ITRI615 - Computer Security
Project Documentation

Affaan Muhammad - 33016763

Joshua Esterhuizen - 30285976

Contents

Section 1

Cryptographic Schemes and Background

1.1 Vigenère Cipher

This cipher was first described by Giovan Bellaso in 1553 but later misattributed to Blaise de Vigenère, hence its name. This cipher works by making use of a series of interwoven Caesar ciphers - which when joined create a Vigenère Table - based on the letters of a given keyword. As such, it is a polyalphabetic substitution cipher. It was still "indecipherable" almost three centuries after its development.

1.1.1 Our Implementation

Our Vigenère algorithm implementation makes use of:

- a function to extend the user given key to match the message or file content's length.

1.1.2 The Algorithm Itself

Our Vigenère algorithm implementation works using calculations involving the ASCII values of the letters, and byte characters in a file's case, to apply the same effect of a Vigenère Table. The encryption adds these values within modulo26 for text (also adding 65 as discussed below) and modulo256 for files while the decryption subtracts these values. When it comes to the text "mode" of this algorithm, we have included a check to ensure that all characters that have the cipher applied are indeed letters. This is done by:

- Before the algorithm takes place, we apply the ".upper()" to the text string to keep all characters as capital letters,
- Setting an if statement that riggers if the ASCII value of a character is not within the capital letter values,
- Immediately append to the array that will be returned.

We could also do the same process above but with lowercase letters with ".lower()" if we so chose.

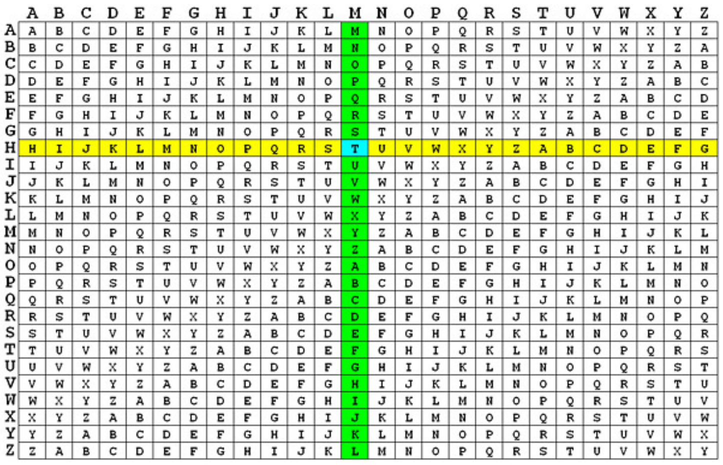
1.1.3 The Key Extender

As with the "normal" and our implementation of the Vigenère cipher algorithm, the user provided key mmust be repeated across the length of the message. This function does nothing if the key provided is the same length as the message. However, if the key is shorter, it it incrementally added onto itself character-by-character until the right length is met. In the event that the message is shorter than the key, this function returns the original key. Due to the way the encryption and decryption algorithms function, this will not matter as the loops within them are tied to message length and not key length.

1.1.4 Logic Behind Using the Sums Taken in Modulo26 of ASCII Values in Place of a Vigenère Table

Using the image below it is clear with the two letters 'M' and 'H', the letter 'T' is produced through the Vigenère cipher. The ASCII values for these are as follows:
'M' = 77
'H' = 72
'T' = 84

Now if we take 77 + 72, we get 149.
Taking 149(mod26) we get 19
Adding 65 and 19 we get 84 (As 65 is the value of 'A' so that we arrive back at the ASCII values of capital letters)
We can then see that the value we get from these calculations, is the same as the ASCII value of 'T' => and with the chr() we can take this number and return the letter 'T'.



	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
A	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
B	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A
C	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B
D	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C
E	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D
F	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E
G	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F
H	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G
I	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H
J	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I
K	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J
L	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K
M	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L
N	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M
O	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N
P	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
Q	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P
R	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q
S	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R
T	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S
U	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T
V	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U
W	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V
X	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W
Y	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X
Z	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y

Figure 1.1: Vigenère table

The above image of the Vigenère table was taken from a website authored by *Dr.*

C.-K. Shene a professor in the Department of Computer Science at Michigan Technological University (circa. 2014) available at https://pages.mtu.edu/~shene/NSF-4/Tutorial/VIG/Vig-Base.html

1.2 Vernam Cipher

The Vernam cipher is a One-Time pad cipher meaning that the key used is only ever used once and never again. It makes use of the positional values of each letter in the alphabet and adds the plaintext to the key with these and then applies a modulo 26 calculation on the resulting values. The key used in this has certain stipulations, the first that it must be truly random (one that computers are somewhat able to do), it must also be as long as the given plain text and then ever be used again.

1.2.1 Our Implementation

Our Vernam algorithm implementation makes use of:

- a randomly generated key for use as the One-Time Pad.

1.2.2 Algorithm Itself

Our implementation of the Vernam cipher is practically the same as it is described. We generate a random key to use only once per encryption, add it numerically to the plaintext or file contents as integers (which represent the bytes) and then decrypt it by subtracting that key again. The text "mode" works as it should, adding the numerical values of the letters' position in the alphabet with those of the random key. However, due to the restrictions placed upon working with file contents while using Python (being unable to change the data type from integer representations of bytes) we simply add the random key as with the text (the exception being that the file's is generated differently as described below) instead of using the XOR function some digital implementations make use of.

1.2.3 The One-Time Pad Random Key Generator

Using python's random library, a key the length of whatever input the algorithm recieves is generated by using;

- the `randrange(65,91)` function, or all capital letter's ASCII values, for text encryption as a string, and
- the `randrange(0,256)` function, or all possible character's ASCII values, for file encryption as an array of integers.

That is to say, it is a key that can consist of some/all 255 ASCII characters in numerical form. This key is then used in a further calculation on the once encrypted data as follows:
For text encryption we make use of the following formula:
`EncryptedValue = (textASCII_Value[ i ] + randomKeyASCII_Value[ i ]% 26) + 65`

(65 is the ASCII value of A)
And with file encryption we made use of the following formula:
`EncryptedValue = fileDataASCII_Value[ i ] + randomKeyASCII_Value[ i ] % 256`
(Taken under modulo256 as character ASCII values are limited to this range)

1.3 Transposition Cipher

A transposition cipher is a means of encryption by which the plaintext characters are shifted according to a regular system. As such, the order of the plaintext is simply reordered. There are also several different methods of doing this, such as the Columnar Transposition which reads the columns in order of the key's character values (i.e. a=1, b=2, and so on). Others involve some form of distortion by adding some spaces or other characters to further mix up the plain text.

1.3.1 Our Implementation

Our Transposition algorithm implementation makes use of:

- a key checking function, and
- a function to convert a string, or array depending on the "mode", into a matrix.

1.3.2 The Algorithm Itself

Before explaining the "formatting" methods used in this implementation of the transposition cipher, we explain which transposition method we made use of. The transposition cipher used is the "*rotate 90 degrees*" variant where the encrypted data is created by reading from a matrix depth-wise through all the columns. To achieve this digitally without any "out of index errors", the matrix is also padded with additional values depending on the "mode" during the matrix formulation function.

1.3.3 The Key Check

As some transposition models make use of an integer value while others a string we decided that ours would accept both:

- In the event an integer is entered, it would stipulate he number of columns within the matrix.
- If a string of characters/letters/special characters/etc. is entered then this function would return the length of that string as the stipulation for the amount of columns in the matrix.

As our front-end makes use of Django's text fields to capture user input, we iterate over each element in the user given key and if all items are found to have the ASCII values of numbers, it is changed from a numerical string to an integer to be used. (It should be noted that this function is also used in our "homebrew" algorithm due to the need to have an integer-based key)

1.3.4 The Matrix Formulation

We have two versions of this function for use:

- One for text encryption and decryption, and
- One for file encryption and decryption.

The reason for this is due to the padding used in this implementation as we cannot pad a files contents with anything other than bytes and as such:

- For text we pad with the ("") character; essentially nothing but adding to the size of the matrix as ("") will have no affect on a string regardless of where it is padded.
- For files we pad the remaining spaces with the last byte accessed - repeating it until the required size is reached.

Regardless of which is used, they both return a matrix (or technically a list of lists).

1.3.5 Removal of padding

Whereas the text version of this transposition essentially pads nothing, the file version pads actual values which must be removed during decryption to return usable "plaintext" file data. To accomplish this we had to store the original file length for use. From a back-end perspective, and when encrypting and then decrypting one file a a time, it was enough to store this length as another variable. However, as a user can encrypt multiple files before decrypting one, we needed a way to tie the original file content's length to its' encrypted version. We accomplished this by sorting the length as an additional array within the encrypted file through the following logic:

- As the length of a files contents' will almost definitely be greater than 255 (the maximum ASCII value) for all but the simplest files, we need to store this integer value as separate bytes for each digit in the number to not exceed this value.
- We then took the length of this number (as it is impossible for it to reach 255 digits long as the largest number a computer can store, $2^31 - 1$, is only ten digits long).
- Therefore, the first byte of the encrypted file contents is the number of times the array of bytes must be popped to be solely "ciphertext" again, still with padding, and to acquire the original length.
- Finally, after the content has been decrypted, we pop() from the back equal to the difference of the matrices full length and the original length to remove all padding bytes.

1.4 J&A Homebrew Algorithm

Our "Homebrew" algorithm makes use of:

- a mathematical calculation to increase the confusion aspect of the cipher text, and
- a randomly generated key of all possible unicode/ASCII values to further enhance the confusion and diffusion aspects.

1.4.1 The Mathematics

With the use of a python mathematics library, such as the simple "math" module or NumPy, this calculation can be quite complex provided the programmer's mathematical knowledge is sufficient. One limitation this algorithm has is that, for files, no multiplication or division can be used as it changes the byte content between a float and an integer which causes it to loose its encoding type. Another is that no mathematical functions that returns multiple values can be used - such as a square root function returning both the positive and negative roots. The integer key a user provides is what is used in these calculations.
As such, for the text encryption we made use of the following formula:
`encryptedValue = (key x plainTextASCII_Value[ i ]) - key`
And with the aforementioned restrictions on file encryption we made use of the following formula:
`encryptedValue = (key x byteASCII_Value[ i ]) - key`
The decryption of this aspect of the algorithm just takes the inverse of these mathematical calculations.

1.4.2 Secondary Random Key

Using python's random library, a key the length of whatever input the algorithm receives is generated by using;

- the `randrange(65,91)` function, or all capital letter's ASCII values, for text encryption, and
- the randrange(32,256) function, or all possible character's ASCII values, for file encryption.

That is to say, it is a key that can consist of some/all 255 ASCII characters in numerical form. This key is then used in a further calculation on the once encrypted data as follows:

For text encryption we make use of the following formula:
`againEncryptedValue = encryptedValue[ i ] + randomKeyASCII_Value[ i ]`
And with the aforementioned restrictions on file encryption we made use of the following formula:
`againEncryptedValue = (encryptedValue[ i ] + randomKeyASCII_Value[ i ]) % 256`
(Taken under modulo256 as character ASCII values are limited to this range)
The decryption of this aspect of the algorithm again just takes the inverse of these mathematical calculations.

1.4.3 The Diffusion Aspect

As the aforementioned random key is needed for decryption, it is actually included in the cipher text or encrypted file in a manner to increase diffusion of the encrypted data. As with the mathematical calculations, it depends on how a programmer wishes to diffuse the key. With this implementation, we have added it as the first half of the encrypted data - which is not too secure now that it's been disclosed :p In the decryption of the encrypted file or cipher text:

- The first half is read as the key, and
- The remaining contents as the encrypted file data or cipher text.

Section 2

Installation and setup

2.1 Project files

The project files can be found on the following GitHub link:
https://github.com/AM-ops/SecurityProject

This was our main code repository. We both have been updating the code as we went along and added details and bug fixes to the project.

To copy the code to your own machine, follow the following steps:

1. Make sure Git is installed. If not it can be downloaded from here:
   https://git-scm.com/
2. Create an empty directory where the code can be copied to
3. Run the following command:

   ```
   git clone https://github.com/AM-ops/SecurityProject.git
   ```

2.2 Virtual Environment

There are multiple advantages of using virtual environments when creating software. The primary reason being we create a layer of separation and abstraction between our host machine's files and our software project.

We made use of a Python virtual environment which was handled by Anaconda. This can be downloaded from the following link:
https://www.anaconda.com/products/individual

2.2.1 Creating a virtual environment

Once Anaconda was installed the following commands were run in the terminal to create a virtual environment called `myDjangoEnv`.

```
conda create --name myDjangoEnv
```

Depending on the version of Anaconda installed you might have to use a leading underscore on Windows machines. The same will apply for commands further down. Below is a demonstration.

```
_conda create --name myDjangoEnv
```

2.2.2 Listing virtual environments

To list all virtual environments on your host machine run the following command.

```
conda info --envs
```

or

```
conda env list
```

2.2.3 Deleting a virtual environment

To delete a virtual environment run the following commands.

```
conda remove --name <name_of_virtual_environment> --all
```

or

```
conda env remove --name <name_of_virtual_environment>
```

2.2.4 Activating and deactivating virtual environments

To activate an environment run the following commands for Windows.

```
conda activate <name_of_virtual_environment>
```

For Linux and MacOS the command is as follows.

```
source activate <name_of_virtual_environment>
```

Once the environment is activated your terminal should change. By default, the active environment, is shown in parentheses () or brackets [] at the beginning of your command prompt as shown below.

```
(<name_of_virtual_environment>) >_
```

Depending on your version of Anaconda to deactivate your environment the commands for Windows is.

```
deactivate
```

or

```
conda deactivate
```

For Linux and MacOS the command will be

```
source deactivate
```

2.2.5 Listing Packages installed

To list all the packages you have installed in an environment there are two methods of listing them. First, if the environment is not activated run the following.

```
conda list -n <name_of_virtual_environment>
```

Secondly, if the environment is activated, then simply run the following.

```
conda list
```

2.2.6 Using pip

Due to the fact that Python is being used for the project it is always necessary to make sure `pip` is installed and functioning. If it is not then run the following commands.

```
conda install -n <name_of_virtual_environment> pip
```

2.3 Frameworks and other packages

2.3.1 Django

The primary framework used for development in this project was `Django`. This is a python based Web framework. The documentation for it can be found here:
https://docs.djangoproject.com/en/3.2/

2.3.2 Bootstrap

Bootstrap is Cascading Style Sheets (CSS) Framework which allows for simple, elegant, and responsive Graphical User Interfaces to be developed for the Web. The documentation for it can be found here:
https://getbootstrap.com/docs/5.0/getting-started/introduction/

For a more seamless integration of Bootstrap with the `Django` Framework an additional package called `django-crispy-forms` was also installed. Its documentation ca be found here:
https://django-crispy-forms.readthedocs.io/en/latest/

2.3.3 Miscellaneous

For typesetting of this documentation, LaTeX was utilised. Additionally, a LaTeX package called `minted` was used to typeset code in this documentation. Its homepage is located at: https://www.ctan.org/pkg/minted

To typeset directory structures in a tree-like manner the LaTeX package `dirtree` was used. Its homepage can be found at: https://ctan.org/pkg/dirtree

To typeset quotations for the Reflection section the LaTeX package `csquotes` was used. Its homepage can bee found at: https://ctan.org/pkg/csquotes?lang=en

Lastly, to typeset code within the HTML pages of our project the JavaScript library called `Rainbow` was implemented. The GitHub link for that is located at:
https://github.com/ccampbell/rainbow

Section 3

Programming of artefact

3.1 Development Tools

3.1.1 Operating Systems

The primary systems on which development was done was Linux and Windows 10. The same systems where utilised for testing and bug fixing purposes.

3.1.2 IDEs

For the purposing of coding the following two Integrated Development Environments were used:

1. Atom. It can be downloaded from: `https://atom.io/`
2. Visual Studio Code, also known as VSCode. It can be downloaded from here: `https://code.visualstudio.com/`

3.1.3 Database Management Tools

For the purposes of database management, TablePlus was the main software we utilised. It was used to see if our `Django` models and cryptographic schemes were correctly implemented. TablePlus can be downloaded from: `https://tableplus.com/`

3.1.4 Hosting

Due to a number of constraints we landed up running our project locally. The server was `localhost` and the port number was `8000`. Therefore the link where we ran our project was: `http://127.0.0.1:8000`

3.2 Prerequisites

3.2.1 Project and Package Initialisation

From here on we will refer to the working directory as the directory where the `manage.py` *file is located. This file is created when the project is setup*

Before our `Django` project can be created we have to install all the packages mentioned above in Section 2.3.1 and 2.3.2. A text file called `requirements.txt` was created which lists the 3 packages we need to install as shown below:

```
django
django-crispy-forms
bootstrap4
```

Thereafter the following command was run in the working directory.

```
pip install -r requirements.txt
```

To start a `Django` project called `SecProj` the following command was run:

```
django-admin startproject SecProj
```

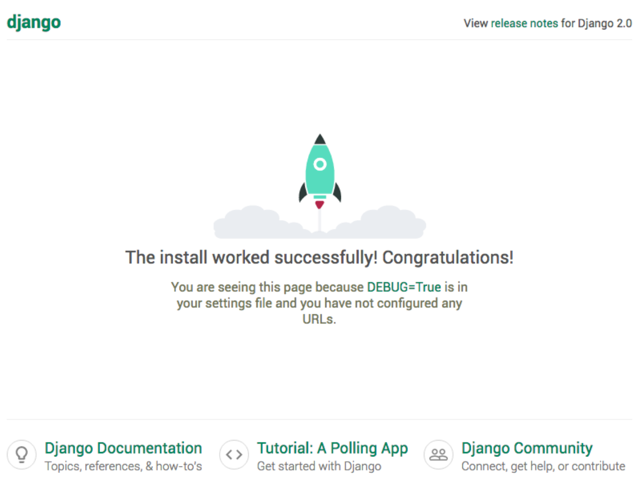
Your directory should look like the following:

```
/
├── manage.py
└── SecProj
    ├── __init__.py
    ├── settings.py
    ├── urls.py
    ├── asgi.py
    └── wsgi.py
```

To verify that your `Django` project is working run the following command in your working directory:

```
python manage.py runserver
```

You should see the following if you open the link: `http://127.0.0.1:8000`


django
View release notes for Django 2.0
Django Documentation
Topics, references, & how-to's
Tutorial: A Polling App
Get started with Django
Django Community
Connect, get help, or contribute

The install worked successfully! Congratulations!

You are seeing this page because DEBUG=True is in your settings file and you have not configured any URLs.

Figure 3.1: Default Homepage of a new Django Project

Now that your Django project is up and running it is time to create a Django 'App' within this project. This App is where we implemented our cryptographic schemes and the bulk of our project. We called our App SecApp and the command to run in your working directory is as follows:

```
python manage.py startapp SecApp
```

Your directory should now look like the following:

```
/
├── manage.py
├── SecProj
│   ├── __init__.py
│   ├── settings.py
│   ├── urls.py
│   ├── asgi.py
│   └── wsgi.py
└── SecApp
    ├── __init__.py
    ├── admin.py
    ├── apps.py
    ├── migrations
    │   └── __init__.py
    ├── models.py
    ├── tests.py
    └── views.py
```

3.2.2 Settings and Admin

The following changes were added to the `settings.py` file. The whole file is not shown below.

```python
from pathlib import Path
import os

# Build paths inside the project like this: BASE_DIR / 'subdir'.
BASE_DIR = Path(__file__).resolve().parent.parent
TEMPLATE_DIR = os.path.join(BASE_DIR, 'templates')

# Application definition

INSTALLED_APPS = [
    'django.contrib.staticfiles',
    'bootstrap4',
    'SecProj',
    'SecApp',
    'accounts',
    'crispy_forms',
]
CRISPY_TEMPLATE_PACK = 'bootstrap4'

ROOT_URLCONF = 'SecProj.urls'
TEMPLATES = [{'DIRS': [TEMPLATE_DIR]}]

# Static files (CSS, JavaScript, Images)
# https://docs.djangoproject.com/en/3.1/howto/static-files/

STATIC_URL = '/static/'
STATICFILES_DIR = [os.path.join(BASE_DIR,'static')]
STATICFILES_DIRS = [
    os.path.join(BASE_DIR, "static"),
]
LOGIN_REDIRECT_URL = 'success'
LOGOUT_REDIRECT_URL = 'thanks'
MEDIA_URL = '/media/'
MEDIA_ROOT = os.path.join(BASE_DIR, 'media')
```

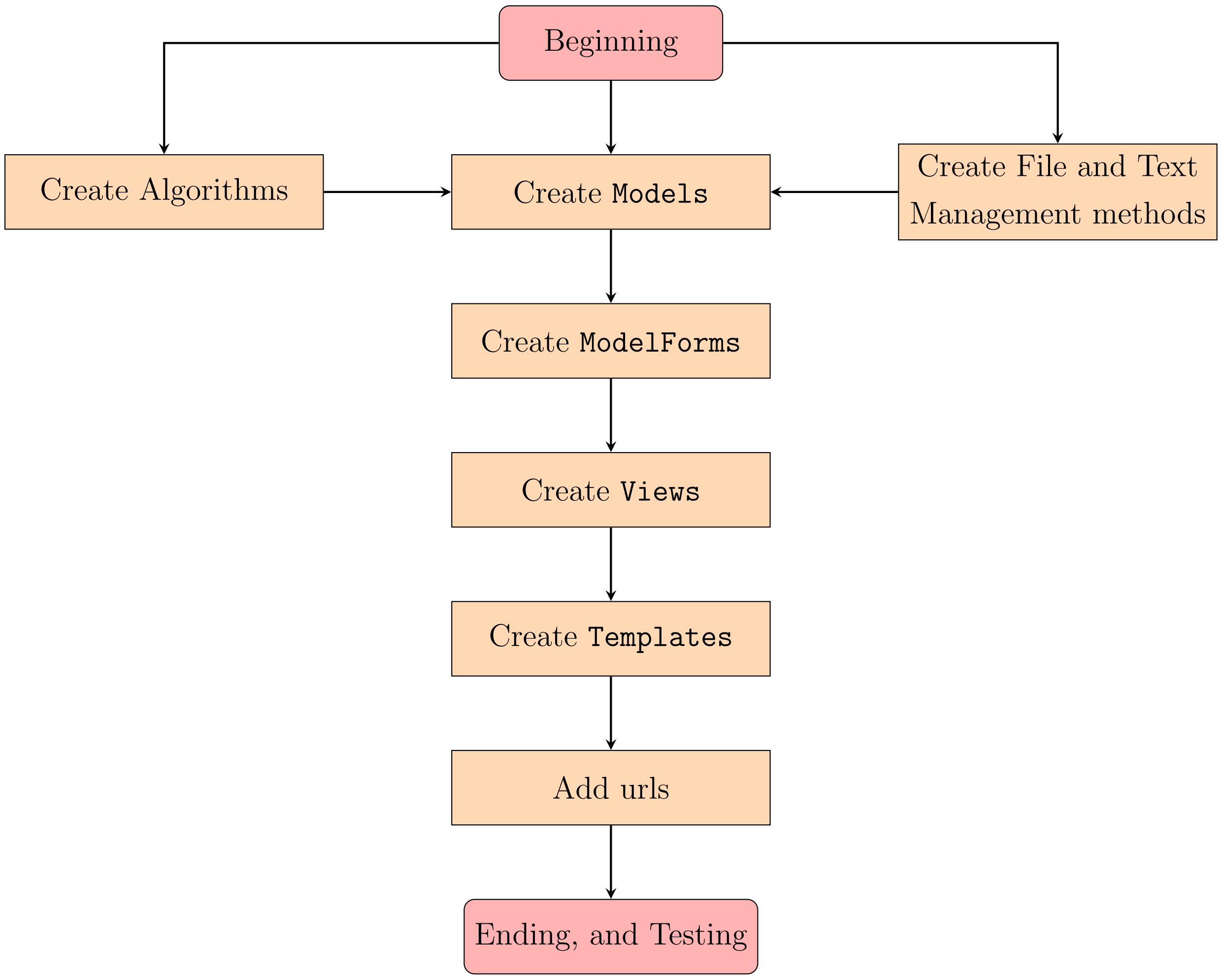
To create a superuser to handle `Django` administration run the following commands and follow the prompts:

```
python manage.py createsuperuser
```

The link for `Django` admininstration for the project is: http://127.0.0.1:8000/admin

3.3 Models

A workflow diagram of how the backend and frontend were created is given below:



The process of creating models was identical for all that were created, therefore for the purpose of brevity we will only look at one text, and one file example. The example will be of the models implemented for the Vigenère cryptographic scheme.

The model for Text Encryption and Decryption with the Vigenère cipher looks exactly the same with a few small differences. Below is the code for Text Encryption and Decryption. This code was added to the `models.py` file.

A few notes to mention before we move to the code:

- The `TextField` was a `ModelField` used due to its capability to store large strings of textual data. It is synonymous with `nvarchar` or `varchar` in other DBMSs.

```python
class VigTextEnc(models.Model):
    user = models.ForeignKey(User, on_delete=models.CASCADE, null=True, blank=True)
    plaintext = models.TextField(null=False, default='')
    ciphertext = models.TextField(null=False,default='')
    key = models.TextField(null=False,default='')
    description = models.TextField(default='Vigenere Text Encryption')

    def save(self, *args, **kwargs):
        self.enc()
        super().save(*args, **kwargs)

    def enc(self, *args, **kwargs):
        self.ciphertext = algorithms.Vigenere_TEXT_Encryption(self.plaintext,self.key)

    def get_absolute_url(self):
        return reverse('SecApp:VigTextEnc_detail', kwargs={'pk':self.pk})

class VigTextDec(models.Model):
    user = models.ForeignKey(User, on_delete=models.CASCADE, null=True, blank=True)
    plaintext = models.TextField()
    ciphertext = models.TextField()
    key = models.TextField(null=False,default='')
    description = models.TextField(default='Vigenere Text Decryption')

    def save(self, *args, **kwargs):
        self.dec()
        super().save(*args, **kwargs)

    def dec(self, *args, **kwargs):
        self.plaintext = algorithms.Vigenere_TEXT_Decryption(self.ciphertext,self.key)

    def get_absolute_url(self):
        return reverse('SecApp:VigTextDec_detail', kwargs={'pk':self.pk})
```

The model for File Encryption and Decryption with the Vigenère cipher looks exactly the same with a few small differences. Below is the code.

A few notes to mention before we move to the code:

- The `FileField` was a `ModelField` used due to its capability to store **metadata** about a file. So in other words its format, url, and path of where the file is located.
- It stores references that point to a certain object after it is uploaded and saved under the `/media/` directory.

```python
class VigFileEnc(models.Model):
    user = models.ForeignKey(User, on_delete=models.CASCADE, null=True, blank=True)
    plaintext = models.FileField(upload_to='', blank=True)
    ciphertext = models.TextField(default='')
    description = models.TextField(default='Vigenere File Encryption')
    key = models.TextField(blank=True,default='')
    ext = models.CharField(max_length=10)

    def save(self, *args, **kwargs):
        super().save(*args, **kwargs)
        self.enc()
        super().save(*args, **kwargs)
    def enc(self, *args, **kwargs):
        THIS_FOLDER = os.path.dirname(os.path.abspath(settings.MEDIA_ROOT))
        new_path = os.path.join(THIS_FOLDER, 'media')
        pt = str(self.plaintext.path)
        plainData = algorithms.fileToByteString(pt)
        cipherData = algorithms.Vigenere_FILE_Encryption(plainData,self.key)
        self.ciphertext = algorithms.byteStringToFile(cipherData,
        ↪  os.path.join(new_path, 'newfile_vig_enc.{0}'.format(self.ext)))
    def get_absolute_url(self):
        return reverse('SecApp:VigFileEnc_detail', kwargs={'pk':self.pk})

class VigFileDec(models.Model):
    user = models.ForeignKey(User, on_delete=models.CASCADE, null=True, blank=True)
    plaintext = models.TextField(default='')
    ciphertext = models.FileField(upload_to='', blank=True)
    description = models.TextField(default='Vigenere File Decryption')
    key = models.TextField(blank=True,default='')
    ext = models.CharField(max_length=10)

    def save(self, *args, **kwargs):
        super().save(*args, **kwargs)
        self.dec()
        super().save(*args, **kwargs)
    def dec(self, *args, **kwargs):
        THIS_FOLDER = os.path.dirname(os.path.abspath(settings.MEDIA_ROOT))
        new_path = os.path.join(THIS_FOLDER, 'media')
        pt = str(self.ciphertext.path)
        plainData = algorithms.fileToByteString(pt)
        cipherData = algorithms.Vigenere_FILE_Decryption(plainData,self.key)
        self.plaintext = algorithms.byteStringToFile(cipherData,
        ↪  os.path.join(new_path, 'newfile_vig_dec.{0}'.format(self.ext)))
    def get_absolute_url(self):
        return reverse('SecApp:VigFileDec_detail', kwargs={'pk':self.pk})
```

Once the `Models` are finalised we can create the `ModelForms`. Below is the snippet of code for the `ModelForms` that use the `Models` shown above.

```python
class VigTextEncModelForm(ModelForm):
    class Meta:
        model = VigTextEnc
        fields = ['plaintext','key']
        labels = {
        "plaintext": "Text to Encrypt",
        "key": "Key",
        }
        widgets = {
        'plaintext': forms.Textarea(attrs={'class':'form-control', 'placeholder':'Enter text here','rows':5,}),
        'key': forms.Textarea(attrs={'class':'form-control', 'placeholder':'Enter ONLY Alphabet Letters','rows':5,}),
        }

        def __init__(self, *args, **kwargs):
            super().__init__(*args, **kwargs)

class VigTextDecModelForm(ModelForm):
    class Meta:
        model = VigTextDec
        fields = ['ciphertext','key']
        labels = {
        "ciphertext": "Text to Decrypt",
        'key':'Key',
        }
        widgets = {
        'ciphertext': forms.Textarea(attrs={'class':'form-control', 'placeholder':'Enter text here','rows':5,}),
        'key': forms.Textarea(attrs={'class':'form-control', 'placeholder':'Enter ONLY Alphabet Letters','rows':5,}),
        }

        def __init__(self, *args, **kwargs):
            super().__init__(*args, **kwargs)
```

The `ModelForms` make it easier to implement a HTML form, thereby streamlining the process. Using `ModelForms` we can specify exactly what fields should be displayed on a form, which ones are mandatory to fill in, which ones are optional, as well as specify additional information to make the User Experience more pleasant for anyone that uses the program.

3.4 Views

Once the Models and ModelForms are completed we can move onto the Views. The primary ones that are used are the CreateView, DetailView, and the TemplateView. These are all classes that inherit from the parent GenericView.

Once again we will look at the Vigenère cipher and the models created for it and how they are integrated with the Views. Below is the snippet of code for the CreateView.

```python
class VigOverviewPage(TemplateView):
    template_name = 'SecApp/vig/overview.html'

class VigTextEncCreate(LoginRequiredMixin,CreateView):
    form_class = forms.VigTextEncModelForm
    template_name = 'SecApp/vig/vig_enc_create_form.html'
    model = models.VigTextEnc

    def form_valid(self, form):
        self.object = form.save(commit=False)
        self.object.user = self.request.user
        self.object.save()
        return super().form_valid(form)

class VigTextDecCreate(LoginRequiredMixin,CreateView):
    form_class = forms.VigTextDecModelForm
    template_name = 'SecApp/vig/vig_dec_create_form.html'
    model = models.VigTextDec

    def form_valid(self, form):
        self.object = form.save(commit=False)
        self.object.user = self.request.user
        self.object.save()
        return super().form_valid(form)

class VigFileEncCreate(LoginRequiredMixin,CreateView):
    form_class = forms.VigFileEncModelForm
    template_name = 'SecApp/vig/vig_enc_create_file.html'
    model = models.VigFileEnc

class VigFileDecCreate(LoginRequiredMixin,CreateView):
    form_class = forms.VigFileDecModelForm
    template_name = 'SecApp/vig/vig_dec_create_file.html'
    model = models.VigFileDec
```

Next we can create the DetailView for the same Models mentioned above.

```python
class VigTextEncDetailView(LoginRequiredMixin,DetailView):
    model = models.VigTextEnc
    context_object_name = 'detail'
    template_name = 'SecApp/vig/vig_text_enc_detail.html'

class VigTextDecDetailView(LoginRequiredMixin,DetailView):
    model = models.VigTextDec
    context_object_name = 'detail'
    template_name = 'SecApp/vig/vig_text_dec_detail.html'

class VigFileEncDetailView(LoginRequiredMixin,DetailView):
    model = models.VigFileEnc
    context_object_name = 'detail'
    template_name = 'SecApp/vig/vig_file_enc_detail.html'

class VigFileDecDetailView(LoginRequiredMixin,DetailView):
    model = models.VigFileDec
    context_object_name = 'detail'
    template_name = 'SecApp/vig/vig_file_dec_detail.html'
```

3.5 Templates

We are almost complete with our implementation. We now have to create `Templates` that link up with the `Views` created above. Below is a snippet of code for Vigenère Text Encryption `View`

```html
{% extends 'base.html'%}
{% load crispy_forms_tags %}
{% block titleblock%}Fill in text{% endblock %}
{% block headblock %}
{% endblock %}
{% block bodyblock %}

<div class="container">
  <div class="container m-5 p-3">
    <h1 style="text-align: center;">Text Encryption with Vigen&egrave;re</h1>
    <form method="post" class="form m-5">
      {% csrf_token %}
      {{ form | crispy}}
      <div style="text-align: center;">
      <input type="submit" value="Encrypt" class="btn btn-success">
      <a class="btn btn-outline-success" href="{% url 'home' %}">Go Back</a>
      </div>
    </form>
  </div>
</div>
```

3.6 URLs

Lastly, we have to add the URLs for the `Views` and `Templates` created above. The code is added to the `urls.py` file. For each `View` that was created a separate URL had to be added. Below is the snippet.

```python
from django.conf.urls import url
from . import views

app_name = 'SecApp'

urlpatterns = [
url(r'^vig/$',views.VigOverviewPage.as_view(),name='vig_overview'),
    url(r'^vig/text/create/enc$',views.VigTextEncCreate.as_view(),
    name='vig_text_create_enc'),
    url(r'^vig/text/create/dec$',views.VigTextDecCreate.as_view(),
    name='vig_text_create_dec'),
    url(r'^vig/file/create/enc$',views.VigFileEncCreate.as_view(),
    name='vig_file_create_enc'),
    url(r'^vig/file/create/dec$',views.VigFileDecCreate.as_view(),
    name='vig_file_create_dec'),
    url(r'vig/text/enc/(?P<pk>\d+)/$',views.VigTextEncDetailView.as_view(),
     name='VigTextEnc_detail'),
    url(r'vig/text/dec/(?P<pk>\d+)/$',views.VigTextDecDetailView.as_view(),
     name='VigTextDec_detail'),
    url(r'vig/file/enc/(?P<pk>\d+)/$',views.VigFileEncDetailView.as_view(),
     name='VigFileEnc_detail'),
    url(r'vig/file/dec/(?P<pk>\d+)/$',views.VigFileDecDetailView.as_view(),
     name='VigFileDec_detail'),
    ]
```

3.7 Migrations

To finally run the program we have to run a few commands. The first make our migrations, the second migrates those changes to our local SQLite database, and the last command runs our server locally.

```
python manage.py makemigrations
python manage.py migrate
python manage.py runserver
```

All these commands, and a few other commands were added to a `script.sh` file for Linux development and in a `script.bat` file for Windows development.

3.8 Algorithms, File and Text Management Methods

First we will display the code located in our `algorithms.py` file then we will discuss the methods in detail.

```python
#Cryptography Project for ITRI615
# J&A's Cryptography Project
import math
import random as r

#Formatting files
#for any type of file :D
def fileToByteString(file):
    byteStream =[]
    f = open(file, 'rb')
    fileData = f.read()
    fileData = bytearray(fileData)

    for item in fileData:
        byteStream.append(item)

    f.close()
    return byteStream

def byteStringToFile(byteStream,file):
    pathStr = str(file)
    f = open(file, 'wb')
    byteStream = bytearray(byteStream)
    f.write(byteStream)
    f.close()
    return pathStr

#Transposition
def keyCheck(key):
    check = True
    for item in key:
        if(not(ord(item) >= 48 and ord(item) <= 57)):
            check = False
    if(check):
        return int(key)
    else:
        return len(key)

#String to Matrix function
def StrToMatrix_TEXT(text,key):
    matrix = []
    row = []
    count = 0
    length = len(text)
    for item in text:
        row.append(item)
        count = count + 1
        length = length - 1
        if(count == key):
            matrix.append(row)
            row = []
            count = 0
        elif(length == 0):
            while(len(row)<key):
                row.append("")
            matrix.append(row)
    return matrix

def StrToMatrix_FILE(text,key):
    matrix = []
    row = []
    count = 0
    length = len(text)
    for item in text:
        row.append(item)
        count += 1
        length -= 1
        if(count == key):
            matrix.append(row)
            row = []
            count = 0
        elif(length == 0):
            while(len(row)<key):
                row.append(item)
            matrix.append(row)
    return matrix

#Text algoritms
def Transposition_TEXT_Encryption(message, key):
    key = keyCheck(key)
    matrix = StrToMatrix_TEXT(message, key)
    encryptedMessage = ""
    for j in range(0,key):
        for item in matrix:
            strList = str(item.pop(0))
            encryptedMessage = encryptedMessage+strList
    return encryptedMessage
```

```

def Transposition_TEXT_Decryption(encryptedMessage, key):
    key = keyCheck(key)
    numberColumns = math.ceil(len(encryptedMessage) / key)
    numberRows = key
    numberBlanks = (numberColumns * numberRows) - len(encryptedMessage)
    decryptedMessage = ['']*numberColumns
    col,row = 0,0

    for item in encryptedMessage:
        decryptedMessage[col] += item
        col += 1
        if (col == numberColumns) or
        (col == numberColumns - 1 and row >= numberRows - numberBlanks):
            col = 0
            row += 1
    return ''.join(decryptedMessage)

def Transposition_FILE_Encryption(message, key):
    key = keyCheck(key)
    matrix = StrToMatrix_FILE(message, key)
    encryptedMessage = []
    origLength = str(len(message))
    lenLen = len(origLength)
    encryptedMessage.append(lenLen)
    for i in range(0,lenLen):
        encryptedMessage.append(ord(origLength[i]))


    for j in range(0,key):
        for item in matrix:
            strList = item.pop(0)
            encryptedMessage.append(strList)
    return encryptedMessage

def Transposition_FILE_Decryption(message,key):
    key = keyCheck(key)

    lenLen = message.pop(0)
    origLengthARR = []
    for i in range(0,lenLen):
        strList = message.pop(0)
        origLengthARR.append(str(chr(strList)))
    origLengthSTR = ''.join(origLengthARR)
    origLength = int(origLengthSTR)

```

```python
    numberColumns = math.ceil(len(message) / key)
    numberRows = key
    numberBlanks = (numberColumns * numberRows) - origLength
    matrix = StrToMatrix_FILE(message, numberColumns)
    decryptedMessageARR = []

    for j in range(0,numberColumns):
        for item in matrix:
            strList = item.pop(0)
            decryptedMessageARR.append(strList)

    for i in range(0,numberBlanks):
        decryptedMessageARR.pop(-1)

    returnMessage =[]
    for item in decryptedMessageARR:
        returnMessage.append(int(item))

    returnMessage = bytearray(returnMessage)

    return returnMessage

#Vignere

#Fixes key length to match size of message
def extendKeyLength(messageLength,key):
    keyLength = len(key)
    newKey = list(key)
    if messageLength == keyLength:
        return(key)

    else:
        for i in range(0,(messageLength - keyLength)):
            newKey.append(key[i % keyLength])
        return ("".join(newKey)).upper()

#Text algorithms
def Vigenere_TEXT_Encryption(message, key):
    message = message.upper()
    messageLength = len(message)
    encryptedMessage = []
    key = extendKeyLength(messageLength, key)
    for i in range(0,messageLength):
        if ord(message[i]) < 65 or ord(message[i]) > 90:
            encryptedMessage.append(message[i])
        else:
            newEncrypted = ord(message[i]) + ord(key[i])
            newEncrypted = newEncrypted % 26
            newEncrypted += ord('A') #gets ascii values back to "letters area"
            encryptedMessage.append(chr(newEncrypted))

    return "".join(encryptedMessage)

def Vigenere_TEXT_Decryption(encryptedMessage, key):
    decryptedMessage = []
    messageLength = len(encryptedMessage)
    key = extendKeyLength(messageLength, key)
    for i in range(0,messageLength):
        if ord(encryptedMessage[i]) < 65 or ord(encryptedMessage[i]) > 90:
            decryptedMessage.append(encryptedMessage[i])
        else:
            newDecrypted = (ord(encryptedMessage[i]) - ord(key[i])) % 26
            newDecrypted += ord('A')
            decryptedMessage.append(chr(newDecrypted))

    return "".join(decryptedMessage)

#File algortitms
def Vigenere_FILE_Encryption(message, key):
    messageLength = len(message)
    encryptedMessage = []
    key = extendKeyLength(messageLength, key)
    for i in range(0,messageLength):
            newEncrypted = ((message[i]) + ord(key[i])) % 256
            encryptedMessage.append(newEncrypted)

    return encryptedMessage

def Vigenere_FILE_Decryption(message, key):
    messageLength = len(message)
    decryptedMessage = []
    key = extendKeyLength(messageLength, key)
    for i in range(0,messageLength):
        newDecrypted = (message[i] - ord(key[i])) % 256
        decryptedMessage.append(newDecrypted)

    return decryptedMessage

#Vernam
#Key generators
def vernam_Key_Generator(messageLength):
    key = []
```

```python
    for i in range(0,messageLength):
        char = chr(r.randrange(65,91))
        key.append(char)
    return ("".join(key)).upper()

def vernam_Key_Generator_FILE(messageLength):
    key = []
    for i in range(0,messageLength):
        char = chr(r.randrange(0,256))
        key.append(ord(char))
    return key

#Text algorithms
def Vernam_TEXT_Encryption(message):
    messageStrip = message.replace(" ","")
    messageStrip = messageStrip.upper()
    messageLength = len(messageStrip)
    key = vernam_Key_Generator(messageLength)
    encryptedMessage = "" + key
    encryptedMessage = list(encryptedMessage)
    for i in range(0,messageLength):
        newEncrypted = (ord(messageStrip[i]) + ord(key[i])) % 26
        newEncrypted += 65
        encryptedMessage.append(chr(newEncrypted))
    return "".join(encryptedMessage)

def Vernam_TEXT_Decryption(message):
    messageLength = int(len(message)/2)
    key = []
    baseMessage = []
    decryptedMessage = []

    for i in range(0,messageLength):
        key.append(message[i])
        baseMessage.append(message[(i+messageLength)])

    for j in range(0,messageLength):
        newDecrypted = (ord(baseMessage[j]) - ord(key[j])) % 26
        newDecrypted += 65
        decryptedMessage.append(chr(newDecrypted))

    return "".join(decryptedMessage)

#File algorithms
def Vernam_FILE_Encryption(message):
    messageLength = len(message)
    key = vernam_Key_Generator_FILE(messageLength)
    baseMessage = []
    encryptedMessage = []
    for j in key:
        encryptedMessage.append(j)
    for i in range(0,messageLength):
        newEncrypted = (message[i] + key[i]) % 256
        baseMessage.append(newEncrypted)


    encryptedMessage = [key,baseMessage]
    returnMessage = []
    for item in encryptedMessage:
        for thing in item:
            returnMessage.append(thing)

    return returnMessage

def Vernam_FILE_Decryption(message):
    messageLength = int(len(message)/2)
    key = []
    baseMessage = []
    decryptedMessage = []

    for i in range(0,messageLength):
        key.append(message[i])
        baseMessage.append(message[(i+messageLength)])

    for j in range(0,messageLength):
        newDecrypted = (baseMessage[j] - key[j]) % 256
        decryptedMessage.append(newDecrypted)

    return decryptedMessage

#Own algorithm
def own_TEXT_Encryption(message, key):
    key = keyCheck(key)
    messageLength = len(message)
    numericEncryptedValues = []
    for item in message:
        numeric = ord((item))
        mathPart = (((key*numeric)) - key)
        numericEncryptedValues.append(mathPart)

    randomKey = []
    for i in range(0,messageLength):
        num = r.randrange(32,256)
        randomKey.append(chr(num))

    encryptedMessage = []
    for i in range(0,messageLength):
        encryptedMessage.append(chr((numericEncryptedValues[i]+ord(randomKey[i]))))

    addedKey = ''.join(randomKey)
    strEncryptedMessage = ''.join(encryptedMessage)

    return (addedKey+strEncryptedMessage)

def own_TEXT_Decryption(message, key):
    key = keyCheck(key)
    messageLength = int(len(message)/2)
    encryptedMessage = []
    randomKey = []

    for i in range(0,messageLength):
        encryptedMessage.append(ord(message[i+messageLength]))
        randomKey.append(ord(message[i]))

    partA = []
    for j in range(0,messageLength):
        partA.append(chr((encryptedMessage[j]-randomKey[j])))

    decryptedMessage = []
    for item in partA:
        numeric = ord(item)
        unMathPart = ((numeric+key)/key)
        decryptedMessage.append(chr(int(unMathPart)))

    return ''.join(decryptedMessage)

def own_FILE_Encryption(message, key):
    key = keyCheck(key)
    messageLength = len(message)
    numericEncryptedValues = []
    for item in message:
        mathPart = (item - key)
        numericEncryptedValues.append(mathPart)

    randomKey = []
    for i in range(0,messageLength):
        num = r.randrange(0,256)
        randomKey.append(num)

    combined = []
    for i in range(0,messageLength):
        combined.append((numericEncryptedValues[i]+randomKey[i])%256)

    encryptedMessage = [randomKey,combined]
    returnMessage = []
    for item in encryptedMessage:
        for thing in item:
            returnMessage.append(thing)

    return returnMessage

def own_FILE_Decryption(message, key):
    key = keyCheck(key)
    messageLength = int(len(message)/2)
    encryptedMessage = []
    randomKey = []

    for i in range(0,messageLength):
        encryptedMessage.append(message[i+messageLength])
        randomKey.append(message[i])

    partA = []
    for j in range(0,messageLength):
        partA.append((encryptedMessage[j]-randomKey[j]))

    decryptedMessage = []
    for item in partA:
        unMathPart = (item+key)
        decryptedMessage.append(unMathPart%256)

    return decryptedMessage
```

`fileToByteString()` Method
This method takes in a file/file path and reads it in binary mode. This data is then converted to a bytes array such that it can be worked on by other functions.

`byteStringToFile()` Method
This function takes in an bytes array of decimal values and writes them to the file/file path provided.

`keyCheck(key)` Method
This function takes the provided key and checks if it is a integer or string and if a string returns the length of it to use as a key.

`StrToMatrix_TEXT()` Method
This function takes in a text string and a numerical key. The key is used to stipulate the number of desired columns in the matrix. The function then rearranges the string into the matrix and accordingly returns it. It also pads the last row if required.

`Transposition_TEXT_Encryption()` Method
This function takes in a string and a numerical key and passes them to `StrToMatrix_TEXT()`. This function then applies a simple transposition cipher by reading the data from the matrix column wise.

`Transposition_TEXT_Decryption()` Method
This function takes in a string and a numerical key. The function then uses various mathematical principles to work out how to read the encrypted text back; it does this by essentially transposing the matrix again. As such, it compiles a matrix to accomplish this and then returns a string through the .join() function.

`StrToMatrix_FILE()` Method
Works similar to `StrToMatrix_TEXT()` with the exception of what is used as padding - a null byte transformed into an integer value; i.e. `ord("\u0000")`.

`Transposition_FILE_Encryption()` Method
Works similar to `Transposition_TEXT_Encryption()` with the exception of returning a numerical array instead of a string as well as taking in a byte array in place of a string.

`Transposition_FILE_Decryption()` Method
Works similar to `Transposition_TEXT_Decryption()` with the exception of returning a numerical array instead of a string and removing the added padding as well as taking in a byte array in place of a string.

`extendKeyLength()` Method
This function takes the length of the string given and the key to be used and ensures the key matches the length and if not duplicates it until it does. There is no need for a shortening function due to how the Vigenere Algorithm uses.

`Vigenere_TEXT_Encryption()` Method
This function takes in a string and a key. It then converts the plaintext to uppercase letters. Instead of making use of a Vigenere Table - this function makes use of the numerical ASCII values of characters and adds them in modulo 26. It then adds the value of 65, or ord('A') to get all character values back to the range of letters and returns these as a string. It also includes a check to ignore any non-letter based characters as a Vigenere table is only alphabet letters.

`Vigenere_TEXT_Decryption()` Method
Works similar to the encryption but instead subtracts the values before adding 65.

`Vigenere_FILE_Encryption()` Method
Works as with Vigenere_TEXT_Encryption() but is applied within modulo 256 to encompass all possible characters as well as taking in a byte array in place of a string.

`Vigenere_FILE_Decryption()` Method
Works as with `Vigenere_TEXT_Decryption()` but is applied within modulo 256 to encompass all possible characters as well as taking in a byte array in place of a string.

The method of adding and subtracting numerical ASCII values has been found by us to consistently produce the same results as using a Vigenere table and as such was used as it would decrease look-up and computational time - especially on larger files.

`vernam_Key_Generator()` Method
This function takes in the length of a message/string and produces a random string of characters (within the capital letter set) to be used in the encryption functions.

`Vernam_TEXT_Encryption()` Method
This function takes in a message/string to be encrypted and calls `vernam_Key_Generator()` to create a unique one-time key. The ascii values of the message and key are added within modulo 26. The function then adds the required value to have the characters to be letter characters again and returns this as a string. Due to the random key generation, the key is stored as part of the resulting encrypted text.

`Vernam_TEXT_Decryption()` Method
This function begins by separating the key from the encrypted text and then subtracts the values in modulo26, takes these values back to the range of letter characters and returns this as a string.

`vernam_Key_Generator_FILE()` Method
This function works similar to `vernam_Key_Generator()` except it generates an array of numerical values (between 0 and 255) as a key.

`Vernam_FILE_Encryption()` Method
Works like `Vernam_TEXT_Encryption()` except it takes in a numerical array of byte values and is done within modulo256.

`Vernam_FILE_Decryption()` Method
Works like `Vernam_TEXT_Decryption()` except it takes in a numerical array of byte values and is done within modulo256.

`own_TEXT_Encryption()` Method
This function takes in a plaintext string and well as a integer key. The individual characters of the string are subjected to a mathematical calculation based on the provided key and added to an array. Afterwards, a randomly generated key of the same length is created consisting of values from 0 to 255 to be added to the array as well before being added to the array as well. A string result of these is then returned.

`own_TEXT_Decryption()` Method
This function takes in a ciphertext string and a key. The cipher text is split into the encrypted text and a random key used in its' generation. The encrypted text is covered to numerical ascii values and have the random-key's values subtracted from it before having the inverse of a mathematical calculation based on the value of the key argument. This result is then concatenated and returned.

`own_FILE_Encryption()` Method
Works very similar to `own_TEXT_Encryption()` except is takes in a byte array in place of a string and the values of the mathematical calculations are done in modulo256 to stay within the 0-255 byte limitations.

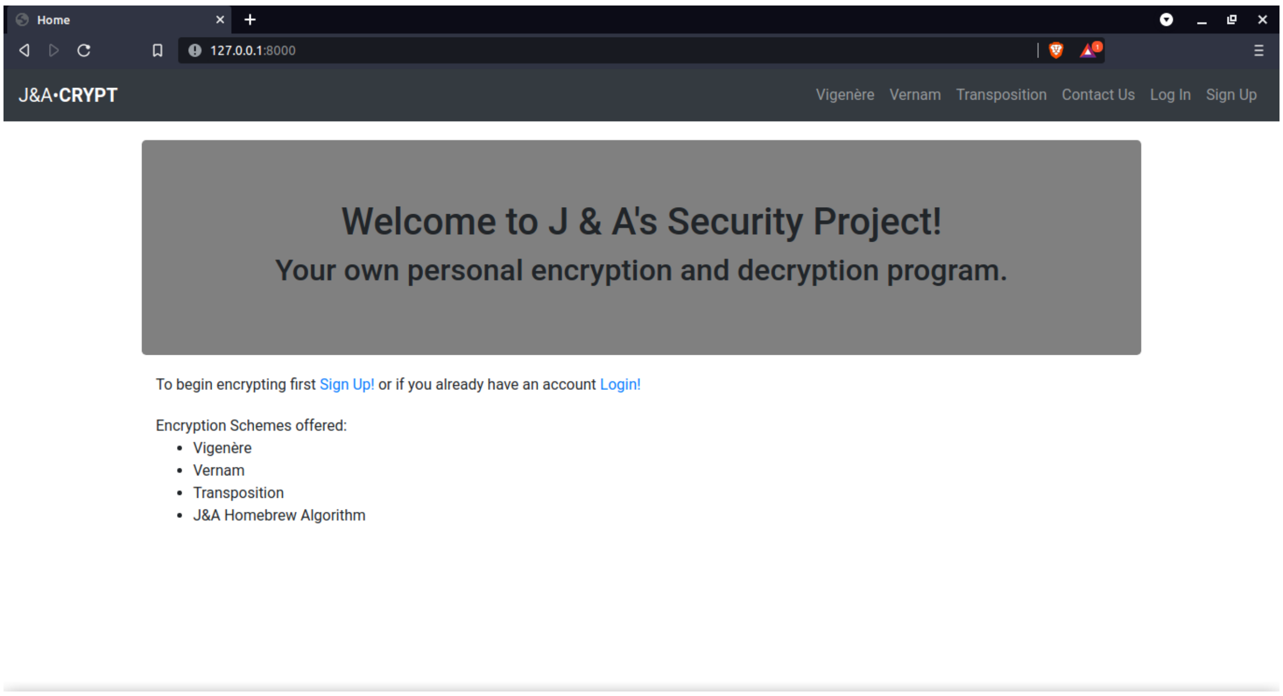
`own_FILE_Decryption()` Method
Works very similar to `own_TEXT_Decryption()` except is takes in a byte array in place of a string and the values of the mathematical calculations are done in modulo256 to stay within the 0-255 byte limitations.

Section 4

User manual

4.1 Homepage

The landing page for our site is located at http://127.0.0.1:8000/ and look like the picture shown below:



Welcome to J & A's Security Project!

Your own personal encryption and decryption program.

To begin encrypting first Sign Up! or if you already have an account Login!

Encryption Schemes offered:

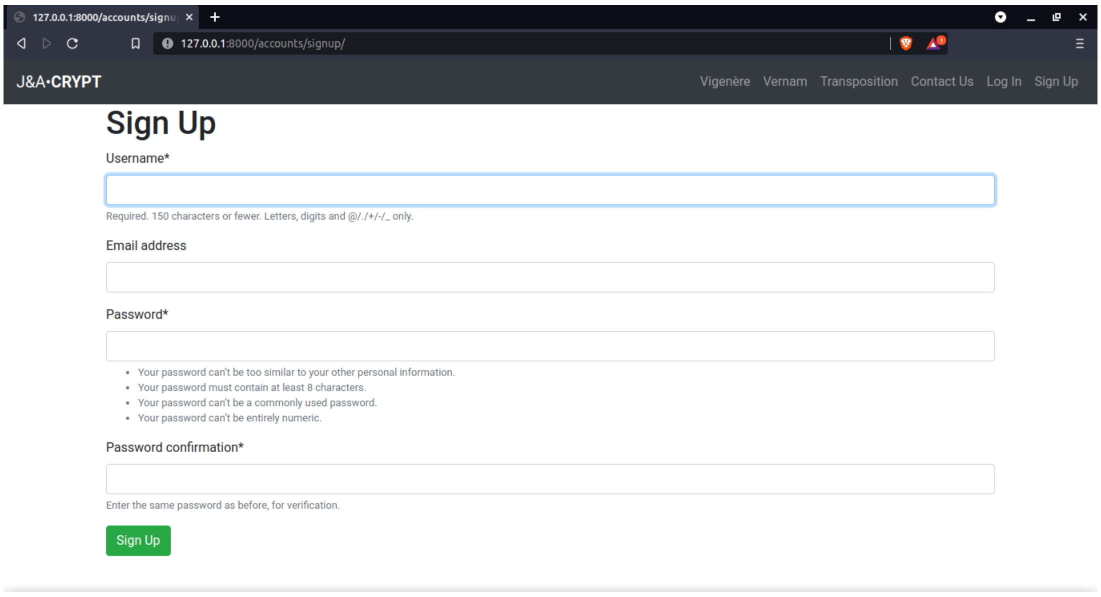
- Vigenère
- Vernam
- Transposition
- J&A Homebrew Algorithm

Figure 4.1: Our Homepage

4.2 Login and Signup

To use the features of our site you have to create an account. This can be done by clicking on the Sign Up button on the navigation bar or by going to the link:
http://127.0.0.1:8000/accounts/signup/

Some key features of our program include the process of password strength testing. This means if your password is weak you will receive feedback telling you to change it.



Sign Up

Username*

Required. 150 characters or fewer. Letters, digits and @/./+/-/_ only.

Email address

Password*

- Your password can't be too similar to your other personal information.
- Your password must contain at least 8 characters.
- Your password can't be a commonly used password.
- Your password can't be entirely numeric.

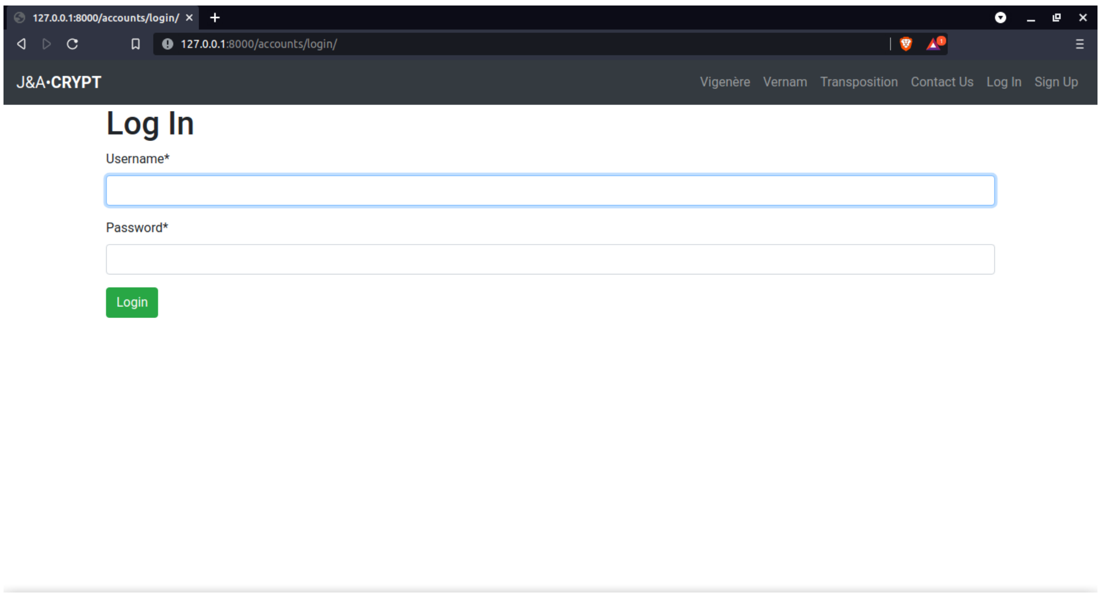
Password confirmation*

Enter the same password as before, for verification.

Sign Up

Figure 4.2: Our Sign Up page

If you are already a registered user you will be able to login. This can be done by clicking on the Log In button or going to the link:
http://127.0.0.1:8000/accounts/login/



Log In

Username*

Password*

Login

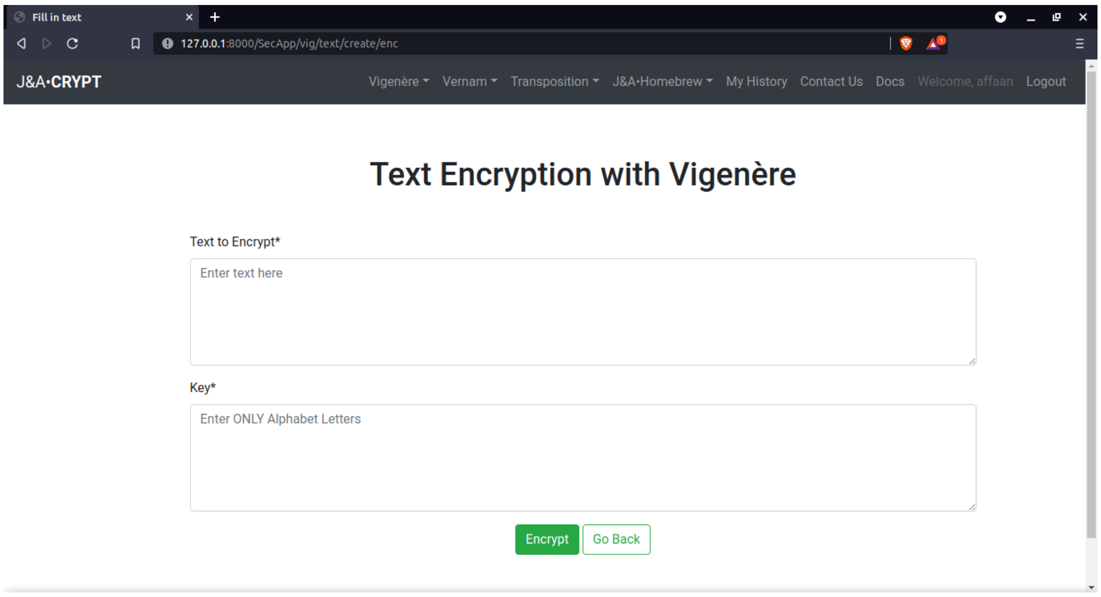
Figure 4.3: Our Log In page

4.3 Text Encryption & Decryption

Once you are logged in you will be able to use the different facilities offered by our program. Our program offers Encryption & Decryption of Text by using 4 different cryptographic schemes i.e. `Vernam`, `Vigenere`, `Transposition`, and our own `J&A Homebrew`. To Encrypt text follow the following steps:

1. Click on drop-down list with the scheme you want to utilise.
2. Select **Text Encryption** or **Text Decryption**.
3. Fill in the form that pops us.
4. Click on **Encrypt** or **Decrypt**

Below is an example Screenshot of a Vigenere Text Encryption and Decryption Form.



Text Encryption with Vigenère

Text to Encrypt*

Enter text here

Key*

Enter ONLY Alphabet Letters

Figure 4.4: Example of our Vigenere Text Encryption form

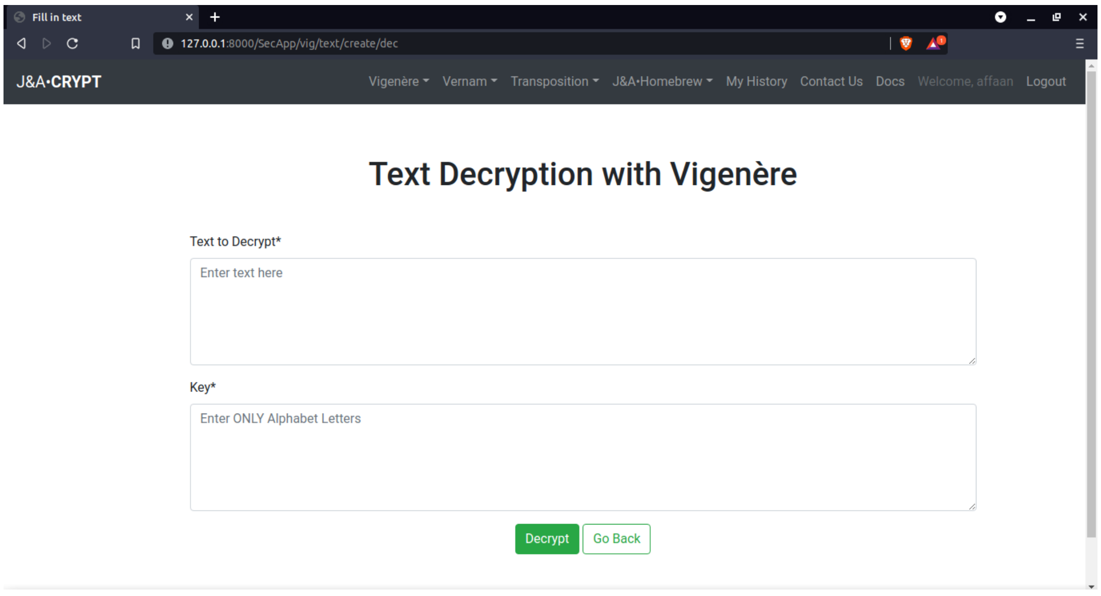


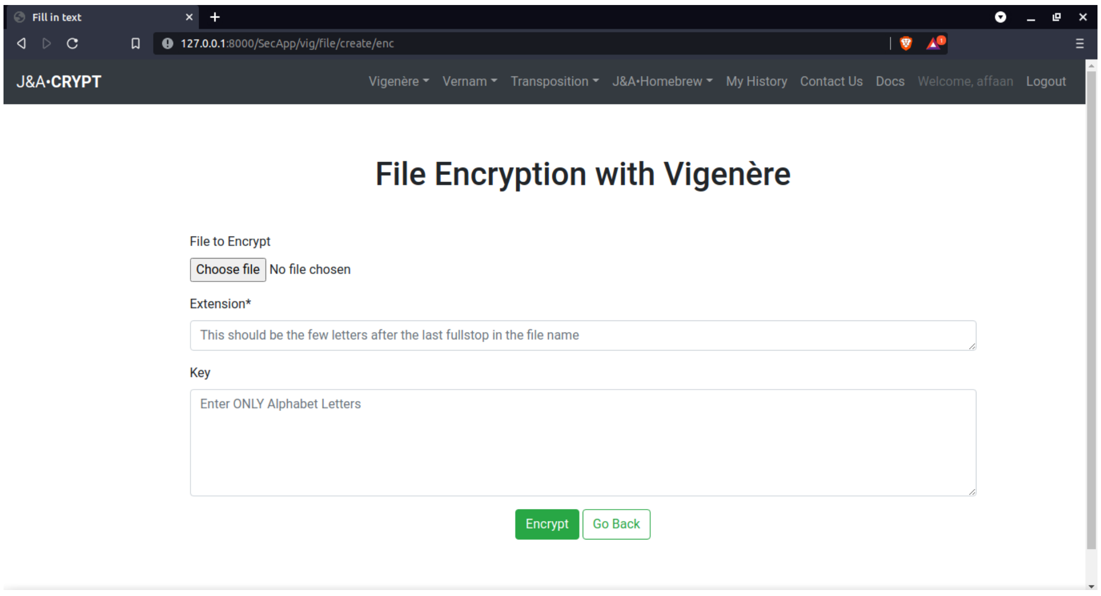
Figure 4.5: Example of our Vigenere Text Decryption form

4.4 File Encryption & Decryption

Once you are logged in you will be able to use the different facilities offered by our program. Our program offers Encryption & Decryption of Files by using 4 different cryptographic schemes i.e. `Vernam`, `Vigenere`, `Transposition`, and our own `J&A Homebrew`. To Encrypt or Decrypt Files follow the following steps:

1. Click on drop-down list with the scheme you want to utilise.
2. Select **File Encryption** or **File Decryption**.
3. Fill in the form that pops us.
4. Click on **Encrypt** or **Decrypt**

Below is an example Screenshot of a Vigenere File Encryption and Decryption Form.



File Encryption with Vigenère

File to Encrypt

Choose file No file chosen

Extension*

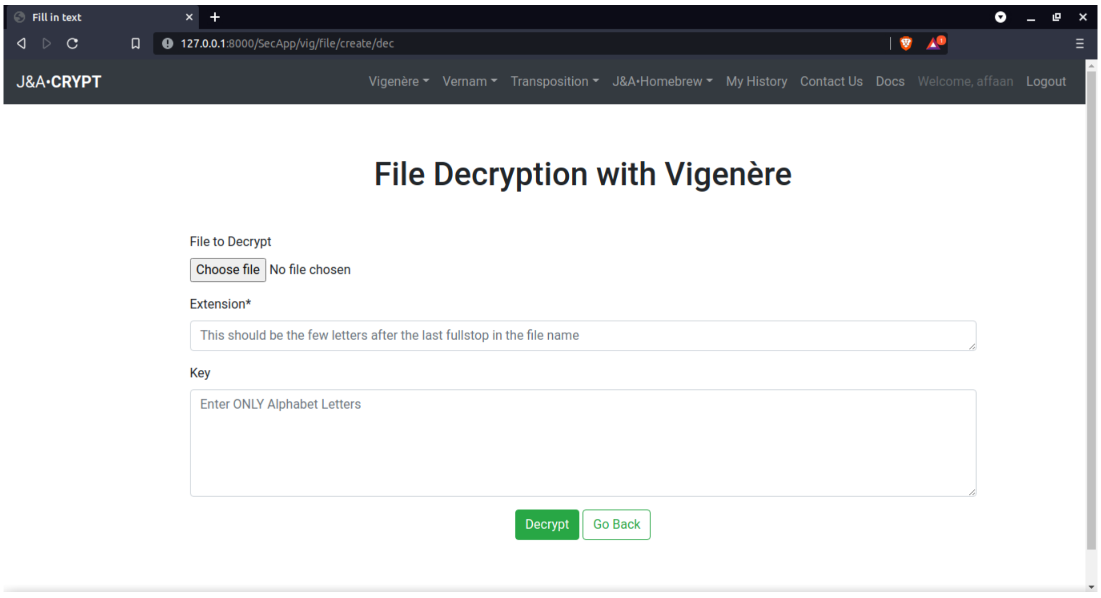
This should be the few letters after the last fullstop in the file name

Key

Enter ONLY Alphabet Letters

Encrypt Go Back

Figure 4.6: Example of our Vigenere Text Encryption form



File Decryption with Vigenère

File to Decrypt

Choose file No file chosen

Extension*

This should be the few letters after the last fullstop in the file name

Key

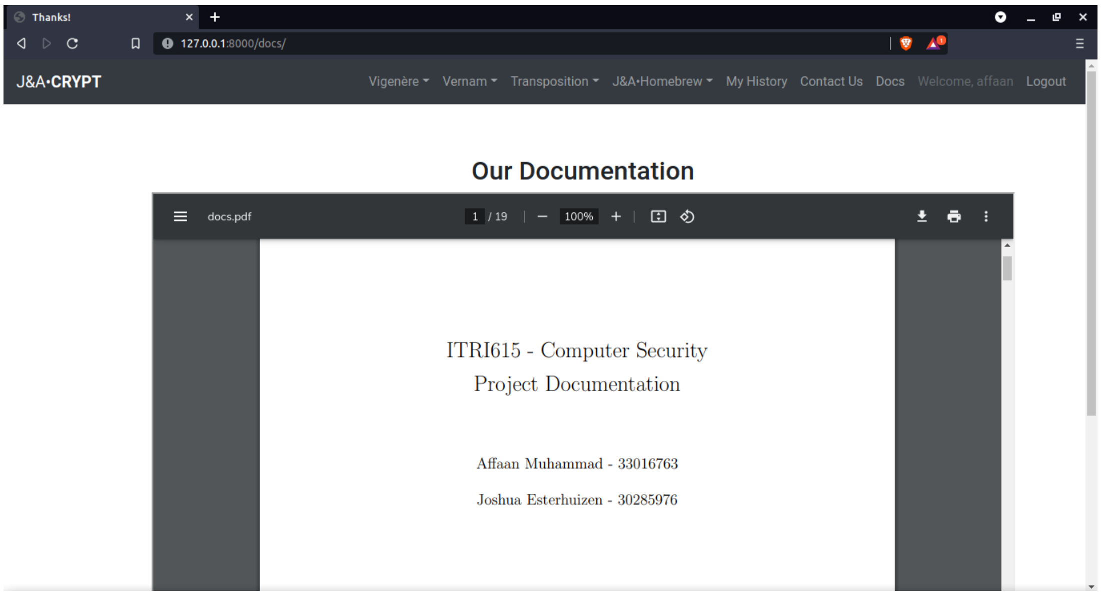
Enter ONLY Alphabet Letters

Decrypt Go Back

Figure 4.7: Example of our Vigenere File Decryption form

4.5 Documentation and Contact

Our documentation can be found at `http://127.0.0.1:8000/docs/` or by clicking on the Docs button on the navigation bar.



Our Documentation

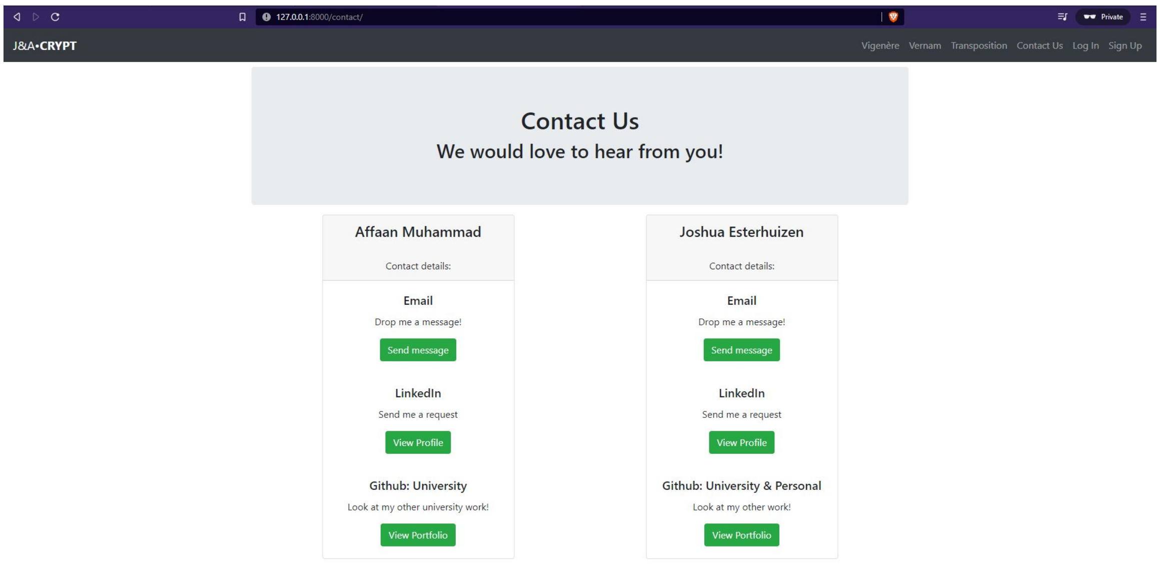
ITRI615 - Computer Security
Project Documentation

Affaan Muhammad - 33016763

Joshua Esterhuizen - 30285976

Figure 4.8: Our Documentation page

Our contact details can be found at http://127.0.0.1:8000/contact/ or by clicking on the Contact Us button on the navigation bar.



Contact Us
We would love to hear from you!

Affaan Muhammad	Joshua Esterhuizen
Contact details:	Contact details:
Email	Email
Drop me a message!	Drop me a message!
Send message	Send message
LinkedIn	LinkedIn
Send me a request	Send me a request
View Profile	View Profile
Github: University	Github: University & Personal
Look at my other university work!	Look at my other work!
View Portfolio	View Portfolio

Figure 4.9: Our Contact Us page

Section 5

Closing remarks

5.1 Reflection

Joshua Esterhuizen had the following to say:

> While developing the Vigenère and Vernam algorithms, it was very interesting how similar they are to each other in their encryption and decryption methods digitally as, through our implementation, both made use of the ASCII values of characters.
> While Python has great success when handling text-based files (.txt, .csv, etc.) it was not as effective when it came to other formats such as .png and .mp3 and as such imposed certain restrictions on how our algorithms could function - the most notable being that we could not alter the data type of the contents of a file to anything other than in integer byte value as it would seem that the encoding used on these is not one of the common ones like UTF-8 or UTF-32 and as such forced us to implement two "modes" for each algorithm - one for text and one for any file (.txt included)
> There was also an instance when testing the Vernam cipher against a .png file where the encrypted file was actually not "corrupted" (as all bytes in a file are used this included file-type specifications) and appeared as an image of a few white stripes (nothing like the original). This is interesting as the OTP generated in that instance must have had a sequence that allowed the file-type specification to still be readable and as such the file could be opened. This does pose an interesting question that if a key could be generated with certain values, could the encrypted file or text mirror the original due to the modulo calculations? While our Vernam implementation shouldn't lead to this as the OTP is diffused within the encrypted contents - it could happen with the Vigenère Cipher as the user must stipulate the key both times and it is not stored. It is very very highly unlikely to happen on file encryption due to the sheer amount of data that this would need to happen to, but for the text encryption, it could (although still very unlikely).

Affaan Muhammad had the following to say:

> Communication was key to collaboration and the success of our project. It gives a good indication of how things will be in the industry where team development is a norm. The communication channel had to remain open and honest throughout the entire process as problems couldn't be fixed on either the backend or the frontend if the understanding was not there. For security to be successful you have to have decent documentation as well as guidance for a user. Therefore we used that in mind to help make it easy for a user to use our program and also receive feedback on what they are doing. File Management seemed to be a bit tricky throughout the development process. Luckily we put our heads together and overcame the issues. Learning to use CI/CD techniques definitely helped us collaborate more easily and keep our work up to date.

5.2 Work Consensus

Below is a table showing how the work was divided in this project amongst the 2 members.

Affaan	Joshua
Graphical User Interface	Algorithms for ciphers
`Models`, `Views`, and `Templates`	File and Text Management methods
Testing	Testing
Bug fixes	Bug fixes
Video	Video
Documentation	Overview and Background of ciphers

Section 6

Sources and Acknowledgements

https://conda.io/projects/conda/en/latest/user-guide/tasks/manage-environments.html
https://www.udemy.com/course/python-and-django-full-stack-web-developer-bootcamp/
https://docs.djangoproject.com/en/3.2/
https://docs.python.org/3/library/math.html
https://docs.python.org/3/library/random.html
https://docs.python.org/3/c-api/list.html
https://docs.python.org/3/tutorial/inputoutput.html
https://youtu.be/h12g__mLboo
https://youtu.be/T9Fe-WD5Bvw