National College of Ireland

Configuration Manual

MSc Research Project
Masters In Cybersecurity

Priyanka Srirangam
Student ID: X21125708

School of Computing
National College of Ireland

Supervisor: Ross Spelman

National College of Ireland

MSc Project Submission Sheet

School of Computing

Student Name: Priyanka Srirangam

Student ID: X21125708

Programme: Masters In Cybersecurity **Year:** 2022

Module: Research Internship

Lecturer: Ross Spelman

Submission Due Date: 13/12/2023

Project Title: Application of Homomorphic Encryption In Open Banking

Word Count: 661 **Page Count:** 11

I hereby certify that the information contained in this (my submission) is information pertaining to research I conducted for this project. All information other than my own contribution will be fully referenced and listed in the relevant bibliography section at the rear of the project.

ALL internet material must be referenced in the bibliography section. Students are required to use the Referencing Standard specified in the report template. To use other author's written or electronic work is illegal (plagiarism) and may result in disciplinary action.

Signature: Priyanka Srirangam

Date: 13/12/2023

PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST

Attach a completed copy of this sheet to each project (including multiple copies)	□
Attach a Moodle submission receipt of the online project submission, to each project (including multiple copies).	□
You must ensure that you retain a HARD COPY of the project, both for your own reference and in case a project is lost or mislaid. It is not sufficient to keep a copy on computer.	□

Assignments that are submitted to the Programme Coordinator Office must be placed into the assignment box located outside the office.

Office Use Only	
Signature:	
Date:	
Penalty Applied (if applicable):	

Configuration Manual

Priyanka Srirangam
Student ID : X2115708

1 Introduction

This document captures the detailed information regarding necessary software and hardware components to build and execute python programs for homomorphic encryption and decryption of Sample open banking API payloads. It would help reader understand and set up similar environment as the researcher to successfully, in a practical manner.

Rest of the document has following key sections –

- Environmental setup
- Coding Implementation
- Sample Payload Encryption
- Sample Payload Decryption

2 Environmental Setup

Your second section. Change the header and label to something appropriate.

2.1 Hardware Requirements

- Available RAM – 32GB
- Minimum RAM – 8GB
- Disk space – 1 TB SSD

2.2 Software Requirements

- Windows 10
- Python 3.8.10
- C++17

2.3 Additional Tools

- Visual Studio Code

2.4 Libraries

The list of libraries used and required have been listed in the table below.

Table 1 : Libraries

Library	Description

Pyfhel[1]	It is an optimized Python API library for C++ based backend for homomorphic encryption schemes such as BGV, BFV and CKKS.
Numpy[2]	It is a Python library which supports large, multi-dimensional arrays and a number of mathematical operations on the arrays.
Base64[3]	This Python module offers functions for encoding binary data into printable ASCII based strings and for decoding ASCII character based strings into Binary data.
Json[4]	It is a library used to parse JSON from string or files on the file system.

3 Coding Implementation

3.1 Encryption Programs

In the following section, screenshots are provided for Pythong programs built to homomorphically encrypt transaction and party API payloads based on BFV scheme, using Pyfhel and other libraries mentioned above.

3.1.1 Encryption of Transaction API using BFV

```python
import json
import numpy as np
from Pyfhel import Pyfhel, PyPtxt, PyCtxt
import base64
from datetime import datetime


class my_dictionary(dict):
    # __init__ function
    def __init__(self):
        self = dict()
    # Function to add key:value
    def add(self, key, value):
        self[key] = value

def is_timestamp(attributeVal):
    try:
        datetime.strptime(attributeVal, '%Y-%m-%dT%H:%M:%S%z')
        return True
    except ValueError:
        return False

def base64ofitem(data):
    return base64.b64encode(data).decode('utf-8')

def encryptPiiIntValue(piiValArr, HE):
    cipherPiiVal = HE.encryptInt(piiValArr)
    decryptedPiiVl = HE.decryptInt(cipherPiiVal)[0]
    print("all good ", decryptedPiiVl)
    return base64ofitem(cipherPiiVal.to_bytes())

```

[1] https://github.com/ibarrond/Pyfhel
[2] https://numpy.org/doc/stable/release/1.26.0-notes.html
[3] https://docs.python.org/3/library/base64.html
[4] https://docs.python.org/3/library/json.html

```
def encryptPiiIntArray(piiValArr, HE, len):
    cipherStringVal = HE.encryptInt(piiValArr)
    decryptedPiiVl = HE.decryptInt(cipherStringVal)[:len]
    print("all good ", decryptedPiiVl)
    return base64ofitem(cipherStringVal.to_bytes())

def encryptPiiStringValue(piiValueString, HE):

    char_array = np.fromstring(piiValueString, dtype='S1')
    print(list(char_array))
    myChrToIntlist =[]
    for character in char_array :
        chrInt = ord(character)
        print(chrInt)
        myChrToIntlist.append(chrInt)
    piiChrValArr = np.array(myChrToIntlist, dtype=np.int64)
    encryptedCharArray = encryptPiiIntArray(piiChrValArr, HE, len(myChrToIntlist))
    return encryptedCharArray

def setHeContextAndSave(data):
    HE = Pyfhel()
    HE.contextGen(scheme='bfv', n=1024, t = 65537)
    HE.keyGen()
    s_context = HE.to_bytes_context()
    data.update({"context": base64ofitem(s_context)})
    s_public_key = HE.to_bytes_public_key()
    data.update({"publicKey": base64ofitem(s_public_key)})
    s_secret_key = HE.to_bytes_secret_key()
    data.update({"privateKey": base64ofitem(s_secret_key)})
    return HE
```

```
def encryptionRoutine(item, value):
    if item in piiAttributes :
            print("PII attribute")
            mylist =[]
            mylist.append(value)
            try:
                piiValArr = np.array(mylist, dtype=np.int64)
                return encryptPiiIntValue(piiValArr, HE)
            except (ValueError):
                if is_timestamp(value):
                    ("Skipping timestamp encryption")
                else:
                    return encryptPiiStringValue(value, HE)
    else:
        return value

def recursiveEncryption(item, value):
    if isinstance(value, dict):
        for nestItem in value:
                print(nestItem)
                nestItemValue = value[nestItem]
                recursiveEncryption(nestItem, nestItemValue)
    else:
        encryptedVal = encryptionRoutine(item, value)
        for idx, itemToUpdate in enumerate(transactionData):
            if item in itemToUpdate:
                transactionItem = transactionData[idx]
                transactionItem.update({item: encryptedVal})
                break
```

```
93
94  f = open('transaction.json')
95  data = json.load(f)
96  transactionData = data['Transaction']
97  piiAttributes = {'TransactionReference','TransactionInformation','Balance'}
98  HE = setHeContextAndSave(data)
99  for keyValue in transactionData:
100     #print(keyValue)
101     for item in keyValue:
102         value = (keyValue[item])
103         recursiveEncryption(item, value)
104
105     with open("TransactionOutput.json", "w") as outfile:
106         json.dump(data, outfile)
107 f.close()
```

3.1.2 Encryption program for Party API

```
1   import json
2   import numpy as np
3   from Pyfhel import Pyfhel, PyPtxt, PyCtxt
4   import base64
5   from datetime import datetime
6
7
8   class my_dictionary(dict):
9       # __init__ function
10      def __init__(self):
11          self = dict()
12      # Function to add key:value
13      def add(self, key, value):
14          self[key] = value
15
16  def is_timestamp(attributeVal):
17      try:
18          # Attempt to parse the input string as a timestamp
19          datetime.strptime(attributeVal, '%Y-%m-%dT%H:%M:%S%z')
20          return True
21      except ValueError:
22          # If an exception is raised, it's not a valid timestamp
23          return False
24
25  def base64ofitem(data):
26      return base64.b64encode(data).decode('utf-8')
27
28  def encryptPiiIntValue(piiValArr, HE):
29      cipherPiiVal = HE.encryptInt(piiValArr)
30      decryptedPiiVl = HE.decryptInt(cipherPiiVal)[0]
31      #assert (decryptedPiiVl) == (piiVal), "Incorrect decryption - check"
32      print("all good ", decryptedPiiVl)
33      return base64ofitem(cipherPiiVal.to_bytes())
```

```
def encryptPiiIntArray(piiValArr, HE, len):
    cipherStringVal = HE.encryptInt(piiValArr)
    decryptedPiiVl = HE.decryptInt(cipherStringVal)[:len]
    #assert (decryptedPiiVl) == (piiVal), "Incorrect decryption - check"
    print("all good ", decryptedPiiVl)
    return base64ofitem(cipherStringVal.to_bytes())

def encryptPiiStringValue(piiValueString, HE):
    # Break down the string into character array
    char_array = np.fromstring(piiValueString, dtype='S1')
    print(list(char_array))
    myChrToIntlist =[]
    for character in char_array :
        # convert individual charatcer to corresponding ASCII integer
        chrInt = ord(character)
        print(chrInt)
        # Encrypt the resulting Integer
        myChrToIntlist.append(chrInt)
    piiChrValArr = np.array(myChrToIntlist, dtype=np.int64)
    encryptedCharArray = encryptPiiIntArray(piiChrValArr, HE, len(myChrToIntlist))
    return encryptedCharArray
```

```
def setHeContextAndSave(data):
    HE = Pyfhel()
    HE.contextGen(scheme='bfv', n=1024, t = 65537)
    HE.keyGen()
    s_context = HE.to_bytes_context()
    data.update({"context": base64ofitem(s_context)})
    s_public_key = HE.to_bytes_public_key()
    data.update({"publicKey": base64ofitem(s_public_key)})
    s_secret_key = HE.to_bytes_secret_key()
    data.update({"privateKey": base64ofitem(s_secret_key)})
    return HE

def encryptionRoutine(item, value):
    if item in piiAttributes :
            print("PII attribute")
            mylist =[]
            mylist.append(value)
            try:
                piiValArr = np.array(mylist, dtype=np.int64)
                return encryptPiiIntValue(piiValArr, HE)
            except (ValueError):
                if is_timestamp(value):
                    # skip timestamp encryption
                    ("Skipping timestamp encryption")
                else:
                    return encryptPiiStringValue(value, HE)
    else:
        return value
```

```
def recursiveEncryption(item, value):
    if isinstance(value, dict):
        for nestItem in value:
                print(nestItem)
                nestItemValue = value[nestItem]
                recursiveEncryption(nestItem, nestItemValue)
    else:
        encryptedVal = encryptionRoutine(item, value)
        for idx, itemToUpdate in enumerate(transactionData):
            if item in itemToUpdate:
                transactionItem = transactionData[idx]
                transactionItem.update({item: encryptedVal})
                break

encryptedOutput = my_dictionary()
f = open('party.json')
data = json.load(f)
transactionData = data['Party']
piiAttributes = {'Name','FullLegalName','AccountRole'}
HE = setHeContextAndSave(data)
for keyValue in transactionData:
    for item in keyValue:
        value = (keyValue[item])
        recursiveEncryption(item, value)

    with open("PartyOutput.json", "w") as outfile:
        json.dump(data, outfile)
f.close()
```

3.2 Decryption Programs

In the following section, screenshots are provided for Pythong programs built to homomorphically decrypt transaction and party API payloads based on BFV scheme, using Pyfhel and other libraries mentioned above.

3.2.1 Decryption program for Party API payload

```
import json
import numpy as np
from Pyfhel import Pyfhel, PyPtxt, PyCtxt
import base64

class my_dictionary(dict):
    # __init__ function
    def __init__(self):
        self = dict()
    # Function to add key:value
    def add(self, key, value):
        self[key] = value

def setHEContext(HE, item, value):
    if (item == 'context') :
            contextBinary = base64.b64decode(value)
            HE.from_bytes_context(contextBinary)
    if (item == 'publicKey') :
        publicKeyBinary = base64.b64decode(value)
        HE.from_bytes_public_key(publicKeyBinary)
    if (item == 'privateKey') :
        privateKeyBinary = base64.b64decode(value)
        HE.from_bytes_secret_key(privateKeyBinary)

def decryptionRoutine(item, value):
    if item in piiAttributes :
        ciphertextBinary = base64.b64decode(value)
        ciphertextBinary = PyCtxt(pyfhel=HE, bytestring=ciphertextBinary)
        plaintext = HE.decryptInt(ciphertextBinary)
        print("plain text array", plaintext)
        plaintextArr = plaintext[plaintext != 0]
        if(len(plaintextArr) > 1) :
```

```
            actualtext = ''.join(chr(code) for code in plaintextArr)
            print("actual text", actualtext)
            return actualtext
        else :
            return plaintextArr[0]
    else:
        return value

def recursiveDecryption(item, value):
    if isinstance(value, dict):
        for nestItem in value:
            #print(nestItem)
            nestItemValue = value[nestItem]
            recursiveDecryption(nestItem, nestItemValue)
    else:
        decryptedVal = decryptionRoutine(item, value)
        for idx, itemToUpdate in enumerate(transactionData):
            if item in itemToUpdate:
                transactionItem = transactionData[idx]
                transactionItem.update({item: decryptedVal})
                break

f = open('PartyOutput.json')
data = json.load(f)
transactionData = data['Party']
piiAttributes = {'Name','FullLegalName','AccountRole'}
contextAttributes = {'context','privateKey','publicKey'}

# Setup Pyfhel conetxt
HE = Pyfhel()
for item in data:
    if (item == 'context') :
        contextBinary = base64.b64decode(data[item])
```

```
        HE.from_bytes_context(contextBinary)
    if (item == 'publicKey') :
        publicKeyBinary = base64.b64decode(data[item])
        HE.from_bytes_public_key(publicKeyBinary)
    if (item == 'privateKey') :
        privateKeyBinary = base64.b64decode(data[item])
        HE.from_bytes_secret_key(privateKeyBinary)
for key in contextAttributes :
        data.pop(key)

for keyValue in transactionData:
    for item in keyValue:
        value = (keyValue[item])
        recursiveDecryption(item, value)

with open("DecryptedTransaction.json", "w") as outfile:
        json.dump(data, outfile)
```

3.2.2 Decryption program for Transaction API

```
import json
import numpy as np
from Pyfhel import Pyfhel, PyPtxt, PyCtxt
import tempfile
import base64

tmp_dir = tempfile.TemporaryDirectory()
tmp_dir_name = tmp_dir.name


class my_dictionary(dict):
    # __init__ function
    def __init__(self):
        self = dict()
    # Function to add key:value
    def add(self, key, value):
        self[key] = value

def setHEContext(HE, item, value):
    if (item == 'context') :
            contextBinary = base64.b64decode(value)
            HE.from_bytes_context(contextBinary)
    if (item == 'publicKey') :
        publicKeyBinary = base64.b64decode(value)
        HE.from_bytes_public_key(publicKeyBinary)
    if (item == 'privateKey') :
        privateKeyBinary = base64.b64decode(value)
        HE.from_bytes_secret_key(privateKeyBinary)


def decryptionRoutine(item, value):
    if item in piiAttributes :
        ciphertextBinary = base64.b64decode(value)
```

```
        ciphertextBinary = PyCtxt(pyfhel=HE, bytestring=ciphertextBinary)
        plaintext = HE.decryptInt(ciphertextBinary)
        print("plain text array", plaintext)
        plaintextArr = plaintext[plaintext != 0]
        if(len(plaintextArr) > 1) :
            actualtext = ''.join(chr(code) for code in plaintextArr)
            print("actual text", actualtext)
            return actualtext
        else :
                return int(plaintextArr[0])
    else:
        return value

def recursiveDecryption(item, value):
    if isinstance(value, dict):
        for nestItem in value:
                #print(nestItem)
                nestItemValue = value[nestItem]
                recursiveDecryption(nestItem, nestItemValue)
    else:
        decryptedVal = decryptionRoutine(item, value)
        for idx, itemToUpdate in enumerate(transactionData):
            if item in itemToUpdate:
                transactionItem = transactionData[idx]
                transactionItem.update({item: decryptedVal})
                break

f = open('TransactionOutput.json')
data = json.load(f)
transactionData = data['Transaction']
piiAttributes = {'TransactionReference','TransactionInformation','Balance','AccountId'}
contextAttributes = {'context','privateKey','publicKey'}

```

```python
# Setup Pyfhel conetxt
HE = Pyfhel()
for item in data:
    if (item == 'context') :
        contextBinary = base64.b64decode(data[item])
        HE.from_bytes_context(contextBinary)
    if (item == 'publicKey') :
        publicKeyBinary = base64.b64decode(data[item])
        HE.from_bytes_public_key(publicKeyBinary)
    if (item == 'privateKey') :
        privateKeyBinary = base64.b64decode(data[item])
        HE.from_bytes_secret_key(privateKeyBinary)
for key in contextAttributes :
        data.pop(key)

for keyValue in transactionData:
    for item in keyValue:
        value = (keyValue[item])
        recursiveDecryption(item, value)

with open("DecryptedTransaction.json", "w") as outfile:
        json.dump(data, outfile)
```

4 Sample Payload Encryption and Decryption

4.1 Pary API Sample payloads (before encryption, after encryption, and after decryption)

4.1.1 Party API Payload Before Encryption


```json
{
    "Party": [
        {
            "PartyId": "PABC123",
            "PartyType": "Sole",
            "Name": "Semiotec",
            "FullLegalName": "Semiotec Limited",
            "LegalStructure": "UK.OBIE.PrivateLimitedCompany",
            "BeneficialOwnership": true,
            "AccountRole": "UK.OBIE.Principal",
            "EmailAddress": "contact@semiotec.co.jp",
            "Relationships": {
                "Account": {
                    "Id": "22289"
                }
            },
            "Address": [
                {
                    "AddressType": "Business",
                    "StreetName": "Street",
                    "BuildingNumber": "15",
                    "PostCode": "NW1 1AB",
                    "TownName": "London",
                    "Country": "GB"
                }
            ]
        }
    ]
}
```

4.1.2 Party API Payload After Encryption

```
{"Party": [{"PartyId": "PABC123", "PartyType": "Sole", "Name": "XqEQBAAAAABxQAAAAAAAAD8fuxtTFLirCSmrQWg
+OEOGIcU0FB1lm6Tfac1IEarlAAIAAAAAAAAAAAQAAAAAAAABAAAAAAAAAAAAAAAAAPA/
AQAAAAAAAABeoRAEAAAAABhAAAAAAAAAAAgAAAAAAAjCoUDAAAAAMPIegMAAAAAQim1AQAAAAD9WCECAAAAANuscAMAAAAA
+mqSBwAAAACcIScHAAAAAPXR8gIAAAAXQctBAAAAAAQpgQHAAAAAIDyiQIAAAAA8DidAQAAAADIedwDAAAAAGfsHQMAAAAAlFdSAQAAAAD+tn4DAAAAAEabxAEAAAAA+Hp
+BQAAAABFIssEAAAAAGc+rQIAAAAA+Xv5AAAAAACqs4QFAAAAAP3loAIAAAAAe6t1AgAAAAATweMEAAAAACvY+QQAAAAAPsANBQAAAAD/
zeIDAAAAALBZKAIAAAAAPFomBwAAAADzPtcDAAAAABJFnAYAAAAA3NBQBAAAAAAhjGAAAAAAAN7AIQEAAAAAYlPABwAAAACaSkkGAAAAAGGgwQAAAAAAFyMgAgAAAABQNAIBAAAAAGvMlg
UAAAAAyoySAgAAAABusbkFAAAAAL5rwgQAAAAAClBYAgAAAAC0ScUAAAAAACvS9gAAAAAA7zlgBwAAAADi4asCAAAAAD8l2wUAAAAArPUvAAAAAABnGG8BAAAAAMkUIQAAAAAA
+34TAQAAAADr400HAAAAALcfwQQAAAAAEi05BgAAABd9y8HAAAAAN6tEAEAAAAA9mNHAQAAAABACmAHAAAAAMVeFwEAAAAAbDV4AAAAAABRCb0DAAAAANnD4wMAAAAAPSubAwAAAAC7/
7YCAAAAAEXO5QUAAAAAGSABBAAAAACm9E8GAAAAAP3bAAIAAAAAkjnUAAAAACea
+YCAAAAAIoJagYAAAAAkrPRAQAAAAB6ItkAAAAAABvDeAEAAAAAI25HAgAAAADsQJ8HAAAAAIKwVwAAAAAAOWu
+AAAAAAAJIoICAAAAAFZgeQAAAAAAQMT6AQAAAAB2uh0GAAAAAHtDdQIAAAAA5TmdBgAAAADYLOQDAAAAAHHVHgIAAAAANhWJAAAAAADxwFkEAAAAABU7rgcAAAAA/
3rBBgAAAAAfrpACAAAAACVX1AIAAAAAljnNBAAAAAC73WIDAAAAAC8slQYAAAAAEsEKAQAAAACenb0BAAAAADEJcwUAAAAAJgViAwAAAAAx4tkBAAAAAHv0KgMAAAAAbfbqBQAAAAAjeM4
CAAAAALYm9gQAAAAArkGbBAAAAABBpxoGAAAAANlDDgIAAAAAmnoABgAAAADwcyIDAAAAAIImxAYAAAAAqBXWAgAAAAAFjEsFAAAAANOwUAUAAAAAonJlBQAAAADiuJoCAAAAAN7RIQUAA
AAA48vTBAAAAADdF4wCAAAAAOCm6AUAAAAA+YBNBQAAAAAuxQsFAAAAADXaHAEAAAAA0rqbBgAAAABwz88FAAAAADUMzgAAAAA5atvAAAAAADLlwoBAAAAABiyKgUAAAAAMYR/
BQAAAABPx+IAAAAAAJQovAAAAAAA4nMrAQAAAABiSiIHAAAAAJ2ZKQAAAAAAWnC8AAAAAAAYqbECAAAAAK
+xoQMAAAAAcMV9AQAAAACvN08CAAAAAPSCEgcAAAArrquBgAAAAD33KsGAAAAAA1gfQAAAAAAvuBgBQAAAADTLNgFAAAAAAkaNAEAAAAAxVjAwAAAAAVoqYHAAAAAA/
2YQIAAAAAgxBcBgAAAAAhzbMHAAAAABP2+wQAAAAAJOgbBgAAACsNZYFAAAAAK69hAcAAAAALxOkBAAAAAB+fckAAAAAAPwJTQAAAAAAGNriBgAAAADtfcMBAAAAAHfTdgYAAAAAXd/
NAQAAAACLShQAAAAAAN5TxgEAAAAA2IDNAAAAAABNRncHAAAAAIE4cAEAAAAA7oCSAwAAAADXpeoEAAAAALUfXgEAAAAAKhnHBwAAAABUlNUEAAAAAMfUUgEAAAAArz5eAgAAAADwakoBA
AAAL9LigQAAAAp316BgAAAABzRakFAAAAAOTbzgEAAAAAXMRqBgAAAADrx2kCAAAAC2frgcAAAAAmKZmAwAAAADtQrgCAAAAACd7QwQAAAAfl1pBQAAAADKNCcHAAAAAMh2OAEAAAA
Abjl7AwAAAACHoW8HAAAAAETTBQIAAAAAoqsBBgAAAAAjpKUDAAAAAB/1jgUAAAAAo
+IJBgAAAD1gEYFAAAAAINIzwQAAAAAkk7yBQAAAADAH6kFAAAAAMX7sAIAAAAAkWJWBgAAAABfKsEHAAAAAClVOAMAAAAASayQAwAAAACPY8ICAAAAABmONgcAAAAAbtb
+AQAAAAC2RGgGAAAAADUvugIAAAAAeHNtBgAAAAC3dF8FAAAAAAjZJQQAAAAAN/MWAAAAAAAgTL4BAAAAALjrSgcAAAAAjCXpAAAAAAA53c8FAAAAAG68nQAAAAAASl4
+BAAAAABrpWsAAAAAAI/WzQUAAAAApiuMBgAAAAAdBM0HAAAAAIeimgEAAAAAgFQUBwAAAAA3qqYFAAAAAO+V6AQAAAAAzrIWAwAAAADLG34BAAAAAC/
sRgcAAAAAoScbAQAAAABjGDUCAAAAAA4xjwUAAAAAJKXqAAAAAC3Dd4FAAAAABxL0QUAAAAAnW/4AwAAAAAn8L0AAAAAAGmRmQYAAAAAVpjbAAAAAAA9uhUGAAAAAD
+FBQUAAAAAZmcPBgAAAArKUsGAAAAAJQWRAcAAAAA9hL/AQAAAACYq/gGAAAAAOJ3ZwQAAAAOvhJBQAAAACcOlMEAAAAANpnygYAAAAADsfAAwAAAABt/
pYBAAAAALvlWwEAAAAJVBVBQAAAAAW/
lEDAAAAAHx4cgQAAAAcWWXAgAAAACiwTEGAAAAAEfdmwYAAAAAwDWyAgAAAABo1gAGAAAAADa7lAQAAAAAbYl4AQAAAABHoFwEAAAAABw87gAAAAAAavIEAQAAAADcUywAAAAAOzW4gA
AAAAAlHvSBgAAAABb0MkHAAAAAFHMiwAAAAAAJkB8BAAAAAA5Bp4CAAAAAEfHNAYAAAAAgefWAQAAAAArz1EFAAAAALuWswYAAAAAg5XuAQAAAAe4c8BAAAAAEHV6AQAAAAaQT8AwAAA
AA8RlIBAAAAAGmiKwMAAAAAkx1eBwAAAABBg/
4CAAAAAFkDxAAAAAAAdXVGAwAAAAAly68HAAAAAEzNdwYAAAAAeOqHBQAAAABtazICAAAAAEfjAAIAAAAAUkB0BgAAAAB3d2gEAAAAAKVgFAMAAAAAnfYFAAAAAABxv/
gAAAAAAEea1wCAAAAAjz9uAwAAAAD6
```


4.1.3 Party API Payload After Decryption

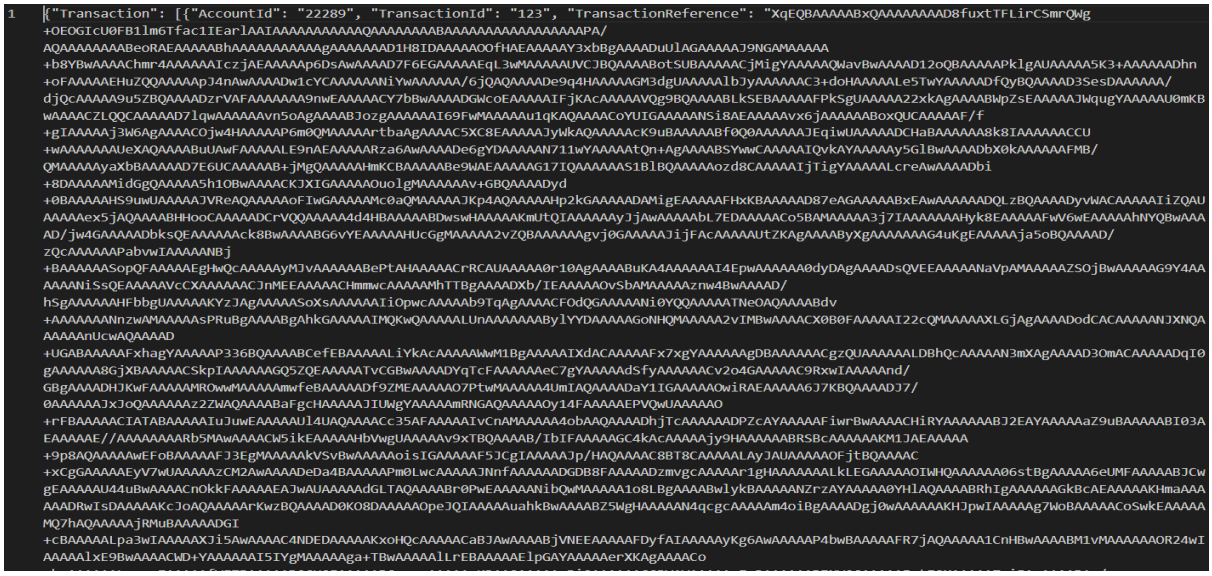
```
{"Party": [{"PartyId": "PABC123", "PartyType": "Sole", "Name": "Semiotec", "FullLegalName": "Semiotec Limited", "LegalStructure": "UK.OBIE.
PrivateLimitedCompany", "BeneficialOwnership": true, "AccountRole": "UK.OBIE.Principal", "EmailAddress": "contact@semiotec.co.jp",
"Relationships": {"Account": {"Id": "22289"}}, "Address": [{"AddressType": "Business", "StreetName": "Street", "BuildingNumber": "15",
"PostCode": "NW1 1AB", "TownName": "London", "Country": "GB"}]}]}
```


4.2 Transaction API Sample payloads (before encryption, after encryption, and after decryption)

4.2.1 Transaction API Payload Before Encryption

```
{} Transaction.json > [ ] Transaction > {} 0 > {} Amount
{
  "Transaction": [
    {
      "AccountId": "22289",
      "TransactionId": "123",
      "TransactionReference": "Ref 1",
      "Amount": {
        "Amount": "10.00",
        "Currency": "GBP"
      },
      "CreditDebitIndicator": "Credit",
      "Status": "Booked",
      "BookingDateTime": "2017-04-05T10:43:07+00:00",
      "ValueDateTime": "2017-04-05T10:45:22+00:00",
      "TransactionInformation": "Cash from Aubrey",
      "BankTransactionCode": {
        "Code": "ReceivedCreditTransfer",
        "SubCode": "DomesticCreditTransfer"
      },
      "ProprietaryBankTransactionCode": {
        "Code": "Transfer",
        "Issuer": "AlphaBank"
      },
      "Balance": {
        "Amount": {
          "Amount": "230.00",
          "Currency": "GBP"
        },
        "CreditDebitIndicator": "Credit",
        "Type": "InterimBooked"
      }
    }
  ]
}
```

4.2.2 Transaction API Payload After Encryption



4.2.3 Transaction API Payload After decryption