Practical Applications of Homomorphic Encryption

Michael Brenner, Henning Perl and Matthew Smith
Distributed Computing Security Group, Leibniz Universitt Hannover, Hannover, Germany
{*brenner, perl, smith*}*@dcsec.uni-hannover.de*

Keywords: Homomorphic Encryption, Private Information Retrieval, Encrypted Search.

Abstract: Homomorphic cryptography has been one of the most interesting topics of mathematics and computer security since Gentry presented the first construction of a fully homomorphic encryption (FHE) scheme in 2009. Since then, a number of different schemes have been found, that follow the approach of bootstrapping a fully homomorphic scheme from a somewhat homomorphic foundation. All existing implementations of these systems clearly proved, that fully homomorphic encryption is not yet practical, due to significant performance limitations. However, there are many applications in the area of secure methods for cloud computing, distributed computing and delegation of computation in general, that can be implemented with homomorphic encryption schemes of limited depth. We discuss a simple algebraically homomorphic scheme over the integers that is based on the factorization of an approximate semiprime integer. We analyze the properties of the scheme and provide a couple of known protocols that can be implemented with it. We also provide a detailed discussion on searching with encrypted search terms and present implementations and performance figures for the solutions discussed in this paper.

1 INTRODUCTION

Fully homomorphic encryption fired many people's imagination in the field of distributed computing security. Architectures have been proposed and many application scenarios have been identified that can benefit from FHE. Encrypted online storage, secure delegation of confidential computation and even privacy for searching the web: the Cloud was about to turn secure. Unfortunately, all implementations of fully homomorphic encryption schemes showed, that this technique is still much too slow for practical applications.

The most important property of FHE is unlimited chaining of algebraic operations in the cipherspace, which means that an arbitrary number of additions and multiplications can be applied to encrypted operands. To achieve this, an FHE scheme must provide a mechanism to reduce the *noise* of cipher values, because these schemes are based on a slightly inaccurate representation of the plaintext values. Every single operation on a ciphertext causes even lower accuracy and eventually, the ciphertext can no longer be properly decrypted.

This paper focuses on somewhat homomorphic encryption, where no re-encryption is required but only a limited number of operations is possible. We discuss an algebraically homomorphic scheme and show for a couple of problems of practical relevance, how these can be solved by a surprisingly small number of operations on encrypted values. We exemplarily discuss solutions to the Millionaires' Problem, one-round Oblivious Transfer and oblivious memory access based on the homomorphic scheme. We also discuss searching over encrypted data with encrypted search terms. Since this is a very important operation in distributed environments, we present our solution to this in more detail in a separate section. We show a delegation scheme, where the remote party operates with encrypted arguments on public data and generates encrypted results.This is useful when posing search requests to a public database while maintaining confidentiality of request and response.

The paper is structured as follows: Section 2 gives a summary of the current status of homomorphic cryptography. Section 3 outlines a somewhat homomorphic encryption scheme and analyses its properties in detail. Section 4 introduces a selection of algorithmic primitives that can be secured using our homomorphic scheme. Section 5 introduces a constant-depth approach to encrypted searching. In Section 6 we present details and performance figures of our implementation. Section 7 concludes the paper.

2 RELATED WORK

Since the breakthrough work of (Gentry, 2009), a number of similar approaches to fully homomorphic encryption appeared, like (Smart and Vercauteren, 2010) or slightly different approaches like (Brakerski and Vaikuntanathan, 2011). Performance figures of actual implementations (Brenner et al., 2011) and applications of FHE show that these systems can be used for small problems only. Due to the computational overhead of current FHE schemes, the question arises, if the underlying SHE schemes can also be used for more practical homomorphic encryption. Recent proposals, like (Naehrig et al., 2011) follow this approach. However, the fully homomorphic encryption is still subject to progress in terms of new considerations of hardness assumptions (Stehl and Steinfeld, 2010) or conceptual simplicity (Coron et al., 2011).

There are different paradigms for secure delegation of computation like *secure function evaluation* (SFE) mostly based on *Yao's Garbled Circuits* (Yao, 1982) and extensions by (Malkhi et al., 2004) or (Kolesnikov et al., 2009b). Garbled circuits have also been combined with homomorphic encryption by (Gentry et al., 2010) and (Kolesnikov et al., 2009a) to overcome their inherent disadvantage of of being limited to static one-pass boolean circuits.

A theoretical approach to achieve privacy of memory access patterns and algorithm execution in a special type of Turing Machines is the Oblivious Random Access Machine (ORAM) by (Goldreich, 1987) (Goldreich and Ostrovsky, 1996). There are recent proposals to reduce the complexity of ORAMs by Pinkas et al. (Pinkas and Reinman, 2010) and further developments towards practical applications by (Damgrd et al., 2011) and (Goodrich and Mitzenmacher, 2011). Section 4 outlines how to achieve oblivious memory access with our scheme.

3 A SOMEWHAT HOMOMORPHIC ENCRYPTION SCHEME

This section describes the encryption scheme and its properties correctness, security and compactness. Our somewhat homomorphic encryption scheme $\mathfrak{E}$ depends on the following parameters:

- the security parameter λ,
- the bit length η of a cipher's initial noise,
- the modulus p which is a large prime integer of order 2^{λ},
- the bit length ρ of the message space, defined as $\lambda - \eta$.

3.1 The Basic Construction

Our scheme $\mathfrak{E}$ is defined as a tuple $\{P,C,K,E,D,\oplus,\otimes\}$ where the elements denote the following:

P is the plaintext space and contains elements from $\mathbb{N}^+$ limited by the prime integer p of order 2^{λ} such that for two plaintext operands $a,b \in \mathbb{N}^P, a \cdot b < p$ and $\mathbb{N}^P := \{x | x < 2^{\eta}\}$.

C is the ciphertext space and contains elements from $\mathbb{N}^+$.

K is the key generator. The secret key is a large prime integer p, the auxiliary compression argument is d with $d \leftarrow 2s + rp$ with $r \in \mathbb{N}^+$ and $s \in \mathbb{N}^C$ with $\forall x \in \mathbb{N}^C, \forall y \in \mathbb{N}^P, 2x < y$ (see *compactness*).

E is the encryption function. We encrypt a bit value b by picking an integer a with $a \equiv b \ mod\ 2$ and adding a random even or odd multiple of the prime modulus, such that $a' = a + (rp)$. If r is composite, it must contain at least one large prime factor of order 2^{λ}. It is mandatory that $a \in \mathbb{N}^+$, i.e. the encryption must add *noise* (see below).

D is the decryption function. The decrypted result is the remainder of a ciphertext modulo the prime key p: $a = a' \ mod\ p$.

$\oplus$ is the addition in ciphertext space. Due to the cipher structure, the addition is performed as an ordinary arithmetic addition. The scheme is mixed additive.

$\otimes$ is the multiplication in ciphertext space. Like the addition, the multiplication is performed as an ordinary arithmetic multiplication. The scheme is mixed multiplicative.

In this scheme, the positive plaintext value is also the *noise* for the ciphertext because it additively interferes with the product of the prime factors that encrypt it. In order to obtain a *probabilistic* encryption scheme, we perform parity ($mod\ 2$) arithmetics, i.e. a plaintext bit is encoded in a random integer of the same parity. Notice that the encryption does not reveal the parity of the plaintext integer, because the encryption function E picks at random even or odd multiples r of the key p. This effectively allows us to hide the parity of the plaintext and to encode binary information.

3.2 Correctness

We show the correctness of the encryption and decryption, as well as the homomorphic operations in the following lemmas, assuming that $p \in \mathbb{N}^+$ be a

large prime integer as a secret key and a and b be two arbitrary positive integers with $a,b \ll p \in \mathbb{N}^+$.

Lemma 1. *The encryption scheme $\mathfrak{E}$ is mixed additive and the addition is correct if $a+b<p$.*

Proof. We perform the encrypted addition as $(a' \oplus b')$ which extends to $(a' \oplus b') = (a+r_1p)+(b+r_2p) = a+b+(r_1+r_2)p$ and decrypted *mod* p yields $(a+b)$. The mixed additive operation is defined as $a' \oplus b = (a+rp)+b = a+rp+b \ mod \ p = a+b$. □

Lemma 2. *The encryption scheme $\mathfrak{E}$ is mixed multiplicative and the multiplication is correct if $a*b<p$.*

Proof. We perform an encrypted multiplication as $a' \otimes b' = (a+r_1p)(b+r_2p) = ab+a(r_2p)+b(r_1p)+(r_1r_2)p^2 \ mod \ p = (ab)$. The mixed multiplicative operation is defined as $a' \otimes b = (a+rp)b = ab+brp \ mod \ p = ab$. □

Lemma 3. *The encryption scheme $\mathfrak{E}$ is capable of computing a sequence of at least $log_2\rho - log_2\eta$ arithmetic operations.*

Proof. The choice of the factors r and p also defines the message space, such that the bit length of the message space ρ is $\lambda-\eta$. The noise of a cipher grows by at most one bit per addition (the carry), so the minimum number of additions is ρ. The minimum number of subsequent multiplications is defined as $log_2\rho - log_2\eta$, assuming that ciphers with the maximum noise are multiplied, since this can produce at most $\eta 2^n$ bits of noise after n multiplications. □

3.3 Security

As shown above, our simple encryption scheme is algebraically correct. This subsection discusses confidentiality of the encrypted data and the security of the encryption scheme. The results of this subsection lead to the choice of appropriate parameters. The attack model focuses an attacker who is in possession of a ciphertext. Since we apply symmetric encryption only, the attacker has no public key. By reduction to integer factorization we show the security of the encryption scheme against the computation of the secret key from a ciphertext.

Lemma 4. *Let the parameters $(p,q) \in \mathbb{N}^+$ be of order 2^λ. Any Attack $\mathcal{A}$ running in time polynomial in λ against the encryption scheme can be converted into an algorithm $\mathcal{B}$ for solving the integer factorization of any composite integer number (pq) running in time polynomial in λ.*

Proof. Consider a poly(λ) time function $\mathcal{A}_{ex}$

$$p \leftarrow \mathcal{A}_{ex}(a')$$

that efficiently computes p from any cipher a'. That means, that $\mathcal{A}_{ex}$ is able to extract p from any integer computed by a term of the form $a+pq$ for arbitrary a,p,q and thus can be applied in a function

$$\mathcal{B}_{fac}(i) : p \leftarrow \mathcal{A}_{ex}(0+i)$$

to factorize arbitrary composite integers (pq) that can be trivially expressed as $0+pq$. □

Now we show the security of the encryption scheme against an attempt of $\mathcal{A}$ to compute the plaintext value of a cipher. We achieve this by showing in addition to Lemma 4 an IND-CPA equivalent property of the scheme, which means that any two ciphertexts are computationally indistinguishable by $\mathcal{A}$.

Lemma 5. *The security of the encryption scheme is IND-CPA equivalent and the success probability $|Pr[Exp_{ind-cpa}=1]-\frac{1}{2}|$ is negligible in λ for any $\mathcal{A}$ from PPT (probabilistic polynomial time) function.*

Proof. Since $\mathcal{A}$ has no public key for encryption, we provide a probabilistic encryption oracle O_{Enc} that takes a plaintext as input and generates a ciphertext under the secret key. Consider the following experiment in the message space $\mathcal{M}$:

$$\begin{array}{ll} Exp^{\lambda}_{ind-cpa} = \{ & \\ \{m_0,m_1\} & \leftarrow \mathcal{M} \\ \quad i & \xleftarrow{R} \{0,1\} \\ \quad c & \leftarrow O_{Enc}(m_i) \\ \quad i_{\mathcal{A}} & \leftarrow \mathcal{A}_{ind-cpa}(c,\{m_0,m_1\}) \\ \text{return} & \begin{cases} 1 & if \ i = i_{\mathcal{A}} \\ 0 & otherwise \end{cases} \\ \} & \end{array}$$

The encryption function equivalent oracle O_{Enc} is defined as follows:

$$\begin{array}{ll} O^{\lambda,p}_{Enc}(m) = \{ & \\ \quad a & \xleftarrow{R} 2[2^\eta] \\ \quad r & \xleftarrow{R} [2^\lambda] \\ \quad c & \leftarrow (m \ mod \ 2)+a+rp \\ \text{return} & c \\ \} & \end{array}$$

The parity of the output of the oracle $O_{Enc}(m)$ has a discrete uniform distribution, such that $Pr[O_{Enc}(m) \equiv_2 0] = Pr[O_{Enc}(m) \equiv_2 1] = \frac{1}{2}$ for any integer $m \in \mathbb{N}^+$, i.e. the encryption function does not leak any parity information. It is sufficient to show that $Pr[O_{Enc}(m) \equiv_2 1] = \frac{1}{2}$.

$$Pr[O_{Enc}(m) =_2 1] = Pr[(a+r\cdot p \equiv_2 1]$$
$$Pr[a \equiv_2 1] \cdot \underbrace{Pr[r\cdot p \equiv_2 0]}_{=0.5} + \underbrace{Pr[a \equiv_2 0]}_{=0} \cdot \underbrace{Pr[r\cdot p \equiv_2 1]}_{=0.5}$$
$$\frac{1}{2}\underbrace{(Pr[a \equiv_2 1])}_{=1} = \frac{1}{2}$$

This leads to the conclusion that ciphertexts are indistinguishable by $\mathcal{A}$, even if he is allowed to provide known plaintext to the encryption function. □

3.4 Compactness

The length of a ciphertext grows exponential in the number of multiplications applied to it. In order to obtain *compact* ciphertexts, i.e. the length of a ciphertext is independent from the number of operations, we suggest generating an encryption of 0 as an auxiliary compression argument d that can be used in a reduction procedure to limit the size of a ciphertext. The security of this delimiter is subject to Lemma 4.

Lemma 6. *The operation $a'' \leftarrow a' \ mod \ (2s + rp)$ reduces the bit length of the ciphertext a'. The reduction is correct for any $2s \in \mathbb{N}^+ < a$, i.e. the parity of the original plaintext a encrypted in a' is preserved in a''. The length of a reduced cipher is $|2s + rp| = 2\lambda$.*

Proof. $a'' \leftarrow a' \ mod \ d \Leftrightarrow a'' \leftarrow a' - \lfloor \frac{a'}{d} \rfloor \cdot d$, $d = 2s + rp$ $\Rightarrow a'' = a' - \lfloor \frac{a + r_1 p}{2s + r_2 p} \rfloor \cdot (2s + r_2 p)$; $I : a + r_1 p < 2s + r_2 p$ $\Rightarrow a'' = a'$; $II : a + r_1 p \geq 2s + r_2 p \Rightarrow n = \lfloor \frac{a + r_1 p}{2s + r_2 p} \rfloor, n \geq 1$; $\delta = n \cdot (2s + r_2 p) = 2ns + nr_2 p$

$$\Rightarrow a'' \leftarrow a + \underbrace{\underbrace{r_1 p}_{mod \ p = 0}}_{a'} \underbrace{- \underbrace{2ns}_{even} - \underbrace{nr_2 p}_{mod \ p = 0}}_{\delta} \ mod \ p \equiv_2 a$$

The reduced result a'' decrypted *mod* p has the same parity as a. □

3.5 Choice of Parameters

As we showed above, the security of our scheme is essentially based on the common assumption, that the factorization of a large integer is hard. So the product of the prime key p and the factor r must be sufficiently large. Taking the results of the former RSA-challenge into account, we suggest factors with a size ¿512 bits. A reasonable configuration with a factor size of $\lambda = 1024$ bits and an initial noise of $\eta = 8$ bits would allow for a sequence of $log_2\lambda - log_2\eta = 7$ multiplications.

Small factors. The encryption function must not issue a ciphertext with $a = 0$ in order to avoid small factors: if it generated two different representations of 0 with $a'_1 = 0 + r_1 p$ and $a'_2 = 0 + r_2 p$, the adversary could easily generate a positive composite integer $b' = |(a'_1 - a'_2)| = |(r_1 - r_2)|p$, with $|(r_1 - r_2)|$ likely containing a small factor, assumed that r_1 and r_2 are in a similar range.

4 SIMPLE APPLICATIONS

This section describes a selection of algorithms that can be encrypted by applying the scheme $\mathfrak{E}$.

4.1 One-round k-Bit Oblivious Transfer

Bob has two arguments and wants to share one of them with Alice. Alice is allowed to pick one argument but doesn't want to reveal to Bob which one she chooses. On the other hand, Bob wants to keep the other argument secret. Using our homomorphic scheme, we can solve this problem for arbitrary k-bit Strings in a single request-reply interaction. This is the protocol:

1. Alice has $a \in \{0,1\}$ and the secret key SK
2. Bob has $m_{0,1} \in \{0,1\}^k$, two k-bit strings
3. Alice picks at random a η-bit integer $t \in \mathbb{N}^+$ such that $t \equiv a \ mod \ 2$ and encrypts $a' \leftarrow E(t)_{SK}$
4. Bob picks at random η-bit integers $m'_{i_j} \in \mathbb{N}^+$ such that $m'_{i_j} \equiv m_{i_j} \ mod \ 2$ for $i \in \{0,1\}$ and $0 \leq j \leq (k-1)$. He computes the (selected) output bits $s_j \leftarrow ((m_{0_j} \cdot (a' + 1)) + (m_{1_j} \cdot a'))$ for $0 \leq j \leq (k-1)$
5. Alice decrypts the selected bit string by computing $s_j \leftarrow D(s'_j)_{SK} \ mod \ 2$ for $0 \leq j \leq (k-1)$

The multiplicative depth of this protocol is 1. Bob determines the result value by applying a binary selector over the two bit strings, addressed by Alice's binary selection argument. He computes $s \leftarrow (m_0 \wedge \neg a) \oplus (m_1 \wedge a)$ by evaluating logic AND-operations after transferring them into an arithmetic representation.

4.2 Oblivious Memory Access

In this scenario, we have a larger number of messages to choose from, so this problem can be solved by applying a $log_2 n$-bit demultiplexer for n messages. The algorithm for this purpose is quite similar to a memory circuit: the switching function of the demux that selects a memory item $m \in \{m_0, m_1, m_2, m_3\}$ addressed by $a \in \{(00), (01), (10), (11)\}$ is given as $m = (\neg a_0 \wedge \neg a_1 \wedge m_0) \vee (a_0 \wedge \neg a_1 \wedge m_1) \vee (\neg a_0 \wedge a_1 \wedge m_2) \vee (a_0 \wedge a_1 \wedge m_3)$. Consider the following protocol:

1. Bob has 4 items $m_0..m_3 \in \{0,1\}$
2. Alice has an argument $a \in \{0,1\}^2$ and the secret key SK
3. Alice picks at random η-bit integers $t_i \in \mathbb{N}^+$ such that $t_i \equiv a_i \ mod \ 2$ for $0 \leq i \leq 2$ and encrypts each $a'_i \leftarrow E(t_i)_{SK}$
4. Bob picks at random η-bit integers $m'_i \in \mathbb{N}^+$ such that $m'_i \equiv m_i \ mod \ 2$ for $0 \leq i \leq 3$. He computes $s' \leftarrow ((a'_0 + 1) \cdot (a'_1 + 1) \cdot m'_0) + (a'_0 \cdot (a'_1 + 1) \cdot m'_1) + ((a'_0 + 1) \cdot a'_1 \cdot m'_2) + (a'_0 \cdot a'_1 \cdot m'_3)$
5. Alice decrypts the selected value $s \leftarrow D(s')_{SK}$.

This protocol has a multiplicative depth of 2 for a range of 4 items. The number of selectable items (i.e. *the memory size*) can be extended by a wider address range which leads to a higher multiplicative depth given as $log_2 n$. It can be extended to select k-bit items by arranging k instances in parallel.

4.3 The Millionaires' Problem

Alice and Bob want to know who is richer but don't want to reveal the amounts of their respective capitals. This paragraph depicts a solution to this problem using homomorphically encrypted binary adders. The protocol is as follows:

1. Alice has $a \in \{0,1\}^k$, containing a (k-1)-bit little-endian binary representation of her capital in $a_0..a_{k-2}$, $a_{k-1} \leftarrow 0$ and the secret key SK.
2. Bob has $b \in \{0,1\}^k$, like a, $b_{k-1} \leftarrow 0$
3. Alice picks at random η-bit integers $t_i \in \mathbb{N}^+$ such that $t_i \equiv a_i \ mod\ 2$ and encrypts $a'_i \leftarrow E(t_i)_{SK}$ for $0 \leq i \leq (k-1)$
4. Bob picks at random η-bit integers $b'_i \in \mathbb{N}^+$ such that $b'_i \equiv b_i \ mod\ 2$ for $0 \leq i \leq (k-1)$. He computes $b'_i \leftarrow b'_i + 1$ for $0 \leq i \leq (k-1)$, picks at random an η-bit integer $t' \in \mathbb{N}^+$ with $t' \equiv 1 \ mod\ 2$, computes $u'_0 \leftarrow b'_0 + t'$ and $c'_0 \leftarrow b'_0 \cdot t'$ and further $u'_i \leftarrow b'_i + c'_{i-1}, c'_i \leftarrow b'_i \cdot c_{i-1}$ for $1 \leq i \leq (k-1)$. Now he has the two's complement of b' in u'. He computes $s'_0 \leftarrow a'_0 + u'_0$ and $c'_0 \leftarrow a'_0 \cdot u'_0$. Finally, he computes $s'_i \leftarrow a'_i + u'_i + c'_{i-1}$ and $c_i \leftarrow (a'_i \cdot u'_i) + (c'_{i-1} \cdot (a'_i + u'_i))$ for $1 \leq i \leq (k-1)$.
5. Alice decrypts $s \leftarrow D(s'_{k-1})_{SK}$ and concludes for both parties that $a < b$ if $s \equiv 1 \ mod\ 2$ and $a \geq b$ otherwise.

Bob first computes the two's complement of his argument and adds Alice's a. The most significant bit of the solution (the sign) indicates the relation between a and b: if the solution is negative, i.e. the sign bit is 1, then obviously b is greater then a. As stated in Section 3.2, Bob injects his data unencryptedly. However, after an operation with an encrypted operand, the result contains the implicitly encrypted plaintext argument. The multiplicative depth of the protocol is 2.

5 SEARCH USING CONSTANT-DEPTH CIRCUITS

This section introduces an approach to encrypted searching using circuits of constant depth that operate on encrypted queries and data. This application of homomorphic encryption is described in more detail than the simple use cases in Section 4, because remote encrypted search is one the basic operations in a distributed setting with a wide range of possible applications. Therefore we split the search scenario in the subtypes *exact search* and *fuzzy search*.

5.1 General Approach

The following components are part of the system: The *Input* This can be a list of words or a stream of data. The *Encrypted circuit* For each word in the input data, an encrypted homomorphic circuit is executed, which calculates one element in the output vector, based on the encrypted search term. An *Encrypted search term* The search term is encrypted and used by the homomorphic circuit. The *Output indicator vector* In the case of an exact search, the output vector is a bit-vector containing a 1 if the search term matches and a 0 else. Yet, it is also possible to do an inexact fuzzy search with this approach. In this case, the output vector would contain the rank of the match. Formally, one cell of the output vector can be written as the value of a function

$$\varphi: \begin{array}{c} \Sigma^* \times \mathfrak{C}^* \to \mathfrak{C}^* \\ (word, searchterm) \mapsto \mathcal{C}(word, searchterm) \end{array} \tag{1}$$

Since the definition of φ is independent of the other words in the database, the output can efficiently be computed in parallel. Another advantage is that the circuit $\mathcal{C}$ depends only on the encrypted search term. Especially, the depth is constant with respect to the size of a database entry. See 5.2.3 for a reduction of the output size.

5.2 Exact-match Search

Building on the general approach for searching introduced above, one can easily generate a circuit which executes an exact-match search. The circuit should have the following property:

$$\varphi(word, searchterm) = \begin{cases} 1 & word == searchterm \\ 0 & \text{else} \end{cases} \tag{2}$$

which translates to a character-level comparison

$$\varphi(word, searchterm) = \bigwedge_{i=0}^{|searchterm|-1} (word_i =_c searchterm_i) \tag{3}$$

The two things that need further specification are a) the concrete implementation of the character-level comparison $=_c$ and b) the (common) case when $|searchterm| \neq |word|$.

5.2.1 Character-level Comparison

Let Σ be the finite alphabet for the word database as well as the search term. Then there exists a strict total order $<$ on Σ. In case of $\Sigma = \{a,\ldots,z,A,\ldots,Z\}$ this may for example be the dictionary order. The bijective funciton $bin(\sigma)$ then returns the binary value of the positon of σ and is defined as

$$bin: \begin{array}{c} \Sigma \to \{0,1\}^{\lceil \log_2 \Sigma \rceil} \\ \sigma \mapsto \begin{cases} 0 \; if \; \sigma = \min(\Sigma) \\ \left(\max_{\sigma' < \sigma}\{bin(\sigma')\} + 1\right)_2 \text{ else.} \end{cases} \end{array} \tag{4}$$

Finally, the relation $=_c$ is defined as

$$=_c: \quad \left\{\{\sigma_1,\sigma_2\} \middle| \bigwedge_{i=0}^{\lceil \log_2 \Sigma \rceil - 1} bin(\sigma_1)_i \oplus bin(\sigma_2)_i \oplus 1\right\} \quad . \tag{5}$$

The depth of the resulting circuit from Equation 5.2 combined with the circuit of $=_c$ from Equation 5 is

$$\begin{array}{c} \text{depth}(\varphi(word, search)) = \\ 2 + \lceil \log_2 \Sigma \rceil + \lceil \log_2 |search| \rceil \in O(\log_2 n) \end{array} \tag{6}$$

Note that the computation of *bin* is just used to transform a character to its binary representation and therefore is not part of the circuit itself. Rather than computing the implicit function given in Equation 4 one would use a code table like ASCII for an efficient transformation.

5.2.2 Padding

In Equation 5.2 the implicit requirement was $|word| \geq |searchterm|$ introduced. In a general search, however, the lengths of the search term and the words may very well be different.

Definition 5.1. *For an alphabet* Σ *and a* padding character $\square$ *with* $\Sigma \cap \square = \emptyset$ *let the* alphabet with padding *be* $\bar{\Sigma} = \Sigma \cup \{\square\}$. $\square^n$ *is a shorthand for* $\underbrace{\square \cdots \square}_{n \text{ times}}$*, an n-character padding.*

Additionally, some random padding may be required for the *searchterm*, because the length of the searchterm is a public information, even if the searchterm itself is encrypted. Also, the length of the searchterm is revealed in the depth of the circuit. In order to hide the actual length the client appends a random number of padding characters to the searchterm prior to encrypting it. The server only evaluates the search function for database items shorter than the searchterm. Despite this modifications, the depth of φ is still in $O(\log n)$, as the only modifications made add a constant to Σ and $|searchterm|$ in Equation 6.

5.2.3 Reducing the Output Size

The output vector of the exact search has a linear dependency in the size if the database that is operated on. Under the assumption of a total strict order of the database *elements*, we have that the search yields at most one positive indicator (or none, if no entry in the databases matches the search term). Following the technique, outlined in the description of the oblivious memory access in 4.2, the reduced output vector for the indicators $I_0..I_n$ can be computed as $\sum_{i=0}^{n} I_i$. This reduces the output space complexity to $O(1)$, i.e. to one single element which contains an encrypted 1, if the search term was found and an encrypted 0 otherwise. To indicate the position of the match in the database, the return value can be computed as $\sum_{i=0}^{n} I_i * i$. The result can be returned as a $log_2 n$ bit element with an encrypted binary representation of the match index i. The reduction cost of the binary index generation is a multiplicative depth which is increased by 1.

5.3 Fuzzy Search

While Section 5.2 provided some fundamental construction of a search using a homomorphic circuit, this section uses that principle to its full extent by allowing more than just a boolean comparison ($=_c$) of the characters. This results in the ability to also compute inexact matches. The main part that differs from the exact search scheme is the construction of ϕ.

5.3.1 Counting Mismatches

The first step towards a circuit evaluating a fuzzy search is done by counting the number of mismatches between the search string and a given word in a database or a substring in a character stream. To accomplish this the multiplicative (AND) combination of all the character-level comparisons like in Equation 5.2 is replaced by a sum, yielding

$$\varphi: \begin{array}{c} \Sigma^* \times \mathfrak{C}^s \to \mathfrak{C}^{\lceil \log_2 s \rceil} \\ (w,s) \mapsto \sum_{i=0}^{|s|-1} (w_i =_c s_i) \end{array} \quad . \tag{7}$$

Since the sum can be expressed as a hamming weight of a bit vector, the application of *symmetric polynomials* leads to a very shallow circuit, used for example in (Smart and Vercauteren, 2010, page 15). The k-th bit of the hamming weight of a vector

$(b_1,\ldots,b_n)$ can be directly expressed by the polynomial

$$e_{2^{k-1}}(b_1,\ldots,b_n) := \sum_{1\le j_1<j_2\cdots j_{2^{k-1}}\le n} b_{j_1}\cdots b_{j_{2^{k-1}}} \quad .$$

The evaluation of the polynomial translates directly to the circuit

$$e_{2^{k-1}}(b_1,\ldots,b_n) := \bigoplus_{1\le j_1<j_2\cdots j_{2^{k-1}}\le n} b_{j_1}\cdots\wedge\cdots b_{j_{2^{k-1}}} \quad .$$

In the case that the maximum number of errors err_{max} is given beforehand, the advantage of symmetric polynomials is that one can directly limit the number of evaluated polynomials to the first $\log_2(err_{max})$ digits of the output vector.

5.3.2 "Softer" Mismatches

Some applications of fuzzy search, for example the mapping of short reads in genomes (Trapnell and Salzberg, 2009), require more differentiation when calculating the match between two characters. In this example, a sequence where only one character (= acid in the DNA) changed to one chemically related acid still strongly indicates a match in the genome. Consequently, if the position of the mismatch would introduce a change to a completely unrelated acid, a match of the sequence would be less likely.

For the rest of this subsection, let the alphabet $\Sigma = \{A,C,G,T\}$ consist of the four acids that build the DNA. The top table in Figure 1 lists the quantified differences (penalities) between these DNA-bases which reflect a computational metric for the *genetic distances* between a searchterm and a processed genome sequence.

In the circuit, a function $pen : \Sigma\times\Sigma\to\mathfrak{C}^*$ is needed that maps two given characters to the penalty of converting one character into the other. Since this function needs to be evaluated inside the circuit, it needs to be written as a boolean circuit itself. The necessary steps (also see Figure 1) are:

1. For each character c in the row and column headers, write $bin(c)$. Additionally, write the penalty in binary.
2. For each digit of the binary value of the penalty, write one table containing only that digit.
3. For each table in the previous step, create a minimal circuit evaluating this table using a Karnaugh map.

Depth analysis of pen. Without loss of generality, assume the resulting circuit be in conjunctive normal form (CNF). The conjunction at the top level results in AND-gates of depth $\log_2(|\Sigma|)$. Since only XOR- and AND-gates are available in the homomorphic circuit, each binary disjunction $a\vee b$ must be written as compound operation $(a\oplus b)\oplus(a\wedge b)$, resulting in a factor two compared to the depth of a circuit using OR gates. Each disjunctive term therefore has depth $\log_2(2|\Sigma|)$. The total depth of the resulting circuit for *pen* is $\log_2(|\Sigma|)+\log_2(2|\Sigma|)\in O(\log|\Sigma|)$, which is the same depth as for $=_c$ in Section 5.2.1.

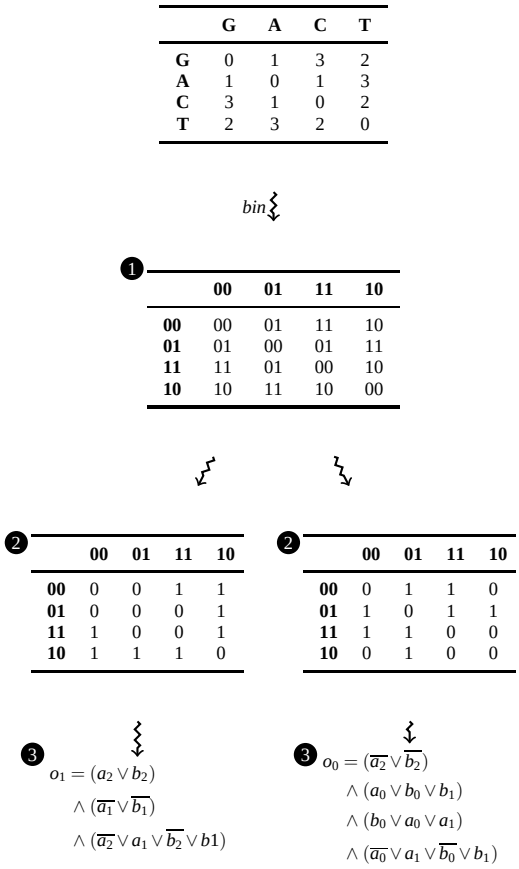

Figure 1: Construction of the *pen* function: transformation of a penalty table to a circuit using a Karnaugh map.

By replacing the character comparison with a penalty function, it is also necessary to sum up the individual penalties from the character-level comparisons for a total penalty of the word match as shown in Section 5.3.1. This results in the circuit

$$\varphi : \begin{array}{c}\Sigma^*\times\mathfrak{C}^s\to\mathfrak{C}^{\lceil\log_2 s\rceil}\\ (w,s)\mapsto\sum_{i=0}^{|s|-1} pen(w_i,s_i)\end{array} \quad . \tag{8}$$

5.4 Results

In this section we introduced an approach to mark the position of a definitive or probable match with respect

to exact or fuzzy search. The circuit φ that needs to be evaluated in the homomorphic cipherspace was shown to have a depth bound to $O(\log_2 |\Sigma|)$ in all cases. This means that the depth of the circuit is known beforehand, and especially that it does *not*depend on the size of the database.

6 IMPLEMENTATION & PERFORMANCE

This section gives a brief summary of our implementation[1]. The platform for the prototype is Java 6 SE on a 2.4 GHz Core 2 Duo with 3 MB L2 cache and 4 GB RAM. We present performance figures for different key and problem sizes.

6.1 Basic Operation Timings

Due to the structure of the homomorphic scheme $\mathfrak{E}$, the algorithms for addition, multiplication and decryption are almost atomic arithmetic operations of the underlying library for large integer handling. The most time consuming operations are K (Keygen) and E (Encrypt). The timing for these and the other basic operations is depicted in Table 1. The reason for the relatively long runtime of Keygen and Encrypt is the determination of a prime number during these operations.

Table 1: Timings for basic operations.

Op.	$\lambda = 512$	$\lambda = 1024$	$\lambda = 2048$
Keygen	35ms	270ms	4242ms
Encrypt	35ms	301ms	3218ms
Decrypt	¡1ms	¡1ms	¡1ms
Add	¡1ms	¡1ms	¡1ms
Mult	¡1ms	¡1ms	¡1ms

6.2 Protocol Timings

Our implementation comprises the protocols outlined in Section 4. The algorithms have been transformed into native Java code which invokes the API of our encryption scheme. The resulting performance is depicted in Table 2, which contains the time consumption figures and Table 3, showing the sizes of the output arguments.

[1]The software can be downloaded at http://www.hcrypt.com

Table 2: Protocol timings.

Protocol	$\lambda = 512$	$\lambda = 1024$	$\lambda = 2048$
OT100	¡1ms	¡1ms	¡1ms
OT10k	8ms	15ms	35ms
OT100k	190ms	290ms	426ms
YMP4	¡1ms	¡1ms	¡1ms
YMP40	2ms	9ms	37ms
YMP80	10ms	34ms	132ms
OMA16	¡1ms	1ms	4ms
OMA32	1ms	4ms	14ms
OMA64	3ms	12ms	43ms

6.3 Parameter Sizes

For the key parameters, factor sizes of $\lambda \in \{512, 1024, 2048\}$ and $\eta = 8$ bits have been chosen. There are measurements of every described protocol for three different problem sizes. Table 3 shows the return sizes for the *Oblivious Transfer* (OT) protocol implementation with 100, 10000 and 100000 bits long plaintext arguments. The timing figures show the runtime required for the determination of the result by Bob. The output sizes of the single return arguments of *Yao's Millionaires' Problem* (YMP) shown in Table 3 are in the range from 4 to 80 plaintext bits and therefore require between 3k and 323k bits of encrypted space. The runtime figures in Table 2 only contain the computational part of Bob. The *Oblivious Memory Access* (OMA) is the selection of exactly one of 16, 32 or 64 elements (memory bits). So the return sizes are the number of bits required by a single encrypted return bit. The calculation of the memory circuit by Bob is shown in the runtime figures.

Table 3: Oblivious transfer parameter sizes (Bits).

Protocol	$\lambda = 512$	$\lambda = 1024$	$\lambda = 2048$
OT100	103k	205k	410k
OT10k	10.3M	20.5M	41M
OT100k	103M	205M	410M
YMP4	3k	6k	12k
YMP40	40k	80k	160k
YMP80	80k	161k	323k
OMA16	4k	8k	16k
OMA32	5k	10k	20k
OMA64	6k	12k	24k

6.4 Search Results

We have implemented the exact search as described in Section 5.2 with different key sizes and database sizes, as well as two different output set reductions. The default reduction yields a one-bit indicator containing an encrypted 1 if the search term matches a

database entry and 0 otherwise. The second reduction type generates an encrypted binary representation of the matching entry's database index. The word size for all experiments is 5 bits. Table 4 summarizes the result figures. The upper section of Table 4 depicts

Table 4: Exact-match search results.

DB size	$\lambda = 512$	$\lambda = 1024$	$\lambda = 2048$
search			
1024	31 ms	115 ms	442 ms
256 k	8.6 s	30.1 s	119 s
512 k	17.4 s	61.1 s	241 s
1 M	36.9 s	124 s	487 s
generate index			
1024	11 ms	22 ms	43 ms
256 k	5.1 s	10.2 s	20.4 s
512k	12 s	24.0 s	48.1 s
1 M	29.8 s	59.5 s	119.5 s
indicator vector size(bits, *compact cipher)			
1024	1 M* / 5.2 M	2 M* / 10.5 M	4 M* / 20.9 M
256 k	256 M* / 1.34 G	512 M* / 2.68 G	1 G* / 5.37 G
512k	512 M* / 2.68 G	1 G* / 5.37 G	2 G* / 10.7 G
1 M	1 G* / 5.37 G	2 G* / 10.7 G	4 G* / 21.4 G
index value size (bits)			
1024	5.1 k	10.2 k	20.5 k
256 k	5.1 k	10.2 k	20.5 k
512k	5.1 k	10.2 k	20.5 k
1 M	5.1 k	10.2 k	20.5 k

the timing results for different key sizes and database sizes. The timings scale almost linearly with the size of the database, whereas larger keys cause an exponential timing behavior. The second section of the table shows the timings of the index-reduction for the different problem sizes. The sum of the index generation time and the basic search time is the total execution time for an index-reduced search. Section 3 of the table summarizes the sizes in number of bits of the returned match-indicator vectors. The lower section of the table contains the return sized of an index-reduced search. The returned argument contains an encrypted binary representation of the $log_2 n$ plaintext index-bits.

7 SUMMARY

In this paper we discussed an algebraically homomorphic scheme of limited multiplicative depth that can be used as an approach to build practical applications that operate on encrypted data. We discussed the properties of the SHE scheme and provided a proof of correctness. We gave a security analysis for different attack models and stated, under what circumstances the scheme is secure. Proof-of-concept implementations of the discussed protocols outlined the characteristics of homomorphically encrypted real-life applications. A detailed formal analysis of exact-match searching with extensions to fuzzy searching on encrypted data with encrypted search terms showed, how the algorithmic primitives of the simple protocols can be combined to solve a problem of higher complexity.

REFERENCES

Brakerski, Z. and Vaikuntanathan, V. (2011). Efficient fully homomorphic encryption from (standard) lwe. Cryptology ePrint Archive, Report 2011/344. http://eprint.iacr.org/.

Brenner, M., Wiebelitz, J., von Voigt, G., and Smith, M. (2011). A smart-gentry based software system for secret program execution. In *Proc. of the International Conference on Security and Cryptography SECRYPT*. SciTePress.

Coron, J.-S., Mandal, A., Naccache, D., and Tibouchi, M. (2011). Fully homomorphic encryption over the integers with shorter public keys. In *Advances in Cryptology CRYPTO 2011*, volume 6841 of *LNCS*. Springer Berlin / Heidelberg.

Damgrd, I., Meldgaard, S., and Nielsen, J. (2011). Perfectly secure oblivious ram without random oracles. In *Theory of Cryptography*, volume 6597 of *LNCS*. Springer Berlin / Heidelberg.

Gentry, C. (2009). Fully homomorphic encryption using ideal lattices. In *Proc. of the 41st annual ACM symposium on Theory of computing*, STOC '09, New York, NY, USA. ACM.

Gentry, C., Halevi, S., and Vaikuntanathan, V. (2010). i-hop homomorphic encryption and rerandomizable yao circuits. In *Advances in Cryptology - CRYPTO 2010*, volume 6223 of *LNCS*. Springer Berlin / Heidelberg.

Goldreich, O. (1987). Towards a theory of software protection and simulation by oblivious rams. In *Proc. of the 19th annual ACM symposium on Theory of computing*, STOC '87, New York, NY, USA. ACM.

Goldreich, O. and Ostrovsky, R. (1996). Software protection and simulation on oblivious rams. *J. ACM*, 43.

Goodrich, M. and Mitzenmacher, M. (2011). Privacy-preserving access of outsourced data via oblivious ram simulation. In *Automata, Languages and Programming*, volume 6756 of *LNCS*. Springer Berlin / Heidelberg.

Kolesnikov, V., Sadeghi, A.-R., and Schneider, T. (2009a). How to combine homomorphic encryption and garbled circuits improved circuits and computing the minimum distance efficiently.

Kolesnikov, V., Sadeghi, A.-R., and Schneider, T. (2009b). Improved garbled circuit building blocks and appli-

cations to auctions and computing minima. In *Cryptology and Network Security*, volume 5888 of *LNCS*. Springer Berlin / Heidelberg.

Malkhi, D., Nisan, N., Pinkas, B., and Sella, Y. (2004). Fairplay - a secure two-party computation system. In *Proc. of the 13th conference on USENIX Security Symposium - Volume 13*, SSYM'04, Berkeley, CA, USA. USENIX Association.

Naehrig, M., Lauter, K., and Vaikuntanathan, V. (2011). Can homomorphic encryption be practical? In *Proc. of the 3rd ACM workshop on Cloud computing security workshop*, CCSW '11, New York, NY, USA. ACM.

Pinkas, B. and Reinman, T. (2010). Oblivious ram revisited. In *Advances in Cryptology - CRYPTO 2010*, volume 6223 of *LNCS*. Springer Berlin / Heidelberg.

Smart, N. and Vercauteren, F. (2010). Fully homomorphic encryption with relatively small key and ciphertext sizes. In *Public Key Cryptography, PKC 2010*, volume 6056 of *LNCS*. Springer Berlin / Heidelberg.

Stehl, D. and Steinfeld, R. (2010). Faster fully homomorphic encryption. In *Advances in Cryptology - ASIACRYPT 2010*, volume 6477 of *LNCS*. Springer Berlin / Heidelberg.

Trapnell, C. and Salzberg, S. (2009). How to map billions of short reads onto genomes. *Nature Biotechnology*, 27(5).

Yao, A. C. (1982). Protocols for secure computations. In *SFCS '82: Proc. of the 23rd Annual Symposium on Foundations of Computer Science*. IEEE Computer Society, Washington, DC, USA.