Coursework: Homomorphic Encryption

William Bowditch
wbowditch@secureworks.com
Edinburgh Napier University - Esecurity (CSN11117)

Abstract

This coursework is focused on homomorphic encryption, a system that allows one to perform arbitrary functions on a cipher-text without the need to decrypt the cipher-text in advance. The rapid development of homomorphic encryption is reviewed by investigating notable papers, mainly from the past decade. Using open source implementations of these papers, a homomorphic exchange is simulated in order to demonstrate the potential homomorphic cryptography has in privacy-preserving surveillance. Security parameters from this exchange are benchmarked in order to optimize noise budget and computational time.

Keywords – Cryptography, SEAL, Paillier, Machine Learning, Homomorphic Encryption

1 Introduction

User privacy has always been a second priority behind revenue for technology companies, and for valid reasons; the revenue of these tech giants rely on online marketing platforms fueled by user behaviour analytics. Companies like Facebook and Google digest raw data that needs to be normalized, sorted, and trained on in order to produce effective machine learning models. While this data can be protected with SSL tunnels on transit, it must be decrypted and stored in plain-text on corporate servers in order to provide any value. A parallel exists in civilian privacy and national security; government agencies like the NSA rely on Internet surveillance programs that search plain-text data in order to detect threats of national security. On the surface, free internet services and effective terrorism countermeasures seem like a reasonable trade in exchange for one's personal data. However, as is often the case in information security, humans are the weakest link in the chain.

Machine learning models (at least government sponsored models) do not use their plain-text access to stalk spouses and ex-lovers [1]. Furthermore, we trust the ride-sharing analytics of Uber to not abuse its data access by tracking the location of billionaires or querying for their phone numbers [2]. With these examples the predicament is clear; we wish to provide data for these models such that they continue to subsidize free internet services and protect homeland security, but we do not trust the human users that inevitably gain access to this data. Fortunately, the cryptographic community has been working on a solution for forty years but it was not until recently that implementations became practical.

Homomorphic encryption, the topic of this coursework, is a cryptographic scheme that allows one to perform arbitrary functions on a cipher-text without the need to decrypt the cipher-text in advance. Furthermore, the decrypted cipher-text is equivalent to the output of the same arbitrary functions performed on the plain-text. Figure 1 illustrates this relationship. For example, fully homomorphic encryption would allow users to send encrypted data to government agencies and technology companies such that models can train and act on this data without knowing or needing to store the plain-text itself. A cryptographic scheme is partially homomorphic if it allows unlimited operations to be performed but with one particular function, while a scheme is somewhat homomorphic if it allows limited operations of any arbitrary function.

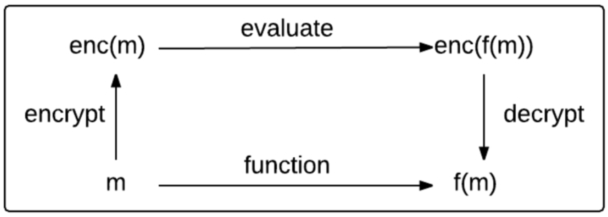

Figure 1: **Homomorphic Encryption** - Visualization

The literature review of this coursework will first explore the partially multiplicative homomorphic properties of the RSA algorithim, the first spark of development in the field of homomorphic encryption. We will then investigate the Paillier crypto system, a probabilistic asymmetric algorithm that allows addition between encrypted messages as well as multiplication with plain-text[3]. The most important literature we will review is Craig Gentry's 2009 seminal paper, which was the first paper to describe a credible fully homomorphic encryption scheme [4], followed by Brakerski and Vaikuntanathan's "Efficient Fully Homomorphic Encryption from (Standard) LWE" as well as Junfeng Fan and Frederik's Vercauteren's "Somewhat Practical Fully Homomorphic Encryption", both papers that build off of Gentry's work and are implemented by Microsoft in the C++ library SEAL [5].

While the state-of-the-art homomorphic methods proposed today are impractical for computationally complex tasks like machine learning without substantial delay [6], the schemes reviewed above are capable of handling machine learning evaluation. The implementation section of this coursework is inspired from a 2017 blogpost [7], and will construct a hypothetical scenario, solved with homomorphic encryption, such that a government agency wishes to use machine learning in order to identify pro-ISIS messages without (a) collecting the messages of citizens and (b) allowing users to reverse engineer the model. Our implementation utilizes the machine learning library scikit-learn [8], the Github repository python-paillier [9], and a Python port [10] of Microsoft SEAL 2.3 [5] in order to benchmark and evaluate the parameters of the Paillier and Fan Vercauteren cryptosystems.

2 Literature Review

2.1 RSA

A year after publishing the original RSA paper [11], Rivest claimed that, "it appears likely that there exist encryption functions which permit encrypted data to be operated on without preliminary decryption of the operands, for many sets of interesting operations" [12]. Rivest's suspicion is due to a peculiar property of the RSA algorithim that allows the homomorphic operations over the multiplicative field. In RSA, a sender Bob can encrypt value m_1 and value m_2 by raising these values to the power of e and performing modulo arithmetic N on the output, where e and N are public.

$$c_1 = m_1^e \ \ (mod \ \ N)$$

$$\mathrm{c}_2 = m_1^e \ \ (mod \ \ N)$$

Due to the algebraic properties of exponentiation, it is trivial to see that multiplying c_1 and c_2 is equivalent to multiplying m_1 and m_2.

$$c_1 \times c_2 = m_1^e \times m_2^e \ \ (mod \ \ N)$$

$$\mathrm{c}_1 \times c_2 = (m_1 \times m_2)^e \ \ (mod \ \ N)$$

After decrypting with a complimentary private key d, it is evident that the multiplication operation was performed successfully without any noise or inaccuracy in the output; as such the operation can be performed infinite times.

2.2 Paillier

Named after Pascal Paillier, the Paillier crypto system was invented in 1999 as a probabilistic asymmetric crytographic scheme [3]. The computational strength of the Paillier system relies on the decisional composite residuosity assumption, which claims that given a composite integer n and integer z, it is difficult for an attacker to determine whether there exist a y such that

$$z \equiv y^n \ \ (mod \ \ n^2)$$

Key Generation The public and private keys are first generated for the Paillier system by choosing two large prime numbers p and q of equal length and computing $n = pq$ along with $\lambda = \phi(n)$ where $\phi(n) = (p-1)(q-1)$. μ can be found easily by calculating $\mu = \phi(n)^{-1} \ \ (mod\, n)$. Consequently, our encryption key and encryption key are (n, g) and (λ, μ), respectively.

Encryption For encryption in the Paillier scheme, Alice must choose a positive integer message less than n, the first component of the public key. She then choose a random positive integer r less than and coprime to n. The cipher-text value is then equal to:

$$c = g^m \times r^n \ \ (mod \ \ n^2)$$

Decryption Decryption of the cipher-text c requires that $c < n^2$. The cipher-text can be decrypted to plain-text m with the following equation:

$$m = L(c^\lambda \ \ mod \ \ n^2) \cdot \mu(mod \ \ n)$$

$$\text{where } L(x) = \tfrac{x-1}{n}$$

The Decisional Composite Residuosity Assumption (DCRA) is the assumption that computing n^{th} residue classes has intractable computational complexity, and thus acts as the trapdoor function for the Paillier scheme. A residue class is a set of integers that are congruent modulo n for some positive integer n. A number z is said to be a n^{th} residue modulo n^2 if there exists a number $y \in \mathbb{Z}_{n^2}^*$ such that:

$$z = y^n \ \ (mod \ \ n^2)$$

Paillier states that the problem of distinguishing n-th residues from non n-th residues is computationally difficult in that it cannot be distinguished in polynomial time. Since inverting the encryption equation of Paillier scheme is the composite residuosity class problem, Paillier ensure semantic security [3].

Additive Homomorphic Properties Paillier demonstrates the homomorphism from $(\mathbb{Z}_{n^2}^*, \times)$ to $(\mathbb{Z}_{n^2}^*, +)$ via a lemma. Let the n-th residuosity class of w with respect to g be denoted as $\|w\|$:

$$\forall w_1, w_2 \in Z_{n^2}^* \qquad \|w_1 w_2\| = \|w_1\|_g + \|w_2\| \ \ (mod \ \ n)$$

This lemma allows the three following homomorphic properties:

- The product of c_1 and c_2 is equal to the sum of m_1 and m_2
- The product of c_1 and g^{m_2} is equal to the sum of m_1 and m_2
- c_1 raised to the power of m_1 is equal to the product of m_1 and m_2

The homomorphic properties of Paillier cipher-text with plain-texts is valuable in the context of the logistic regression implementation found in Section 3.

2.3 Gentry

Craig Gentry broke new ground in the field of homomorphic encryption with his seminar paper, "Fully Homomorphic Encryption Using Ideal Lattices" [4]. Gentry's method relies on a somewhat homomorphic lattice-based crypto scheme; the scheme is limited in the number of operations that can be performed on a cipher-text before "noise", a by-product of the probabilistic nature of the scheme, grows so large such that the plain-text mapping is inaccurate. The monumental insight gained from Gentry's work was the concept of bootstrapping, a technique that refreshes the noise of a cipher-text by decrypting the cipher-text with a new key without revealing the plain-text. While strictly following Gentry's algorithm was unrealistic due to Big-O complexity, his method was the foundation for practical implementations such as HELib and SEAL, the latter of which is utilized in Section 3.

Lattice Based Cryptography In linear algebra, a basis of a vector space is a set of n independent vectors such that any coordinate point on said space is a linear combination of these basis vectors. The lattice of a vector space is the set of basis linear combinations with integer coefficients; for example, all (x, y) points where $x, y \in \mathbb{Z}$ on a Euclidean vector space make up the lattice. Ideal lattices are, "lattices corresponding to ideals in rings of the form $\mathbb{Z}[x]/(f)$ for some irreducible polynomial of degree n" [13]. Ideal lattices are essential to the semantic security of Gentry's FHE method due to the intractable nature of the closest vector problem - given a vector v outside of any lattice points, which lattice point is closest to v? The closest vector problem forces one to perform lattice basis reduction in order to be solved, but at the cost of exponential time. When the vector without

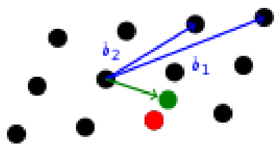

Figure 2: **Closest Vector Problem** - basis vectors in blue, external vector in green, closest vector in red [13]

error is known by a party, this closest vector problem allows this party to "hide" an encoded message m_1 with an error if the message space is $mod\ p$ for some integer p, the cipher space is $mod\ q$ for some integer $q >> p$, and the error is divisible by p, allowing simple future removal of the error. Consequently, the error is calculated by randomly generating e from a uniform distribution and multiplying e by p, thus ensuring this divisibility and clean error removal. Furthermore, it is essential that the chosen p is much less than q since all operations in the scheme are performed $mod\ q$ [14].

Due to the algebraic properties of vector addition and multiplication, it is possible to calculate the sum and product of two cipher-texts with the respective sum and products of the error. When the error is removed after the operation $F(c_1, c_2)$, via decryption, the output is equivalent is $F(m_1, m_2)$. However, this growth in the error is why lattice-based cryptography is somewhat homomorphic; if the error grows too large then the closest lattice vector during decryption is no longer accurate. For example, on a euclidean space if the correct lattice vector is $(1, 1)$ but the error is $x = .3$ and $y = 0.6$, then the decryption will incorrectly decrypt to $(1, 2)$.

Bootstrapping The solution Craig Gentry proposes to counter noise growth in lattice-based cryptography is a "bootstrapping" technique, thus transforming the scheme from partially homomorphic to full homomorphic cryptography. The technique is based off the intuitive notion that the only way to remove noise is to decrypt the cipher-text; therefore if the decryption operations are performed with the private key k_1 encrypted with a new private key k_2 as well as the cipher-text c_1 encrypted with the new key k_2, then the error is reduced to the noise added by the homomorphic decryption operation. Due to the circuit complexity of the decryption operation, Gentry invents a squashing technique to the decryption function that in a sense provides a "hint" to the decryption process for the evaluator, but relies on the intractability of the subset sum problem; given a set of integers find a non empty subset with a sum of 0 [4]. The majority of homomorphic research in the past decade has been built upon Gentry's proposal, mainly focusing on (1) reducing the homomorphic operation cost of decryption and (2) reducing the resources necessary to encrypt an already encrypted cipher-text.

2.4 Brakerski and Vaikuntanathan

In 2011, Zvika Brakersi and Vinod Vaikuntanathan published a fully homomorphic encryption scheme that improved on Gentry's scheme in both efficency and simplicity [15]. This paper introduces two key concepts: a re-linearization technique that removes the need for intractability assumptions regarding ideal lattices, and a dimension-modulus technique that removes the need for squashing and the above mentioned intractability assumption of the subset sum problem.

Re-Linearization Braverski and Vaikuntanathan migrated the assumption from ideal lattice cryptography to general lattices by utilizing learning with errors, which states that given a basis, a linear combination of the basis vector with small error, and a lattice point, finding the latter vector from the former is computationally difficult. Since ideal lattices are a relatively new field of study in the mathematics community as opposed to general lattices, Brakersi and Vaikuntanathan claim that the community has, "a much better understanding of the complexity of lattice problems (thanks to [LLL82, Ajt98, Mic00] and many others), compared to the corresponding problems on ideal lattices"[15], thus providing more confidence to this scheme's semantic security. By using learning with errors, a homomorphic multiplication optimization called re-linearizaton can be performed, which utilizes a new key to decrease the degree of cipher-text. This optimization prevents noise from growing exponentially during multiplication, replacing the growth factor with a constant dependent on the initial security parameter λ.

Dimension-Modulus Reduction As mentioned above, Gentry's 2009 paper utilizes a "squashing" technique in order to ensure that homomorphic operations were possible on the decryption circuits by relying on the hardness of the subset sum problem. Braverski and Vaikuntanathan demonstrate that a learning with error homomorphic scheme with dimension-modulus reduction only requires a relatively small decryption, thus making any squashing of the decryption circuit unnecessary. Dimension-modulus reduction is the process of converting a cipher-text with dimension n and cipher modulo q and mapping this cipher-text with new parameters of dimension k where $k << n$ and modulo $log(p)$ where p is the plain-text modulus. The semantic security of this dimension-modulus reduction relies the hardness of learning with errors for dimension k modulo $log(p)$. Consequently, this reduction allows Braverski and Vaikuntanathan's scheme to be boot-strappable, thus fully homomorphic, without assumptions beyond the intractability of the learning with errors problem.

2.5 Fan Vercauteren Scheme

Jufeng Fan and Frederik Vercauteren's "Somewhat Practical Fully Homomorphic Encryption" directly builds off of Braverski's learning with errors homomorphic scheme by introducing a ring variant of the learning with error problem [16]. The 2012 paper optimizes Braverski's re-linearization with the aid of smaller re-linearization keys, as well as a modulus switching trick in order to simplify bootstrapping.

Ring Learning with Errors In mathematics, a ring R is a set with two binary operations that allows generalization from normal arithmetic to other frames like polynomials and functions. Thus, a polynomial ring can be $R = \mathbb{Z}/f(x)$ where $f(x) \in Z[x]$ is a monic irreducible polynomial of degree d. Fan and Vercauteren utilize polynomial rings in creating the hardness of their scheme: Definition 1 (Decision-RLWE):

> For security parameter λ, let $f(x)$ be a cyclotomic polynomial $\omega_m(x)$ with $deg(f) = \sigma(m)$ depending on λ and set $R = Z[x]/(f(x))$. Let $q = q(\lambda) \geq 2$ be an integer. For a random element $s \in R_q$ and a distribution $\chi = \chi(\lambda)$ over R, denote with $A(a)$ the distribution obtained by choosing a uniformly random element $a \leftarrow \chi$ and outputting $(a, [a \cdot s + e]_q)$. The Decision-RLWE problem is to distinguish between the distribution $A^q_{s,\chi}$ and the uniform distribution $U(R^2_q$. [16]

Modulus Switching Trick During re-linearization in Braverski's scheme, the secret key s^2 is masked such that the error term e_1 is multiplied with the cipher-text c_2. In order to avoid excess error modulo q, a masked version of s^2 is substituted by using modulo $p \cdot q$ for some integer q. Consequently, this technique allows efficient transformation from cipher-text encrypted under $mod\ p$ into a cipher-text under this new modulus $p \cdot q$ but with reduced noise.

3 Implementation

3.1 Background

As mentioned in Section 1, internet technology companies and government agencies each have an imperative that require data analytics. However, the requirement for data analytics does not imply a need for data collection; these organizations do not relish data silo maintenance, data breach countermeasures, and the damage control against inevitable internal employee misuse. Homomorphic cryptography can diminish these vulnerabilities by separating the data evaluators from the data owners. For example, imagine a government agency who wishes to detect messages related to terrorist activity exchanged on a public network; we will refer to this agency as Big Brother. But unlike Orwell's dystopic counterpart, our Big Brother has regulations in place that prevent the collection of plain-text messages. Our Big Brother requires probable cause before being granted a warrant for a citizen's data, thus ensuring the honest citizen's data privacy. Big Brother's necessity for surveillance can be met by using a homomorphic scheme akin to a metal detector in an airport. Rather than forcing a body search of every passenger, airport security use metal detectors to single out the potentially dangerous passengers. Furthermore, since dangerous passengers can not experiment with an airport metal detector from home, the ability to reverse engineer or trick the airport detector is severely limited.

In our implementation of homomorphic cryptography, our metal detector is a homomorphically encrypted logistic regression model trained to detect pro-ISIS tweets. In our hypothetical scenario, Big Brother trained his model using pro-ISIS messages collected from previous investigations and synthetic ISIS-related data; in reality, the pro-ISIS ($pro\_isis = 1$) and ISIS-related data ($pro\_isis = 0$) was collected from a Kaggle dataset [17]. After training, Big Brother encrypts the weights and y-intercept of his model using either the Paillier scheme or Fan Vercauterene scheme, and sends the encrypted model with any necessary evaluation parameters to the messaging devices of his citizens. When an arbitrary citizen, who we will refer to as Winston Smith, sends a message from his device, the message is evaluated by the encrypted model resulting in an encrypted prediction that is sent to Big Brother. Winston can not discern the result nor purpose of the encrypted prediction, and it is not possible for Winston to disable this feature. Upon receiving the encrypted prediction, Big Brother decrypts the prediction m with his private key and inputs this value into the logistic function $p = \frac{1}{1=e^{-m}}$ and ignores any probability p below an arbitrary threshold γ. However, a probability greater than γ is equivalent to a $beep$ in airport security and can be used as probable cause for a warrant in order to lawfully obtain Winston's plain-text messages (as wells as more ground truth to improve the accuracy of Big Brother's model). Consequently, two major goals have been achieved: lawful citizens of the state are ensured data privacy, and the state can protect national security without collecting personal data nor revealing the source code of their surveillance to potential criminals. A sample of script demonstrating this exchange is illustrated on Listing 1, as well as available live through the docker image of this coursework.

Listing 1: Example Exchange

```
def example_situation(classifier, vectorizer, trainset, testset,
    tweet):

    seal_scheme = SealScheme()
    seal_crypto = seal_scheme.getCrypto()

    big_brother = BigBrother(classifier, vectorizer,
        seal_crypto, trainset, testset)

    big_brother.train()

    encrypted_model = big_brother.get_encrypted_model()

    seal_eval = seal_scheme.getEval()
    winston = WinstonSmith(testset, seal_eval, vectorizer,
        encrypted_model)

    tweet_vector = winston.vectorize(tweet.tweets)

    encrypted_prediction = winston.predict(tweet_vector)

    decrypted_prediction = big_brother.decrypt_result(
        encrypted_prediction)

```

3.2 Logistic Regression

Logistic regression is a statistical technique used to estimate the parameters of a binary model, binary in that there are two possible outputs for the dependent variable discerned from independent variables. The estimated parameters, or weights, along with the y-intercept and logistic function allow us to calculate the probability for either one of the dependent target variables. A logistic model is trained using a maximum likelihood estimate and labeled data, with the goal being to find the model parameters that most minimize the training error. Logistic regression is a fundamental machine learning method, and as such is implemented in the open source Python machine learning library scikit-learn [8]. The code used to trained our ISIS message detector is shown from lines 14 to 23 of Listing 4. Before submitting text data for training in natural language processing, one must decide how they wish to quantify the representation of text. For the sake of simplicity, we will use a bag-of-words approach that tokenizes each message into a vector where each index represents a distinct word from the "vocabulary" of the training set. Since there are 44,000 unique words in our 37,000 message dataset, the training matrix has 44,000 columns and 37,000 rows. We can reduce this sparse column vector to any size we choose for the sake of optimization at the cost of increasing hash collisions and accuracy; the number of features used in our implementation is benchmarked in Section 4.

3.3 Python Paillier

The Paillier implementation used in this coursework is from a publicly available Github repository owned by N1 Analytics [9]. The initialization of the scheme is trivial; the API key-pair generation function is given a requested key size in bits and return a randomly generated public and private key pair. The precision argument is used during the encryption process for rounding floating point integers, as the Paillier scheme only works on integer values. When Big Brother uses his public key to encrypt the model parameters, an EncryptedNumber object is instantiated containing the cipher-text along with Big Brother's public key; no further information needs to be provided to Winston in order to evaluate the cipher-text.

Listing 2: Paillier Scheme Initialization

```
class PaillierScheme(HomomorphicScheme):

def __init__(self, n_length=2048, precision = 4):
    self.pubkey, self.private_key =
        paillier.generate_paillier_keypair(n_length)

    self.precision = precision

def getCrypto(self):
    return PaillierCrypto(self.pubkey, self.private_key,
        self.precision)

def getEval(self):
    return PaillierEval()

```

3.4 PySEAL

The PySEAL library is a Python port of the C++ library SEAL, developed by a homomorphic research team at Microsoft [10] [5]. SEAL is an implementation of the Fan Vercauteren homomorphic scheme described in Section 2.5, including further optimizations introduced by the 2016 paper "A Full RNS Variant of FV like Somewhat Homomorphic Encryption Schemes" [18]. Due to the fully homomorphic nature of the Fan Vercauteren scheme, the initialization parameters are more complex relative to the Paillier scheme.

The first parameter configured on line 8 of Listing 3 is the polynomial modulus. The polynomial modulus is required to be a power-of-two cyclotomic polynomial of the form $1x^{power-of-two} + 1$. The size of the polynomial modulus is proportional to the security level of the homomorphic scheme and inversely proportional to the computation time because a larger polynomial modulus increases the size of cipher-texts.

The second parameter chosen is the coefficient modulus, which is directly proportional to the allowance of noise accumulated on a cipher-text before the plain-text message is unrecoverable. However, increasing the coefficient modulus will also decrease the security level of the scheme. The SEAL documentation advises to use the helper function illustrated on lines 11 to 16, where $bit\_strength = n$ denotes that it would take 2^n operations to break the cipher. Furthermore, the plain-text modulus is configured in order to determine the size of the plain-text data, and is inversely proportional to the noise budget. The importance of these parameters on the remaining noise budget and computation time will be investigated in Section 4. Ultimately, the noise budget of an encrypted cipher-text can be estimated with:

$$noise\_budget \approx log_2(\frac{\text{coefficient\_modulus}}{\text{plain\_modulus}})$$

The fractional encoder utilized on line 31 of Listing 3 allows the encoding of a fixed-precision rational number into a plain-text polynomial. Given a base b, the fractional encoder maps a rational number to a polynomial where $x = b$, and will limit the number of fractional and integral coefficients by fractional_coeff and fractional_base, respectively. The fractional encoder will also move any fractional component x of the number to the degree of the polynomial modulus minus the degree of the fractional component multiplied by -1; 0.75 where $b = 2$ and the polynomial modulus degree is 12 would be encoded as $-1x^{11} - 1x^{10}$. To further illustrate this encoding method, a fraction encoded with polynomial degree $n = 2048$, base $b = 2$, $\text{fractional\_coeff} = 1$, and $\text{fractional\_base} = 64$ will be encoded as:

$$24.2351 = 1x^4 + 1x^3 - 1x^{-2047}$$

Listing 3: SEAL Scheme Initialization

```
1  class SealScheme(HomomorphicScheme):
2
3      def __init__(self, poly_modulus = 2048, bit_strength = 128,
4          plain_modulus = 1<<8, integral_coeffs = 64,
5          fractional_coeffs = 32, fractional_base = 3):
6
7          parms = EncryptionParameters()
8          parms.set_poly_modulus("1x^{} + 1"
9          .format(poly_modulus))
10
11         if (bit_strength == 128):
```

```
            parms.set_coeff_modulus(
                seal.coeff_modulus_128(poly_modulus))
        else:
            parms.set_coeff_modulus(
                seal.coeff_modulus_192(poly_modulus))

        parms.set_plain_modulus(plain_modulus)

        self.parms = parms
        context = SEALContext(parms)

        keygen = KeyGenerator(context)
        public_key = keygen.public_key()
        secret_key = keygen.secret_key()

        self.encryptor = Encryptor(context, public_key)
        self.evaluator = Evaluator(context)
        self.decryptor = Decryptor(context, secret_key)

        self.encoder = FractionalEncoder(context
        .plain_modulus(), context.poly_modulus(),
        integral_coeffs, fractional_coeffs, fractional_base)

    def getCrypto(self):
        return SealCrypto(self.encoder, self.encryptor,
            self.decryptor, self.parms)

    def getEval(self):
        return SealEval(self.encoder, self.evaluator, self.parms)

```

3.5 Big Brother and Winston Smith

Big Brother Our Big Brother is initialized with a classifier object, a vectorizer object, a cryptographic scheme, a test set, and a trainset. The classifier is the machine learning model used for surveillance, while the vectorizer is used to extract independent variables from raw text. The cryptographic scheme is a 'HomomorphicCryptography' object with the decrypt and encrypt implementations abstracted from Big Brother, allowing simple substitution of any homomorphic cryptographic method.

Listing 4: Benevolent Big Brother

```
class BigBrother:

    def __init__(self , classifier, vectorizer,
        homomorphic_cryptography,
        testset = None, trainset = None):

        self.classifier = classifier
        self.vectorizer = vectorizer
        self.homomorphic_cryptography =
            homomorphic_cryptography
        self.testset = testset
        self.trainset = trainset

    def train(self):
        vectorize_text = self.vectorizer.fit_transform(
            self.trainset.tweets.values.astype('U'))
        self.classifier = self.classifier.fit(vectorize_text,
            self.trainset.pro_isis)
        vectorize_text = self.vectorizer.transform(
            self.testset.tweets.values.astype('U'))
        score = self.classifier.score(vectorize_text,
            self.testset.pro_isis)
        return score

    def get_encrypted_model(self):
        self.encrypted_model = self.homomorphic_cryptography
            .encrypt(self.classifier)
        return self.encrypted_model

    def decrypt_result(self, encrypted_prediction):
        value = self.homomorphic_cryptography.decrypt(
            encrypted_prediction)
        return 1/(1+np.exp(-value))

    def plain-text_predict(self,tweet):
        return self.classifier.predict_proba(
            self.vectorizer.transform(tweet))[:,1][0]

```

Winston Smith Winston Smith is initialized with test data, a homomorphic evaluator, a vectorizer, and the encrypted model. The test data is used to simulate original messages written by Winston Smith. The evaluator contains any parameters necessary to evaluate his messages. The vectorizer is used transform his raw message to a token vector; while this reveals the number of features used in the encrypted model, it does not reveal the importance or weight of any feature. The encrypted model is supplied by Big Brother, and is a tuple of the encrypted weights and encrypted y-intercept.

Listing 5: Winstom Smith

```
class WinstonSmith:
    def __init__(self, data, homomorphic_eval, vectorizer,
        encrypted_model):
        homomorphic_eval.set_encrypted_model(
            encrypted_model)
        self.data = data
        self.homomorphic_eval = homomorphic_eval
        self.vectorizer = vectorizer

    def vectorize(self, text):
        return np.array(self.vectorizer.transform(text)
            .todense())[0]


    def predict(self, vector):
        return self.homomorphic_eval.evaluate(vector)

    def talk(self):
        return self.data.sample(1)

```

4 Evaluation

A truly practical homomorphic cryptographic scheme needs to be able to calculate arbitrary calculations without developers needing to fine tune their evaluation code or wait substantial time for calculations that are relatively fast on plain-text data. Fortunately, the computational requirements for a logistic regression prediction are small; multiplication against plain-text data is not only possible in the Fan Vercauteren and Paillier scheme, but also has no affect on the noise budget of cipher-texts in the former. A logistic regression prediction, where n represents the number of features used in the model, x_i represents the data token at index i, w_i represents the weight coefficent at index i, and y_o represents the intercept is calculated as such:

$$Y = w_0 \cdot x_0 + w_{11} + ... + w_{n-1} \cdot x_{n-1} + y_0$$

This formula illustrates that addition will be the primary culprit for any noise accumulated during Winston's evaluation. Furthermore, the expansion and exponentiation of the plain-text model to the cipher-text model will be time intensive for both the Paillier and FV scheme.

4.1 PySEAL Benchmark

The PySEAL benchmark was performed by timing different components of our exchange while changing one exchanging parameter. The different components are: the scheme initialization, the model encryption, the homomorphic evaluation, and the prediction decryption. The benchmark was performed on the same message throughout the experiment. Unless otherwise stated, the default parameters for this experiment are as follows:

Parameter	Default Value
nfeatures	1000
polymodulus	1024
plainmodulus	16
integralcoeffs	32
fractionalcoeffs	64

Polynomial Modulus For the FV scheme of PySEAL, the importance of the polynomial modulus degree on the computation time and remaining noise budget of the final cipher-text $E(Y)$ was evaluated for 2^n where $10 \leq n \leq 14$. As mentioned in Section 3.4, the polynomial modulus increases the size of the cipher-text dramatically; although not recorded precisely in this coursework, the benchmark scripts memory usage grew linearly with the modulus degree such that a degree of 2^{10} required $400MB$ of RAM while a degree of 2^{14} required $20G$ of RAM. It can be seen from the below table how the degree of the polynomial modulus has a postive relationship with the available noise budget; this increased noise budget allows us to evaluate the cipher-text further without needing to re-linearize or otherwise refresh the noise, an expensive operation.

Polynomial Modulus	Noise Budget
1024	10
2048	36
4096	90
8192	198

Figure 3 demonstrates the cost that increasing the polynomial modulus and thus the cipher-text size has on the computation time.

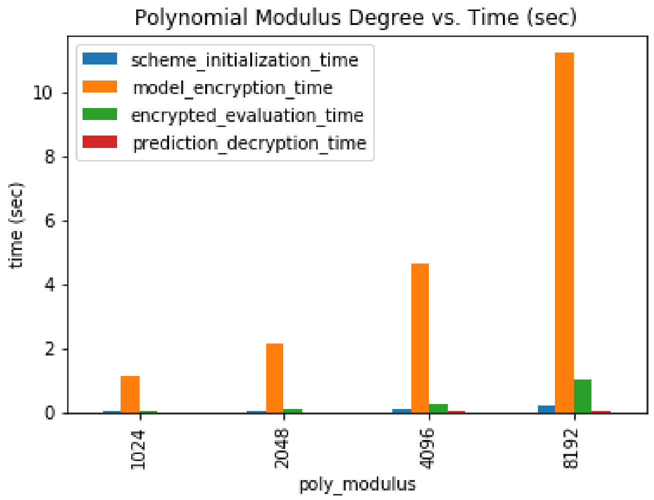

Figure 3: **Polynomial Modulus**

Bit Strength As mentioned in Section 3, the bit strength denotes the level of security for our scheme. While Figure 4 illustrates that while increasing the bit strength from 128 to 192 has little effect on the computation time for our model with 1000 features, the bit strength table below demonstrates that it does have a negative effect on the remaining noise budget of the evaluated cipher-text.

Bit Strength	Noise Budget
128	10
192	1

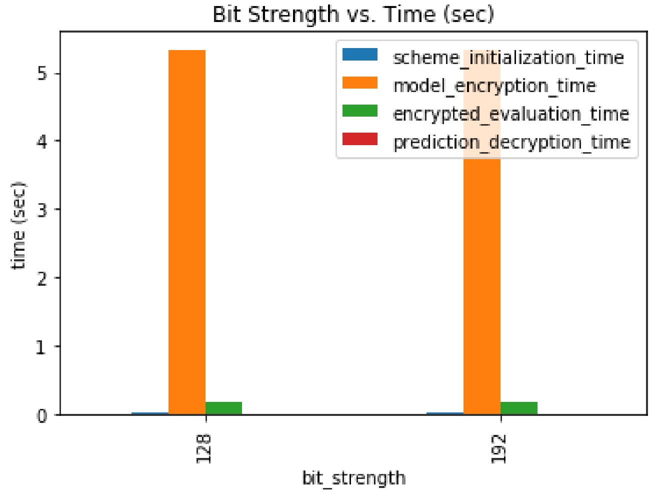

Figure 4: **Bit Strength**

Plain Modulus The effect of the plain modulus degree on the noise budget is evident by the negative relationship demonstrated in the table below. Furthermore, Figure 5 illustrates that the plain modulus is uncorrelated with the time complexity of the FV cryptographic scheme.

Plain Modulus	Noise Budget
64	8
128	7
256	6
512	5
1024	4
2048	3
4096	2
8192	1
16384	0

Number of Features The number of features used by the logistic regression model has a direct correlation with the accuracy of the model. Figure 6 shows that the number of features is also correlated with the model encryption and evaluation time because more weights need to be encrypted and evaluated.

Features	Accuracy
1000	0.9304
2000	0.9396
3000	0.9445
4000	0.9459
5000	0.9502
6000	0.9534
7000	0.9548
8000	0.9517
9000	0.9548
10000	0.9545

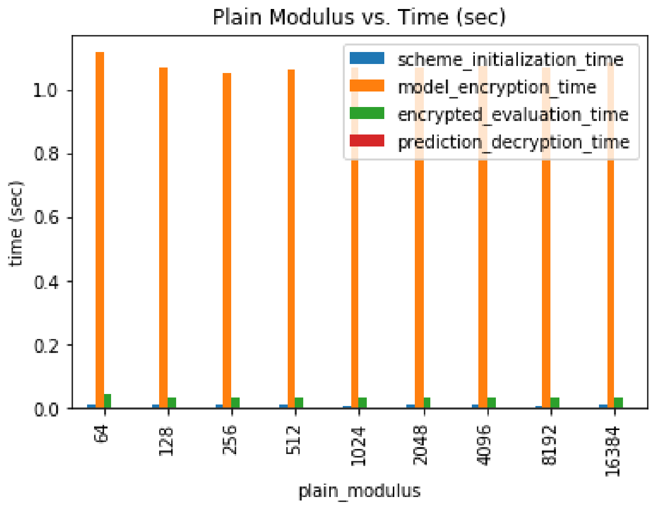

Figure 5: **Plain Modulus**

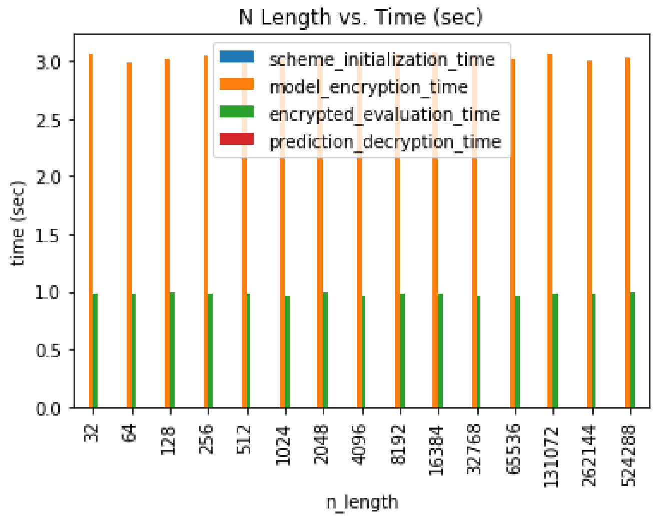

Figure 7: **Features**

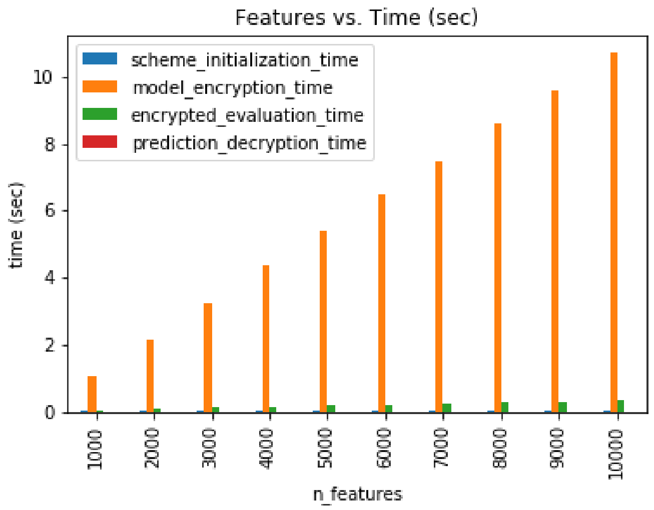

Figure 6: **Features**

4.2 Paillier Benchmark

Due to the partially homomorphic nature of the Paillier cryptosystem, there is no noise budget to manage as such there does not exist a noticeable difference in time required to perform logistic regression evaluation. Figure 7 demonstrates the private key lengths lack of correlation with the encryption and evaluation time of Paillier cryptography. The benchmark of Paillier was performed using a count vectorizer with 44,000 features, as opposed to 1000 features for SEAL; this difference demonstrates the light-weight efficiency of the Paillier cryptosystem for this addition-intensive specific evaluation.

5 Conclusions

This coursework is by no means an advocate of state-surveillance evaluated on private devices for the purpose of incrimination. Nonetheless, we have entered an age where personal privacy has been deemed second to profit and homeland-security. While the cryptographic exchange in this coursework offers one potential compromise that could prevent further eroding of civil liberties, there are many other potential homomorphic applications that are not as dystopic. For example, homomorphic encryption would allow medical data from multiple private institutions to be centralized and analyzed without privacy concerns, accelerating medical research development. Companies are beginning to realize the potential of homomorphic cryptography; engineers at an American cyber-security company are researching homomorphic cryptography to allow clients to build machine learning models with other client data without breach of privacy. Homomorphic cryptography is a rapidly developing field of study; the three most recent papers reviewed in Section 2 were published all within 3 years. A noble goal for homomorphic encryption is to integrate this system into the public channel as effectively as previous crypto holy grails, like public key cryptography, exist today.

References

[1] A. Selyukh, "Nsa staff used spy tools on spouses, ex-lovers: Watchdog," 2013 (accessed April 20, 2019).

[2] Pymnts, "Lyft accused of giving access to rider data, including zuck's phone number," 2018 (accessed April 20, 2019).

[3] P. Paillier, "Public-key cryptosystems based on composite degree residuosity classes," in *International Conference on the Theory and Applications of Cryptographic Techniques*, pp. 223–238, Springer, 1999.

[4] C. Gentry *et al.*, "Fully homomorphic encryption using ideal lattices.," in *Stoc*, vol. 9, pp. 169–178, 2009.

[5] "Microsoft SEAL (release 2.3)." http://sealcrypto.org, Dec. 2017. Microsoft Research, Redmond, WA.

[6] K. Laine, R. Gilad-Bachrach, H. Chen, K. Lauter, K. Han, Z. Huang, and A. Jalali, "Logistic regression

over encrypted data from fully homomorphic encryption," October 2018.

[7] iamtrask, "Safe crime detection," 2017 (accessed April 20, 2019).

[8] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, "Scikit-learn: Machine learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

[9] "python-paillier." https://github.com/n1analytics/python-paillier, Dec. 2012. n1analytics.

[10] "Pyseal." https://github.com/Lab41/PySEAL, Dec. 2017. Lab41.

[11] R. L. Rivest, A. Shamir, and L. Adleman, "A method for obtaining digital signatures and public-key cryptosystems," *Communications of the ACM*, vol. 21, no. 2, pp. 120–126, 1978.

[12] R. L. Rivest, L. Adleman, M. L. Dertouzos, *et al.*, "On data banks and privacy homomorphisms," *Foundations of secure computation*, vol. 4, no. 11, pp. 169–180, 1978.

[13] Wikipedia contributors, "Lattice problem — Wikipedia, the free encyclopedia," 2004. [Online; accessed 22-April-2019].

[14] F. Raynal, "A brief survey of fully homomorphic encryption, computing on encrypted data."

[15] Z. Brakerski and V. Vaikuntanathan, "Efficient fully homomorphic encryption from (standard) lwe," *SIAM Journal on Computing*, vol. 43, no. 2, pp. 831–871, 2014.

[16] J. Fan and F. Vercauteren, "Somewhat practical fully homomorphic encryption.," *IACR Cryptology ePrint Archive*, vol. 2012, p. 144, 2012.

[17] F. Tribe, "How isis uses twitter," 2016. We scraped over 17,000 tweets from 100+ pro-ISIS fanboys from all over the world since the November 2015 Paris Attacks. , https://www.kaggle.com/fifthtribe/how-isis-uses-twitter.

[18] J.-C. Bajard, J. Eynard, M. A. Hasan, and V. Zucca, "A full rns variant of fv like somewhat homomorphic encryption schemes," in *International Conference on Selected Areas in Cryptography*, pp. 423–442, Springer, 2016.