Inequality Comparisons over Homomorphically Encrypted Data

Sophia Pietsch *

Abstract

Fully homomorphic encryption allows the computation of functions over encrypted data, eliminating the need to trust entities that run calculations on sensitive data. Computations over ciphertexts create noise, which must be kept small through expensive bootstrapping operations or by limiting the number of consecutive multiplications. Efficiency is especially important for operations like inequality comparisons, which are required in many algorithms.

In this paper, we design an inequality comparison and compare its complexity with the complexity of an existing algorithm. The designed algorithm reduces the inequality comparison to equality comparisons and uses an equality operator over constant-weight codes. Unlike the existing algorithm, the parameters of the proposed algorithm can be chosen to achieve an arbitrarily low multiplicative depth for integers of any size.

1 Introduction

Homomorphic encryption schemes allow computations directly over encrypted data, without the need for decryption. A fully homomorphic encryption (FHE) scheme allows an unlimited number of operations over encrypted data [AAUC18]. Most homomorphic encryption schemes support the addition and multiplication of ciphertexts, which is enough to represent arbitrary functions. Some cryptosystems can fit multiple integers into one ciphertext and use element-wise operations.

Operations over encrypted data increase the amount of noise present in ciphertexts. Noise must be kept low for correct decryption, so special bootstrapping operations reduce noise to a constant level. Bootstrapping is very expensive, leading to the design of algorithms that minimize the generation of noise. As repeated multiplications create the most noise, algorithms aim to minimize the multiplicative depth: the number of consecutive multiplications necessary on any input.

Instead of reducing each algorithm down to additions and multiplications, protocols for simple tasks are designed and incorporated into more complex algorithms. Equality and inequality comparisons of integers are an example of a commonly used task. For example, comparisons are essential for private queries over encrypted databases and machine learning over encrypted data.

The multiplicative depth for comparisons of n bit integers depends on n in most proposed algorithms [BST19]. Recently, Mahdavi and Kerschbaum designed an equality comparison with constant multiplicative depth using constant-weight codes [MK]. An open question is whether a similar algorithm can be constructed for inequality comparisons.

1.1 Main Result and Proof Overview

We investigate the complexity of inequality comparison algorithms in terms of both multiplications and multiplicative depth. The number of multiplications determines the runtime, while the multi-

*University of Waterloo, email: stpietsc@uwaterloo.ca.

plicative depth determines how much noise is generated. First, we analyze a less-than comparison LT_{VFE} developed by Tan et al [TLW+20]. They split integers into k digits in $\mathbb{Z}_{p^l}$. We show that

Theorem 1.1. [TLW+20] *LT_{VFE} performs approximately $l(4p-2+\log l)+\lfloor \log k \rfloor$ multiplications and has $\leq 2\lceil \log p \rceil + 2\lceil \log(l-1) \rceil + \log(\lfloor \log(p-1) \rfloor + 1) + \lfloor \log k \rfloor + 3$ multiplicative depth.*

High-level ideas of the proof: The less-than comparison is split into less-than and equality operations over the k digits of the two inputs. The less-than comparison over digits is split up further and computed using Lagrange Interpolating polynomials, while the equality comparison over digits uses Fermat's Little Theorem. We compute the complexity of these operations and the complexity required to combine them to obtain the result.

Next, we propose a less-than-or-equal comparison LTE that allows for constant multiplicative depth for n-bit integers. Numbers are encoded as binary strings of length m with exactly k ones.

Theorem 1.2. *LTE uses $m+k-1$ multiplications and has $\lceil \log(k-1) \rceil + 1$ multiplicative depth.*

High-level ideas of the proof: By representing an interval by a Best Range Cover, we reduce the less-than-or-equal comparison to $n+1$ equality comparisons. These comparisons are computed with one call to Mahdavi and Kerschbaum's equality operator [MK], which has the required complexity.

1.2 Related Work

Many algorithms encrypt each bit of an n bit integer in a separate ciphertext. For example, Garay et al. reduced a less-than comparison to bitwise operations, requiring $3n$ multiplications and $\lceil \log n \rceil$ multiplicative depth [GSV07]. Damgard et al. proposed an algorithm with constant multiplicative depth by finding the most significant differing bit. However, with a depth of 19 the algorithm would only be more efficient than other algorithms for integers with at least 2^{19} bits [DFK+06].

Cheon et al. worked on a greater-than comparison using batching techniques, encrypting the bits of an integer into different slots of the same ciphertext. When using an encryption scheme that allows shifting and cyclic rotations of the plaintext slots, they reduced the number of multiplications needed to $2n-4$. Their algorithm still has a multiplicative depth of $\log n$ [CKK15].

Next, researchers started investigating word-wise encryption, where an integer in $\mathbb{Z}_{p^l}$ is encrypted in one plaintext slot. Kim et al. designed an equality operator for FHE schemes that support efficient exponentiation. Their operator has $\lceil \log(p-1) \rceil + \lceil \log n \rceil$ multiplicative depth [KLLW18].

Tan et al. combined the batching techniques with word-wise encryption, allowing efficient computations on larger numbers by splitting them into k digits in $\mathbb{Z}_{p^l}$. The multiplicative depth of their less-than operator is logarithmic in all of p, l and k [TLW+20]. Iliashenko and Zucca optimized this algorithm, slightly reducing the required number of multiplications [IZ21].

In contrast, Mahdavi and Kerschbaum encoded integers as binary strings with length m and exactly k ones before encrypting them. Their equality operator uses $m+k-1$ multiplications and has $\lceil \log(k-1) \rceil + 1$ multiplicative depth [MK].

1.3 Organization

In the next section, we present some background knowledge that is used throughout the paper. After that, in Section 3 we present and analyze the less-than comparison introduced by Tan et al.

[TLW+20]. In Section 4 we develop and analyze an algorithm for less-than-or-equal comparisons that encodes integers as constant-weight codes. Lastly, we conclude in Section 5.

2 Preliminaries

2.1 Fan-Vercauteren (FV) Homomorphic Encryption System

The plaintexts in the Fan-Vercauteren homomorphic encryption system are elements of $R_t = \mathbb{Z}_t[X]/(x^d + 1)$ for some $t > 1$, while the ciphertexts are vectors of polynomials from $R_q = \mathbb{Z}_q[X]/(x^d + 1)$. To use the cryptosystem, values for t, d and q must be specified.

The FV scheme supports element-wise addition and multiplication, allowing simultaneous computations on up to d numbers. These operations increase the amount of noise in the ciphertexts. An expensive bootstrap operation reduces the noise back to a constant value [FV12]. When $t = p^l$ for some prime p, the FV scheme supports shifting and rotation of the d encrypted numbers as well as efficient element-wise exponentiation for exponents of the form p^i for some $i \geq 0$ [TLW+20].

2.2 Finite Extension Fields

This subsection states some Theorems from Finite Extension Fields that are used in Section 3.

Theorem 2.1. (Fermat's Little Theorem on $\mathbb{Z}_{p^l}$) *Let p be a prime and $l > 0$. Then, for any $\alpha \in \mathbb{Z}_{p^l} \backslash \{0\}$, we have $\alpha^{p^l - 1} = 1$.*

Theorem 2.2. (Lagrange Interpolation) *Given the outputs of a function f on all points $(x, y) \in \mathbb{Z}_p \times \mathbb{Z}_p$, a polynomial expression $P_f(x, y)$ can be constructed such that $P_f(x, y) = f(x, y)$ for any $(x, y) \in \mathbb{Z}_p \times \mathbb{Z}_p$. The degrees of x and y in P_f are at most $|\mathbb{Z}_p| - 1$ each.*

Theorem 2.3. [TLW+20, Lemma 3] *Let T be a $\mathbb{Z}_p$-linear map on $\mathbb{Z}_{p^l}$ for a prime p and $l \geq 0$. There is a unique set of constants $c_i \in \mathbb{Z}_{p^l}$ for $0 \leq i < l$ such that $T(x) = \sum_{i=0}^{l-1} c_i x^{p^i}$.*

2.3 Computing Running Products

Cheon et al. designed an algorithm for computing the running product $\pi_x = \left(x_0, x_0x_1, \ldots, \prod_{i=0}^{n-1} x_i\right)$ from $x = (x_0, x_1, \ldots, x_{n-1})$. Here, $x^{(i)} = (1, \ldots, 1, x_0, x_1, \ldots, x_{n-1-i})$ is computed using shift operations [CKK15]. For a proof of correctness see Cheon et al. [CKK15] or Tan et al [TLW+20].

Algorithm 1 RUNNING PRODUCT [CKK15]

Input: $x = (x_0, x_1, \ldots, x_{n-1}) \in \mathbb{Z}^n$

Output: $\pi_x = \left(x_0, x_0x_1, \ldots, \prod_{i=0}^{n-1} x_i\right)$

1: $z \leftarrow x$
2: **for** $i = 0$ to $\lfloor \log n \rfloor$ **do**
3: $\quad z \leftarrow z \cdot z^{(i)}$
4: **return** z

Lemma 2.4. **[TLW$^+$20]** *When the elements of x are stored in the same ciphertext, Algorithm 1 requires $\lfloor \log n \rfloor$ multiplications and multiplicative depth.*

Lemma 2.5. **[TLW$^+$20]** *When the elements of x are stored in different ciphertexts, Algorithm 1 requires approximately $n \log n$ multiplications and has $\lfloor \log n \rfloor$ multiplicative depth.*

2.4 Evaluating Polynomials over $\mathbb{Z}_p$

Tan et al. introduce the following strategy of evaluating a polynomial $f(x, y) = \sum_{j=0}^{n} \sum_{i=0}^{n} f_{i,j} x^i y^j$ of degree at most n in each variable [TLW$^+$20]:

First, compute powers of x and y up to the nth power. Aung et al. show this requires only $n-1$ multiplications [ALTW18]. Tan et al. optimize their algorithm to obtain a multiplicative depth of $\log(\lfloor \log n \rfloor + 1) + \lfloor \log n \rfloor$ [TLW$^+$20]. Next, evaluate n polynomials $f_j(x) = \sum_{i=0}^{n} f_{i,j} x^i$ using no ciphertext multiplications. Then, evaluate $f(x, y) = \sum_{j=0}^{n} f_j(x) y^j$ using $n-1$ multiplications and one multiplicative depth. Overall, the evaluation of the polynomial $f(x, y)$ requires $3n-3$ multiplications and $\log(\lfloor \log n \rfloor + 1) + \lfloor \log n \rfloor$ multiplicative depth.

2.5 Constant-weight codes

A binary constant-weight code is a binary string of length m with exactly k ones. We denote with $CW(m, k)$ all binary constant-weight codes with length m and k ones. Such codes can encode $\binom{m}{k}$ different values, so we can define a mapping from $[0, n-1]$ to $CW(m, k)$ if $\binom{m}{k} \geq n$. Mahdavi and Kerschbaum give an example of such a mapping [MK].

2.6 Best Range Covers

Best Range Covers (BRC) are a method of representing ranges of numbers and were originally used by Kiayias et al. to delegate pseudorandom functions [KPTZ13].

Definition 2.6. Given $n = 2^k$ for some $k \geq 0$, construct a binary tree with n leaves labeled from 0 to n. Then given $0 \leq a \leq b \leq n$, BRC[a,b] is the smallest set of nodes such that the set of all leaves that are in the subtree of some $v \in BRC[a, b]$ is equal to $\{a, a+1, \ldots, b\}$.

Algorithm 2 BEST RANGE COVER [KPTZ13]

Input: A number $a = a_{n-1} \cdots a_0$ with $a \leq n-1$
Output: $BRC[a, n-1]$

1: $BRC \leftarrow \{\}$
2: **if** $a = n-1$ **then return** $\{a\}$
3: $t \leftarrow \max\{i : a_i \neq 1\}$
4: **if** $\forall i \leq t : a_i = 0$ **then return** $\{a_{n-1} \cdots a_{t+1}\}$
5: **else**
6: $\quad \mu \leftarrow \min\{i | i < t \wedge a_i = 1\}$
7: $\quad$ **for** $i = t-1$ to $\mu + 1$ **do**
8: $\quad\quad$ **if** $a_i = 0$ **then** $BRC \leftarrow BRC \cup \{a_{n-1} \cdots a_{i+1} 1\}$
9: $\quad BRC \leftarrow BRC \cup \{a_{n-1} \cdots a_\mu, a_{n-1} \cdots a_{t+1} 1\}$
10: $\quad$ **return** BRC

Kiayias et al. give an algorithm for finding $BRC[a, b]$ and prove its correctness [KPTZ13]. The prefix of a number denotes a node and the empty set denotes the root. Algorithm 2 is simplified for the case $b = n - 1$.

Lemma 2.7. *$BRC[a, n-1]$ contains at most one node at each level of the binary tree.*

Proof. If we return on line 2 or 4, then $BRC[a, n-1]$ only contains one element. Otherwise, we add at most one node on each of the levels $t-1$ to $\mu-1$ and $\mu+1$ in the for loop. No element is added on level μ since $a_\mu = 1$. On line 9, nodes at levels μ and t are added. □

3 Less-than Comparisons with Word-wise Encoding and Batching

Tan et al. introduce a less-than comparison that exploits features of the plaintext space of the FV scheme introduced in Section 2.1 [TLW+20]. They choose the plaintext modulus $t = p^l$ for some prime p. Tan et al. use the vector of field elements (VFE) encoding, where integers are split into k digits in the field $\mathbb{Z}_{p^l}$ by representing the integers base p^l. We assume the digits are indexed from 0 to $k-1$, with $k-1$ being the most significant digit. For all digits of one integer to fit into one ciphertext, we choose the parameter $d \geq k$.

3.1 Less-than Comparison with Digits in $\mathbb{Z}_{p^l}$

First, Tan et al. assume the existence of an equality operator EQ and less-than operator LT over elements of $\mathbb{Z}_{p^l}$. Given these operations, they give a less-than comparison LT_{VFE} over VFE encodings [TLW+20]. As the FV scheme uses component-wise operations, the algorithm computes all equality and less-than comparisons over pairs x_i, y_i with one call to each of EQ and LT.

Algorithm 3 LESS-THAN OPERATOR OVER VFE (LT_{VFE}) [TLW+20]

Input: $x, y \in \left(\mathbb{Z}_{p^l}\right)^k$

1: $lt \leftarrow LT(x, y), eq \leftarrow EQ(x, y)$
2: $e \leftarrow lt[k-1] + \sum_{i=0}^{k-2} lt[i] \cdot \prod_{j=i+1}^{k-1} eq[j]$
3: **return** e

Theorem 3.1. [TLW+20] *If EQ and LT implement equality and less-than operators on $\mathbb{Z}_{p^l}$, then $LT_{VFE}(x, y) = 1$ if $x < y$ and $LT_{VFE}(x, y) = 0$ otherwise.*

Proof. Assume EQ and LT are correct. Let x_i and y_i be the ith digit of the VFE encodings of x and y respectively. Then $x < y$ if and only if there is a unique $0 \leq i < k$ such that $x_i < y_i$ and $x_j = y_j$ for all $i < j < k$.

Line 2 of LT_{VFE} checks this condition for all values of i and adds the results. Since i is unique, at most one of the conditions evaluates to one and the rest evaluate to zero. Hence, $LT_{VFE}(x, y) = 1$ if $x < y$ and $LT_{VFE}(x, y) = 0$ otherwise. □

3.2 Equality comparison over $\mathbb{Z}_{p^l}$

Tan et al. [TLW+20] use Kim et al.'s [KLLW18] equality operator as EQ.

Algorithm 4 EQUALITY OPERATOR (EQ) [KLLW18]

Input: $x, y \in \mathbb{Z}_{p^l}$

1: $e \leftarrow (x-y)^{p-1}$
2: $e \leftarrow \prod_{i=0}^{l-1} e^{p^i}$
3: **return** 1 - e

Theorem 3.2. [KLLW18] *$EQ(x,y) = 1$ if $x = y$ and $EQ(x,y) = 0$ otherwise.*

Proof. The final exponent of $x - y$ is equal to

$$(p-1)(p^{l-1} + p^{l-2} + \cdots + p + 1) = p^l + p^{l-1} + \cdots + p - p^{l-1} - p^{l-2} - \cdots - 1 = p^l - 1$$

Thus, $e = 1 - (x-y)^{p^l-1}$ and the claim follows from Fermat's Little Theorem (Theorem 2.1). □

Theorem 3.3. [TLW+20] *The EQ algorithm performs $\lceil \log p \rceil + l - 1$ multiplications and has $\lceil \log p \rceil + \lceil \log(l-1) \rceil$ multiplicative depth.*

Proof. On line 1, $(x-y)^{p-1}$ is computed using $\lceil \log p \rceil$ multiplications and multiplicative depth. The exponents on line 2 are computed using efficient exponentiation. Then, e is obtained using $l-1$ multiplications that require $\lceil \log(l-1) \rceil$ multiplicative depth. Overall, we use $\lceil \log p \rceil + l - 1$ multiplications and $\lceil \log p \rceil + \lceil \log(l-1) \rceil$ multiplicative depth. □

3.3 Less than comparison over $\mathbb{Z}_{p^l}$

Tan et al. split the elements in $\mathbb{Z}_{p^l}$ into l digits in $\mathbb{Z}_p$ and then decompose the less-than operator into equality and less-than comparisons using the same strategy as in Section 3.1 [TLW+20]. Now, Lagrange Interpolation is used for both the equality and less-than comparisons to obtain polynomials $P_{EQ}(x,y)$ and $P_{LT}(x,y)$ over $\mathbb{Z}_p$. By Theorem 2.2, the degree of these polynomials is at most $p-1$ in each of the variables x and y.

The decomposition from $\mathbb{Z}_{p^l}$ to $\mathbb{Z}_p$ needs to be computed over encrypted data. Since $a \in \mathbb{Z}_{p^l}$ can be expressed as $a = \sum_{j=0}^{l-1} a_j p^j$, it suffices to find functions T_0 to T_{l-1} such that $T_i(a) = a_i$.

Algorithm 5 LESS-THAN OPERATOR (LT) [TLW+20]

Input: $x, y \in \mathbb{Z}_{p^l}$

1: $lt, eq \leftarrow 0$
2: **for** $i = 0$ to $l-1$ **do**
3: $\quad a \leftarrow T_i(x), b \leftarrow T_i(y)$
4: $\quad lt[i] \leftarrow P_{LT}(a,b), eq[i] \leftarrow P_{EQ}(a,b)$
5: $e \leftarrow lt[l-1] + \sum_{i=0}^{l-2} lt[i] \cdot \prod_{j=i+1}^{l-1} eq[j]$
6: **return** e

Lemma 3.4. [TLW+20] *When viewing $\mathbb{Z}_{p^l}$ as an l-dimensional $\mathbb{Z}_p$-vector space, T_i is a linear map for $0 \le i < l$.*

Proof. Let $a = \sum_{j=0}^{l-1} a_j p^j$, $b = \sum_{j=0}^{l-1} b_j p^j$ and $c \in \mathbb{Z}_p$. For any $0 \le i < l$, $T_i(a + cb) = T_i\left(\sum_{j=0}^{l-1} (a_j + cb_j) p^j\right) = a_i + cb_i = T_i(a) + cT_i(b)$. □

By applying Lemma 3.4 and Theorem 2.3, the functions T_i are polynomials over $\mathbb{Z}_{p^l}$. Hence, $T_i(a)$ can be computed over ciphertexts to decompose each $x \in \mathbb{Z}_{p^l}$ into l digits in $\mathbb{Z}_p$.

Remark 3.5. For LT_{VFE} (Algorithm 3), we need LT to compute k less-than comparisons corresponding to the different slots of the ciphertexts x and y. Hence, the LT algorithm uses different ciphertexts for each $exp_x[i]$, $exp_y[i]$, $lt[i]$ and $eq[i]$, as each of these will need to contain k entries.

Theorem 3.6. [TLW+20] *$LT(x, y) = 1$ if $x < y$ and $LT(x, y) = 0$ otherwise.*

Proof. The decomposition from $\mathbb{Z}_{p^l}$ to l digits in $\mathbb{Z}_p$ is correct by Lemma 3.4 and Theorem 2.3. The polynomials P_{LT} and P_{EQ} return the correct results by Lagrange Interpolation (Theorem 2.2). Line 5 correctly computes $LT(x, y)$ by Theorem 3.1, where the same strategy is used. □

Theorem 3.7. [TLW+20] *The LT algorithm performs approximately $l(4p - 3 + \log l)$ multiplications and has $\log(\lfloor \log(p-1) \rfloor + 1) + \lfloor \log(p-1) \rfloor + \lfloor \log(l-1) \rfloor + 2$ multiplicative depth.*

Proof. By Theorem 2.3, the powers of x and y required for the polynomials T_i can be computed using efficient exponentiation without multiplication of ciphertexts.

The strategy outlined in Section 2.4 is used to evaluate P_{LT} and P_{EQ}. The powers of a and b only have to be computed once, giving $4p-4$ multiplications and $\log(\lfloor \log(p-1) \rfloor + 1) + \lfloor \log(p-1) \rfloor + 1$ multiplicative depth. For all l iterations, $l(4p - 4)$ multiplications are required.

As explained in Remark 3.5, the elements $eq[j]$ are stored in different ciphertexts. By Lemma 2.5, the running product of elements $eq[j]$ on line 5 requires approximately $(l - 1)\log(l - 1) \approx l \log l$ multiplications and has multiplicative depth $\lfloor \log(l-1) \rfloor$. We then multiply by elements $lt[i]$, adding approximately l more multiplications and one additional multiplicative depth.

In total, the algorithm LT requires approximately $l(4p-3+\log l)$ multiplications and $\log(\lfloor \log(p-1) \rfloor + 1) + \lfloor \log(p-1) \rfloor + \lfloor \log(l-1) \rfloor + 2$ multiplicative depth. □

3.4 Complexity analysis

Recall that for LT_{VFE} (Algorithm 3), integers x and y are represented by k digits in $\mathbb{Z}_{p^l}$. Here, the complexity of $LT_{VFE}(x, y)$ is given in terms of k, p and l.

Theorem 3.8. [TLW+20] *LT_{VFE} uses approximately $l(4p - 2 + \log l) + \lfloor \log k \rfloor$ multiplications and has at most $2\lceil \log p \rceil + 2\lceil \log(l-1) \rceil + \log(\lfloor \log(p-1) \rfloor + 1) + \lfloor \log k \rfloor + 3$ multiplicative depth.*

Proof. By Theorem 3.3 and Theorem 3.7, the calls to EQ and LT require approximately $\lceil \log p \rceil + l(4p - 2 + \log l)$ multiplications and at most $2\lceil \log p \rceil + 2\lceil \log(l-1) \rceil + \log(\lfloor \log(p-1) \rfloor + 1) + 2$ multiplicative depth.

The elements of eq are stored in one ciphertext, so by Lemma 2.4 the running product on line 2 can be computed with $\lfloor \log k \rfloor$ multiplications and multiplicative depth. Multiplying the results by the ciphertext corresponding to lt gives one additional multiplication and multiplicative depth.

Overall, we get approximately $\lceil \log p \rceil + l(4p - 2 + \log l) + \lfloor \log k \rfloor$ multiplications and a multiplicative depth of at most $2\lceil \log p \rceil + 2\lceil \log(l-1) \rceil + \log(\lfloor \log(p-1) \rfloor + 1) + \lfloor \log k \rfloor + 3$. □

Table 1 and Table 2 show the number of multiplications and multiplicative depths LT_{VFE} requires for 64 and 128-bit numbers respectively. Compared with Garay et al.'s bitwise algorithm, which requires $3n$ multiplications and $\lceil \log n \rceil$ multiplicative depth for n-bit integers [GSV07], LT_{VFE} requires significantly fewer multiplications but up to double the multiplicative depth. For a given n, all choices of p, k and l give a similar multiplicative depth, limiting the amount of computation possible after the comparison operation without needing to use bootstrapping.

p	l	k	Multiplications	Mult. Depth
5	4	7	80	16
5	1	28	22	14
2	8	8	64	14
2	1	64	11	11

Table 1: Complexity of LT_{VFE} for different combinations of p and l and k when $n = 64$.

p	l	k	Multiplications	Mult. Depth
5	2	28	44	14
5	1	56	23	15
2	8	16	72	15
2	4	32	40	14

Table 2: Complexity of LT_{VFE} for different combinations of p and l and k when $n = 128$.

4 Less-than-or-equal Comparisons with Constant-weight Codes

Mahdavi and Kerschbaum [MK] describe equality operators for integers encoded as constant-weight codes (see Section 2.5) with very small multiplicative depths, allowing for more computations after the equality comparison. Here, we extend this technique to less-than-or-equal comparisons by using the BRC construction to reduce $a \leq b$ to equality operations over constant-weight codes.

4.1 Reduction to Constant-weight Codes

Since the largest possible unsigned n-bit integer is $2^n - 1$, $a \leq b$ if and only if $b \in [a, 2^n - 1]$. From Section 2.6, the interval $[a, 2^n - 1]$ can be represented by $BRC[a, 2^n - 1]$, a set of nodes in a binary tree with leaves labeled from 0 to $2^n - 1$. Let $PATH[b] = \{r, b_{n-1}, b_{n-1}b_{n-2}, \ldots, b_{n-1} \cdots b_0\}$ be the set of nodes in the path from the root of the binary tree to the leaf b, where r represents the root.

Lemma 4.1. $|BRC[a, 2^n - 1] \cap PATH[b]| \leq 1$

Proof. If $|BRC[a, 2^n - 1] \cap PATH[b]| > 1$, there exist $u = b_{n-1} \cdots b_i, v = b_{n-1} \cdots b_j \in BRC[a, 2^n - 1]$ with $i > j$. As the subtree of v is contained in the subtree of u, $BRC[a, 2^n - 1] \backslash \{v\}$ covers all leaves in $[a, 2^n - 1]$. This contradicts the minimality of $BRC[a, 2^n - 1]$. □

Lemma 4.2. $|BRC[a, 2^n - 1] \cap PATH[b]| = 1$ *if and only if* $b \in [a, 2^n - 1]$.

Proof. Assume $|BRC[a, 2^n - 1] \cap PATH[b]| = 1$, say $v \in BRC[a, 2^n - 1] \cap PATH[b]$. By definition, all leaves in the subtree of v are in $[a, 2^n - 1]$. Since b is a leaf of the subtree of v, $b \in [a, 2^n - 1]$.

Now, assume $b \in [a, 2^n - 1]$. There exists $v \in BRC[a, 2^n - 1]$ such that b is in the subtree of v. Hence, $b = b_{n-1} \cdots b_0 = vb_i \cdots b_0$ for some i. This gives $v = b_{n-1} \cdots b_{i+1} \in PATH[b]$. Hence $|BRC[a, 2^n - 1] \cap PATH(b)| \geq 1$ and by Lemma 4.1, $|BRC[a, 2^n - 1] \cap PATH(b)| = 1$. □

Lemma 2.7 implies the existence of an array $BRC[a, 2^n - 1]$ of length $n+1$ such that $BRC[a, 2^n - 1][i]$ contains the node in $BRC[a, 2^n - 1]$ at the ith level of the binary tree, or -1 if no such node exists. Then Lemma 4.2 implies that $a \leq b$ if and only if there exists $0 \leq i \leq n$ such that $BRC[a, 2^n - 1][i] = b_{n-1} \cdots b_i$.

We can choose m and k so that the elements of $BRC[a, 2^n - 1]$ and $PATH[b]$ can be mapped to $CW(m, k)$. When $BRC[a, 2^n - 1][i] = -1$, use a length m string of zeros. Then, checking whether $a \leq b$ reduces to $n + 1$ equality operations over constant-weight codes.

4.2 Equality Comparisons over Constant-weight Codes

For the equality operations, we use the arithmetic equality operator AEQ developed by Mahdavi and Kerschbaum [MK]. We encode the constant-weight codes x and y bitwise with one ciphertext

for each bit. To obtain the modular arithmetic for line 2 of Algorithm 6, set $t = p$ in the FV scheme introduced in Section 2.1. We can use more plaintext slots to simultaneous compute up to d equality comparisons.

Algorithm 6 ARITHMETIC EQUALITY OPERATOR (AEQ) [MK]

Input: $x, y \in CW(m,k) \cup CW(m,0)$, p a prime with $p > k$.

1: $k' = \sum_{i=1}^{m} x[i] \cdot y[i]$

2: $e = \frac{1}{k!} \prod_{i=0}^{k-1} (k' - i) \mod p$

3: **return** e

Theorem 4.3. [MK, Theorem 1] For $x, y \in CW(m,k)$, $AEQ(x,y) = 1$ if $x = y$ and $AEQ(x,y) = 0$ otherwise. If at least one of x or y is in $CW(m,0)$, $AEQ(x,y) = 0$.

Proof. After line 1, k' is the inner product of x and y. For $x, y \in CW(m,k)$, $k' = k$ if and only if $x = y$ and $0 \leq k' < k$ otherwise. If at least one of x or y is in $CW(m,0)$, we have $k' = 0$. On line 2, $e = 1$ if $k' = k$ and $e = 0$ if $0 \leq k' < k$. □

Theorem 4.4. [MK] AEQ performs $m+k-1$ multiplications and has $\lceil \log(k-1) \rceil + 1$ multiplicative depth.

Proof. The computation of k' requires m multiplications and one multiplicative depth. On line 2, k elements are multiplied, requiring $k-1$ multiplications and $\lceil \log(k-1) \rceil$ multiplicative depth. In total, AEQ uses $m + k - 1$ multiplications and has $\lceil \log(k-1) \rceil + 1$ multiplicative depth. □

4.3 Inequality Comparison Algorithm

We now implement the method developed in Section 4.1. First, we find $BRC[a, 2^n - 1]$ and $PATH[b]$ and map their elements to $n+1$ pairs of constant-weight codes. Choosing $d \geq n+1$ in the FV scheme, the pairs of constant-weight codes can be encoded in one pair of ciphertext arrays. These arrays contain m ciphertexts, the ith ciphertext containing the ith bit of $n+1$ constant-weight codes. Then, one call to AEQ computes $n+1$ equality comparisons. This gives the following algorithm LTE for computing $a \leq b$.

Algorithm 7 LESS THAN OR EQUAL COMPARISON (LTE)

Input: Arrays $BRC[a, 2^n - 1]$, $PATH[b]$ with elements in $CW(m,k) \cup CW(m,0)$

1: $c \leftarrow EQ(BRC[a, 2^n - 1], PATH[b])$

2: $e = \sum_{i=0}^{n} c[i]$

3: **return** e

To compute e, we use the shift operations as all elements of c are in the same ciphertext.

Theorem 4.5. *For two n-bit integers a, b, $LTE(BRC[a, 2^n - 1], PATH[b]) = 1$ if $a \leq b$ and $LTE(BRC[a, 2^n - 1], PATH[b]) = 0$ otherwise.*

Proof. We showed in Section 4.1 that if $a \leq b$ then there is some i such that $c[i] = 1$ and $c[j] = 0$ for $i \neq j$. The section also shows that if $a > b$, $c[i] = 0$ for all i. □

Corollary 4.6. *LTE uses $m + k - 1$ multiplications and has $\lceil \log(k-1) \rceil + 1$ multiplicative depth.*

Proof. The only multiplications are performed in the single call to AEQ. Hence, the complexity of LTE follows from Theorem 4.4. □

Remark 4.7. If a smaller value of d is desired to speed up computation, $l < n + 1$ values can be batched into one ciphertext and a total of $\lceil (n+1)/l \rceil$ calls to AEQ are needed. This modification requires more multiplications but has the same multiplicative depth.

4.4 Complexity analysis

Section 2.5 implies that for n-bit numbers, we need $\binom{m}{k} \geq 2^n$ to uniquely map elements on the ith level of $BRC[a, 2^n - 1]$ and $PATH[b]$ to $CW(m, k)$. Many multiplications are needed for poor choices of m and k as the number of multiplications is the sum of both (see Corollary 4.6).

For 64-bit numbers, multiplications required are minimal for $m = 73$ and $k = 25$ with 97 multiplications and 6 multiplicative depth. For 128-bit numbers, multiplications are minimal for $m = 138$ and $k = 52$, with 189 multiplications and 7 multiplicative depth. These optimized choices have the same depth as Garay et al.'s bitwise algorithm [GSV07], but significantly fewer multiplications. In contrast to the LT_{VFE} algorithm (Algorithm 3) developed by Tan et al. [TLW+20], the choices of m and k can be modified to reduce the multiplicative depth as desired.

5 Conclusion and Open Problems

In this paper, we introduced a new algorithm LTE for less-than-or-equal comparisons over homomorphically encrypted data by representing integers with ranges and paths and encoding these as constant-weight codes. The main motivation for this approach is to construct inequality comparisons with very low multiplicative depth. Algorithms designed for FHE schemes avoid expensive bootstrapping operations by achieving a low multiplicative depth. Comparisons in particular benefit from a low multiplicative depth, as are often used as a subroutine in more complex algorithms.

We contrast our algorithm with a less-than operator LT_{VFE} introduced by Tan et al., which splits integers into several digits and encodes them in one ciphertext [TLW+20]. One limitation of LT_{VFE} is the multiplicative depth, which given the size of integers is almost constant for all parameter choices. The LTE algorithm does not have this limitation, as the parameters can be chosen to achieve any multiplicative depth independent of the size of the inputs. However, more multiplications are required for small multiplicative depths. A limitation of the LTE algorithm is the specific input format that it requires. The format may not be useful for other computations, limiting the use of LTE as a subroutine for other algorithms.

An interesting future direction is the parallelization of the algorithms presented. With parallelization, multiplicative depth may be much more important than the number of multiplications, making the LTE algorithm proposed very promising. When encrypted data must be transmitted, another important consideration is the amount of communication required by different encoding schemes. Here, algorithms like LT_{VFE} that encode whole integers in only one ciphertext are much more efficient than bitwise encoding schemes.

Acknowledgments

This paper was inspired by my URA, where Florian Kerschbaum and Rasoul Mahdavi introduced me to their work on constant-weight codes and Best Range Covers.

References

[AAUC18] Abbas Acar, Hidayet Aksu, A. Selcuk Uluagac, and Mauro Conti. A survey on homomorphic encryption schemes: Theory and implementation. *ACM Comput. Surv.*, 51(4), July 2018.

[ALTW18] Khin Aung, Hyung Lee, Benjamin Tan, and Huaxiong Wang. Fully homomorphic encryption over the integers for non-binary plaintexts without the sparse subset sum problem. *Theoretical Computer Science*, 771, 11 2018.

[BST19] Florian Bourse, Olivier Sanders, and Jacques Traoré. Improved secure integer comparison via homomorphic encryption. Cryptology ePrint Archive, Report 2019/427, 2019. `https://ia.cr/2019/427`.

[CKK15] Jung Cheon, Miran Kim, and Myungsun Kim. Search-and-compute on encrypted data. pages 142–159, 01 2015.

[DFK+06] Ivan Damgård, Matthias Fitzi, Eike Kiltz, Jesper Nielsen, and Tomas Toft. Unconditionally secure constant-rounds multi-party computation for equality, comparison, bits and exponentiation. volume 3876, pages 285–304, 03 2006.

[FV12] Junfeng Fan and Frederik Vercauteren. Somewhat practical fully homomorphic encryption. 01 2012.

[GSV07] Juan Garay, Berry Schoenmakers, and Jose Villegas. Practical and secure solutions for integer comparison. volume 4450, pages 330–342, 04 2007.

[IZ21] Ilia Iliashenko and Vincent Zucca. Faster homomorphic comparison operations for bgv and bfv. *Proceedings on Privacy Enhancing Technologies*, 2021:246 – 264, 2021.

[KLLW18] Myungsun Kim, Hyung Tae Lee, San Ling, and Huaxiong Wang. On the efficiency of fhe-based private queries. *IEEE Transactions on Dependable and Secure Computing*, 15(2):357–363, 2018.

[KPTZ13] Aggelos Kiayias, Stavros Papadopoulos, Nikos Triandopoulos, and Thomas Zacharias. Delegatable pseudorandom functions and applications. In *Proceedings of the 2013 ACM SIGSAC Conference on Computer & Communications Security*, CCS '13, page 669–684, New York, NY, USA, 2013. Association for Computing Machinery.

[MK] Rasoul Mahdavi and Florian Kerschbaum. Scalable pir using constant-weight codewords.

[TLW+20] Benjamin Tan, Hyung Lee, Huaxiong Wang, Shu Ren, and Aung Khin. Efficient private comparison queries over encrypted databases using fully homomorphic encryption with finite fields. *IEEE Transactions on Dependable and Secure Computing*, PP:1–1, 01 2020.