. Bellow the images shows the XML file opened with a text editor.

```
<Contacts>
  <Property Contact-ID="{5.70002.d4}" Public-Key=
  "MIIBIjANBgkqhkiG9w0BAQEFAAOCAQ8AMIIBCgKCAQEAkCixIYmIchaex4gLUuzBodpJmExWlnTSsTXswfuxLmAHHvd3QtD8ERhicJ2IhlFdlTijJwIGd+m3i0+
  LbG84hoWKV5GFTldQu2Cjk70UAr6kfikOYDWMZcPoHzO/gHUzbb67dYmYnCq4klrDAIKPOTmlVFf8pXYOydxllJVmUfn9glaZV9m0QNIXAl/ubmmZGp2ColLkIMu
  olMUBHxyEeQrCL5/DXcx7Va7YFi5nnHsRojaCAkDU9SUFcamTNTggEDkl7a3vwlH7J8mmFR9SxTHQQn22WN+z+b7/9T2c397unGg82FmgzDqK4zsIJMEePp7Gxb3
  JfIbPkILikYL5vQIDAQAB" />
  <Property Contact-ID="{5.70002.195}" Public-Key="N/A" />
  <Property Contact-ID="{5.70002.1fd}" Public-Key="N/A" />
  <Property Contact-ID="{5.70002.149}" Public-Key="N/A" />
</Contacts>
```

The Tags <Contacts> mark the boundaries of the list and each property tag represents a contact. The Contact-ID attribute has a value of the unique ID and the Public-Key attribute will have the

RSA public key if its imported by the user. In the image above we can see that only the first entry has a public key imported. Since the application is using XML serialization to store these values they cannot be in binary format, they must always be converted to base 64 string.

To read this file we need code that will convert these values into the original object state on runtime, for this purpose the method ReadContactsXml() is used.

```
12      public static void ReadContactsXml()
13      {
14          StorageFolder folder = ApplicationData.Current.LocalFolder;
15
16          using (StreamReader stream = new StreamReader(Path.Combine(folder.Path, "Contacts.xml")))
17          {
18              XDocument document = XDocument.Load(stream);
19
20              XElement contacts = document.Element("Contacts");
21              foreach (XElement property in contacts.Elements("Property"))
22              {
23                  string contactID = property.Attribute("Contact-ID")?.Value ?? throw new NullReferenceException(message: "Contact id can not be null");
24                  string publicKeyValue = property.Attribute("Public-Key")?.Value ?? throw new NullReferenceException(message: "Public key can not be null");
25                  publicKeyValue = CheckIfNullPublicKey(publicKeyValue);
26                  byte[] publicKeyBytes;
27                  IBuffer publicKey;
28                  if (publicKeyValue != null)
29                  {
30                      publicKeyBytes = Convert.FromBase64String(publicKeyValue);
31                      publicKey = publicKeyBytes.AsBuffer();
32                  }
33                  else
34                  {
35                      publicKey = null;
36                  }
37
38                  Base.PublicKeyByContact.Add(contactID, publicKey);
39              }
40          }
41      }
```

The file loads as a stream and loads as an XDocument, it’s a class provided by Microsoft to process XML documents.

For the element Contacts we go through all the elements in it and we parse the values of each attribute to proper variables, in the end of the loop domain we add that info in a public available list named PublicKeyByContact.

Moving back to the method InitializeContactsAsync() we use this list to load the actual contacts from the Peoples app.

```
            XmlReader.ReadContactsXml();
            ContactStore contactStore = await ContactManager.RequestStoreAsync(ContactStoreAccessType.AllContactsReadOnly);
            foreach(string contactID in Base.PublicKeyByContact.Keys.ToList())
            {
                Contact actualContact = await contactStore.GetContactAsync(contactID);
                ListViewItem item = new ListViewItem();
                ContactItemView itemView = new ContactItemView();
                itemView.Name.Text = actualContact.DisplayName;
                try
                {
                    foreach (ContactEmail mail in actualContact.Emails)
                    {
                        if (mail.Address != null)
                            itemView.Info1.Text = mail.Address;
                        break;
                    }
                }
                catch (Exception ex)
                {
                    itemView.Info1.Text = "No available e-mail";
                }
                if (actualContact.SmallDisplayPicture != null)
                {
                    BitmapImage bitmapImage = new BitmapImage();
                    var image = await actualContact.SmallDisplayPicture.OpenReadAsync();
                    bitmapImage.SetSource(image);
                    itemView.Image.ImageSource = bitmapImage;
                }
                item.Content = itemView;
                Base.ContactByListViewItem.Add(item, actualContact.Id);
                item.Tapped += Item_Tapped;
                contactsListView.Items.Add(item);
            }

        }
```

When the PublicKeyByContact Dictionary is ready we request access to the contacts of People app using the class ContactStore, a special class that provides ways to a special storage location in Windows that is only for contacts.

For each one of the keys in the dictionary we need to initialize a new ListViewItem, a nea ContactItemView and a new Contact object.

The contact object will be the actual contact loaded by contact storage using the unique ID, as soon as the contact object is ready, we read the full name, email and the image and we assign it on the contact view item respective properties.

ListViewItem is the container that will hold the ContactItemView component, and the view list will contain the list view items.

Each ListViewItem will have attached on the tapped event a method that will contain code to be executed every time an item is clicked.

The Add button will initialize the contact picker, after the user select the contacts that wants to import the application will check if the contacts are already imported, then if they are not they will be added in the view and the serialized XML file will be updated.

```
98      private async void AddContactsAsync()
99      {
100         var contactPicker = new ContactPicker();
101         contactPicker.CommitButtonText = "Select";
102         contacts = await contactPicker.PickContactsAsync();
103         ContactStore contactStore = await ContactManager.RequestStoreAsync(ContactStoreAccessType.AllContactsReadOnly);
104
105         if (contacts != null && contacts.Count > 0)
106         {
107             foreach (Contact contact in contacts)
108             {
109                 Contact actualContact = await contactStore.GetContactAsync(contact.Id);
110                 if (Base.PublicKeyByContact.Keys.Count == 0)
111                 {
112                     Base.PublicKeyByContact.Add(actualContact.Id, null);
113                     string xml = XmlWriter.CreateContactsXmlDocument();
114                     await XmlWriter.SaveContactsAsync(xml);
115                 }
116                 else
117                 {
118                     if (Base.PublicKeyByContact.ContainsKey(actualContact.Id))
119                     {
120                         MessageDialog messageDialog = new MessageDialog(content: "This contact has been added before.");
121                         await messageDialog.ShowAsync();
122                         return;
123                     }
124                     foreach (string existingContactId in Base.PublicKeyByContact.Keys.ToList())
125                     {
126                         try
127                         {
128                             Base.PublicKeyByContact.Add(actualContact.Id, null);
129                             string xml = XmlWriter.CreateContactsXmlDocument();
130                             await XmlWriter.SaveContactsAsync(xml);
131                         }
132                         catch (Exception ex)
133                         {
134                             Debug.WriteLine(ex.Message);
135                         }
136                     }
137                 }
```

Below is the rest of the AddContactAsync method

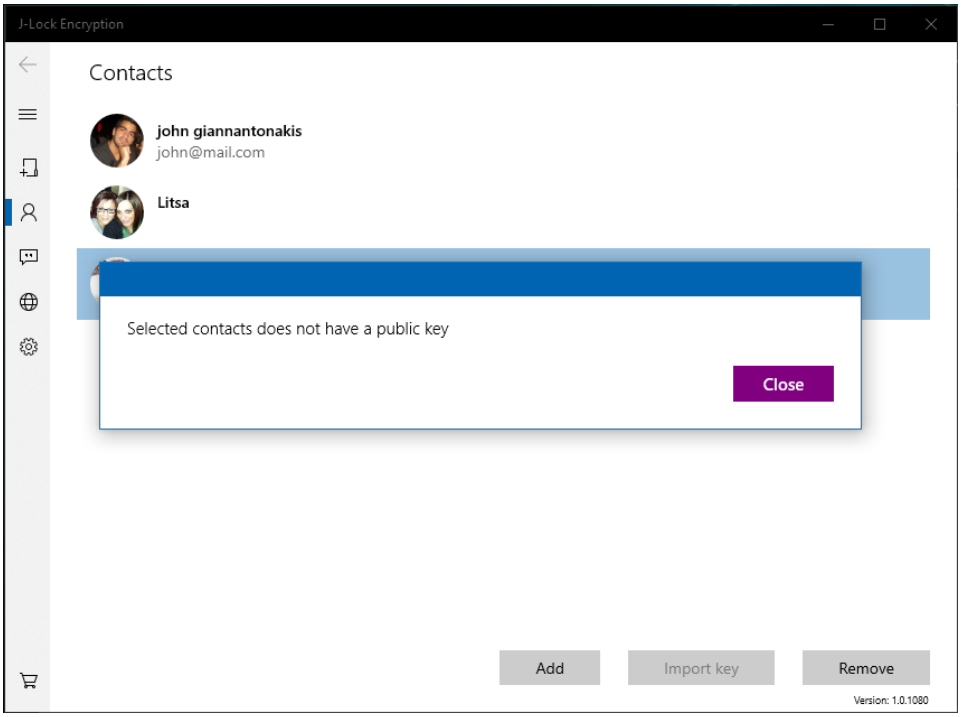
```
                ListViewItem item = new ListViewItem();
                ContactItemView itemView = new ContactItemView();
                itemView.Name.Text = actualContact.DisplayName;
                try
                {
                    foreach (ContactEmail mail in actualContact.Emails)
                    {
                        if (mail.Address != null)
                            itemView.Info1.Text = mail.Address;

                        break;
                    }
                }
                catch (Exception ex)
                {
                    itemView.Info1.Text = "No available e-mail";
                }
                if (actualContact.SmallDisplayPicture != null)
                {
                    BitmapImage bitmapImage = new BitmapImage();
                    var image = await actualContact.SmallDisplayPicture.OpenReadAsync();
                    bitmapImage.SetSource(image);
                    itemView.Image.ImageSource = bitmapImage;
                }
                item.Content = itemView;
                Base.ContactByListViewItem.Add(item, actualContact.Id);
                item.Tapped += Item_Tapped;
                contactsListView.Items.Add(item);
            }
        }
        else
        {

        }

    }
```

The Remove button will remove the selected contact from the XML file and the view as well.

```
248         private async void Button_Click_1(object sender, RoutedEventArgs e)
249         {
250             string contact = Base.ContactByListViewItem[contactsListView.SelectedItem as ListViewItem];
251             Base.PublicKeyByContact.Remove(contact);
252             contactsListView.Items.Remove(contactsListView.SelectedItem);
253             string contactsXml = XmlWriter.CreateContactsXmlDocument();
254             await XmlWriter.SaveContactsAsync(contactsXml);
255         }
```

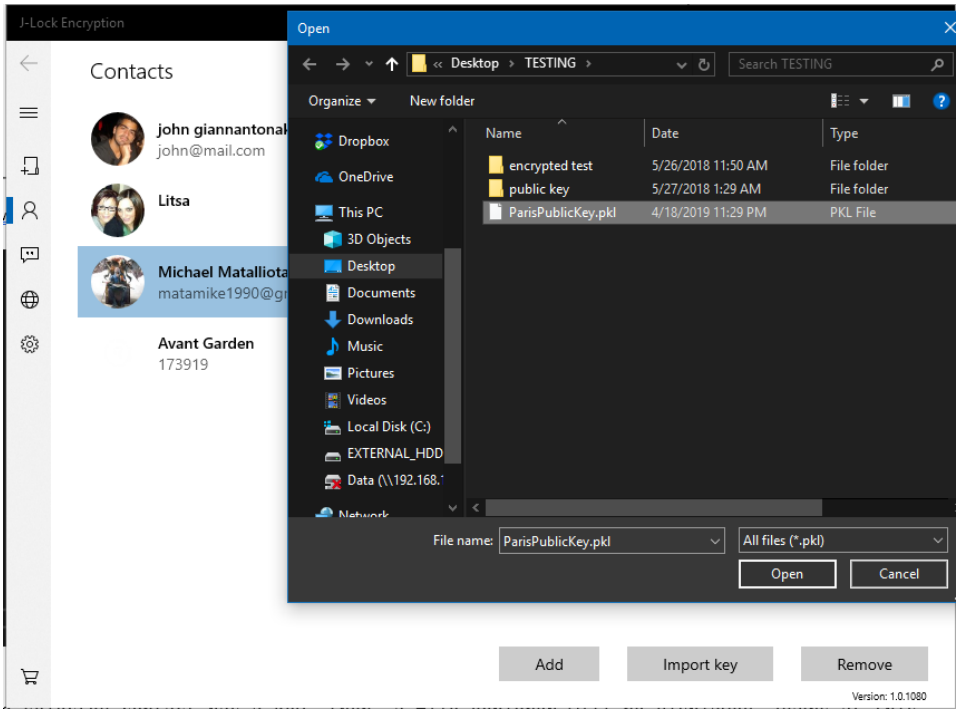
When the user selects a contact the tapped event will check if the contact has an imported RSA public key, if not the button import key will be enabled and the user can import a key for this contact, once this done the XML file will update the specified contact with the key.

```
176         private async void Item_Tapped(object sender, TappedRoutedEventArgs e)
177         {
178             MessageDialog messageDialog = new MessageDialog(content:"Selected contacts does not have a public key");
179             string selectedContactID = Base.ContactByListViewItem[contactsListView.SelectedItem as ListViewItem];
180             if (Base.PublicKeyByContact[selectedContactID] == null)
181             {
182                 await messageDialog.ShowAsync();
183                 importKeyButton.IsEnabled = true;
184                 return;
185             }
```



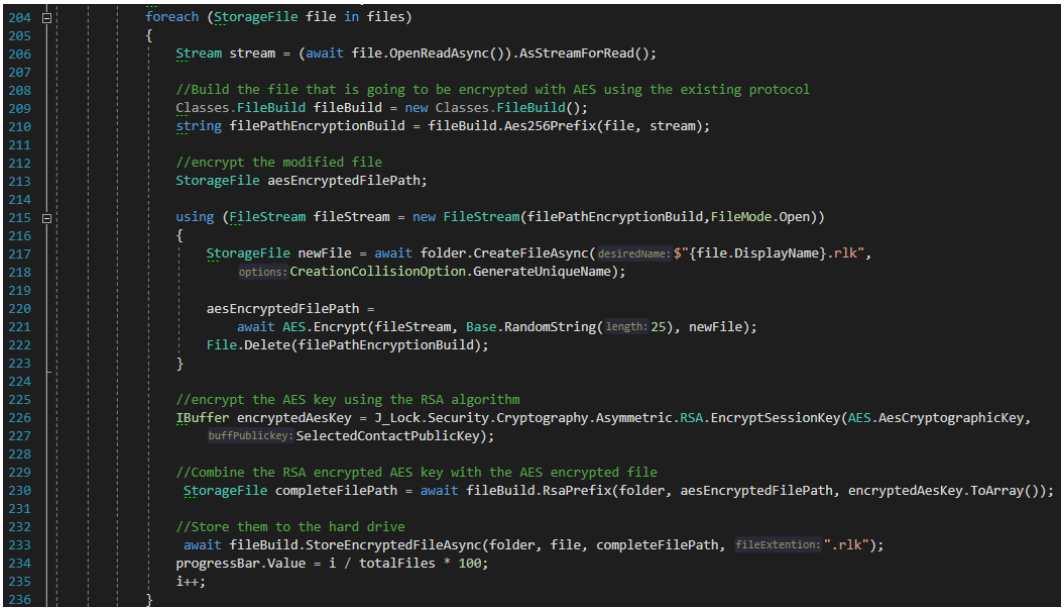
Below is the method that is executed once users click this button for the selected contact.

```
257     private async void Button_Click_2(object sender, RoutedEventArgs e)
258     {
259
260         var picker = new Windows.Storage.Pickers.FileOpenPicker();
261         picker.ViewMode = Windows.Storage.Pickers.PickerViewMode.Thumbnail;
262         picker.FileTypeFilter.Add(item: ".pkl");
263         Windows.Storage.StorageFile file = await picker.PickSingleFileAsync();
264         var PublicKey = await file.OpenReadAsync();
265         byte [] buffer = new byte[PublicKey.Size];
266         await PublicKey.ReadAsync(buffer.AsBuffer(), (uint)buffer.Length, InputStreamOptions.None);
267         IBuffer contactPublicKey = buffer.AsBuffer();
268
269         string selectedContact = Base.ContactByListViewItem[contactsListView.SelectedItem as ListViewItem];
270         Base.PublicKeyByContact[selectedContact] = contactPublicKey;
271         string xml = XmlWriter.CreateContactsXmlDocument();
272         await XmlWriter.SaveContactsAsync(xml);
273
274
275         importKeyButton.IsEnabled = false;
276     }
```



Once the public key is imported into the contact then the user can click on the contact and encrypt files using this key.

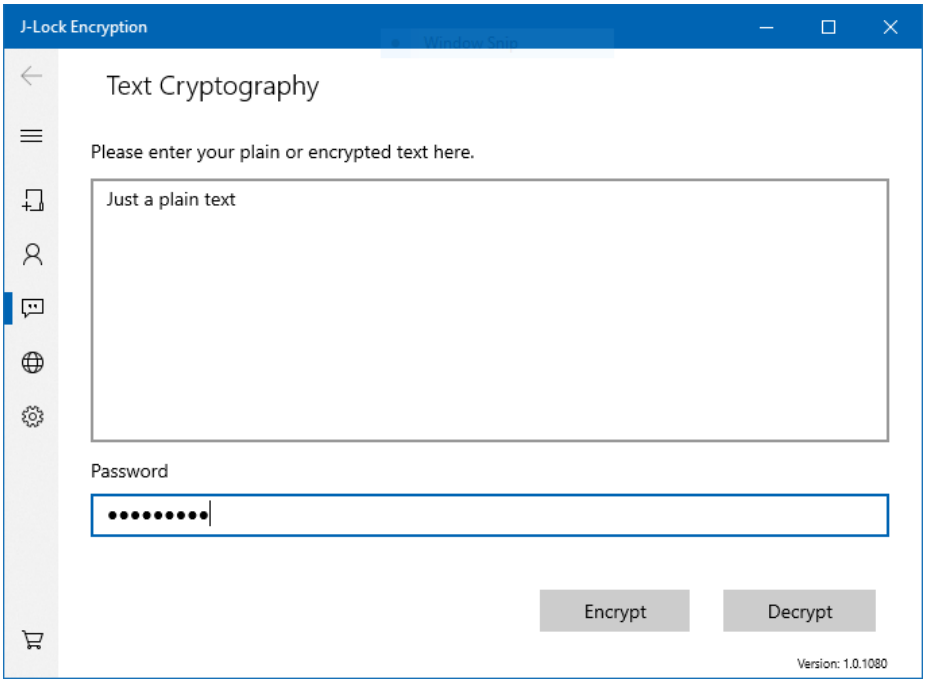
If the selected contact has a key, then, a file explorer will be displayed, users at this state can select any number of files that they will be encrypted using the public key of this contact as explained earlier. Users will be asked to select a folder to save the encrypted files. A progress bar will be displayed showing how many files left to encrypt before the action completes.

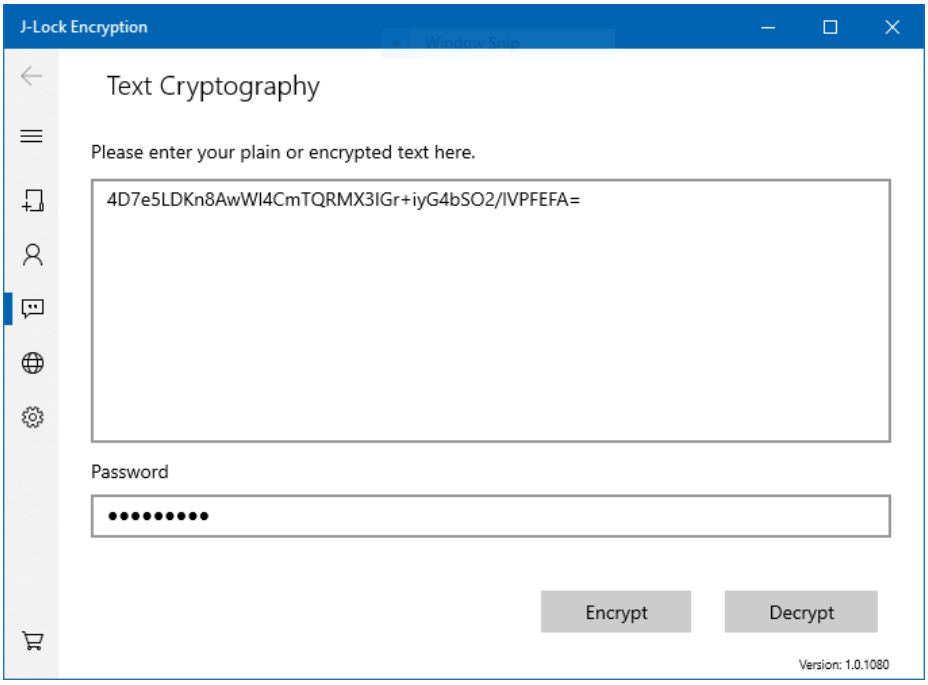


```
foreach (StorageFile file in files)
{
    Stream stream = (await file.OpenReadAsync()).AsStreamForRead();

    //Build the file that is going to be encrypted with AES using the existing protocol
    Classes.FileBuild fileBuild = new Classes.FileBuild();
    string filePathEncryptionBuild = fileBuild.Aes256Prefix(file, stream);

    //encrypt the modified file
    StorageFile aesEncryptedFilePath;

    using (FileStream fileStream = new FileStream(filePathEncryptionBuild,FileMode.Open))
    {
        StorageFile newFile = await folder.CreateFileAsync(desiredName:$"{file.DisplayName}.rlk",
            options:CreationCollisionOption.GenerateUniqueName);

        aesEncryptedFilePath =
            await AES.Encrypt(fileStream, Base.RandomString(length:25), newFile);
        File.Delete(filePathEncryptionBuild);
    }

    //encrypt the AES key using the RSA algorithm
    IBuffer encryptedAesKey = J_Lock.Security.Cryptography.Asymmetric.RSA.EncryptSessionKey(AES.AesCryptographicKey,
        buffPublickey:SelectedContactPublicKey);

    //Combine the RSA encrypted AES key with the AES encrypted file
     StorageFile completeFilePath = await fileBuild.RsaPrefix(folder, aesEncryptedFilePath, encryptedAesKey.ToArray());

    //Store them to the hard drive
     await fileBuild.StoreEncryptedFileAsync(folder, file, completeFilePath, fileExtention:".rlk");
    progressBar.Value = i / totalFiles * 100;
    i++;
}
```

4.9.4 Text area view

The text area view contains a simple text editor and a password filed, users can type any text and use a password to replace the plain text in the editor with the encrypted version of the text. This tool utilizes the AES – 256 algorithm, users can agree to a common password and exchange encrypted messages through any media like email and chat services. Of course users can copy and paste a cipher text in the editor use the common password and decrypt the text.





The XAML code for this view is quite simple, it contains a text field, a password filed and two buttons for encryption and decryption.

```
<Grid Background="{ThemeResource ApplicationPageBackgroundThemeBrush}" HorizontalAlignment="Stretch">
    <Grid HorizontalAlignment="Stretch" Height="auto" VerticalAlignment="Stretch" Width="auto">
        <TextBlock HorizontalAlignment="Left" Margin="0,10,0,0" Text="Please enter your plain or encrypted text here." TextWrapping="Wrap" VerticalAlignment="Top" />
        <TextBox x:Name="txtinput"  Text="" Margin="0,40,0,150" VerticalContentAlignment="Stretch" HorizontalAlignment="Stretch" VerticalAlignment="Stretch"/>
        <TextBlock HorizontalAlignment="Left" Margin="0,0,0,120" Text="Password" TextWrapping="Wrap" VerticalAlignment="Bottom" />
        <PasswordBox x:Name="txtpassword" HorizontalAlignment="Stretch" Margin="0,0,0,80" VerticalAlignment="Bottom"/>
        <StackPanel HorizontalAlignment="Stretch" Height="auto" VerticalAlignment="Bottom" Width="auto" Orientation="Horizontal"
                    Margin="10,10,10,10" FlowDirection="RightToLeft">
            <Button Content="Decrypt" HorizontalAlignment="Stretch" VerticalAlignment="Stretch" Margin="0,0,25,0" Padding="30,4,30,4" Tapped="Button_Tapped"/>
            <Button Content="Encrypt" HorizontalAlignment="Stretch" VerticalAlignment="Stretch" Margin="0,0,25,0" Padding="30,4,30,4" Tapped="Button_Tapped_1"/>
        </StackPanel>
    </Grid>
```

The code behind is quite simple as well, two methods one for the encrypt button and one for the decrypt button will get the text from the text editor and it will be encrypted or decrypted respectively using the provided password.

```
1 reference | DESKTOP-1PKLJKN\paris, 62 days ago | 1 author, 3 changes
private void Button_Tapped_1(object sender, TappedRoutedEventArgs e)
{
    txtinput.Text =
        Convert.ToBase64String(J_Lock.Security.Cryptography.AES.Encrypt(txtpassword.Password,
            Encoding.UTF8.GetBytes(txtinput.Text)));
}
```

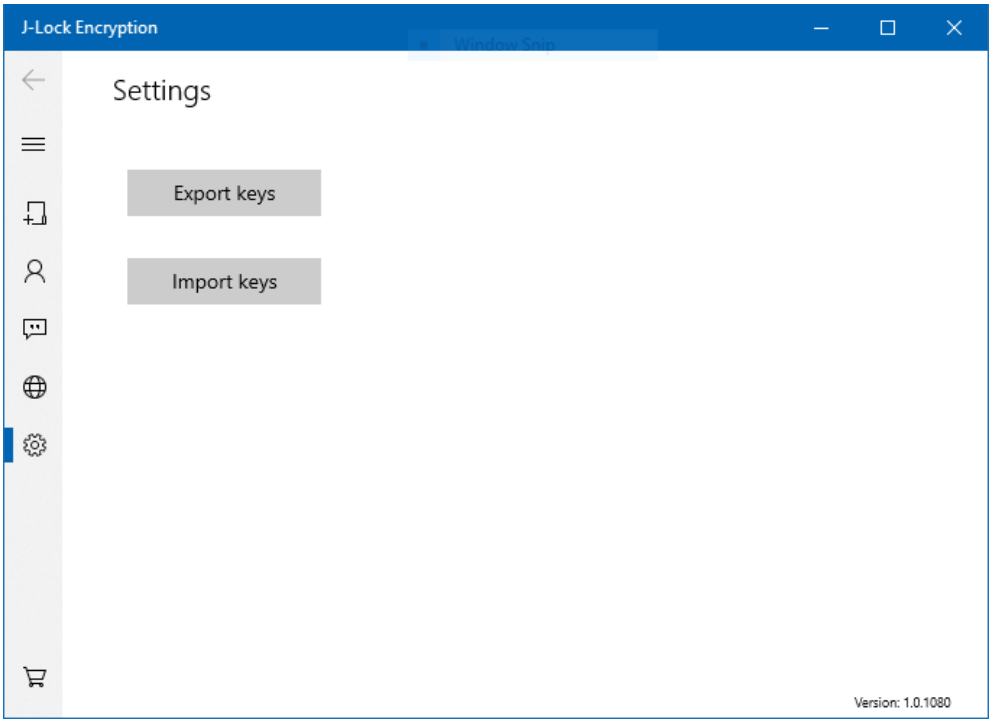
```
1 reference | DESKTOP-1PKLJKN\paris, 62 days ago | 1 author, 3 changes
private async void Button_Tapped(object sender, TappedRoutedEventArgs e)
{
    try
    {

        txtinput.Text = Encoding.UTF8.GetString(J_Lock.Security.Cryptography.AES.Decrypt(txtpassword.Password, Convert.FromBase64String(txtinput.Text)));

    }
    catch(Exception)
    {
        ContentDialog noWifiDialog = new ContentDialog
        {
            Title = "Failed",
            Content = "Operation not completed. Please check you password!",
            CloseButtonText = "Close",
            DefaultButton = ContentDialogButton.Close
        };
        ContentDialogResult result = await noWifiDialog.ShowAsync();
    }

}
This method was edited by 1 author in 3 unique commits.
```

On decryption if the password is wrong a respective message will be displayed to the user.

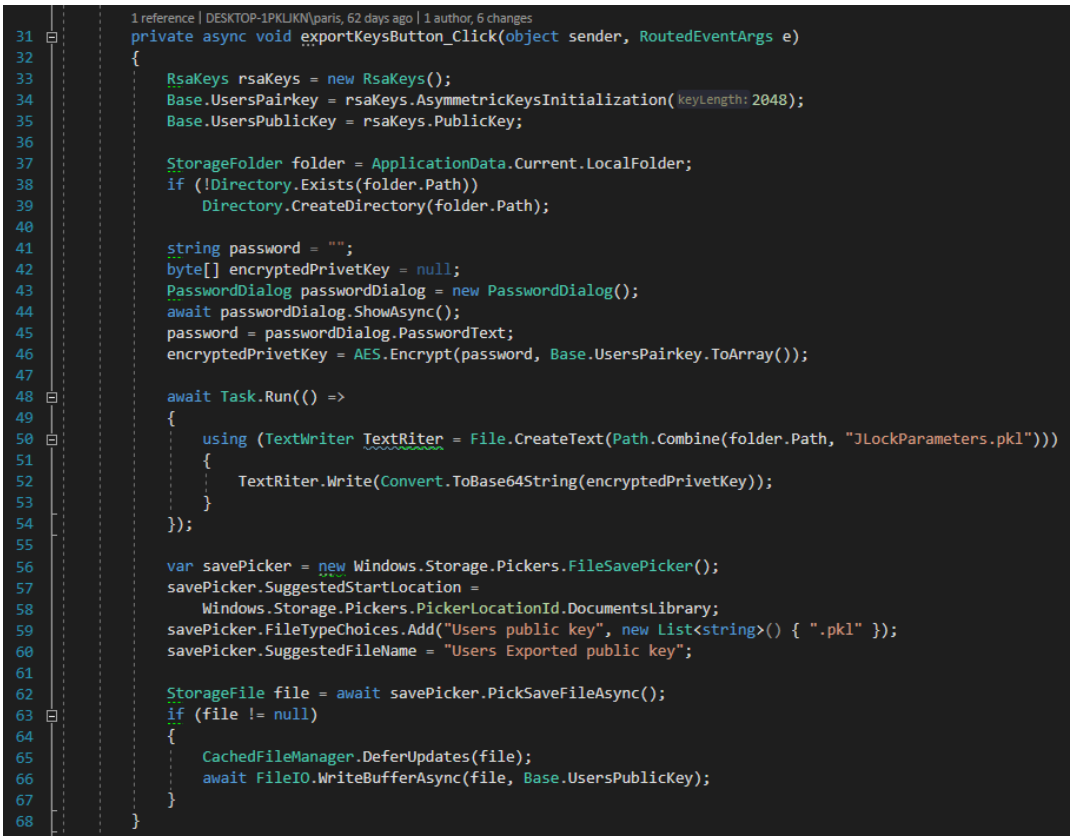
4.9.5 Settings view

The settings view gives the option to user to generate and export a new RSA pair of keys for personal use. Users can save their public and private key on a location they desire, it is safe to store the RSA key pair to any location since they are encrypted with a master password that the user provides. Moreover in the case a user changes a machine or after a system recovery there is an option to import an old pair of keys.

Of course it is possible to keep many RSA key pairs and import them at will, in case the machine is used by many users each user can import their key and after use the can reset the application. Another use case is when a user needs to have multiple keys for identification purposes. Users can know each other from the key they use to decrypt received messages.



The code behind is composed by two methods, one for each option that are called when the respective button is clicked.



```
1 reference | DESKTOP-1PKLJKN\paris, 62 days ago | 1 author, 6 changes
private async void exportKeysButton_Click(object sender, RoutedEventArgs e)
{
    RsaKeys rsaKeys = new RsaKeys();
    Base.UsersPairkey = rsaKeys.AsymmetricKeysInitialization(keyLength: 2048);
    Base.UsersPublicKey = rsaKeys.PublicKey;

    StorageFolder folder = ApplicationData.Current.LocalFolder;
    if (!Directory.Exists(folder.Path))
        Directory.CreateDirectory(folder.Path);

    string password = "";
    byte[] encryptedPrivetKey = null;
    PasswordDialog passwordDialog = new PasswordDialog();
    await passwordDialog.ShowAsync();
    password = passwordDialog.PasswordText;
    encryptedPrivetKey = AES.Encrypt(password, Base.UsersPairkey.ToArray());

    await Task.Run(() =>
    {
        using (TextWriter TextRiter = File.CreateText(Path.Combine(folder.Path, "JLockParameters.pkl")))
        {
            TextRiter.Write(Convert.ToBase64String(encryptedPrivetKey));
        }
    });

    var savePicker = new Windows.Storage.Pickers.FileSavePicker();
    savePicker.SuggestedStartLocation =
        Windows.Storage.Pickers.PickerLocationId.DocumentsLibrary;
    savePicker.FileTypeChoices.Add("Users public key", new List<string>() { ".pkl" });
    savePicker.SuggestedFileName = "Users Exported public key";

    StorageFile file = await savePicker.PickSaveFileAsync();
    if (file != null)
    {
        CachedFileManager.DeferUpdates(file);
        await FileIO.WriteBufferAsync(file, Base.UsersPublicKey);
    }
}
```

The export keys option will generate a new RSA key pair and they will override the existing one if any.
On creation the user is asked to provide a password to encrypt the pair, then the public key will be extracted as a copy of the public part.

The user will be asked to select a location to store the public key and the RSA pair as well, the public key is plain while the pair is encrypted. Users can give any name to the exported public key but the encrypted pair should remain intact.

The pair which contains the private key as well will be stored to the selected location and a copy of it will be stored to the special application folder encrypted as well.

Importing an RAS pair is very simple, the application will open a file explorer and the user can select the stored pair to restore it.

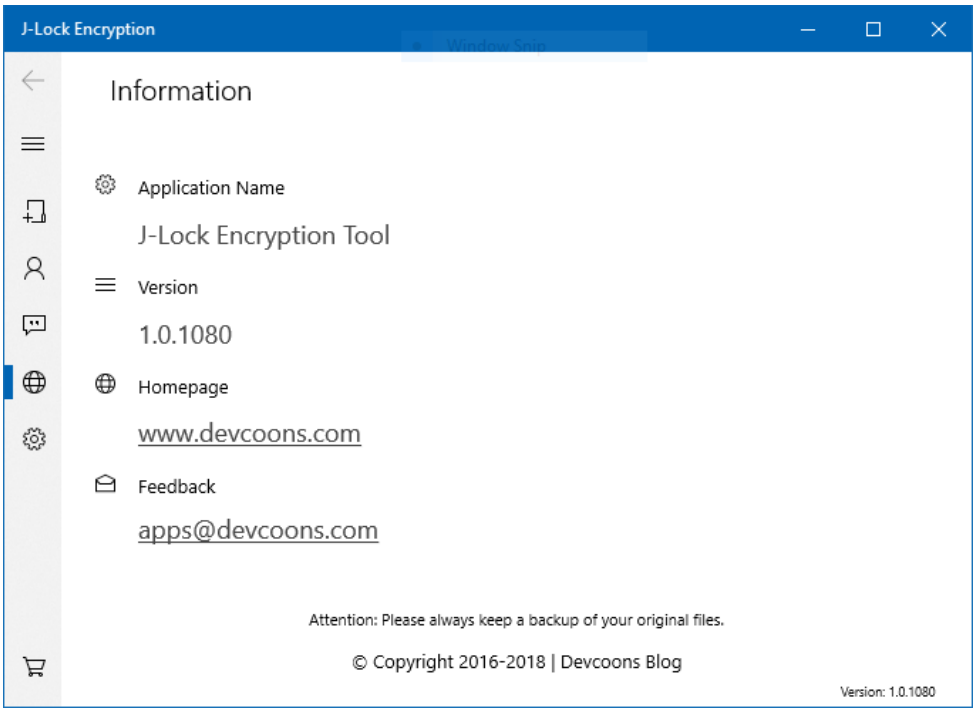
```
70      private async void Button_ClickAsync(object sender, RoutedEventArgs e)
71      {
72          StorageFolder folder = ApplicationData.Current.LocalFolder;
73          if (!Directory.Exists(folder.Path))
74              Directory.CreateDirectory(folder.Path);
75
76          var picker = new Windows.Storage.Pickers.FileOpenPicker();
77          picker.ViewMode = Windows.Storage.Pickers.PickerViewMode.Thumbnail;
78          picker.FileTypeFilter.Add(item: ".pkl"); // or other file type that is the public key
79          StorageFile file = await picker.PickSingleFileAsync();
80          try
81          {
82              string privateKey = await FileIO.ReadTextAsync(file);
83              await Task.Run(() =>
84              {
85                  using (TextWriter TextRiter = File.CreateText(Path.Combine(folder.Path, "JLockParameters.pkl")))
86                  {
87                      TextRiter.Write(privateKey);
88                  }
89              });
90
91              PasswordDialog passwordDialog = new PasswordDialog();
92              passwordDialog.Title = "Password protected keys, please provide the password for these keys.";
93              passwordDialog.PrimaryButtonText = "Import";
94              await passwordDialog.ShowAsync();
95              string password = passwordDialog.PasswordText;
96              byte[] plainPrivateEkyBytes = AES.Decrypt(password, Convert.FromBase64String(privateKey));
97
98              IBuffer plainPrivateKey = plainPrivateEkyBytes.AsBuffer();
99              AsymmetricKeyAlgorithmProvider algorithmProvider = AsymmetricKeyAlgorithmProvider.OpenAlgorithm(AsymmetricAlgorithmNames.RsaPkcs1);
100             CryptographicKey keyPair = algorithmProvider.ImportKeyPair(plainPrivateKey);
101
102             var savePicker = new Windows.Storage.Pickers.FileSavePicker();
103             savePicker.SuggestedStartLocation =
104                 Windows.Storage.Pickers.PickerLocationId.DocumentsLibrary;
105             savePicker.FileTypeChoices.Add("Users public key", new List<string>() { ".pkl" });
106             savePicker.SuggestedFileName = "Users Exported public key";
107
108             StorageFile publickeyFile = await savePicker.PickSaveFileAsync();
109             if (publickeyFile != null)
110             {
111                 CachedFileManager.DeferUpdates(publickeyFile);
112                 await FileIO.WriteBufferAsync(publickeyFile, keyPair.ExportPublicKey());
113             }
114
115             plainPrivateEkyBytes = null;
116             plainPrivateKey = null;
117
118         }
119         catch (Exception)
120         {
121
122         }
123     }
```

As soon the file is imported the application will ask for the password that protects it, if users don't remember the password it will not be possible to restore it or use it. Then a file explorer will be prompt asking the user to select a location to store the public key and the pair.

If the is already a pair of keys, they will be overridden.

4.9.6 Information view

The information view provides basic information about the application with contact details for the developer. It is mandatory in Windows store published applications to have a section with these details. Also is useful for the users in case they want to send feedback or ask for a feature.



The XAML code is very simple it contains only text and link components and there is no code behind.

```
<Grid Background="{ThemeResource ApplicationPageBackgroundThemeBrush}">
    <TextBlock HorizontalAlignment="Left" Margin="0,30,0,0" Text="" TextWrapping="Wrap" VerticalAlignment="Top" FontFamily="Segoe MDL2 Assets"/>
    <TextBlock HorizontalAlignment="Left" Margin="30,29,0,0" Text="Application Name" TextWrapping="Wrap" VerticalAlignment="Top" FontSize="13"/>
    <TextBlock HorizontalAlignment="Left" Margin="0,100,0,0" Text="" TextWrapping="Wrap" VerticalAlignment="Top" FontFamily="Segoe MDL2 Assets"/>
    <TextBlock HorizontalAlignment="Left" Margin="0,170,0,0" Text="" TextWrapping="Wrap" VerticalAlignment="Top" FontFamily="Segoe MDL2 Assets"/>
    <TextBlock HorizontalAlignment="Left" Margin="0,240,0,0" Text="" TextWrapping="Wrap" VerticalAlignment="Top" FontFamily="Segoe MDL2 Assets"/>
    <TextBlock HorizontalAlignment="Left" Margin="30,99,0,0" Text="Version" TextWrapping="Wrap" VerticalAlignment="Top" FontSize="13"/>
    <TextBlock HorizontalAlignment="Left" Margin="30,169,0,0" Text="Homepage" TextWrapping="Wrap" VerticalAlignment="Top" FontSize="13"/>
    <TextBlock HorizontalAlignment="Left" Margin="30,239,0,0" Text="Feedback" TextWrapping="Wrap" VerticalAlignment="Top" FontSize="13"/>
    <TextBlock HorizontalAlignment="Left" Margin="30,60,0,0" Text="J-Lock Encryption Tool" TextWrapping="Wrap" VerticalAlignment="Top" FontSize="18" Foreground="#FF464646"/>
    <HyperlinkButton NavigateUri="http://www.devcoons.com" Content="www.devcoons.com" HorizontalAlignment="Left" Margin="30,193,0,0" VerticalAlignment="Top" FontSize="18" Foreground="#FF464646"/>
    <TextBlock x:Name="TxtVersion" HorizontalAlignment="Left" Margin="30,130,0,0" Text="" TextWrapping="Wrap" VerticalAlignment="Top" FontSize="18" Foreground="#FF464646"/>
    <HyperlinkButton NavigateUri="mailto:apps@devcoons.com" Content="apps@devcoons.com" HorizontalAlignment="Left"  Margin="30,261,0,0" VerticalAlignment="Top" FontSize="18" Foreground="#FF464646"/>
    <!--UI:AdControl ApplicationId="9nddldh0fz1g"  AdUnitId="1100017107" Horizonta                    t="250"  VerticalAlignment="Center" Width="300" Margin="0,0,20,0"/-->
    <TextBlock HorizontalAlignment="Center" Margin="0,0,0,0" Text="© Copyright 201                    extWrapping="Wrap" VerticalAlignment="Bottom" FontSize="13"/>
    <TextBlock HorizontalAlignment="Center" Margin="0,0,0,30" Text="Attention: Please always keep a backup of your original files."
               TextWrapping="Wrap" VerticalAlignment="Bottom" TextAlignment="Center" FontSize="11"/>
</Grid>
```

ContentControl.Content

CHAPTER 5: RESULTS

5.1 Conclusion

5.1.1 Results

The application was able to encrypt 4GB of data using AES-256 in 1 minute and 30 seconds on the testing machine using a single thread on CPU, this might vary on different hardware depending on the frequency of the CPU. Using only one thread for the process emulates the behavior of a CPU with one core.

RSA encryption will take only fractions of a second more since it is used only to encrypt 32 bytes that is the AES-256 key.

The application is using in average 45 MB of RAM but at 99% of that usage are the graphical components and animations of the user interface, while encrypting the RAM consumption is limited to the buffer's size of the stream which is 16KB, it means that at any given time the encryption process it self is consuming only 16KB of ram.

Decrypting the same file of 4GB completed in 1 minute and 12 seconds and RAM consumptions are of course at the same level since decrypting process is done using the same streaming techniques. Decrypting an RSA protected file takes again fractions of a second more.

All the above results are analogous to the size of the data, if the same amount of data is by multiple files it takes almost the same time, few fractions of seconds more to find the next file and initialize the new stream. On all tests the results were approximately the same.

Regarding security and penetration tests, assuming a subject that has access on the source code an in RAM pointers could possibly obtain the password of the user however this is out of the applications scope since RAM and operating systems security is a matter of the distributer (Microsoft), even if someone manages to do so it would take a large amount of time effort and money to penetrate a valid copy of Windows without being noticed by an antivirus or the build in protection.

Brute forcing the file is technically impossible, the password is not used as it produce a 256bit array and then it immediately its disposed while this array is used as the actual key. Brute forcing a 256bit key is not a practical option.

5.1.2 Benefits of thesis

This thesis contains theoretical and practical information regarding some of the best practices of cryptography so far.

Subjects of interest are how to use AES-256 and RSA, the theoretical model under the code but also how to implement them on another software regardless the programming language the techniques and the theory behind the code are the same.

Also, another benefit is how to manipulate, manage and handle any size of data without reaching a machine to its limits and potentially crash it while at the same time utilizing the maximum speed the machine may give. Streams and how they work are the key of handling and distributing extremely large amounts of data. Again, the same techniques and logic are valid for any platform and any programming language.

Moreover, the reader may benefit on how to make a complicated process effortless for an average user, the UI practices keep the logic of 3-4 clicks to complete a task. All modern applications should implement this logic to make a secure and user-friendly application.

5.2 Future work and extensions

This thesis and the application itself in a generic perspective is a good base as a showcase of a basic completed product, this could help others to have a guide line of what it takes to create an application ready to be published on a platform.

In a specific perspective it an application made for security and it can be extended to more fields for example to utilize the network for direct secure file exchange through a custom secured channel on local networks or the internet. The backend code of the application could be used to create a network platform of secure file exchange.

Regarding performance the application could be extended to utilize more than one threads during cryptographic operations, this could minimize the time by a large amount, the same data could be shared among the threads to process them at the same time.
For example, a 4GB file could be separated to 8 parts of 500 MB each part could be a handled by a thread. So that means 8 threads would process the file at the same time minimizing the time to complete almost at the 1/8 of the time that need one thread to process the 4GB file.

Another extension would be to implement microservices into the system as part of the application, this can utilize file exchange within a network. On top of that there could be mobile application sharing the same logic, that means, in a network the security is available on PC and mobile devices.

Encrypting files through a mobile device or PC and sharing it though the secure channels of the microservices running on a local network may benefit companies and organizations.

Finally there is a new way of cryptography called block chain, the same algorithms used for cryptocurrencies, the application could extend to use these techniques to be at the edge of modern technology in encryption.

REFERENCES

1. https://medium.com/coinmonks/github-vs-bitbucket-vs-gitlab-d2b5c28276b0
2. https://en.wikipedia.org/wiki/.NET_Framework
3. https://www.techopedia.com/definition/26272/c-sharp
4. https://sourcemaking.com/design_patterns
5. https://searchsecurity.techtarget.com/definition/Advanced-Encryption-Standard
6. https://www.di-mgt.com.au/cryptopad.html
7. https://en.wikipedia.org/wiki/RSA_%28cryptosystem%29
8. https://en.wikipedia.org/wiki/Side_channel_attack
9. https://en.m.wikipedia.org/wiki/Black-box_testing