Coursework: Homomorphic Encryption

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

Abstract

Test

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 a second party to perform arbitrary functions on a ciphertext without the need to decrypt the ciphertext in advanced. Furthermore, the decrypted ciphertext is equivalent to the output of the same arbitrary functions performed on the plaintext. 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 plaintext itself. A cryptographic schemes 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.

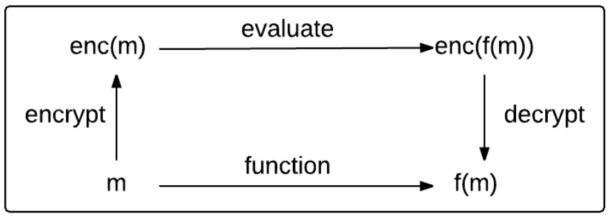
$$\begin{tikzcd} \mathrm{enc}(m) \arrow[r, "\text{evaluate}"] & \mathrm{enc}(f(m)) \arrow[d, "\text{decrypt}"] \\ m \arrow[u, "\text{encrypt}"] \arrow[r, "\text{function}"] & f(m) \end{tikzcd}$$

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 [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 Junfeng Fan and Frederik's Vercauteren's "Somewhat Practical Fully Homomorphic Encryption", a paper the builds off of Gentry's work and was implemented by Microsoft in the C++ library SEAL [5].

The implementation section of this coursework is inspired from a 2017 blogpost [6], 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 be aware of what the model is predicting. Our implementation utilizes machine learning library scikit-learn [7], the Github repository python-paillier [8], and a Python port [9] of Microsoft SEAL 2.3 [5] in order to benchmark and evaluate the parameters of the Paillier and FV cryptosystems.

2 Literature Review

2.1 RSA

A year after publishing the original RSA paper [10], 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" [11]. 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 V_1 and value V_2 by raising these values to the power of e and performing modulo arithmetic N on the output, where e is private and N is public.

$$C_1 = V_1^e \ (mod \ N)$$
$$\mathrm{C}_2 = V_1^e \ (mod \ N)$$

Due to the algebraic properties of exponentiation, it is trivial to see that multiplying C_1 and C_2 is equivalent to multiplying V_1 and V_2.

$$C_1 \times C_2 = V_1^e \times V_2^e \ (mod \ N)$$
$$\mathrm{C}_1 \times C_2 = (V_1 \times V_2)^e \ (mod \ N)$$

After decrypting with a complimentary 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 ciphertext can be decrypted to plaintext m with the following equation:

$$m = L(c^\lambda \ mod \ n^2) \cdot \mu(mod \ n)$$
$$\text{where } L(x) = \frac{x-1}{n}$$

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 ciphertext with plaintexts is valuable in the context of the logistic regression implementation 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 ciphertext by decrypting the ciphertext with a new key without revealing the plaintext. 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". 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 error is known by a party, this learning with error problem allows the 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

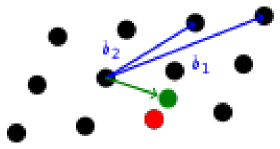

Figure 2: **Closest Vector Problem** - basis vectors in blue, external vector in green, closest vector in red [12]

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 $mod\ q$ [13].

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 p_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 [14]. 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"[14], 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 ciphertext 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 [15]. 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 , let f(x) be a cyclo- tomic polynomial m(x) with deg(f) = (m) depending on and set R = Z[x]/(f(x)). Let q = q() 2 be an integer. For a random element s Rq and a distribution = () over R, denote with A(q) the distribution obtained by choosing a uniformly randomelementaRq andanoisetermeandoutputting(a,[a·s+e]q).The Decision-RLWE problem is to distinguish between the distribution A(q) and the d,q, uniform distribution U(Rq2).

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, financially and politically, that require data analysis. However, the requirement for data analytics does not imply a need for data collection; these organizations do not relish data silo maintenance, investing in security against data breach, and the damage control against never-ending unethical employee scandals. 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 the public network; we will refer to this agency as Big Brother. But unlike Orwell's dystopic counterpart, this Big Brother has regulations in place that prevent the collection of plain-text messages (most likely due to past scandals). Our Big Brother requires probable cause before being granted a warrant on a citizen. This conundrum can be solved using a homomorphic scheme akin to a metal detector at an airport. Rather than forcing a touch down 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. Logistic regression is a machine learning algorithm ¡define logistic regression here¿. 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$_i sis = 1) and ISIS-related data (pro_i sis = 0) was collected from a Kaggle dataset. After training, Big Brother encrypts the weights and y-intercept of his model using either the Paillier scheme or Fan Vercauteren escheme, and sends the encrypted model with any necessar$
However, a probability greater than γ is equivalent of 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 his model). Consequently, two major goals have been achieved: lawful citizens of this state have been 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.

3.2 Python Paillier

Listing 1: Paillier Scheme Initialization

```
class PaillierScheme(HomomorphicScheme):

    def __init__(self, n_length=64, precision = 4):
        self.pubkey, self.private_key =
            paillier.generate_paillier_keypair(n_length=64)

        self.precision = precision

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

    def getEval(self):
        return PaillierEval()
```

3.3 PySEAL

Listing 2: SEAL Scheme Initialization

```
class SealScheme(HomomorphicScheme):

    def __init__(self, poly_modulus = 2048, bit_strength = 128,
        plain_modulus = 1<<8, integral_coeffs = 64,
        fractional_coeffs = 32, fractional_base = 3):

        parms = EncryptionParameters()
        parms.set_poly_modulus("1x^{} + 1"
        .format(poly_modulus))

        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)
```

3.4 Logistic Regression

Prediction function

3.5 BenevolentBigBrother and Winston-Smith

Listing 3: Benevolent Big Brother

```
class BenevolentBigBrother:

    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 plaintext_predict(self,tweet):
        return self.classifier.predict_proba(
            self.vectorizer.transform(tweet))[:,1][0]

```

Listing 4: Winston 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)

```

Listing 5: Example Exchange

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

    seal_scheme = PaillierScheme()
    seal_crypto = seal_scheme.getCrypto()

    big_brother = BenevolentBigBrother(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)



```

4 Evaluation

4.1 Code Listing

You can load segments of code from a file, or embed them directly.

```
for $i = 0$ to $100$ do
    print_number = true;
    if i is divisible by 3 then
        print "Fizz";
        print_number = false;
    end
    if i is divisible by 5 then
        print "Buzz";
        print_number = false;
    end
    if print_number then
        print i;
    end
    print a newline;
end
```

Algorithm 1: FizzBuzz

4.2 PseudoCode

5 Conclusions

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

[7] 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.

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

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

[10] 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.

[11] 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.

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

[13] F. Raynal, “A brief survey of fully homomorphic encryption, computing on encrypted data.”

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

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