Non-interactive and Output Expressive Private Comparison from Homomorphic Encryption

Wen-jie Lu
riku@mdl.cs.tsukuba.ac.jp
University of Tsukuba

Jun-jie Zhou
zhou@mdl.cs.tsukuba.ac.jp
University of Tsukuba

Jun Sakuma
jun@cs.tsukuba.ac.jp
University of Tsukuba
JST CREST
RIKEN Center for AIP

ABSTRACT

Private comparison is about privately determining whether $a > b$, given two input integers a and b which are held as private information. Private comparison is an important building block for applications such as secure auction and privacy-preserving decision tree evaluation.

In this paper, we consider a variant setting in which the inputs a and b as well as the result bit $\mathbf{1}\{a > b\}$ are encrypted. Using a ring variant of a fully homomorphic encryption scheme, our solution takes as input the ciphertexts of a and b and produces only one ciphertext which decrypts to the bit $\mathbf{1}\{a > b\}$ without any interaction with the secret key holder. Our approach does not encrypt the inputs bit-wisely and requires only a multiplicative depth of one, giving about $48 - 90$ fold speed up over previous solutions.

CCS CONCEPTS

• **Security and privacy → Privacy-preserving protocols**;

KEYWORDS

Secure Comparison; Non-interactive Crypto-computing; Secure Classification

ACM Reference Format:
Wen-jie Lu, Jun-jie Zhou, and Jun Sakuma. 2018. Non-interactive and Output Expressive Private Comparison from Homomorphic Encryption. In *ASIA CCS '18: 2018 ACM Asia Conference on Computer and Communications Security, June 4–8, 2018, Incheon, Republic of Korea.* ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/3196494.3196503

1 INTRODUCTION

Private comparison, which allows an evaluator to determine whether $a > b$ without revealing the input integers a and b to the evaluator, is an important building block for building privacy-preserving applications such as secure auction [10], private data access [12], secure face recognition [21], and privacy-preserving decision tree evaluation [2, 25].

A non-interactive private comparison protocol that allows the private comparison to be done locally on the evaluator's side is essential for non-interactive privacy-preserving applications that use private comparison as a building block. Such a non-interactive application allows the users to go offline when the application has started, saving electricity and network bandwidth for devices, which is especially important for low-powered users such as cell phones and the IoT.

However, to the best of our knowledge, no existing private comparison protocol is satisfactory enough for building non-interactive applications that involve integer comparison. The reason is twofold. The explicit one is that the private comparison protocol itself [2, 12] already requires active communication between the protocol players. The other one is less explicit and subtle. The comparison protocols themselves are non-interactive [1, 9–11, 26], however, only very limited operations, such as the logic AND, can be carried out on the output of these protocols. Applications that build on these private comparison protocols are usually forced to be interactive [2, 12, 25].

The objective of this work is to develop a private comparison protocol that is both non-interactive and output expressive. Namely, this private comparison protocol should allow evaluating locally the comparison $a > b$ from two encrypted integers a and b. Also, expressive arithmetic operations, such as addition and multiplication, should be possible on the resulting ciphertext of the protocol without extra interactions with the decryption key holder.

One possible solution for these two requirements is to use fully homomorphic encryption (FHE). A such FHE solution usually requires a multiplicative depth of $O(\log \delta)$ where δ is the input bit length. However, the bootstrapping operations of [3, 7, 13] used by the FHE solution might become the bottleneck when δ is relatively small, e.g., $\delta \leq 16$.

In this work, we present XCMP, a more practical private comparison protocol for relatively small domains (i.e., less than 16 bits) that satisfies both the non-interactivity and output expressiveness requirements. XCMP uses a ring variant of an FHE scheme but only requires a multiplicative depth of one, and thus it is more efficient than the naive bit-wise encryption FHE solution for small domains. Briefly, XCMP allows the evaluator to locally compute a ciphertext ctxt which decrypts to the comparison bit $\mathbf{1}\{a > b\}$, i.e., $\mathsf{ctxt} \leftarrow \mathsf{XCMP}([\![\pi(a)]\!], [\![\pi(b)]\!])$. Here, we use an encoding π to map a and b into the message space of the FHE scheme. Also, the additive homomorphism property of ctxt is preserved. We can homomorphically sum the outputs from several calls of XCMP without any interactions with the decryption key holder. The multiplication of the outputs from several calls of XCMP is not supported, while we can still multiply an integer (plain or encrypted) to the output of XCMP.

ASIA CCS '18, June 4–8, 2018, Incheon, Republic of Korea

ACM ISBN 978-1-4503-5576-6/18/06…$15.00
https://doi.org/10.1145/3196494.3196503

Table 1: Comparing XCMP with other HE-based solutions. δ denotes the input bit length. Notice that the way of encrypting inputs differs for each method, and we simply denote it as $[\![\cdot]\!]$.

Methods	Input	Interactions	Expressiveness
[11]	$[\![a]\!]$, $[\![b]\!]$	0	barely none
[12]	$[\![a]\!]$, $[\![b]\!]$	$\log \delta$	ADD, MULT
[2]	$[\![a]\!]$, $[\![b]\!]$	4	ADD
[10], [1, 9]	$[\![a]\!]$, b	0	logic AND
Naive FHE [7]	$[\![a]\!]$, $[\![b]\!]$	0	ADD, MULT
XCMP	$[\![a]\!]$, $[\![b]\!]$	0	ADD (limited) MULT
XCMP	$[\![a]\!]$, b	0	ADD (limited) MULT

Related Work. Non-interactive private comparison protocols can be built from homomorphic encryption (HE). In Table 1, we summarize some of the-state-of-the-art HE based protocols, and compare them with XCMP. The method of [11] allows comparing two encrypted integers by encrypting bits as the quadratic and non-quadratic residues of an RSA modulus, at the cost of introducing errors. By running the protocol λ times, we can limit the error rate within $2^{-\lambda}$. [2] is a variant of [11] and gives exact results while it introduces 4 more rounds of interaction and requires decryption in the middle of the protocol. Most of the other HE-based methods such as [1, 9, 10, 26] only work for the case that a or b are plaintexts. Currently, the most efficient solution is considered as [9] and its improvement [26]. Also, the approach of [1] allows parameterizing the comparison result to a pre-defined value while it is much slower than [9, 26].

None of these HE-based approaches satisfies both the non-interactivity and expressive reusability requirements. [2] requires decryption in the middle of the protocol, and thus is interactive. On the other hand, the approaches [1, 9–11] are non-interactive, but the resulting ciphertexts are hardly reusable. That is because these methods compute multiple ciphertexts and one of the computed ciphertexts will decrypt to zero if $a > b$ holds, otherwise all these ciphertexts decrypt to non-zero random values. As a result, we can only perform limited computation with these ciphertexts, i.e., the logic AND of two comparison results. The naive bit-wise encryption FHE solution (e.g., [7]) satisfies these two requirements but is not practical enough. To the best of our knowledge, XCMP is the first non-interactive private comparison protocol that allows the evaluator to perform more expressive arithmetic operations on the resulting ciphertexts, without bit-wise encryption.

Private comparison protocols can also be built from Yao's garbled circuit (GC) [17, 18], secret sharing [5], and from oblivious transfer (OT) [8]. All these GC and OT solutions are designed as two-party protocols and require multiple rounds of interaction. Thus they cannot be employed directly in the non-interactive setting.

Contributions. In summary, our protocol offers some novel features that are only partially possible in existing protocols.

- *Non-interactive.* We present a new encoding π for the comparison functionality. As a result, an evaluator can locally compute a ciphertext that decrypts to the comparison bit without any assistance from the decryption key holder.
- *Output expressive.* The output ciphertext from XCMP preserves (to some extent) additive homomorphism so that the evaluator can directly use this ciphertext for a higher level functionality without asking the decryption key holder for decryption.
- *Two encrypted inputs.* XCMP can take as input two encrypted integers. From our experimental results, XCMP was about $48 - 90$ times faster than [11] in terms of evaluation time.
- *Parametrizable.* The protocol output can be parameterized as two distinct integers μ_0 and μ_1. That is the ciphertext

$$\mathsf{ctxt} \leftarrow \mathsf{XCMP}([\![\pi(a)]\!], [\![\pi(b)]\!]; \mu_0, \mu_1)$$

  decrypts to μ_1 if $a > b$ holds, otherwise to μ_0.

These properties of XCMP enable us to outsource sophisticated functionalities to a public cloud without compromising privacy. As a concrete example, we present a private decision tree evaluation protocol in Appendix B.

Table 2: Notation

Notation	Description
$\mathbf{1}\{\mathcal{P}\}$	returns 1 if $\mathcal{P}$ is true, else returns 0
$e \xleftarrow{\$} \mathcal{E}$	uniform random sample e from set $\mathcal{E}$
$\boldsymbol{x}$, x_i	vector $\boldsymbol{x}$ and its i-th entry
P, $P[i]$	polynomial P and its i-th coefficient
X	indeterminate of polynomials
m, p	parameters used in FHE
$\mathbb{A}_p$	ring defined as $\mathbb{Z}_p[X]/(X^m + 1)$
$[\![P]\!]$	a ciphertext of $P \in \mathbb{A}_p$
$\oplus$, $\otimes$, $\ominus$	homomorphic operations

2 PRELIMINARIES

Table 2 summarizes the notation used in this paper. Typically, the symbol X is used to denote the indeterminate of polynomials.

2.1 Stakeholders & Non-interactivity

We consider three kinds of stakeholders, i.e., encryptor, evaluator and decryptor. The decryption key is held by the decryptor while the corresponding encryption key is shared among the encryptor(s). The encryptor sends encrypted data to the evaluator using the shared encryption key. The evaluator runs the private comparison on the collected ciphertexts. The evaluator then gives the computed encrypted result to the decryptor for decryption. This setting reduces to the outsourcing setting of [12] if the encryptor and decryptor are identical, by viewing the client as the decryptor and the cloud as the evaluator.

We state a private comparison protocol is non-interactive if except the final result-responding interaction, there is no further interaction between the evaluator and the decryptor. The inevitable single interaction between the encryptor(s) and evaluator is not counted.

Input: $a, b \in \mathbb{Z}_m$ for positive integer m.
Output: $\mathbf{1}\{a > b\}$.

Figure 1: Ideal functionality $\mathcal{F}_{\mathsf{cmp}}$ for comparison.

2.2 Ring-variant FHE

In this paper, we need a ring-variant of FHE whose message space is defined by the polynomial ring $\mathbb{A}_p := \mathbb{Z}_p[X]/(X^m + 1)$ where m is a power of 2 and p is a prime number. The security level of the encryption scheme are determined by the value of m. We respectively choose $m = 2^{12}$ and $m = 2^{13}$, aiming to provide at least 80-bit and 128-bit security level, according to the security analysis of [13, 14]. We take advantage of the property $X^m = -1 \mod X^m + 1$ in our construction.

We give brief descriptions related to FHE. Let (sk, pk) be a private-public key pair. We write $[\![\cdot]\!]$ to denote the encryption function, and $\mathsf{Dec}_{\mathsf{sk}}(\cdot)$ for the decryption. For elements $A, B \in \mathbb{A}_p$, we leverage the following properties of FHE in our construction.

- Additive homomorphism:

$$\mathsf{Dec}_{\mathsf{sk}}([\![A]\!] \oplus [\![B]\!]) = A + B, \quad \mathsf{Dec}_{\mathsf{sk}}([\![A]\!] \oplus B) \quad = A + B$$

- Multiplicative homomorphism:

$$\mathsf{Dec}_{\mathsf{sk}}([\![A]\!] \otimes [\![B]\!]) = A \times B, \quad \mathsf{Dec}_{\mathsf{sk}}([\![A]\!] \otimes B) \quad = A \times B$$

- Automorphism: $\mathcal{M}_q : X \mapsto X^q$ $(q \in \mathbb{Z}^*_{2m})$

$$\mathsf{Dec}_{\mathsf{sk}}(\mathcal{M}_q([\![A]\!])) = A(X^q).$$

  For example, $\mathcal{M}_3(1+3X^2) = 1+3X^6$. This operation is much faster than the homomorphic multiplication.

The operators $\oplus$, $\ominus$ and $\otimes$ indicate homomorphic addition, subtraction and multiplication, respectively.

3 PROPOSED PROTOCOL

3.1 Basic Idea

The message space of FHE is a polynomial ring, and thus we can put integers in the coefficient or the degree of the polynomials. In XCMP, we encode $0 \le a, b < m$ in the degree of the polynomials. Remember that m is the parameter of the underlying FHE scheme which is usually set as a few thousands, e.g., $m \le 2^{13}$. Specifically, we use the encoding $\pi : \mathbb{Z} \mapsto \mathbb{A}_p$, given by $\pi(a) = X^a$. The core idea behind XCMP is to use the polynomial

$$C_0 = T_0 \times \pi(a) \times \pi(-b) \mod (X^m + 1), \tag{1}$$

where $T_0 = 1 + X + \cdots + X^{m-1}$. Notice that the polynomial with a negative degree X^{-b} is equivalent to $-X^{m-b}$ modulo $X^m + 1$. We argue that the 0-th coefficient $C_0[0] \in \{1, -1\}$. When $a \le b$, $C_0[0]$ comes from $T_0[b-a]$, and thus $C_0[0] = 1$. On the other hand, when $a > b$, $C_0[0]$ comes from the $(m-(a-b))$-th coefficient of T_0, but in this case $C_0[0] = -1$ due to the degree wrap around, i.e., $X^{m-(a-b)} \times X^{a-b} = -1 \mod X^m + 1$. In other words, $C_0[0] = 1$ if $a \le b$, else $C_0[0] = -1$.

Algorithm 1 Private Comparison XCMP.

Input: $[\![X^a]\!], [\![X^b]\!]$ where $a, b \in \mathbb{Z}_m$.
(1) Compute $\bar{\mu} = 2^{-1} \mod p$ and $\mu = -\bar{\mu} \mod p$
(2) Set the polynomial $T = \mu + \mu X + \cdots + \mu X^{m-1}$
(3) Sample polynomial $R \xleftarrow{\$} \mathbb{A}_p$ and set $R[0] = \bar{\mu}$
(4) Negate the degree $[\![X^{-b}]\!] \leftarrow [\![X^b]\!]$ (see Section 3.2.1)
(5) Compute $[\![C]\!] = [\![X^a]\!] \otimes [\![X^{-b}]\!] \otimes T \oplus R$
(6) Output $[\![C]\!]$

Algorithm 2 Comparison functionality $\mathcal{F}_{\mathsf{cmp}}$ from XCMP.

Input of encryptor: $a, b \in \mathbb{Z}_m$
Input of decryptor: sk
Input of evaluator, decryptor: $\emptyset$
Output of decryptor: $\mathbf{1}\{a > b\}$
Output of encryptor, evaluator: $\emptyset$
(1) Encryptor computes and sends $[\![X^a]\!], [\![X^b]\!]$ to evaluator.
(2) Evaluator invokes $[\![C]\!] \leftarrow \mathsf{XCMP}([\![X^a]\!], [\![X^b]\!])$, and sends the ciphertext $[\![C]\!]$ to decryptor.
(3) Decryptor decrypts $[\![C]\!]$ and outputs $C[0]$.

3.2 Private Comparison Protocol

We now present our private comparison XCMP in Algorithm 1. In Step 1 and Step 2, we compute a polynomial T which is used to parameterize the comparison result to 0 and 1. The details of how to negate the degree of X^b to X^{-b} homomorphically (Step 3) will be presented in the following section. In Step 5, we basically evaluate Equation 1 on the ciphertexts $[\![X^a]\!]$ and $[\![X^b]\!]$ with an additional random polynomial R added to the result. The coefficients of R are uniformly sampled from $\mathbb{Z}_p$ except that $R[0] = \bar{\mu}$ so that we hide the difference of a and b in the decryption of $[\![C]\!]$. The private implementation of Figure 1 is then directly constructed as Algorithm 2.

Theorem 3.1 (Correctness). *Algorithm 2 correctly implements the functionality $\mathcal{F}_{\mathsf{cmp}}$ of Figure 1.*

Proof. We consider when $a \le b$. In this case, we have exactly one term, i.e., μX^{b-a} of T which leads the 0-th coefficient of the polynomial $(X^a \times X^{-b} \times T)$ being μ. As a result, $C[0] = \mu + R[0] = \mu + \bar{\mu} = 0$. On the opposite side, we consider the case $a > b$. In this case, this coefficient becomes $-\mu$ because $\mu X^m = -\mu \mod X^m + 1$. As a result $C[0] = -\mu + R[0]$ which equals to $2\bar{\mu} = 1$. This completes our proof that $C[0] = \mathbf{1}\{a > b\}$. □

Theorem 3.2 (Privacy). *Algorithm 2 privately implements $\mathcal{F}_{\mathsf{cmp}}$ of Figure 1 in the semi-honest setting.*

The proof of this theorem is given in Appendix A.

The proposed method can work well for a relatively small domain, e.g., less than 16 bits, according to our experiments in Section 4. We argue that such a small domain might be insufficient for numerical computation, however, it is sufficient enough for many countable data. For example, the *Heart-disease Data set* and *Spambase Data set* [19] that are used in our experiments in Appendix B

3.2.1 Negating the Degree $[\![X^{-b}]\!] \leftarrow [\![X^b]\!]$. Indeed, Step 4 of Algorithm 1 can be avoided if the encryptor sends four ciphertexts $\{[\![X^a]\!], [\![X^{-a}]\!]\}$ and $\{[\![X^b]\!], [\![X^{-b}]\!]\}$ to the evaluator. However, this doubles the encryptor's computation and communication overhead. We now present a way that allows the evaluator to homomorphically compute $[\![X^{-b}]\!]$ from $[\![X^b]\!]$. We achieve this negate step by applying homomorphically the automorphism map $\mathcal{M} : X \mapsto X^{2m-1}$

$$\mathcal{M}(X^b) = X^{2mb-b} = (-1)^{2b} \cdot X^{-b} \mod X^m + 1.$$

This automorphism is possible on the FHE ciphertext since $2m-1 \in \mathbb{Z}^*_{2m}$. We refer to [13, 16] for more details about the automorphism on FHE ciphertexts.

3.2.2 Parametrisable XCMP. We can further parameterize $C[0]$ of Algorithm 2 as two distinct integers $\mu_0, \mu_1 \in \mathbb{Z}_p$. That is, $C[0] = \mu_0$ if $a \le b$, else $C[0] = \mu_1$. To do so, we need to change Step 1 of Algorithm 1 as $\bar{\mu} = (\mu_0 + \mu_1) \cdot 2^{-1} \mod p$ and $\mu = \mu_0 - \bar{\mu} \mod p$. The correctness proof in this setting is similar to the proof of Theorem 3.1 which is a special case of $\mu_0 = 0$ and $\mu_1 = 1$.

3.3 Preserving Homomorphism

Arithmetic addition and multiplication are still possible on the output ciphertexts of XCMP. To be precise, the resulting ciphertext from XCMP encrypts a polynomial of the form $C = \mathbf{1}\{a > b\} + \sum_{j=1}^{m-1} r_j X^j$ where r_js' are randomly generated integers. We can homomorphically sum the outputs from several calls of XCMP. For example, from the decryption of

$$\mathsf{XCMP}([\![X^a]\!], [\![X^{b_1}]\!]) \oplus \mathsf{XCMP}([\![X^a]\!], [\![X^{b_2}]\!]),$$

we can know $\mathbf{1}\{a > b_1\} + \mathbf{1}\{a > b_2\}$.

For the multiplication, we require some constraints. If we operate directly $\mathsf{XCMP}([\![X^a]\!], [\![X^{b_1}]\!]) \otimes \mathsf{XCMP}([\![X^a]\!], [\![X^{b_2}]\!])$, we can not obtain any meaningful result due to the random coefficients introduced in XCMP. In other words, we can not multiply the outputs from several calls of XCMP directly. On the other hand, we can only multiply the encryption of an integer $e \in \mathbb{Z}_p$ to the encrypted result of XCMP, giving a ciphertext $[\![e \cdot C]\!]$. Thereby, after the decryption, we can obtain a meaningful result $e \cdot \mathbf{1}\{a > b\}$ from the 0-th coefficient.

3.4 Feasible Domain Extension

By exploiting the polynomial structure, our protocol can efficiently compare two encrypted integers. However, the feasible input domain of XCMP is relatively small, that is $\mathbb{Z}_m$, and this is the major limitation of XCMP. We now present a method to expand the feasible domain of XCMP to $\mathbb{Z}_{m^2}$ while the non-interactive and output expressive properties of XCMP remain unchanged.

Our idea is very simple, that is to separately compare each digit of the inputs, from the most significant digit down to the least significant one. More precisely, an integer $a' \in \mathbb{Z}_{m^2}$ is partitioned into two digits $0 \le a'_1, a'_0 < m$ such that $a' = a'_1 \cdot m + a'_0$. We can see that $\mathbf{1}\{a' > b'\}$ is equivalent to

$$\mathbf{1}\{a'_1 \neq b'_1\} \cdot (\mathbf{1}\{a'_1 > b'_1\} - \mathbf{1}\{a'_0 > b'_0\}) + \mathbf{1}\{a'_0 > b'_0\}. \quad (2)$$

The two comparisons in Equation 2 can be done through XCMP, and the preserved additive homomorphism of XCMP allows us to perform the subtraction and addition. The main challenge for evaluating this equation lies in the computation of the bit $\mathbf{1}\{a'_1 \neq b'_1\}$, and to multiply this bit to the output of XCMP.

We now show how to compute $[\![\mathbf{1}\{a'_1 \neq b'_1\}]\!]$ from the ciphertexts $[\![X^{a'_1}]\!]$ and $[\![X^{b'_1}]\!]$. We also present a method to convert this ciphertext to a XCMP-compatible form so that we can homomorphically multiply $[\![\mathbf{1}\{a'_1 \neq b'_1\}]\!]$ to the private comparison result, i.e.,

$$\mathsf{XCMP}([\![X^{a'_1}]\!], [\![X^{b'_1}]\!]) \ominus \mathsf{XCMP}([\![X^{a'_0}]\!], [\![X^{b'_0}]\!]).$$

3.4.1 Inequality Test. We can simply compute inequality test through $[\![A]\!] = 1 \ominus [\![X^{a'_1}]\!] \otimes [\![X^{-b'_1}]\!]$. Specifically, when $a'_1 \neq b'_1$, A is a polynomial of $1 - X^{a'_1 - b'_1} \neq 0$. If $a'_1 = b'_1$, A downgrades to zero. In other words, the 0-th coefficient of A gives the inequality bit $\mathbf{1}\{a'_1 \neq b'_1\}$.

3.4.2 Converting to XCMP*-Compatible Form.* An encrypted integer value can be multiplied to the output of XCMP, and thus is XCMP-compatible. However, the ciphertext $[\![A]\!]$ above is not XCMP-compatible, that is because $[\![A]\!]$ might encrypt a polynomial more than an integer. Thus, we need to convert $[\![A]\!]$ to an encrypted integer value if $A \neq 0$ (i.e., when $a'_1 \neq b'_1$), otherwise we should obtain a ciphertext of zero. We now show how to do this conversion.

The core idea is to employ Fermat's little theorem under the polynomial modulo $X^m + 1$. That is $A^{p^d - 1} = 1 \mod X^m + 1$ if $A \neq 0$, where $p^d = 1 \mod m$. When $A = 0$, A^{p^d-1} is still zero. The naive way needs a multiplicative depth of $d \log p$ for computing $[\![A^{p^d-1}]\!]$ from $[\![A]\!]$. According to [16, 23], we can use automorphisms to reduce this depth. That is, we first compute the exponent $[\![A^{p-1}]\!]$ from $[\![A]\!]$ which needs a multiplicative depth of $\log_2 p$. Then we use d automorphism maps $\mathcal{K}_j : X \mapsto X^{p^{j-1}}$ for $j \in [d]$. Finally, the product of d ciphertexts gives $[\![A^{p^d-1}]\!]$

$$\prod_{j\in[d]} \mathcal{K}_j([\![A^{p-1}]\!]) = \prod_{j\in[d]} [\![A^{(p-1)p^{j-1}}]\!] = [\![A^{(p-1)\sum_{j\in[d]}(p^{j-1})}]\!] = [\![A^{p^d-1}]\!]$$

In total, we need a multiplicative depth of $\log_2 p + \log_2 d$ and d automorphisms for this conversion step.

The FHE parameter p defines the message space which is usually determined by the application scenario. In order to limit the overhead of this conversion step, we tend choosing such p that d is a small value, e.g., $d = 2$. The domain extension procedure is still costly compared to XCMP itself, and we intend to improve it in the future work.

4 EVALUATION

Implementation Details. We instantiated XCMP using HElib [23] and SEAL [6] separately. SEAL and HElib use a different representation for representing the plain polynomials from $\mathbb{Z}_p[X]/(X^m + 1)$. HElib's representation for plain polynomials needs more computation effort than SEAL. As a result, for the ciphertext-plaintext comparison case, XCMP instantiated from SEAL can provide a better evaluation performance than the HElib based one. However, HElib allows automorphisms which are not supported in SEAL

Table 3: Experimentally comparing XCMP with previous HE-based private comparison protocols. δ denotes the input bit length. κ is the security parameter. FF and ECC mean that the protocols were instantiated with a finite filed scheme and an elliptic curve scheme, respectively. Timing numbers were measured as the mean of a thousand runs with extra fifty warm up runs.

δ, κ	Method	ENC (ms)	EVAL (ms)		DEC (ms)
			two ciphertexts inputs	single ciphertext input	
12, 80	[11] (λ = 20, FF)	18.63 ± 0.388	574.44 ± 6.563	-	2.93 ± 0.546
	[11] (λ = 40, FF)	18.87 ± 0.488	1164.23 ± 10.834	-	3.291 ± 0.715
	[1] (FF)	22.27 ± 0.776	-	115.98 ± 3.658	20.54 ± 0.701
	[9] (FF)	26.50 ± 0.217	-	54.24 ± 0.377	16.99 ± 0.113
	[9] (ECC)	0.83 ± 0.038	-	**0.799 ± 0.037**	0.45 ± 0.016
	Ours (HElib)	2.93 ± 0.069	**13.07 ± 0.114**	2.28 ± 0.112	1.68 ± 0.060
	Ours (SEAL)	6.97 ± 0.102	automorphism not supported	1.00 ± 0.021	0.60 ± 0.015
13, 128	[11] (λ = 20, FF)	21.77 ± 0.281	672.96 ± 4.678	-	63.80 ± 12.365
	[11] (λ = 40, FF)	21.78 ± 0.146	1346.97 ± 10.167	-	64.97 ± 13.562
	[1] (FF)	437.77 ± 0.399	-	2224.54 ± 1.433	410.32 ± 0.354
	[9] (FF)	352.24 ± 16.569	-	705.61 ± 17.857	340.58 ± 15.749
	[9] (ECC)	2.11 ± 0.150	-	**1.98 ± 0.145**	1.10 ± 0.074
	Ours (HElib)	6.17 ± 0.195	**28.22 ± 0.843**	4.68 ± 0.247	3.65 ± 0.138
	Ours (SEAL)	16.54 ± 0.237	automorphism not supported	4.04 ± 0.022	2.20 ± 0.019

yet. The automorphism allows the "degree-negating" operation described in Section 3.2.1.

We implement the HE based comparison protocols of [1, 9, 11] using the GMP library. The method in [11] uses the GM encryption scheme with the AND-gate extension technique [22]. The protocols of [1, 9] use an additively HE which was instantiated as the Paillier encryption [20]. We also instantiated [9] with lifted ElGamal over elliptic curves [24].

We used two security parameters $\kappa \in \{80, 128\}$. We set the public key sizes of $\{1024, 3072\}$-bits for the GM encryption and Paillier encryption, and $\{160, 256\}$-bits for the elliptic curve. In addition, we set $\lambda \in \{20, 40\}$ in the probabilistic method of [11]. For the FHE scheme, we set the polynomial degree $m = 2^{12}$ and $m = 2^{13}$.

All the programs were written in C++ and compiled with g++-6.3. We ran experiments on a machine equipped with an Intel Xeon E5-2640 v3@2.60 GHz CPU and 32GB RAM running Ubuntu 14.04 [1].

Measurements. We ran benchmarks for the performance of XCMP and previous non-interactive private comparison methods as described in Table 1. Specifically, we measured the computation time of three parts: 1) time for encrypting the inputs, 2) time for evaluating comparison, and 3) time for decrypting the resulting ciphertext(s).

4.1 Comparison with TFHE

Private comparison, e.g., [7], that encrypts each bit of a and b using FHE satisfies both the non-interactive and output expressive requirements. However, for a relative small domain, e.g., less than 16 bits, these bit-wise FHE solutions might take a longer evaluation time than XCMP due to the bootstrapping operation. We compared the evaluation time of XCMP with the bit-wise FHE solution from TFHE [7]. The results are shown in Figure 2. For bit length $\delta \in \{8, 9, 10\}$, we used the same parameter $m = 2^{10}$ for

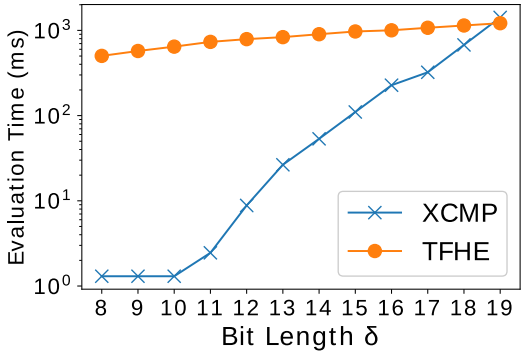

Figure 2: Evaluation time of XCMP and the bit-wise solution from TFHE [7] for comparing two encrypted values aspect to various bit length.

XCMP due to the 80-bit security level requirement. We can see that the performance of the TFHE method increased linearly with the bit length while our method increased exponentially. Also, the current TFHE implementation is a symmetric encryption scheme while XCMP was instantiated with an asymmetric scheme. Even though, for small domains (i.e., less than 16-bits), our method was more than 5 – 500 times faster than TFHE. We admit that for a larger input bit length, the TFHE method will outperform XCMP, while for applications that require only a relatively small domain, XCMP can be more efficient.

[1] Source codes are available from https://github.com/fionser/XCMP.

4.2 Comparison with HE-based Methods

The experimental results are reported in Table 3. The probabilistic method of [11] is able to compare two encrypted integers within an error rate of $2^{-\lambda}$. The complexity of this method is linear in λ. As a result, the method of [11] has high computation and communication complexity for a negligible error rate, e.g., $\lambda = 40$.

For the FF-based implementations, our method outperformed [1, 9, 11] in terms of evaluation time. We have two settings, that is two-ciphertexts case, and single-ciphertext case. In the first case, XCMP was about $45 - 90$ times faster than [11]. On the other hand, in the second case, XCMP can be $54 - 175$ times faster than [1, 9] in terms of evaluation time. Notice that, when only one input was encrypted, XCMP itself was even about $5 - 6$ times faster than with two encrypted inputs.

Our method took more evaluation time than the ECC-based implementations of [9], but the performance gap was less than one order of magnitude. Note that the elliptic curve library (i.e., [24]) we used is better optimized than HElib. By using the more recent FHE library SEAL, we reduced this performance gap. We thus conclude that our comparison algorithm that exploits the structure of the polynomial ring $X^m + 1$ is efficient. Remember that the output of XCMP still offers additive homomorphism, and multiplicative homomorphism under a certain condition. This property is absent in all previous HE-based solutions and is helpful for constructing a higher level protocol beyond integer comparison. We provide a concrete example of using this property for private decision tree evaluation in Appendix B.

Acknowledgment. We would like to thank the anonymous reviewers for their helpful comments. This work was supported by JST CREST with grant number JPMJCR1302 and Grants-in-Aid for Scientific Research with grant number 16H02864.

REFERENCES

[1] Ian F. Blake and Vladimir Kolesnikov. December 5-9, 2004. Strong Conditional Oblivious Transfer and Computing on Intervals. In *ASIACRYPT 2004, Jeju Islan, Korea*. 515–529.
[2] Raphael Bost, Raluca Ada Popa, Stephen Tu, and Shafi Goldwasser. February 8-11, 2015. Machine Learning Classification over Encrypted Data.. In *22nd Annual Network and Distributed System Security Symposium NDSS*. San Diego, CA, USA.
[3] Zvika Brakerski, Craig Gentry, and Vinod Vaikuntanathan. January 8-10, 2012. (Leveled) fully homomorphic encryption without bootstrapping. In *Innovations in Theoretical Computer Science 2012*. Cambridge, MA, USA, 309–325.
[4] Ran Canetti. 2000. Security and Composition of Multiparty Cryptographic Protocols. *J. Cryptology* 13, 1 (2000), 143–202.
[5] Octavian Catrina and Sebastiaan de Hoogh. September 13-15, 2010.. Improved Primitives for Secure Multiparty Integer Computation. In *SCN 2010, Amalfi, Italy*. 182–199.
[6] Hao Chen, Kim Laine, and Rachel Player. 2017. Simple Encrypted Arithmetic Library - SEAL v2.1. *IACR Cryptology ePrint Archive* 2017 (2017), 224.
[7] Ilaria Chillotti, Nicolas Gama, Mariya Georgieva, and Malika Izabachène. December 4-8, 2016. Faster Fully Homomorphic Encryption: Bootstrapping in Less Than 0.1 Seconds. In *ASIACRYPT 2016, Hamoi, Vietnam*. 3–33.
[8] Geoffroy Couteau. 2016. Efficient Secure Comparison Protocols. *IACR Cryptology ePrint Archive* 2016 (2016), 544.
[9] Ivan Damgård, Martin Geisler, and Mikkel Krøigaard. 2009. A correction to 'efficient and secure comparison for on-line auctions'. *International Journal of Advancements in Computing Technology (IJACT)* 1, 4 (2009), 323–324.
[10] Ivan Damgård, Martin Geisler, and Mikkel Krøigaard. July 2-4, 2007. Efficient and Secure Comparison for On-Line Auctions. In *ACISP 2007, Townsville, Australia*. 416–430.
[11] Marc Fischlin. April 8-12, 2001. A Cost-Effective Pay-Per-Multiplication Comparison Method for Millionaires. In *CT-RSA 2001, San Francisco, CA, USA*. 457–472.
[12] Craig Gentry, Shai Halevi, Charanjit S. Jutla, and Mariana Raykova. Jun 2-5, 2015. Private Database Access with HE-over-ORAM Architecture. In *ACNS 2015, New York, NY, USA*. 172–191.
[13] Craig Gentry, Shai Halevi, and Nigel P. Smart. April 15-19, 2012. Fully Homomorphic Encryption with Polylog Overhead. In *EUROCRYPT 2012, Cambridge, UK*. 465–482.
[14] Craig Gentry, Shai Halevi, and Nigel P. Smart. August 19-23, 2012. Homomorphic Evaluation of the AES Circuit. In *CRYPTO 2012, Santa Barbara, CA, USA*. 850–867.
[15] Oded Goldreich. 2009. *Foundations of cryptography: volume 2, basic applications*. Cambridge University Press.
[16] Shai Halevi and Victor Shoup. August 17-21, 2014. Algorithms in HElib. In *CRYPTO 2014, Santa Barbara, CA, USA*. 554–571.
[17] Vladimir Kolesnikov, Ahmad-Reza Sadeghi, and Thomas Schneider. December 12-14, 2009. Improved Garbled Circuit Building Blocks and Applications to Auctions and Computing Minima. In *CANS 2009, Kanazawa, Japan*. 1–20.
[18] Vladimir Kolesnikov and Thomas Schneider. July 7-11, 2008. Improved Garbled Circuit: Free XOR Gates and Applications. In *Automata, Languages and Programming, 35th International Colloquium, Proceedings, Part II*. Reykjavik, Iceland, 486–498.
[19] M. Lichman. 2013. UCI Machine Learning Repository. (2013). http://archive.ics.uci.edu/ml
[20] Pascal Paillier. 1999. Public-key cryptosystems based on composite degree residuosity classes. In *International Conference on the Theory and Applications of Cryptographic Techniques*. 223–238.
[21] Ahmad-Reza Sadeghi, Thomas Schneider, and Immo Wehrenberg. December 2-4, 2009. Efficient Privacy-Preserving Face Recognition. In *ICISC 2009, Seoul, Korea*. 229–244.
[22] Tomas Sander, Adam L. Young, and Moti Yung. 17-18 October, 1999. Non-Interactive CryptoComputing For NC^1. In *40th Annual Symposium on Foundations of Computer Science, FOCS '99, New York, NY, USA*. 554–567.
[23] Victor Shoup Shai Halevi. 2017. HELib. http://shaih.github.io/HElib. (2017).
[24] Misunari Shigeo. 2017. mcl: a generic and fast pairing-based cryptography library. https://github.com/herumi/mcl. (2017). Accessed: 2017-11-10.
[25] Raymond K. H. Tai, Jack P. K. Ma, Yongjun Zhao, and Sherman S. M. Chow. September 11-15, 2017. Privacy-Preserving Decision Trees Evaluation via Linear Functions. In *ESORICS 2017, Oslo, Norway*. 494–512.
[26] Thijs Veugen. December 2-5, 2012. Improving the DGK comparison protocol. In *WIFS 2012, Costa Adeje, Tenerife*. 49–54.

A SECURITY MODEL

Our protocols are private under the semi-honest assumption, that is the stakeholders follow the protocol specification. Our security definitions follow the real world/ideal world paradigm of [4, 15]. Specifically, we compare the protocol execution in the real world to an execution in an ideal world. In the real world, protocol players follow the specification of a protocol Π, and in the ideal world, protocol players have access to a trusted third party that evaluates the functionality. The protocol execution is viewed as occurring in the existence of an adversary $\mathcal{A}$ and cooperated with an environment $C = \{C_\kappa\}$ which is modeled as a class of polynomial-size circuits parameterized by a security parameter κ. The role of the environment is to choose the input to the protocol execution and to distinguish experiments in the real world and the ideal world. Informally, a protocol Π privately implements a functionality $\mathcal{F}$ if for any polynomial-sized circuit C, it can not distinguish between the real and ideal world experiments.

A.1 Proof of Theorem 3.2

We now prove Theorem 3.2.

Proof (sketch). The view of the evaluator during the protocol execution consists of only ciphertexts, and thus the security against an adversary evaluator is reduced to the semantic security of the FHE scheme. On the other hand, the view of the decryptor consists of one polynomial C whose constant term is the comparison result. Beside this constant term, all other terms of C are randomized through a uniform random polynomial R, and thus the coefficients of these terms are distributed uniformly in $\mathbb{Z}_p$. □

Table 4: Data sets used for the PPDT evaluation. γ and N respectively denotes the number of features and the number of internal nodes of the decision tree.

	Heart-disease	Housing	Spambase	Artificial
γ	13	13	57	16
N	5	92	58	500

B PRIVACY-PRESERVING DECISION TREE (PPDT) EVALUATION

Decision tree is a fundamental and popularly used classification algorithm, in which a number of comparisons are evaluated between the elements of an input vector $\boldsymbol{a} \in \mathbb{Z}_p^{\gamma}$ and N thresholds $\{\tau_k \in \mathbb{Z}_p\}_{k\in[N]}$. According to the comparison results, a label that indicates the category of $\boldsymbol{a}$ is obtained. The entity that owns the decision tree model (i.e., $\{\tau_k \in \mathbb{Z}_p\}_{k\in[N]}$) is called model holder. Privacy of the decision tree model requires the values of τ_k to be kept secret.

PPDT protocols such as [2, 25] are usually designed as a two-party computation protocol, in which a client sends its input $\boldsymbol{a}$ to a model holder, who holds the thresholds $\{\tau_k\}$, for the classification label of $\boldsymbol{a}$. The PPDT protocol offers privacy to both the client and the model holder. That is, after the execution of the PPDT protocol, the model holder does not learn about the client's input $\boldsymbol{a}$, except the number of features (i.e., input privacy), while the client learns nothing about the thresholds $\{\tau_k\}$, except the result label (i.e., model privacy). Notice that the existing PPDT protocols requires multiple rounds of interaction between the model holder and the client.

New Non-interactive PPDT. We can obtain a non-interactive variant protocol of [25] by instantiating the private comparison with XCMP, as opposed to [25] which used the DGK's private comparison protocol [9]. The other parts of our PPDT protocol are identical to the semi-honest protocol of [25], and thus we refer to their paper for the details. The lack of output expressiveness of [9] forced [25] to use multi-rounds of interactions between the client and the model holder. Thereby, in [25], the client cannot be offline during the protocol execution. Also, the DGK protocol [9] only works for one encrypted input. As a result, the thresholds $\{\tau_k\}$ must be known to the evaluator (i.e., the model holder himself in [25]).

By replacing the DGK's private comparison with XCMP, our protocol is superior to its origin [25] in two points. The first is that our protocol allows the client to be offline after he has sent the ciphertexts of $\boldsymbol{a}$ to the evaluator. The second is that our PPDT protocol allows the model holder to delegate the evaluation to a third-party evaluator, e.g., a public cloud server, without compromising the model privacy. That is, the model holder can encrypt the thresholds $\{\tau_k\}$, and place the ciphertexts on the public cloud. Then the cloud can process classification upon the (encrypted) request from the client using the ciphertexts of $\{\tau_k\}$ for the model holder. Also, during the classification process, no interaction between the cloud, the model holder and the clients is needed.

B.1 Evaluation

We trained decision tree models through the scikit-learn library on three real data sets, heart-disease (HD), housing (HG), and spambase (SP) from the UCI repository [19]. We also used an artificial data to show the scalability of our PPDT protocol. The number of features and internal nodes of the trained models are summarized in Table 4. Note that in the training phase was done on plain data without any crypto. We used the same machines mentioned in Section 4 for the following experiments.

Measurements. We measured the performance of our PPDT protocol for the case that both the client's input and the tree model are encrypted. In particular, we measured the encryption and decryption time on the client's side, and the evaluation time on the cloud's side.

Empirical Results. The benchmark performance of our PPDT protocol on the trained models are given in Table 5. The evaluation time on the server's side was linear with the number of internal nodes N, which can be accelerated easily through multi-threads. However, the client needs to decrypt $O(2N)$ ciphertexts which still takes high computation effort.

Comparison with the-State-of-the-Art. We also compared our PPDT protocol to its origin [25], which is the most efficient HE-based PPDT protocol. Remind that, in [25], the model thresholds $\{\tau_k\}$ were plaintext due to the DGK's private comparison, while in our PPDT protocol both the client's input and the model thresholds were encrypted. The performance details are shown in Table 6. We can see that the performance gap between our PPDT protocol and [25], was smaller than than 25-fold, counting the differences of bit length δ. Our PPDT protocol can be hosted on a public cloud. Thus we can take advantage of the abundant computing resources of the cloud to easily accelerate the protocol execution, e.g., by parallelism.

Table 5: Performance of our PPDT protocol on the trained decision tree model. The feasible domain of XCMP was $m = 2^{13}$.

Data set	ENC	EVAL	DEC
HD	44.18 ± 2.59 ms	0.32 ± 0.06 s	0.32 ± 0.06 s
HG	46.35 ± 3.71 ms	5.11 ± 0.08 s	5.12 ± 0.08 s
SP	193.58 ± 7.27 ms	3.34 ± 0.07 s	3.35 ± 0.07 s
ART	54.62 ± 1.96 ms	28.16 ± 0.61 s	28.15 ± 0.61 s

Table 6: Comparing total computation time of our PPDT with [25]. γ and N respectively denotes the number of features and the number of internal nodes of the decision tree. δ is the input bit length.

Data set	(γ, N)	Ours ($\delta = 13$)	[25] ($\delta = 64$)
Heart-disease	(13, 5)	0.59s	0.25s
Housing	(13, 92)	10.27s	1.98s
Spambase	(57, 58)	6.88s	1.80s
Artificial	(16, 500)	56.37s	10.42s