INFORMATION SECURITY LABORATORY
333 Owen Hall ○ Oregon State University ○ +1 (541) 737-4861

On The Secure Evaluation Of Encrypted Polynomials Over $\mathbb{Z}/n\mathbb{Z}$

DRAFT

Colin W. van Dyke
vandyke@ece.orst.edu

Abstract

An interesting application of cryptography is in the reduction of software piracy and the protection of intellectual property within executable code. Many techniques have been developed for software protection, none of which have provable security and are often circumvented relatively easily by a malicious party. Provably secure software protection is achievable via cryptographic methods, sometimes referred to as *Mobile Cryptography*, and is an active area of research in engineering and mathematics. Concepts and ideas in cryptographic software protection rely on fundamental ideas of abstract and boolean algebra, and the reader is therefore desired to have some experience in these areas. This paper is intended to be a thorough coverage of the current status of research in this field and to be used as a foundation for any interested in pursuing research in this area. It introduces the reader to the fundamental theory required for computing with encrypted functions, and provides protocols which allow for the secure evaluation of polynomials over the algebraic ring $\mathbb{Z}/n\mathbb{Z}$, where n is smooth. We will also address the benefits and shortcomings in current research, and identify future directions necessary to make the evaluation of encrypted functions feasible for Boolean circuits, which would theoretically allow for the cryptographic protection of all software.

1 Introduction

Often in practice the real world of technological advancement is simply a cat-and-mouse game between two parties, with one edging above the other until the converse is occurs. Nowhere is this more evident than in the world of software piracy and intellectual property theft. The software vendors are constantly developing methods of protecting their software from illegal distribution and their algorithms from malicious decompilation, while pirates as well as driven intellectual property thieves work to keep ahead of the protection mechanisms. With the constant and rapid advancement of computing technology, and the growing market for software vendors, it is increasingly necessary to develop stronger protection mechanisms to reduce revenue and property loss. Oftentimes, protection mechanisms rely on the concept of 'security through obscurity', which is heavily frowned upon as inherently insecure due to the fact that there is no concrete, mathematical proof of its strength against attack. Therefore, it is necessary to build up a provably secure foundation for the protection of software against both piracy and decompilation.

The concept of *secure circuit evaluation* was first studied by Yao in [yao82]. A two-player protocol for *secure circuit evaluation* was first introduced in [abadi90]. Abadi and Feigenbaum were able to effectively present a secure method for two parties to hide information from each other while still computing the desired results, but their method relied on a high number of communication rounds between the two parties which has made it highly unpopular in practice. Moreover, their protocol dealt with the concept of *computing with encrypted data* (CED) wherein the actual data being processed was encrypted. A notable advancement pertains to the notion of *computing with encrypted functions* (CEF), where rather than evaluating encrypted data the functions themselves are encrypted such that they maintain their executable properties. This concept is often called *function hiding*, and is the primary emphasis of this paper.

The general goal of function hiding is twofold [loureiro99]:

- algorithm confidentiality (i.e. concealing the internal behavior of a program), and
- integrity of execution (i.e. if an attacker cannot derive the algorithm of the program, then he is unable to find the best way of changing it to his benefit, as in a pirate attempting to circumvent protection mechanisms).

The remainder of this paper is as follows: section 2 introduces the concept of evaluating encrypted functions along with the necessary theoretical prerequisites; section 3 introduces a protocol for the secure evaluation of a special class of polynomials over rings; and section 4 presents the results of the work and areas of future research.

2 Evaluating Encrypted Functions

The evaluation of encrypted functions (EEF) seems to be the most promising method of code protection. It allows for a vendor to distribute an encrypted, yet executable, form of their software program which is provably secure against decompilation and piracy. In this section, we will build a theoretical foundation for and introduce the necessary concepts for evaluating encrypted functions.

2.1 History

The notion of evaluating encrypted functions is the logical extension of the work by Abadi and Feigenbaum in [abadi90]. The most considerable contribution to this work came from Sander and Tschudin in [tmc] while at the International Computer Science Institute in Berkeley, CA. In their work they defined the basic prerequisites and foundations for computing with encrypted

functions, as well as the basic two-party protocol for such computations. We will now examine these foundations.

2.2 Prerequisites

In order for us to successfully evaluate an encrypted function its encrypted form must be executable. The general problem of *evaluation of encrypted functions* (EEF) can be described as follows [sanderXX]:

> Alice (the originating host) has an algorithm to compute function f. Bob (the remote host) has an input x and is willing to compute $f(x)$ for her, but Alice wants Bob to learn nothing "substantial" about f. Moreover, Bob should not need to interact with Alice during the computation of $f(x)$.

The general protocol for EEF is as follows [sanderXX]:

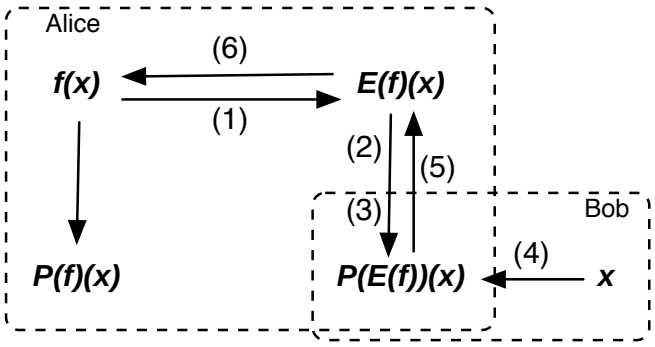

Figure 1: Protocol for Evaluating Encrypted Functions

(1) Alice encrypts f.
(2) Alice creates a program $P(E(f))$ which implements $E(f)$.
(3) Alice sends $P(E(f))$ to Bob.
(4) Bob executes $P(E(f))$ at x.
(5) Bob sends $P(E(f))(x)$ to Alice.
(6) Alice decrypts $P(E(f))(x)$ and obtains $f(x)$.

The above protocol has an optimal number of communication rounds in that it has no interactions between the two parties other than the exchange of the program and the resulting value at the end [sanderXX]. In order to realize such a scheme, some desirable properties of an encryption system must be enforced. In the next section, we will explore those desirable properties.

2.3 Homomorphic Encryption Schemes

Our goal is to map a cleartext program P to an encrypted program P_E such that Alice can recover $P(x)$ from $P_E(x)$ for some unknown input x of Bob. The encrypted program can not just be any data stream but must be an executable program. As a result, most of the ordinary data encryption techniques can not be applied. Furthermore, we want the cleartext program and its encrypted form to be "compatible" with each other[sander97]. In 1978, Rivest et al. [rivestHES] identified that to process encrypted data the encryption scheme must have certain *homomorphic* properties. Homomorphism is a fundamental concept of group theory, and commonly refers to the mapping of one algebraic system to a like algebraic system that preserves

structure [Herstein64]. A formal definition is as follows:

Definition: Homomorphisms
A mapping ϕ from a group G to a group G' is said to be a *homomorphism* if for all $a, b \in G$, $\phi(ab) = \phi(a)\phi(b)$.

It is important to note that on the left side of the previous relation ($\phi(ab)$), the product ab is computed in G using the product of elements of G whereas on the right side of this relation ($\phi(a)\phi(b)$), the product is that of elements in G' . This notion that $\phi(x + y) = \phi(x) + \phi(y)$ is often referred to as a "compatability" requirement. That is, $\phi(x + y)$ is compatible with $\phi(x) + \phi(y)$. At this point, it may be helpful to look at an example, which will illustrate the fundamental idea of homomorphism in a familiar setting:

Example:
Let G be the group of integers under addition and let $G' = G$. For the integer $x \in G$ define ϕ by $\phi(x) = 2x$. That ϕ is a homomorphism then follows from $\phi(x+y) = 2(x+y) = 2(x) + 2(y) = \phi(x) + \phi(y)$.

For more on group theory and homomorphism, the reader is referred to [Herstein64].

2.3.1 Homomorphism and EEF

It is now important to show how homomorphic encryption schemes enable the realization of EEF. In 1991, Feigenbaum and Merritt in [cite] extended the work of Rivest et al. in [rivestHES] and asked more specifically:

Is there a public-key encryption function E such that both E(x + y) and E(xy) are easy to compute from E(x) and E(y)?

It can be readily identified that an encryption function having the above properties allows one to evaluate polynomial expressions in the encrypted data without revealing the input and the result. In addition, we can observe that for computing with encrypted polynomials, it is not necessary to satisfy the multiplicative property. Rather, it is sufficient to support addition and *mixed multiplication* . From this we can formulate the following definition [sanderXX]:

Definition: *Let R and S be rings. We call an encryption function* $E : R \to S$
- *additively homomorphic* if there exists an efficient algorithm PLUS to compute *E(x + y)* from *E(x)* and *E(y)* that does not reveal x and y.
- *mixed multiplicatively homomorphic* if there exists an efficient algorithm MIXED-MULT to compute *E(xy)* from *E(x)* and y that does not reveal x .

We can observe that the process of MIXED-MULT is implied through PLUS within $\mathbb{Z}/n\mathbb{Z}$ cryptosystems. That is, we can obtain *E(xy)* by simply adding *E(x)* with itself y times. Therefore, any additively homomorphic scheme on $\mathbb{Z}/n\mathbb{Z}$ is also mixed-multiplicatively homomorphic. Using the above properties, it can be readily seen that an encryption scheme satisfying them would, theoretically, allow for EEF. The main problem is in identifying such a scheme, which has proven fruitless until recently.

2.3.2 Algebraic Homomorphic Encryption Schemes

From a cryptographic viewpoint, the mathematical "compatibility" requirement by using homomorphisms may be too strong. Recall that the map we use to transform a cleartext program into an encrypted program should be hard to invert for an adversary. The problem is that there may not be enough mathematical homomorphisms available as encryption functions or they may be

too easy to invert and therefore not suitable for our application. Consider, for example, the ring $\mathbb{Z}/n\mathbb{Z}$ where the only functions that satisfy the additively homomorphic requirement are linear ($x \to cx$) which are insecure. However, the required "compatibility" is considerably weaker in a computational framework than in the mathematical framework. Instead of requiring for a map between groups that $\phi(x + y) = \phi(x) + \phi(y)$, it is for computational purposes sufficient that $\phi(x + y)$ can be efficiently computed from $\phi(x)$ and $\phi(y)$ [sander97].

There are already schemes on $\mathbb{Z}/n\mathbb{Z}$ that enable one to compute *E(x + y)* from *E(x)* and *E(y)*. Two examples are the Naccache-Stern public key encryption function [cite] and Ferrer's "privacy homomorphism" [cite]. The one problem is that they are infeasible for computing with encrypted functions. The first approach is computationally infeasible because in order to guarantee the correctness of the results the number of calls to PLUS needed to perform CEF require the system paramenter p to be exponentially large. The Ferrer privacy homomorphism is multiplicatively homomorphic, but not mixed-multiplicatively homomorphic. The encryption function E would have to be published in order to perform CEF. However, the security of this scheme relies on the secrecy of E , thus making it unacceptable [sander97].

Alternatively, one can use a scheme based on exponentiation as follows:

> The exponentiation map $E : \mathbb{Z}/(p-1)\mathbb{Z} \to \mathbb{Z}/p\mathbb{Z}, x \to g^x$, for a prime p and a generator g of $(\mathbb{Z}/p\mathbb{Z})^x$ is additively homomorphic: the function PLUS is the simple multiplication because *E(x + y) = E(x)E(y)*. To recover x from *y := E(x)* one has to solve the discrete logarithm problem $y = g^x$ which is believed to be hard. Alice chooses a prime p such that the discrete logarithm problem is easy to solve (e.g., if *p-1* has only small prime factors she can use the Pohlig-Hellmann algorithm for computing discrete logarithms efficiently). She further chooses a generator g of $(\mathbb{Z}/p\mathbb{Z})^x$ which she keeps secret. Given this additively homomorphic encryption scheme $E : \mathbb{Z}/n\mathbb{Z} \to \mathbb{Z}/m\mathbb{Z}$ one can realize CEF for polynomials [sander97].

The security of this scheme relies on the secrecy of g . An additively homomorphic encryption scheme based on discrete logarithms which does not have this shortcoming (i.e. which can be published), was developed by Lipton and Sander in [cite]. Additionally, their scheme is probabalistic, which significantly reduces the information leakage about the original polynomial [sander97]. However, it is somewhat long and detailed. Alternatively, we will look at a useful homomorphic encryption scheme in the next section which we can use as the foundation for our protocol later in the paper.

2.3.3 A (useful) additively homomorphic encryption scheme

In a practical sense, we can rely on the Goldwasser-Micali encryption scheme that is, when applied to a one-bit message, additively homomorphic on $\mathbb{Z}/2\mathbb{Z}$. In Goldwasser-Micali, Alice's private key is two large primes p and q and her public key is the modulus $n = pq$ along with a quadratic non-residue $y \pmod n$ with jacobi symbol 1 [sandercodeprotection].

The main ideas of Goldwasser-Micali are concerned with whether, for a given number a, there is x with $x^2 \equiv a \pmod n$. Such values of a are called residues. Their encryption scheme $E : \mathbb{Z}/2\mathbb{Z} \to \mathbb{Z}/m\mathbb{Z}$ is based upon the following lemmas:

Lemma 1: *If a,b are residues, then $a \cdot b$ is a residue. If a is a residue and b is not, then $a \cdot b$ is not a residue.*

Lemma 2: *a is a residue mod n iff it is a residue mod p and a residue mod q.*

Lemma 3: *Let $h = \frac{p-1}{2}$. If a is a residue mod p, $a^h \equiv 1 \pmod p$. If a is not a residue,*

$$a^h \equiv -1.$$

These imply that, if p and q are known, it is easy to decide whether a is a residue. The encryption scheme depends on the assumption (Quadratic Residue Assumption) that this problem is very difficult if p and q are unknown. In addition, this encryption scheme draws upon the following:

Lemma 4: *Half of the numbers from 1 to (p - 1) are residues mod p. Take the numbers from 1 to n and leave out those divisible by p or by q. Divide the remaining (p-1)(q-1) numbers into four groups according to whether they are residues or not modulus p and also modulus q. There are* $\frac{(p-1)(q-1)}{4}$ *numbers in each group.*

Numbers which are not residues mod p and also not residues mod q are called *non-residues*. For example, if $p = 5$ and $q = 7$, the residues mod 35 are 1, 4, 9, 16, 29, 11 ($29 \equiv 8^2$, $11 \equiv 9^2$; note that we did not include 25 and 14, which are divisible by p and q). The non-residues must be congruent to 2 or 3 mod 5 and 3, 5 or 6 mod 7, so they are 17, 12, 27, 3, 33 and 13. The encryption scheme is concerned with the union of the set of residues and non-residues, denoted as $\mathcal{Z}_n^1$. As exactly half of the members of $\mathcal{Z}_n^1$ are residues, just saying that a number is a residue is only correct half of the time. The QRA states that no algorithm that runs in a reasonable amount of time will do better than this. In addition to announcing n, the entity receiving messages announces one non-residue y, as previously stated. To send a sequence of binary numbers, the sending entity converts them as follows: for each number in the sequence, an x is chosen *randomly*. A zero value is converted into $x^2 \bmod n$ (a residue) and a one value is converted into $yx^2 \bmod n$ (a non-residue). Based on lemma 4, each value zero or one in the sequence can be converted based on the choice of x into one of $\frac{(p-1)(q-1)}{4}$ different numbers. For instance, if the message is of length 500 bits and $p, q \approx 10^{100}$, the message can be encoded into $\frac{1}{4} \cdot 10^{100000}$ different possible ciphertexts. By lemma 1, zeros are converted to residues and ones are converted to non-residues. As the receiving entity knows p and q, they can efficiently decode the message using lemmas 2 and 3 [citeweb].

The Goldwasser-Micali encryption scheme is additive because the encrypted sum of two values x and y is obtained by multiplying their encrypted values modulo n (i.e., $E(x+y) = E(x)E(y)$) [sandercodeprotection]. A better scheme which is homomorphic for rings $\mathbb{Z}/n\mathbb{Z}$, n smooth and ≥ 2, was identified by Lipton and Sander in [cite].

Proposition: *The Goldwasser-Micali encryption scheme is additively homomorphic on* $\mathbb{Z}/2\mathbb{Z}$. *For this encryption function E we have E(a + b) = E(a)E(b).*

Proof: We define the canonical isomorphism $\psi : \mathbb{Z}/N\mathbb{Z} \to \bigoplus_{i=1}^{k} \mathbb{Z}/N_i\mathbb{Z}, x \mapsto (x \bmod N_1, ..., x \bmod N_k)$. So, an element $x \bmod N$ is a residue in $\mathbb{Z}/N\mathbb{Z}$ iff $x \bmod N_i$ is a residue in $\mathbb{Z}/N_i\mathbb{Z}\ \forall\ i$. In the Goldwasser-Micali scheme, we have $E_i : \mathbb{Z}/2\mathbb{Z} \to \mathbb{Z}/N_i\mathbb{Z}$. Abstractly, we can identify the new encryption process of an element $a = (a_1, ..., a_k)$ under E with the process of first encrypting a_i with E_i by the value $Ei(a_i)$ in $\mathbb{Z}/N_i\mathbb{Z}$ and then combining these values together via the Chinese Remainder Theorem. The proof continues formally [sandercodeprotection]:

$$\begin{aligned} E(a)E(b) &= \psi^{-1}(\psi(E(a)E(b))) \\ &= \psi^{-1}[(E_1(a_1), ..., E_k(a_k))(E_1(b_1), ..., E_k(a_k))] \\ &= \psi^{-1}[(E_1(a_1)E_1(b_1), ..., E_k(a_k)E_k(b_k))] \\ &= \psi^{-1}[(E_1(a_1 + b_1), ..., E_k(a_k + b_k)] \\ &= E(a + b) \end{aligned}$$

Q.E.D.

Now that we have identified a useful additively homomorphic encryption scheme, we can continue to evaluate the necessary steps to achieve EEF for the important class of rings $\mathbb{Z}/n\mathbb{Z}$.

3 Secure Evaluation of Encrypted Polynomials over $\mathbb{Z}/n\mathbb{Z}$

In this section we will present a protocol introduced in [sanderXX] that allows for evaluation for encrypted polynomials (EEP) over rings $\mathbb{Z}/n\mathbb{Z}$ which executes non-interactively. The one restriction of this protocol is that it considers only polynomial/rational functions, which limits its real-world applicability. The necessary steps for widespread application of function hiding are covered in section 4.

3.1 A Protocol for Evaluating Encrypted Polynomials over $\mathbb{Z}/n\mathbb{Z}$

Based on the concepts outlined in section 2.3, we can give a protocol for solving EEF for encrypted polynomials for the important class of rings $\mathbb{Z}/n\mathbb{Z}$. The protocol is outlined as follows:

Let $E : \mathbb{Z}/n\mathbb{Z} \to S$ be an additively homomorphic encryption scheme. We can implement non-interactive EEF for polynomials $p \in \mathbb{Z}/n\mathbb{Z}[X_1, ..., X_s]$ with E.

Proof: Note that every additively homomorphic encryption scheme on $\mathbb{Z}/n\mathbb{Z}$ is also mixed-multiplicative as shown in section 2.3.1. Let p be the polynomial $\sum a_{i_1...i_s} X_1^{i_1} ... X_x^{i_s}$

(1) Alice creates a program *P(X)* that implements p in the following way:
- each coefficient $a_{i_1...i_s}$ of p is replaced by $E(a_{i_1...i_s})$,
- the monomials of p are evaluated on the input $x_1, ..., x_s$ and stored in a list $L := [..., (x_1^{i_1} ... x_s^{i_s}), ...]$,
- the list $M := [..., E(a_{i_1...i_s} x_1^{i_1} ... x_s^{i_s}), ...]$ is produced by calling the function MIXED-MULT for the elements of L and the coefficients $E(a_{i_1...i_s})$,
- the elements of M are added up by calling PLUS.

(2) Alice sends the program P to Bob.

(3) Bob runs P on his private input $x_1, ..., x_s$ and obtains $P(x_1, ..., x_s)$.

(4) Bob sends the result $P(x_1, ..., x_s) = E(p(x))$ back to Alice.

(5) Alice decrypts the result by applying E^{-1} and obtains $p(x)$.

Thus the security of the program P and the input $x_1 ... x_s$ remains intact. Note the minimal number of communication rounds between the two parties. The realization of the aforementioned protocol allows for a number of security enhancements to the executing code. Most notably, the client (Bob) is forced to have their result decrypted by the server (Alice), which allows for strict enforcement of program distribution. If a client which is not authorized to execute the program requests for the decryption of their generated output, the server can simply refuse the request. This makes the process of pirating software much more difficult if not impossible. In addition, the growing trend of 'software as a service' can be realized in that the server can distribute the program P for free, and charge for the decryption of the generated results on a per-use basis [sanderprogramsecurity]. We will now look at the security of EEP, and what if any, information is leaked in the process.

3.2 Security of EEP over $\mathbb{Z}/n\mathbb{Z}$

It is necessary for us to analyze the security of this protocol and the information, if any, that is leaked. The aforementioned protocol leaks information about the coefficients of the unencrypted polynomials, also known as the *skeleton* of f . Note that, depending on the value of n used, the leakage may reveal all information needed to reconstruct the original polynomial. Take, for instance, an encryption function $E : \mathbb{Z}/2\mathbb{Z} \to \mathbb{Z}/m\mathbb{Z}$, where $m = 2$; given the elements of the

message space {0,1}, any non-zero coefficient is easilly obtained as 1.

A formal analysis of the security of our protocol can be explained as follows: *E(f)* can be easily constructed from the program P . A set of monomials $\mathcal{U}$ containing the monomials with non-zero coefficients of f is inherently revealed by our protocol. This set $\mathcal{U}$ is called the *skeleton* of f . If we think of the monomials of $\mathcal{U}$ as given in a certain order we can identify f with the list of coefficients of f that we call m . We can also identify *E(f)* with its list of coefficients *E(m)* . So, the message spaces M_n consist of n-tuples of elements from $\mathbb{Z}/n\mathbb{Z}$ on which our encryption function E operates componentwise. We can now say that if our protocol does not leak any information about f except its skeleton, it does not leak any (useful) information when applied to elements of the message spaces M_n. Given this information, we can say that our encryption scheme is *polynomial time indistinguishable* [sanderXX].

Definition: Polynomial Time Indistinguishable
An encryption function is called *polynomial time indistinguishable* if, when applied to elements of its message space, it does not leak any important information.

Theorem: *Let E be an additively homomorphic encryption scheme on $\mathbb{Z}/n\mathbb{Z}$. Our protocol realizes non-interactive EEF for polynomials $f \in \mathbb{Z}/n\mathbb{Z}[X_1, ..., X_s]$, and is polynomial time indistinguishable. Given these properties, no information is leaked about f except its skeleton.*

The additively homomorphic encryption scheme covered in section 2.3.3 is polynomial time indistinguishable under the hardness of the Power Residue Hypothesis, which is a generalization of the Quadratic Residue Hypothesis to residues of higher degree. So, if one allows the leakage of the skeleton, the problem of non-interactive EEF for polynomials in $\mathbb{Z}/n\mathbb{Z}[X_1, ..., X_s]$ with n smooth, is solved [sanderXX].

It is important to note that there are special cases where the leakage of the skeleton, even for a polynomial time indistinguishable encryption scheme, may allow for an adversary to recover the full polynomial f . As an example, assume that the adversary knows the encrypted function f is an RSA function ($x \to x^d$), which has only a single non-zero coeffecient. If the input/output pairs (*x,f(x)*) are known, then an adversary can find f and consequently d by evaluating every element of $\mathcal{U}$ on the inputs x , with difficulty depending on the computational resources of the attacker [sanderXX].

It is also important to note a special attack against the previously mentioned protocol called the *coefficient attack*. In this attack, Bob can recover the plaintext coeffecients of f even though they are not explicitly leaked by our protocol. The attack proceeds as follows: Bob, instead of sending his program output to Alice for decryption, sends her the encrypted coefficients of *E(f)*. According to the protocol, Alice would decrypt them and send Bob the very information she was attempting to hide. This is a general problem that EEF schemes have to face if the encryption of some "data" about the function and the output of the encrypted function are encrypted by the same scheme. This is defeated via a more complicated protocol outlined in [sanderinfomrationhiding].

4 Conclusions and Future Work

Using the above scheme, it would seem that the problem of EEF for polynomials is solved. However, this covers just a small subset of the problem. Theoretically, if one were able to obtain promising results for boolean circuits, the problem of EEF would be solved. To this date, this has proven to be a much more difficult problem.

In [abadi90], Abadi and Feigenbaum described a protocol in which one could securely evaluate a Boolean circuit in encrypted data. They further reduced EEF for boolean circuits to *processing encrypted data* by representing the Boolean circuit that is to be hidden as data fed to a universal Boolean circuit. This reduction, for practical application, is infeasible. It is true that a Boolean circuit $f : \mathcal{B}^n \rightarrow \mathcal{B}$ can be realized by substituting in a universal Boolean circuit $U : \mathcal{B}^n \rightarrow \mathcal{B}$ certain variables specifying the function f. However, for a function $U : \mathcal{B}^n \rightarrow \mathcal{B}$ that is universal for the class of boolean circuits $\{f : \mathcal{B}^n \rightarrow \mathcal{B}\}$, we have that $m \geq \frac{2^n}{log(2n+2)}$. Thus, we get an exponential blowup by using universal Boolean circuits [sander97]. The main question now is this: how does the above work with algebraic circuits relate to Boolean circuits? Under reasonable assumptions, every algebraic circuit on finite fields can be simulated efficiently by Boolean circuits. The converse of this is wide open and is one possible approach to CEF for boolean circuits. Shifting from a Turing machine model to Boolean circuits is not a restriction: every language in $\mathcal{P}$ (which can be recognized by a deterministic Turing machine in polynomial time), can be recognized by uniform Boolean circuits of polynomial size [citefrom sander97]. Therefore, there may be programs that cannot be efficiently simulated using algebraic circuits but which are feasible for Boolean circuits. One approach for CEF for Boolean circuits concedes that some information about the original circuit may be revealed. The requirement to hide *all* information about a Boolean circuit is too strong as it leads to the use of universal Boolean circuits which we previously identified as computationally infeasible. For many applications it may be sufficient to hide only partial information about the Boolean circuit, allowing one to circumvent the construction of a universal Boolean circuit. This area is open to research, and to this date has not been solved.

Another limitation is the fact that evaluating functions in encrypted form yields encrypted results when utilizing homomorphic properties. An interesting extension of this work would be to possibly construct a protocol such that the result produced by Bob's execution of P over his input $x_1...x_s$ produced a result such that it was in *plaintext* form, thus reducing the required communication rounds to one. Note, however, that in doing this some security provisions of EEF are nullified, such as the ability to constrain who can execute the encrypted program.

Note: The author admits that the previous information is a brief introduction to the theoretical aspects of EEF for boolean circuits. As a result, another paper will be written which is intended to be a broad coverage of the various concepts and ideas necessary for a solid theoretical foundation for research in the field of computing with encrypted functions over Boolean circuits.