ORTHOGONAL MATRIX CRYPTO-SYSTEM

JAMES DELLA-GIUSTINA

1. Introduction

We aim to implement a cryptography system proposed by Yeray Santana [4] that utilizes properties of orthogonal matrices in order to encode and decode plain-text. The premise of Santana's system hinges on using an orthogonal matrix A to encode/decode some cipher-text numerical vector M that would produce an encrypted vector X:

$$X = AM$$

Where the decryption process to recover your cipher-text numerical values would be:

$$M = A^{-1}X$$

Let's identify the properties of orthogonal matrices that we can exploit in order to replicate the cryptosystem.

For any orthogonal matrix Q:

(1) Possesses a full set of distinct eigenvalues λ_i.
(2) $\forall \lambda_i \in \mathbb{R}$.
(3) Property (1) automatically implies that Q is diagonalizable such that $Q = D\Lambda D^{-1}$ where $\Lambda =$

$$\begin{pmatrix} \lambda_1 & 0 & 0 & \dots & 0 \\ 0 & \lambda_2 & 0 & \dots & 0 \\ 0 & 0 & \lambda_3 & \dots & 0 \\ \vdots & \ddots & \ddots & \ddots & \vdots \\ 0 & \dots & \dots & \dots & \lambda_i \end{pmatrix}$$

(4) Each column of Q is linearly independent, so $Q^{-1} = Q^T$ iff the norm of each column vector of Q is equal to 1 because Q^TQ:

$$q_i^T q_j = \begin{cases} 1 \text{ iff } i = j \\ 0 \text{ iff } i \neq j \end{cases}$$

So since $Q^TQ = \mathbb{I}$, $Q^T = Q^{-1}$.

(5) Given Q is orthogonal, then so is the eigenvector matrix D and from Property (4) we have $D^{-1} = D^T$.

All of these properties are established and proven in [5]. Additionally, we know that for any diagonalizable matrix $Q = D\Lambda D^{-1}$, then the matrix exponential:

$$\begin{aligned} e^{Qt} &= I + Qt + \frac{(Qt)^2}{2!} + \frac{(Qt)^3}{3!} + \dots \\ &= D(I + Qt + \frac{(Qt)^2}{2!} + \frac{(Qt)^3}{3!} + \dots)D \\ \therefore \quad &= De^{\Lambda t}D^{-1} \end{aligned}$$

Santana states that for any function $f(Q)$ with a Taylor series expansion and a diagonal matrix Λ, then:

$$f(Q) = D \begin{pmatrix} f(\lambda_1) & 0 & \dots & 0 \\ 0 & f(\lambda_2) & \dots & 0 \\ \vdots & \ddots & \ddots & \vdots \\ 0 & \dots & \dots & f(\lambda_n) \end{pmatrix} D^{-1}$$

Santana's first step to construct a crypto-system by taking some plain-text and converting it to some numerical values according to a bijective mapping, breaking the list of values into 'blocks' of 4 values. Each cipher-text numerical value corresponds to some λ_i in a diagonal matrix Λ, where an encryption process produces an encrypted vector as:

$$\texttt{encrypt} = (D^T)^j \begin{pmatrix} f(\lambda_1) & 0 & \dots & 0 \\ 0 & f(\lambda_2) & \dots & 0 \\ \vdots & \ddots & \ddots & \vdots \\ 0 & \dots & \dots & f(\lambda_n) \end{pmatrix} D^j \begin{pmatrix} \lambda_1 \\ \lambda_2 \\ \vdots \\ \lambda_n \end{pmatrix}$$

and a decrypted vector as:

$$\begin{aligned} \texttt{decrypt} &= D^j\ \texttt{encrypt} \\ &\equiv D^j\ (D^T)^j \begin{pmatrix} f(\lambda_1) & 0 & \dots & 0 \\ 0 & f(\lambda_2) & \dots & 0 \\ \vdots & \ddots & \ddots & \vdots \\ 0 & \dots & \dots & f(\lambda_n) \end{pmatrix} D^j \begin{pmatrix} \lambda_1 \\ \lambda_2 \\ \vdots \\ \lambda_n \end{pmatrix} \\ &= \begin{pmatrix} f(\lambda_1) & 0 & \dots & 0 \\ 0 & f(\lambda_2) & \dots & 0 \\ \vdots & \ddots & \ddots & \vdots \\ 0 & \dots & \dots & f(\lambda_n) \end{pmatrix} D^j \begin{pmatrix} \lambda_1 \\ \lambda_2 \\ \vdots \\ \lambda_n \end{pmatrix} \end{aligned}$$

We note that for each encryption/decryption step, we raise our orthogonal matrix D and its transpose to the $\texttt{j}^{th}$ power and increment `j`. The function we chose to implement was the hyperbolic cosine denoted *cosh*, of which we give justification in Section 2.

2. Implementation

To implement the crypto-system, we chose Python for its extensive linear algebra libraries.

```python
import numpy as np
import sympy as sym
```

```
from numpy import pad
from sympy import *
from sympy.functions import log
from scipy.stats import ortho_group
import itertools
from sympy.matrices import Matrix, eye, zeros, ones, diag, GramSchmidt
```

Specifically we choose to import the libraries numerical python (NumPy), symbolic python (SymPy), itertools, and orthogonal group from SciPy which is a library used for scientific computing. Our requirements for the project included using matrices of size at least 7×7 as opposed to the 4×4 matrices used in the paper, where 7 and 4 correspond to the lengths of the cipher-text being encoded at each step. Explicitly this means for the 4×4 matrices (D^T, Λ, D), we are encoding 4 numbers (corresponding to our cipher-text) at each step in the encryption/decryption process. We generate a random 8×8 orthogonal matrix x calling from Scipy's ortho_group function.

```
x = ortho_group.rvs(8)
```

In order to have equal access to the code base among all team members, we utilized GoogleColaboratory which is the analog of GoogleDocs, but for programming in Python. To read in a file for encryption we needed to 'mount' a users GoogleDrive and read in a file.

```
with open('/content/drive/MyDrive/text_to_encode.txt', 'r') as f:
textfromfile = f.read()
```

Now Python has read everything in from our file, we can go ahead and now use a 'dictionary' datatype to change from the plain-text to a cipher-text and appropriately named encoder. Notice that decoder is essentially the inverse map that encoder defines.

```
# Setup this empty string encodedtext
encodedtext = []
# Change plaintext into 'ciphertext'
encoder = {
       'a': '27', 'b': '28', 'c': '29', 'd': '30', 'e': '31', 'f': '32',
       'g': '33', 'h': '34', 'i': '35', 'j': '36', 'k': '37', 'l': '38',
       'm': '39', 'n': '40', 'o': '41', 'p': '42', 'q': '43', 'r': '44',
       's': '45', 't': '46', 'u': '47', 'v': '48', 'w': '49', 'x': '50',
       'y': '51', 'z': '52',

       '0': '68', '␣': '72','2':'56', '3' :'57', '4':'58','5':'59',
       '6': '53', '7':'54', '8':'55','9':'56', '␣␣':'71',

       '!': '61', '.': '62', '?': '63', ':': '64', ';': '65', '/': '66',
       '"': '66','\'': '67', '': '69', ',': '70',

       'A': '60', 'B': '2', 'C': '3', 'D': '4', 'E': '5', 'F': '6',
       'G': '7', 'H': '8', 'I': '9', 'J': '10', 'K': '11', 'L': '12',
       'M': '13', 'N': '14', 'O': '15', 'P': '16', 'Q': '17', 'R': '18',
       'S': '19', 'T': '20', 'U': '21', 'V': '22', 'W': '23', 'X': '24',
        'Y': '25', 'Z': '26',
}
# Change 'ciphertext' into plaintext
decoder = {};
for i in encoder:
    decoder[encoder[i]] = i

# Reading in our plain text and encoding it
for i in range(0, len(textfromfile)):
    encodedtext.append(encoder[textfromfile[i]])

# Pad the encoded text with 72's so that its divisible by 8
encodedtext_pad = np.pad(encodedtext, (0, 8-len(encodedtext)%8),
'constant', constant_values=72)

# Convert from string to int so that we can use it as arguments for our
   functions
encodedtext_pad = [int(i) for i in encodedtext_pad]
```

The second portion of the code block above performs three important tasks that will then allow us to focus on the linear algebra operations of our cryptosystem. Since we are required to have square matrices of size at least 7×7, we instead aim to use square matrices of one dimension higher, that is 8×8. For this requirement, we need to make sure that the length of our text is divisible by 8. We use `numpy.pad` to check this requirement is met, and if not we append 1's on the end of text until `len(textfromfile)`$\equiv 0 \ mod \ 8$. We chose 72's to be spaces in our `encoder` dictionary such that it would not affect our final decrypted message. The last function loops over every entry in `encodedtext_pad` and converts from the datatype from string to integers.

Now that we have read in, encoded, and changed to the appropriate datatype, we can implement the mathematically significant portion of the system. The next block performs both the encryption and decryption process within a single `for` loop, but it should be noted that implementing this for a two party system would not require extensive additional work. We also note that this was implemented using matrix methods from the symbolic processing package `SymPy`, such that we do not have numerical evaluations at each step in the process as we would have with the numerical package `NumPy`. The rationale behind this will be more fully explored in Section 3.

```
j = 1
# Empty array
plain=[]
for i in range(0, len(encodedtext_pad), 8): # Step sizes of 8
    # Constructing diagonal matrices with the COSH function on the main
    # diagonal
  a = diag(.5*(sym.exp(encodedtext_pad[i])+sym.exp(-encodedtext_pad[i])),
        .5*(sym.exp(encodedtext_pad[i+1])+sym.exp(-encodedtext_pad[i+1])),
        .5*(sym.exp(encodedtext_pad[i+2])+sym.exp(-encodedtext_pad[i+2])),
        .5*(sym.exp(encodedtext_pad[i+3])+sym.exp(-encodedtext_pad[i+3])),
        .5*(sym.exp(encodedtext_pad[i+4])+sym.exp(-encodedtext_pad[i+4])),
        .5*(sym.exp(encodedtext_pad[i+5])+sym.exp(-encodedtext_pad[i+5])),
        .5*(sym.exp(encodedtext_pad[i+6])+sym.exp(-encodedtext_pad[i+6])),
        .5*(sym.exp(encodedtext_pad[i+7])+sym.exp(-encodedtext_pad[i+7])))

    # Create a vector with numerical values that correspond to our
    # ciphertext
  vect = Matrix([[encodedtext_pad[i]],
              [encodedtext_pad[i+1]],
              [encodedtext_pad[i+2]],
              [encodedtext_pad[i+3]],
              [encodedtext_pad[i+4]],
              [encodedtext_pad[i+5]],
              [encodedtext_pad[i+6]],
              [encodedtext_pad[i+7]]])

    # To create our encrypted vector, use the orthogonal matrix x that
    # we generated
  encrypt= (x.T**j) @ a @ (x**j) @ vect

    # Decrypt the encrypted vector
  decrypt = a @ (x**j) @ vect

    # Pulls out the power of the exponential in each decrypted vector
  for i in range(0, len(decrypt)):
      plain += [decrypt[i].atoms(exp)]

    # Run over all j until all of our ciphertext is operated on
  j = j + 1

    # Remove set brackets from plain array and just store absolute
    # value of entries and use step sizes of 2 to isolate only one value
    # for each entry
merged = list(itertools.chain.from_iterable(plain))
d_text = []
for i in range(0, len(merged),2):
    d_text.append(abs(ln(merged[i])))
# Convert back from int to string
dec_text = [str(i) for i in d_text]
```

Though we have commented the code, let's walk through line by line so that we can be explicit about what is happening within this block. We need to iterate through each 'block' of 8 integers in our cipher-text and so we set an iterator variable `j=1` so that we can raise our matrix `x` to the `j`th power for each successive block. We also initialize an empty list `plain=[]` to store our integers from the exponentials that we will then use to decipher at a later step. We setup a `for` loop that iterates from 0 to the length of `encodedtext_pad` and works in step sizes of 8. At every step in this loop, we construct a diagonal matrix named `a` that has `.5*(sym.exp(encodedtext_pad[i])+sym.exp(-encodedtext_pad[i])` for entry $a_{1,1}$ and `.5*(sym.exp(encodedtext_pad[i+k])+sym.exp(-encodedtext_pad[i+k])` for entries $a_{1+k,1+k}$ respectively.

Our next step is construct a vector of integers corresponding to each entry in `encoded_text` and appropriately named `vect`. The encryption process takes the matrix `x`, raises it to the power `j`, multiplies (on the left) using the `@` operator on our diagonal matrix `a`, then multiplies by the transpose `x.T` (which is equal to x^{-1} by Property 4 in Section 1) raised to the `j`th power and a final left multiplication on `vect`. This will yield the `encrypt` vector:

$$\texttt{encrypt=}\begin{pmatrix} -7.5e^{3} - 7.5e^{-3} + 10.5e^{-18} + 10.5e^{18} \\ 10.5e^{-18} + 7.5e^{-3} + 7.5e^{3} + 10.5e^{18} \\ 4.5e^{-25} + 20.5e^{-16} + 20.5e^{16} + 4.5e^{25} \\ -4.5e^{25} - 4.5e^{-25} + 20.5e^{-16} + 20.5e^{16} \\ 2.5e^{-20} + 17.5e^{-15} + 17.5e^{15} + 2.5e^{20} \\ -2.5e^{20} - 2.5e^{-20} + 17.5e^{-15} + 17.5e^{15} \\ -5.5e^{7} - 5.5e^{-7} + 12.5e^{-18} + 12.5e^{18} \\ 12.5e^{18} + 5.5e^{-7} + 5.5e^{7} + 12.5e^{18} \end{pmatrix}$$

To create the decrypted vector, we simply multiply the vector `encrypt` on the left by `x` raised to the power `j`:

$$\texttt{decrypt=}\begin{pmatrix} -15e^{3} - 15e^{-3} \\ 21e^{-18} + 21e^{18} \\ 9e^{-25} + 9e^{25} \\ 41e^{-16} + 41e^{16} \\ 5e^{-20} + 5e^{20} \\ 35e^{-15} + 35e^{15} \\ -11e^{7} - 11e^{-7} \\ 25e^{-18} + 25e^{18} \end{pmatrix}$$

This is exactly the desired output, since every entry in the `decrypt` vector is:

$$\texttt{decrypt=}\begin{pmatrix} a_1 \cdot cosh(3) \\ a_2 \cdot cosh(18) \\ a_3 \cdot cosh(25) \\ a_4 \cdot cosh(16) \\ a_5 \cdot cosh(20) \\ a_6 \cdot cosh(15) \\ a_7 \cdot cosh(7) \\ a_8 \cdot cosh(18) \end{pmatrix}$$

for $a_i \in \mathbb{R}$. From here we need to read the powers off of the exponential for each entry in the `decrypt` vector. We can use a neat Python trick that examines each entry in `decrypt` and reads the argument being passed to the exponential, storing that argument into our `plain` list. We increment `j=j+1` and return to the beginning of our `for` loop, performing each encryption and decryption and storing the arguments of the exponentials along the way. Finally when we have reached the stopping conditions of the `for` loop we are left with `plain=[{exp(-3), exp(3)}, {exp(18), exp(-18)}, {exp(25), exp(-25)}, {exp(16), exp(-16)},...]`. This is obviously not our desired form and so we need to remove everything except the integers and convert to a string type, eventually to use the inverse map `decoder` defined above. We set up an intermediate variable named `merged` that calls upon a python toolbox called `itertools` and uses the function `.chain(plain)` to remove the set braces. Setting up another intermediate list `d_text` we utilize another `for` loop to take the absolute value of the natural logarithm ln of each entry, thus leaving us with a list of the integers corresponding to our cipher-text. However, notice that since there are 2 exponential terms of the same power for each entry in `decrypt`, and so this last `for` loop operates in step sizes of 2. Finally we set up yet another list called `dec_text` that converts every integer in `d_text` to a string. This is what we will pass to the inverse dictionary `decoder` to convert our cipher-text back to the original plain-text that was read in. We can use essentially the same `for` loop used for the encoding process but merge all characters at the end:

```
plaintext = []
plainmerged = []
for i in range(0, len(dec_text)):
    plaintext.append(decoder[dec_text[i]])
plainmerged[0:len(plaintext)] = [''.join(plaintext[0:len(plaintext)])]
```

Which returns exactly the text that was read from the file. The exact sample file, cipher-text, and decrypted text can be found in the Appendix.

3. Analysis & Security

Arguably the most glaring security flaw here is the use of the symbolic evaluation library `SymPy`. If we were to instead numerically evaluate at every step of the encryption and decryption process, then we would simply have real numbers as entries for both vectors `encrypt` and `decrypt`. While one may think that it would undoubtedly make the system more secure, in order to find the power of the exponential for each decrypted entry we would have to take the natural logarithm of every single character in our encoder dictionary for each entry (70 times for our dictionary). Even if this were not computationally infeasible, we would also have to know the scalar a_i and its natural logarithm $ln(a_i)$ multiplying that exponential such that we would have $ln(a_i) \times m$ for $m \in \mathbb{Z}$ in order to identify m.

Lastly, the performance of our crypto-system was severely lacking. To encode and decode our sample text found in the Appendix, run-time was measured at 6 minutes and 57 seconds which is $\sim .84$ seconds per word. Further removing all `print` statements reduced the run-time to 5 minutes and 50 seconds. It's generally well known that matrix operations can be carried out extremely fast on computers due to the large amount of work done to optimize such routines (such as `BLAS` [1]). We can speculate and claim this poor performance to be an artifact of the symbolic computations, in which extra overhead to compute quantities like the coefficients of each exponential may be unnecessarily long [3]. Indeed, implementing all matrix methods from `Numpy` resulted in run times of less than 1 second, fully indicating that `SymPy` was indeed the culprit behind poor performance. After security, the performance of a cryptosystem is arguably the most important factor. We want our communications to be secure, but we also want a system in which there is no significant performance degradation, this system fails in both regards when using `SymPy`.

4. Future Direction

To continue working on improving both the performance and security of the system, there are a number of methods we wanted to implement but couldn't due to a lack of time. The paper does employ a permutation matrix to permute the entries of the encrypted vector in hopes of increasing security. A permutation matrix P is comprised of the basis vectors $e_1, e_2, \ldots, e_n$ as columns but in differing order than that of the identity matrix $\mathbb{I}_n$ (where the orderings are $e_1, e_2, \ldots e_n$ for $\mathbb{I}_n \in \mathbb{R}^n$) [2]. Since the permutation matrices P are also orthogonal matrices, then we enjoy all the same properties described in Section 1.

In order to further improve security but still employ numerical evaluations to obscure the entries in `encrypt`, it may be possible to ask `SymPy` to somehow convert $t \in \mathbb{R}$ to a scalar r times some exponential e^k for $k \in \mathbb{Z}$ such that $t = r \cdot e^k$, but we were unable to find any built in function to accomplish this. Other programming languages such as `Mathematica` or `Matlab` may have more accessible ways to force such an expression.

However, due to both limited time and the potential exploitation of the symbolic package `SymPy`, the permutation matrices P were not implemented.

5. Conclusion

While Santana does present a novel system based on orthogonal matrices, the text lacks a cohesive explanation for many of the processes that could potentially satisfy the stringent security required for a working crypto-system. Additionally, symbolic computations and performance issues hinder the system to be robust in almost all aspects. It may be that certain key aspects of the paper were overlooked, but the poor grammar and lack of fluency in the English language magnified the technical deficiencies that ultimately hindered us in implementing Santana's desired result.

Appendix

The text from a file was a simple 500 word essay that was found on Google about not wasting time. The text was read-in as:

```
500 Words Essay on Time Essay On Time: Time is very precious and we should
   not waste it in any way. Likewise, we can earn the money we spent but we
    cannot get back the time we have lost. So, this makes the time more
   valuable than money. Hence, we should utilize the time in the most
   possible way. Importance of Time This the most valuable and precious
   thing in the world. Also, we should use it for our good as well as for
   the good of others around us. This will help us and the society to
   progress towards a better tomorrow. Moreover, we should teach our
   children the importance and value of time. Also, wasting time will only
   lead you to cause an issue to you and the people around you.Effective
   Utilization of Time For effectively utilizing the time we must consider
   some points which will help us in our whole life. This utilization
   includes setting goals, prepare work lists, prioritize task, and take
   adequate sleep and various others.For effectively utilizing time set
   long and short term goals these goals will help you in remaining
   productive. Moreover, they will prove as a driving force that will keep
   you motivated. Also, this will give the willingness to achieve something
    in life. In the beginning, it will feel like a boring task but when you
    do it regularly then you will realize that that it only helps you to
   increase your productivity. Ultimately, this will force you to achieve
   more in life.Prioritizing task is a very effective way of managing time.
    Also, because of it, you will know the importance of various task and
   jobs. Apart from that, if your club and perform a similar activity in a
   go then it also increases your productivity. Hence, it will help you to
   achieve more in life. Being productive does not mean that you engage
   yourself in different tasks every time. Taking proper sleep and
   exercising is also part of being productive. Besides, proper exercise
   and sleep maintain a balance between body and mind which is very
   important for being productive and efficient. Value of Time Although
   most people do not understand how valuable time is until they lost it.
   Besides, there are people in the world who prioritize money over time
   because according to them, time is nothing. But, they do not realize the
    fact that it is time that has given them the'
```

The text was converted to:

```
['59', '68', '68', '72', '23', '41', '44', '30', '45', '72', '5', '45',
 '45', '27', '51', '72', '41', '40', '72', '20', '35', '39', '31', '72',
  '5', '45', '45', '27', '51', '72', '15', '40', '72', '20', '35', '39',
 '31', '64', '72', '20', '35', '39', '31', '72', '35', '45', '72', '48',
 '31', '44', '51', '72', '42', '44', '31', '29', '35', '41', '47', '45',
 '72', '27', '40', '30', '72', '49', '31', '72', '45', '34', '41', '47',
 '38', '30', '72', '40', '41', '46', '72', '49', '27', '45', '46', '31',
 '72', '35', '46', '72', '35', '40', '72', '27', '40', '51', '72', '49',
 '27', '51', '62', '72', '12', '35', '37', '31', '49', '35', '45', '31',
 '70', '72', '49', '31', '72', '29', '27', '40', '72', '31', '27', '44',
 '40', '72', '46', '34', '31', '72', '39', '41', '40', '31', '51', '72',
 '49', '31', '72', '45', '42', '31', '40', '46', '72', '28', '47', '46',
 '72', '49', '31', '72', '29', '27', '40', '41', '46', '72', '33', '31',
 '46', '72', '28', '27', '29', '37', '72', '46', '34', '31', '72', '46',
 '35', '39', '31', '72', '49', '31', '72', '34', '27', '48', '31', '72',
 '38', '41', '45', '46', '62', '72', '19', '41', '70', '72', '46', '34',
 '35', '45', '72', '39', '27', '37', '31', '45', '72', '46', '34', '31',
 '72', '46', '35', '39', '31', '72', '39', '41', '44', '31', '72', '48',
 '27', '38', '47', '27', '28', '38', '31', '72', '46', '34', '27', '40',
 '72', '39', '41', '40', '31', '51', '62', '72', '8', '31', '40', '29',
 '31', '70', '72', '49', '31', '72', '45', '34', '41', '47', '38', '30',
 '72', '47', '46', '35', '38', '35', '52', '31', '72', '46', '34', '31',
 '72', '46', '35', '39', '31', '72', '35', '40', '72', '46', '34', '31',
 '72', '39', '41', '45', '46', '72', '42', '41', '45', '45', '35', '28',
 '38', '31', '72', '49', '27', '51', '62', '72', '9', '39', '42', '41',
 '44', '46', '27', '40', '29', '31', '72', '41', '32', '72', '20', '35',
 '39', '31', '72', '20', '34', '35', '45', '72', '46', '34', '31', '72',
 '39', '41', '45', '46', '72', '48', '27', '38', '47', '27', '28', '38',
 '31', '72', '27', '40', '30', '72', '42', '44', '31', '29', '35', '41',
 '47', '45', '72', '46', '34', '35', '40', '33', '72', '35', '40', '72',
 '46', '34', '31', '72', '49', '41', '44', '38', '30', '62', '72', '60',
 '38', '45', '41', '70', '72', '49', '31', '72', '45', '34', '41', '47',
 '38', '30', '72', '47', '45', '31', '72', '35', '46', '72', '32', '41',
 '44', '72', '41', '47', '44', '72', '33', '41', '30', '72', '27', '45',
 '72', '49', '31', '38', '38', '72', '27', '45', '72', '32', '41', '44',
 '72', '46', '34', '31', '72', '33', '41', '41', '30', '72', '41', '32',
 '72', '41', '46', '34', '31', '44', '45', '72', '27', '44', '41', '47',
 '40', '30', '72', '47', '45', '62', '72', '20', '34', '35', '45', '72',
 '49', '35', '38', '38', '72', '34', '31', '38', '42', '72', '47', '45',
 '72', '27', '40', '30', '72', '46', '34', '31', '72', '45', '41', '29',
 '35', '31', '46', '51', '72', '46', '41', '72', '42', '44', '41', '33',
 '44', '31', '45', '72', '46', '41', '49', '27', '44', '30', '45', '72',
 '27', '72', '28', '31', '46', '46', '31', '44', '72', '46', '41', '39',
 '41', '44', '44', '41', '49', '62', '72', '13', '41', '44', '31', '41',
 '48', '31', '44', '70', '72', '49', '31', '72', '45', '34', '41', '47',
 ’38’, ’30’, ’72’, ’46’, ’31’, ’27’, ’29’, ’34’, ’72’, ’41’, ’47’, ’44’,
 ’72’, ’29’, ’34’, ’35’, ’38’, ’30’, ’44’, ’31’, ’40’, ’72’, ’46’, ’34’,
 ’31’, ’72’, ’35’, ’39’, ’42’, ’41’, ’44’, ’46’, ’27’, ’40’, ’29’, ’31’,
 ’72’, ’27’, ’40’, ’30’, ’72’, ’48’, ’27’, ’38’, ’47’, ’31’, ’72’, ’41’,
 ’32’, ’72’, ’46’, ’35’, ’39’, ’31’, ’62’, ’72’, ’60’, ’38’, ’45’, ’41’,
 ’70’, ’72’, ’49’, ’27’, ’45’, ’46’, ’35’, ’40’, ’33’, ’72’, ’46’, ’35’,
 ’39’, ’31’, ’72’, ’49’, ’35’, ’38’, ’38’, ’72’, ’41’, ’40’, ’38’, ’51’,
 ’72’, ’38’, ’31’, ’27’, ’30’, ’72’, ’51’, ’41’, ’47’, ’72’, ’46’, ’41’,
 ’72’, ’29’, ’27’, ’47’, ’45’, ’31’, ’72’, ’27’, ’40’, ’72’, ’35’, ’45’,
 ’45’, ’47’, ’31’, ’72’, ’46’, ’41’, ’72’, ’51’, ’41’, ’47’, ’72’, ’27’,
 ’40’, ’30’, ’72’, ’46’, ’34’, ’31’, ’72’, ’42’, ’31’, ’41’, ’42’, ’38’,
 ’31’, ’72’, ’27’, ’44’, ’41’, ’47’, ’40’, ’30’, ’72’, ’51’, ’41’, ’47’,
 ’62’, ’␣5’, ’32’, ’32’, ’31’, ’29’, ’46’, ’35’, ’48’, ’31’, ’72’, ’21’,
 ’46’, ’35’, ’38’, ’35’, ’52’, ’27’, ’46’, ’35’, ’41’, ’40’, ’72’, ’41’,
 ’32’, ’72’, ’20’, ’35’, ’39’, ’31’, ’72’, ’␣6’, ’41’, ’44’, ’72’, ’31’,
 ’32’, ’32’, ’31’, ’29’, ’46’, ’35’, ’48’, ’31’, ’38’, ’51’, ’72’, ’47’,
 ’46’, ’35’, ’38’, ’35’, ’52’, ’35’, ’40’, ’33’, ’72’, ’46’, ’34’, ’31’,
 ’72’, ’46’, ’35’, ’39’, ’31’, ’72’, ’49’, ’31’, ’72’, ’39’, ’47’, ’45’,
 ’46’, ’72’, ’29’, ’41’, ’40’, ’45’, ’35’, ’30’, ’31’, ’44’, ’72’, ’45’,
 ’41’, ’39’, ’31’, ’72’, ’42’, ’41’, ’35’, ’40’, ’46’, ’45’, ’72’, ’49’,
 ’34’, ’35’, ’29’, ’34’, ’72’, ’49’, ’35’, ’38’, ’72’, ’34’, ’31’, ’38’,
 ’42’, ’72’, ’47’, ’45’, ’72’, ’35’, ’40’, ’72’, ’41’, ’47’, ’44’, ’72’,
 ’49’, ’34’, ’41’, ’38’, ’31’, ’72’, ’38’, ’35’, ’32’, ’31’, ’62’, ’72’,
 ’20’, ’34’, ’35’, ’45’, ’72’, ’47’, ’46’, ’35’, ’38’, ’35’, ’52’, ’27’,
 ’46’, ’35’, ’41’, ’40’, ’72’, ’35’, ’40’, ’29’, ’38’, ’47’, ’30’, ’31’,
 ’45’, ’72’, ’45’, ’31’, ’46’, ’35’, ’40’, ’33’, ’72’, ’33’, ’41’, ’27’,
 ’38’, ’45’, ’70’, ’72’, ’42’, ’44’, ’31’, ’42’, ’27’, ’44’, ’31’, ’72’,
 ’49’, ’41’, ’44’, ’37’, ’72’, ’38’, ’35’, ’45’, ’46’, ’45’, ’70’, ’72’,
 ’42’, ’44’, ’35’, ’41’, ’44’, ’35’, ’46’, ’35’, ’52’, ’31’, ’72’, ’46’,
 ’27’, ’45’, ’37’, ’70’, ’72’, ’27’, ’40’, ’30’, ’72’, ’46’, ’27’, ’37’,
 ’31’, ’72’, ’27’, ’30’, ’31’, ’43’, ’47’, ’27’, ’46’, ’31’, ’72’, ’45’,
 ’38’, ’31’, ’42’, ’72’, ’27’, ’40’, ’30’, ’72’, ’48’, ’27’, ’44’, ’35’,
 ’41’, ’47’, ’45’, ’72’, ’41’, ’46’, ’34’, ’31’, ’44’, ’45’, ’62’, ’␣6’,
 ’41’, ’44’, ’72’, ’31’, ’32’, ’31’, ’29’, ’46’, ’35’, ’48’, ’31’, ’38’,
 ’51’, ’72’, ’47’, ’46’, ’35’, ’38’, ’35’, ’52’, ’35’, ’40’, ’33’, ’72’,
 ’46’, ’35’, ’39’, ’31’, ’72’, ’45’, ’31’, ’46’, ’72’, ’38’, ’41’, ’40’,
 ’33’, ’72’, ’27’, ’40’, ’30’, ’72’, ’45’, ’34’, ’41’, ’44’, ’46’, ’72’,
 ’46’, ’31’, ’44’, ’39’, ’72’, ’33’, ’41’, ’27’, ’38’, ’45’, ’72’, ’46’,
 ’34’, ’31’, ’45’, ’31’, ...
```

Notice that Python truncated the output and this corresponds to the ... found at the end of the list.

A single sample of the encrypted vectors:

```
encrypt=
Matrix([[-2.57777311255397*exp(41) - 2.5614861272219*exp(23) - 2.5614861272219*exp(-23) - 2.57777311255397*exp
    (-41) + 9.97018461778186*exp(-72) + 19.3088080031526*exp(-68) + 0.0493527166817119*exp(-59) +
    0.09601852020538*exp(-44) + 5.21489538195426*exp(-30) + 5.21489538195426*exp(30) + 0.09601852020538*exp
    (44) + 0.0493527166817119*exp(59) + 19.3088080031526*exp(68) + 9.97018461778186*exp(72)],
    [-8.18884714735408*exp(41) - 0.0539941324447804*exp(44) - 8.18884714735408*exp(-41) - 0.0539941324447804*
    exp(-44) + 29.5785480217721*exp(-72) + 2.30728069272846*exp(-68) + 1.98173514629551*exp(-59) +
    6.21327156252196*exp(-30) + 2.16200585648088*exp(-23) + 2.16200585648088*exp(23) + 6.21327156252196*exp
    (30) + 1.98173514629551*exp(59) + 2.30728069272846*exp(68) + 29.5785480217721*exp(72)],
    [-5.28634866187704*exp(68) - 0.0196698894965541*exp(44) - 3.24331950872679*exp(23) - 3.24331950872679*exp
    (-23) - 0.0196698894965541*exp(-44) - 5.28634866187704*exp(-68) + 18.8157808450884*exp(-72) +
    0.524149830277781*exp(-59) + 17.5682634026753*exp(-41) + 5.64114398205882*exp(-30) + 5.64114398205882*exp
    (30) + 17.5682634026753*exp(41) + 0.524149830277781*exp(59) + 18.8157808450884*exp(72)],
    [-0.442774714968151*exp(59) - 0.0220315867228809*exp(44) - 11.2402199241631*exp(30) - 11.2402199241631*
    exp(-30) - 0.0220315867228809*exp(-44) - 0.442774714968151*exp(-59) + 17.3157620458047*exp(-72) +
    21.7876654139539*exp(-68) + 7.60154235079304*exp(-41) + 1.00005641530246*exp(-23) + 1.00005641530246*exp
    (23) + 7.60154235079304*exp(41) + 21.7876654139539*exp(68) + 17.3157620458047*exp(72)],
    [-17.0669724982491*exp(68) - 2.90441514031984*exp(59) - 2.90441514031984*exp(-59) - 17.0669724982491*exp
    (-68) + 27.6627879839732*exp(-72) + 0.0407460818573004*exp(-44) + 0.267485943147551*exp(-41) +
    0.443925897757188*exp(-30) + 3.05644173183367*exp(-23) + 3.05644173183367*exp(23) + 0.443925897757188*exp
    (30) + 0.267485943147551*exp(41) + 0.0407460818573004*exp(44) + 27.6627879839732*exp(72)],
    [-4.82972342666242*exp(72) - 2.84317709328129*exp(59) - 1.9634685173297*exp(41) - 0.0531902822390813*exp
    (44) - 1.9634685173297*exp(-41) - 0.0531902822390813*exp(-44) - 2.84317709328129*exp(-59) -
    4.82972342666242*exp(-72) + 23.1496914513541*exp(-68) + 7.00191610828398*exp(-30) + 0.0379517598744478*
    exp(-23) + 0.0379517598744478*exp(23) + 7.00191610828398*exp(30) + 23.1496914513541*exp(68)],
    [-3.26139918665466*exp(41) - 3.12044487470863*exp(30) - 3.12044487470863*exp(-30) - 3.26139918665466*exp
    (-41) + 14.7837875186197*exp(-72) + 11.3117457894056*exp(-68) + 1.37248899222393*exp(-59) +
    0.0306510656074057*exp(-44) + 0.883170695506642*exp(-23) + 0.883170695506642*exp(23) +
    0.0306510656074057*exp(44) + 1.37248899222393*exp(59) + 11.3117457894056*exp(68) + 14.7837875186197*exp
    (72)], [-15.4830452525177*exp(72) - 15.4830452525177*exp(-72) + 7.87089623630474*exp(-68) +
    0.967660739721617*exp(-59) + 0.0287953500262812*exp(-44) + 10.523678753011*exp(-41) + 5.14197890520912*
    exp(-30) + 5.95003526824497*exp(-23) + 5.95003526824497*exp(23) + 5.14197890520912*exp(30) +
    0.0287953500262812*exp(44) + 10.523678753011*exp(41) + 0.967660739721617*exp(59) + 7.87089623630474*exp
    (68)]])
```

The decrypted text:

```
['500␣Words␣Essay␣on␣Time␣Essay␣On␣Time:␣Time␣is␣very␣precious␣and␣we␣should
   ␣not␣waste␣it␣in␣any␣way.␣Likewise,␣we␣can␣earn␣the␣money␣we␣spent␣but␣
   we␣cannot␣get␣back␣the␣time␣we␣have␣lost.␣So,␣this␣makes␣the␣time␣more␣
   valuable␣than␣money.␣Hence,␣we␣should␣utilize␣the␣time␣in␣the␣most␣
   possible␣way.␣Importance␣of␣Time␣This␣the␣most␣valuable␣and␣precious␣
   thing␣in␣the␣world.␣Also,␣we␣should␣use␣it␣for␣our␣god␣as␣well␣as␣for␣
   the␣good␣of␣others␣around␣us.␣This␣will␣help␣us␣and␣the␣society␣to␣
   progress␣towards␣a␣better␣tomorrow.␣Moreover,␣we␣should␣teach␣our␣
   children␣the␣importance␣and␣value␣of␣time.␣Also,␣wasting␣time␣will␣only␣
   lead␣you␣to␣cause␣an␣issue␣to␣you␣and␣the␣people␣around␣you.Effective␣
   Utilization␣of␣Time␣For␣effectively␣utilizing␣the␣time␣we␣must␣consider␣
   some␣points␣which␣will␣help␣us␣in␣our␣whole␣life.␣This␣utilization␣
   includes␣setting␣goals,␣prepare␣work␣lists,␣prioritize␣task,␣and␣take␣
   adequate␣sleep␣and␣various␣others.For␣effectively␣utilizing␣time␣set␣
   long␣and␣short␣term␣goals␣these␣goals␣will␣help␣you␣in␣remaining␣
   productive.␣Moreover,␣they␣will␣prove␣as␣a␣driving␣force␣that␣will␣keep␣
   you␣motivated.␣Also,␣this␣will␣give␣the␣willingness␣to␣achieve␣something
   ␣in␣life.␣In␣the␣beginning,␣it␣will␣feel␣like␣a␣boring␣task␣but␣when␣you
   ␣do␣it␣regularly␣then␣you␣will␣realize␣that␣that␣it␣only␣helps␣you␣to␣
   increase␣your␣productivity.␣Ultimately,␣this␣will␣force␣you␣to␣achieve␣
   more␣in␣life.Prioritizing␣task␣is␣a␣very␣effective␣way␣of␣managing␣time.
   ␣Also,␣because␣of␣it,␣you␣will␣know␣the␣importance␣of␣various␣task␣and␣
   jobs.␣Apart␣from␣that,␣if␣your␣club␣and␣perform␣a␣similar␣activity␣in␣a␣
   go␣then␣it␣also␣increases␣your␣productivity.␣Hence,␣it␣will␣help␣you␣to␣
   achieve␣more␣in␣life.␣Being␣productive␣does␣not␣mean␣that␣you␣engage␣
   yourself␣in␣different␣tasks␣every␣time.␣Taking␣proper␣sleep␣and␣
   exercising␣is␣also␣part␣of␣being␣productive.␣Besides,␣proper␣exercise␣
   and␣sleep␣maintain␣a␣balance␣between␣body␣and␣mind␣which␣is␣very␣
   important␣for␣being␣productive␣and␣efficient.␣Value␣of␣Time␣Although␣
   most␣people␣do␣not␣understand␣how␣valuable␣time␣is␣until␣they␣lost␣it.␣
   Besides,␣there␣are␣people␣in␣the␣world␣who␣prioritize␣money␣over␣time␣
   because␣according␣to␣them,␣time␣is␣nothing.␣But,␣they␣do␣not␣realize␣the
   ␣fact␣that␣it␣is␣time␣that␣has␣given␣them␣the␣']
```

This is exactly our desired output.

References

[1] *BLAS Support*. URL: https://www.gnu.org/software/gsl/doc/html/blas.html.

[2] Ron Larson. *Elementary Linear Algebra*. 8th ed. Brooks Cole, 2016. ISBN: 1305658000-9781305658004.

[3] *Numeric Computation*. URL: https://docs.sympy.org/latest/modules/numeric-computation.html.

[4] Yeray Cachon Santana. *Orthogonal Matrix in Cryptography*. 2014. URL: https://arxiv.org/abs/1401.5787.

[5] Gilbert Strang. *Linear Algebra and Its Applications*. 4th. Brooks Cole, 2005. ISBN: 0030105676,9780030105678.