A Taxonomy of Secure Two-party Comparison Protocols and Efficient Constructions

Nuttapong Attrapadung[†], Goichiro Hanaoka[†], Shinsaku Kiyomoto[*], Tomoaki Mimoto[*], Jacob C. N. Schuldt[†]
[*]KDDI Research, Inc
2-1-15 Ohara, Fujimino, Saitama 356-8502, Japan
[†]National Institute of Advanced Industrial Science and Technology
2-3-26 Aomi, Koto-ku, Tokyo 135-0064, Japan

***Abstract*—Secure two-party comparison plays a crucial role in many privacy-preserving applications, such as privacy-preserving data mining and machine learning. In particular, the available comparison protocols with the appropriate input/output configuration have a significant impact on the performance of these applications. In this paper, we firstly describe a taxonomy of secure two-party comparison protocols which allows us to describe the different configurations used for these protocols in a systematic manner. This taxonomy leads to a total of 216 types of comparison protocols. We then describe conversions among these types. While these conversions are based on known techniques and have explicitly or implicitly been considered previously, we show that a combination of these conversion techniques can be used to convert a perhaps less-known two-party comparison protocol by Nergiz et al. (IEEE SocialCom 2010) into a very efficient protocol in a configuration where the two parties hold shares of the values being compared, and obtain a share of the comparison result. This setting is often used in multi-party computation protocols, and hence in many privacy-preserving applications as well. We furthermore implement the protocol and measure its performance. Our measurement suggests that the protocol outperforms the previously proposed protocols for this input/output configuration, when off- line pre-computation is not permitted.**

I. Introduction

Multi-party computation (MPC) is a powerful cryptographic tool often used to obtain privacy-preserving applications such as privacy-preserving data mining. In general, MPC enables a set of parties to jointly compute a function of their private inputs i.e. for a function f, n parties each holding private input x_i ($i = 1, \ldots, n$), are able to compute $f(x_1, \ldots, x_n)$ without having to reveal their private inputs x_i. The security guarantee provided by MPC is quite strong; a party l obtaining $f(x_1, \ldots, x_n)$ as part of the protocol will learn nothing about the inputs x_i for $i \neq l$, except the information that can be derived from $f(x_1, \ldots, x_n)$ and x_l. In other words, MPC provides the best possible guarantee regarding input privacy, which is a highly desirable property in privacy-preserving data mining.

While MPC can theoretically be realized for any function f [1], [2], the resulting protocols are often too complex and inefficient to be used in practice. This has led to the development of bespoke MPC protocols that efficiently implements specific functionalities required by various privacy-preserving applications. For example, Bunn and Ostrovsky [3] proposed a two-party k-means clustering algorithm that enables two servers, each holding a separate data set, to compute a k-means clustering of their data sets combined without having to disclose individual data points to the other server. (Note that k-means clustering is a very commonly used technique in data mining and machine learning, e.g. see [4] for a discussion.) Bost et al. [5] proposed various protocols achieving privacy-preserving machine learning classification, in which a server holding model M and a client holding an input x, can jointly classify x according to M without the server revealing M to the client or the client revealing x to the server. Other examples include privacy-preserving biometric data matching [6], [7] and privacy-preserving recommender systems [8]. A common approach to the design of these protocols is to use a modular design in combination with efficient sub-protocols for specific low-level operations.

In this paper, we focus on efficient *two-party comparison* protocols. Comparison plays a fundamental role in computation, and secure comparison protocols are frequently used sub-protocols in the design of efficient MPC protocols for more complex functionalities and privacy-preserving protocols. In particular, comparison plays a crucial role in the above mentioned protocols for k-means clustering and machine learning classification. However, depending on how the values to be compared are stored by the parties, different types of comparison protocols are needed. Unfortunately, these different "types" are scattered throughout the vast research literature of secure computation. This makes it somewhat hard to find an efficient construction which can be used in a particular target applications, and might lead to comparison protocols being designed anew each time whenever a slightly less common configuration is required. In this paper, we show that efficient comparison protocols for a very wide range of configurations can obtained via conversions of existing protocols.

A. Our Contribution

We firstly propose a "taxonomy" that aims to systematize how one can describe each specific type of two-party comparison protocols. Specifically, given values x and y which are to be compared, a number of different scenarios often occur: for example, x and y might be stored in the clear by different parties, each party might hold a secrete share of both x and y, or one party might hold the encryption of both x and y whereas the other party might hold the corresponding secret key. Further variations are obtained when considering the output of the protocol which might be required to be obtained by only one party, might be required to be secret shared among the parties, or might be required to be in encrypted form. More precisely, we identify 6 types of "semantics" for data stored by the two parties, for each of the three values considered in a comparison protocol, namely the two inputs x, y and the comparison result which is the output of the protocol. This overall give us $6^3 = 216$ different types of comparison protocols.

We then describe conversions among these types of protocols. The conversions are either based on known techniques, have previously been considered, or are implicitly used in the literature. However, these conversions allows us to construct a comparison protocol for one setting by converting an existing protocol from a different setting. We describe more details on motivations of having such conversions below (Section I-C).

We complete our framework by suggesting a specific "base" protocol; using this in combination with the above mentioned conversions allows to obtain an efficient comparison protocol for any of the configurations captured in our taxonomy. We do not directly construct a new base protocol but rather recall a perhaps less-known protocol proposed by Nergiz et al. [9].[1] This protocol has the nice property that it is simple due to its combinatorial nature (as it uses tree-based structures) and the processing can be done in *parallel*, although this was not pointed out in the original paper [9].

Finally, we end our discussion by focusing on the particular setting in which the values x and y are secret shared among the two parties, and the result of the protocol is required to also be secret shared among the parties. We will refer to this setting as *Type 1* in the paper. This setting is the most common setting in generic 2PC/MPC protocols, which makes composition with existing protocols easy. We note that, in this setting, Nishide and Ohta [11], Catrina and de Hoogh [12], and Garay, Shoenmakers and Villegas [13] have proposed efficient protocols. However, all of these require a computationally heavy off-line phase. On the other hand, a protocol that requires no off-line phase can be obtained via generic MPC techniques, but as highlighted above, this yields a relatively inefficient protocol. This leaves the DGK comparison protocol by Damgård, Geisler and Kroigaard [14] as the best candidate so far for an efficient comparison protocol (in Type 1) without the need for an off-line phase.[2] We construct and implement a protocol in Type 1 from our framework, namely, starting with the above base protocol of [9] and applying a combination of conversions, we obtain a Type 1 protocol. We then measure the performance of our protocol, and compare this with the measurements of the (optimized) DGK protocol from [10]. This suggests that our protocol outperforms the DKG protocol in computation time (thanks to our observation on the parallel computation property of the base protocol) while retaining almost the same communication cost. Furthermore, we also show that our protocol is more efficient than the Type-1 protocol by Bunn and Ostrovsky [3] (used as a part of their secure k-means clustering protocol) and the more recent protocol by Gentry et al. [15] (used as a part of their ORAM protocol), converted to Type 1.

B. Related work

Research on secure comparison protocols have a vast literature, e.g., [1], [16], [17], [18], [11], [13], [14], [19], [12], [20], and we would like to point the reader to an excellent survey published recently in 2015 by Veugen et al. [10] for a detailed overview. Regarding the "categorization" of comparison protocols, which is our main theme, there is one notable related work by Bost et al., in which they describe syntax and schemes for 5 configurations mostly consisting of encrypted input/output settings. All of these are also included in our categorization (among the 216 types) as well.

C. Motivations for Conversions

As described above, a direct benefit of conversions is that one can obtain a new type of secure comparison protocol without having to design this from scratch, simply by identifying an existing protocol with the desired properties, and then applying the appropriate conversions.

Additionally, it is useful to have a conversion that converts one type of comparison protocol that is perhaps "easier" to design into another type that is seemingly more difficult to design. For example, a protocol with *encrypted* inputs is arguably more difficult to design than a protocol with *plain* inputs. Intuitively, this is since, for plain inputs, each party can construct the bit decomposition of the inputs directly, and from this, comparison can simply be done via a boolean comparison circuit. Veugen also already described this intuition in [20]. (In this paper, we use more efficient representation, which is the tree-based point/range encoding.) As illustrated by the results in this paper, this approach might also lead to protocols with performance advantages, if a particular setting allows

[1] This protocol does not appear to be cited by many papers and did not appear in the recent state-of-the-art survey paper by Veugen et. al. [10], which suggests that it is somewhat less known.

[2] This is to the best of our knowledge, and is also supported by the performance comparisons in the survey paper of [10].

very efficient instantiations of comparison protocols to be constructed.

II. Preliminaries

A. Homomorphic Encryption

We use additive homomorphic encryption as a building block. The syntax consists of the following algorithms. The key generation algorithm outputs a public key pk and a secret key sk, where we write $(pk, sk) \leftarrow \mathsf{KGen}$. The encryption algorithm encrypts a message $X \in \mathcal{M}$, where a ring $\mathcal{M}$ is the message space, to a ciphertext $C = Enc_{pk}(X)$. The decryption algorithm decrypts C to X, using sk. The additive homomorphic property allows us to homomorphically add encrypted messages as $Enc_{pk}(M_1) + Enc_{pk}(M_2)$ which is decrypted to $M_1 + M_2$, and to multiply by a known value r as $r \cdot Enc_{pk}(M)$ which is decrypted to $r \cdot M$.

B. Secure Two-party Computation

In this work, we consider secure two-party computation for comparison functionalities. We label the two parties as Alice and Bob. We refer syntax and security to, e.g., [21], but briefly give some intuition here. The correctness and security are completely defined by the inputs and the outputs of Alice and Bob. In particular, the security requires that each party should not be able to learn anything beyond what can be inferred from its own input and output. We consider the semi-honest adversary model, where we assume a computationally bounded adversary who tries to learn additional information from the messages seen during the protocol execution. In contrast to the stronger malicious adversary, the semi-honest adversary is not allowed to deviate from the protocol.

III. A Taxonomy of Comparison Protocols

A. Types of Comparison Protocols

In a comparison protocol, we have two integer inputs x, y, where we assume $0 \leq x, y < n$, and an output $\delta \in \{0, 1\}$ defined as

$$\delta = \begin{cases} 1 & \text{if } x \geq y \\ 0 & \text{if } x < y \end{cases}.$$

Here, the integer n denotes the bound, and we usually assume n to be a power of 2, say $n = 2^\ell$.

Types of Protocols. We categorize types of comparison protocols by the following criteria:

- semantic of x
- semantic of y
- semantic of δ

where the "semantic" of a value z (which can be either x, y, δ) is defined as a pair of values; the first value is the data regarding z held by Alice whereas the second is the data regarding z held by Bob. We require these values (Alice's data and Bob's data) can be combined to recover z efficiently. [3]

We refer to the semantic type of the pair

$$(\text{Alice's data regarding } z,\ \text{Bob's data regarding } z)$$

using the following two-letter abbreviations.

1) ES: (Encrypted data, Secret key),
2) SE: (Secret key , Encrypted data),
3) SS: (Shared data , Shared data),
4) PN: (Plain data , None),
5) NP: (None , Plain data),
6) PP: (Plain data , Plain data).

The above semantic types are defined as follows.

1) ES-semantic:

   Alice holds $E_{pk}(z)$, Bob holds sk

   where $(pk, sk) \leftarrow \mathsf{KGen}$ is generated by Bob.
2) SE-semantic:

   Alice holds sk, Bob holds $E_{pk}(z)$

   where $(pk, sk) \leftarrow \mathsf{KGen}$ is generated by Alice.
3) SS-semantic:

   Alice holds z^A, Bob holds z^B

   where $z^A, z^B \in \mathbb{Z}_N$ and $z^A + z^B \equiv z \bmod N$. Here, N is a known integer. For inputs x, y, we assume N to be much larger than n. For an output δ, we use $N = 2$ (hence, the addition then becomes XOR).
4) PN-semantic:

   Alice holds z, Bob holds nothing
5) NP-semantic:

   Align holds nothing, Bob holds z
6) PP-semantic:

   Align holds z, Bob holds z

Categorization. We can systematically specify a protocol type as

XX-YY-DD

where each $\text{XX}, \text{YY}, \text{DD} \in \mathcal{S} := \{\text{ES}, \text{SE}, \text{SS}, \text{PN}, \text{NP}, \text{PP}\}$ represents the semantic of x, y, δ, respectively. *Therefore, overall we obtain* $6^3 = 216$ *protocol types.*

A few of the combinations will be trivial to construct (and, as a result, carry no practical meaning of secure computation), as we conclude in the following lemma.

Lemma 1. *Consider any choice of DD* $\in \mathcal{S}$*, we have that PN-PN-DD, PP-PN-DD, PN-PP-DD, NP-NP-DD, PP-NP-DD, NP-PP-DD, PP-PP-DD are trivial.*

[3] Each value can be thought of as a "share" of z as in a secret sharing scheme where Alice and Bob will hold a share each. However, we do not directly use the terminology of secret sharing since not all of our semantics fit into the formal syntax of secret sharing; one example is the plain-input setting.

The proof is straightforward. For example, PN-PN-DD is trivial since Alice knows both x, y. The other cases can be argued similarly.

Looking forward, in this paper, by combining the base protocol in Section V and conversions in Section IV, we will have an efficient secure construction for all the remaining non-trivial $216 - 7 \times 6 = 174$ types.

B. Concrete Examples for Commonly Used Types

For concreteness, we list the *commonly used* types:

- Type 1: SS-SS-SS,
- Type 2: SE-SE-SE,
- Type 3: PN-NP-SE,
- Type 4: PN-NP-SS,
- Type 5: PN-NP-NP,

where, for example, Type 1 is considered in [3], [11], [13], [14], [12], Type 2 in in [14], [5], Type 3 in [5], while Type 5 is the original Yao's millionaire protocol [22] and is also considered in [9], who consider Type 4 as well.

We also consider some types which have not been considered previously in the literature but which could potentially be useful in protocols using multiple types of "shared" data:

- Type 6: PN-SE-SS,
- Type 7: PP-SS-ES.

We provide explicit descriptions of these below.

Type 1: Shared inputs/ Shared output.

Type 1			
	Alice	Bob	Constraints
Input	x^A, y^A	x^B, y^B	$x^A + x^B \equiv x \bmod N$, $y^A + y^B \equiv y \bmod N$
Output	δ^A	δ^B	$\delta^A \oplus \delta^B = \delta$, $\delta = (x \geq y)$

Type 2: Encrypted inputs/ Encrypted output.

Type 2			
	Alice	Bob	Constraints
Input	sk	$E_{pk}(x), E_{pk}(y)$	$(pk, sk) \leftarrow \mathsf{KGen}$
Output		$E_{pk}(\delta)$	$\delta = (x \geq y)$

Type 3: Plain inputs/ Encrypted output.

Type 3			
	Alice	Bob	Constraints
Input	x	y	
Output	sk	$E_{pk}(\delta)$	$\delta = (x \geq y)$, $(pk, sk) \leftarrow \mathsf{KGen}$

Type 4: Plain inputs/ Shared output.

Type 4			
	Alice	Bob	Constraints
Input	x	y	
Output	δ^A	δ^B	$\delta^A \oplus \delta^B = \delta$, $\delta = (x \geq y)$

Type 5: Plain inputs/ Plain output.

Type 5			
	Alice	Bob	Constraints
Input	x	y	
Output		δ	$\delta = (x \geq y)$

Type 6: Plain, Encrypted inputs/ Shared output.

Type 6			
	Alice	Bob	Constraints
Input	x, sk	$Enc_{pk}(y)$	$(pk, sk) \leftarrow \mathsf{KGen}$
Output	δ^A	δ^B	$\delta^A \oplus \delta^B = \delta$, $\delta = (x \geq y)$

Type 7: Plain, Encrypted inputs/ Encrypted output.

Type 7			
	Alice	Bob	Constraints
Input	x, sk_1	$x, Enc_{pk_1}(y)$	$(pk_1, sk_1) \leftarrow \mathsf{KGen}$
Output	$Enc_{pk_2}(\delta)$	sk_2	$(pk_2, sk_2) \leftarrow \mathsf{KGen}$, $\delta = (x \geq y)$

IV. Conversions

We are interested in constructing a protocol of type XX-YY-DD from another protocol of type XX′-YY′-DD′, that is, the former uses the latter as a subroutine. If this is possible, we write XX′-YY′-DD′ $\Rightarrow$ XX-YY-DD. We also use $\Leftrightarrow$ if conversions in both directions are possible.

Section IV-A and IV-B provide rather obvious conversions, while Section IV-C provides a sophisticated conversion that converts a seemingly weaker type (Type 3) to a stronger one (Type 2).

A. Conversions between Shared/Encrypted Data

We describe a conversion from a state of shared data to an equivalent state of encrypted data, and vice versa. For simplicity, we assume that $\mathcal{M} = \mathbb{Z}_N$. These conversions seem to be implicitly used in the literature.[4]

Conversion 1 (Shared $\rightarrow$ Encrypted data).

SS $\rightarrow$ SE			
	Alice	Bob	Constraints
Input	x^A	x^B	$x^A + x^B \equiv x \bmod N$
Output	sk	$E_{pk}(x)$	$(pk, sk) \leftarrow \mathsf{KGen}$

[4]However, we cannot find a reference that does exactly this. A closely related conversion can be found in, e.g., [23], though.

1) **Alice**:
   - Generate $(pk, sk) \leftarrow \mathsf{KGen}$.
   - Send pk and $E_{pk}(x^A)$ to Bob.
2) **Bob**:
   - Compute $E_{pk}(x) = E_{pk}(x^A) + E_{pk}(x^B)$.

Conversion 2 (Encrypted → Shared data).

SE → SS			
	Alice	Bob	Constraints
Input	sk	$E_{pk}(x)$	$(pk, sk) \leftarrow \mathsf{KGen}$
Output	x^A	x^B	$x^A + x^B \equiv x \bmod N$

1) **Bob**:
   - Randomly choose $x^B \in \mathbb{Z}_N$.
   - Compute $E_{pk}(x^A) = E_{pk}(x) - E_{pk}(x^B)$ and send to Alice.
2) **Alice**:
   - Decrypt $E_{pk}(x^A)$ to x^A.

Security should guarantee that each party will not have learned more about x at the output state than in the input state (besides the information that can be derived from the output). Conversion 1 is secure due to the semantic security of encryption, while Conversion 2 is secure due to the randomness of x^B.

From these conversions, we have the following lemma.

Lemma 2. *For any XX, YY, DD ∈ 𝒮, we have*

- *SE-YY-DD ⇔ SS-YY-DD ⇔ ES-YY-DD.*
- *XX-SE-DD ⇔ XX-SS-DD ⇔ XX-ES-DD.*
- *XX-YY-SE ⇔ XX-YY-SS ⇔ XX-YY-ES.*

B. Conversion between Plain Data and Shared Data

Conversion 3 (Plain → Shared *Input*).

PN → SS			
	Alice	Bob	Constraints
Input	x		
Output	x^A	x^B	$x^A + x^B \equiv x \bmod N$

1) **Alice**:
   - Randomly choose $x^A \in \mathbb{Z}_N$.
   - Send $x^B = x - x^A \bmod N$ to Bob.

Conversion 4 (Shared → Plain *Output*).

SS → PN			
	Alice	Bob	Constraints
Input	δ^A	δ^B	$\delta^A \oplus \delta^B = \delta$
Output	δ		

1) **Bob**:
   - Send δ^B to Alice.
2) **Alice**:
   - Compute $\delta = \delta^A \oplus \delta^B$.

Note that these capture the ordinary "sharing" and "reconstructing" procedures of the (2-out-of-2) secret sharing scheme, however when rephrasing them into our terminology, we obtain the conversions, which we collect in the following lemma.

Lemma 3. *For any XX, YY, DD ∈ 𝒮, and any TT ∈ { PN,NP,PP }, we have*

- *SS-YY-DD ⇒ TT-YY-DD.*
- *XX-SS-DD ⇒ XX-TT-DD.*
- *XX-YY-SS ⇒ XX-YY-TT.*

The conversions in the first two lines are obtained from Conversion 3. For example, if our target type is PN-YY-DD, that is, Alice has plain x as input, we can construct the protocol for this type by just letting Alice share x (as x^B) to Bob and then running a protocol of type SS-YY-DD. The conversions in the last line is done via Conversion 4. For example, if our target type is XX-YY-PN, we first run XX-YY-SS to obtain the shared outputs, Bob then send his share to Alice who then reconstructs δ.

It is also straightforward to see that XX-YY-PN ⇒ XX-YY-PP (by just letting Alice send her output to Bob).

C. Constructing a Type 2 Protocol from Type 3 [20]

In this subsection, we describe the most sophisticated conversion so far: it is for PN-NP-SE ⇒ SE-SE-SE. That is, it converts Type 3 protocol to Type 2.

Consider $a, b \in \mathbb{Z}_N$, and $n << N$, we denote $a \operatorname{div} n = \lfloor a/n \rfloor$, hence we have $a = (a \operatorname{div} n) \cdot n + (a \bmod n)$. Define

$$\beta_{a,b,n} := \begin{cases} 1 & (a+b) \bmod n < b \bmod n \\ 0 & (a+b) \bmod n \geq b \bmod n \end{cases}.$$

Lemma 4 ([20])**.** $(a+b) \operatorname{div} n = a \operatorname{div} n + b \operatorname{div} n + \beta_{a,b,n}$.

The idea of protocol is to use the fact that $0 \leq x, y < n$, and when considering $a = n + (x - y)$, we have the identity:

$$\begin{aligned} x \geq y &\Leftrightarrow a \operatorname{div} n = 1, \\ x < y &\Leftrightarrow a \operatorname{div} n = 0. \end{aligned} \tag{1}$$

The amount $a \operatorname{div} n$ is then securely computed via

$$a \operatorname{div} n = (a+b) \operatorname{div} n - b \operatorname{div} n - \beta_{a,b,n},$$

while $a + b$ and b will be known privately to Alice and Bob, respectively. (Hence, in particular, its value mod n and div n are also computable privately by each.) The encrypted value of $\beta_{a,b,n}$ is obtained via running a type-3 protocol.

Conversion 5 (protocol of Type 2 from Type 3) [20]. We construct a Type-2 protocol, for which we recall the input-output table below.

Type 2			
	Alice	Bob	Constraints
Input	sk	$E_{pk}(x), E_{pk}(y)$	$(pk, sk) \leftarrow \mathsf{KGen}$
Output		$E_{pk}(\delta)$	$\delta = (x \geq y)$

In the following, we denote $a = n + (x - y)$.

1) **Bob**:
   - Randomly choose $b \in \mathbb{Z}_N$.
   - Set $y' = b \bmod n$.
   - Compute $E_{pk}(a+b) = E_{pk}(n) + E_{pk}(x) - E_{pk}(y) + E_{pk}(b)$ and send this to Alice.
2) **Alice**:
   - Decrypt $E_{pk}(a+b)$ to $a+b$.
   - Set $x' = (a+b) \bmod n$.
   - Compute $E_{pk}((a+b) \operatorname{div} n)$ and send to Bob.
3) **Alice** and **Bob**
   - Together run a protocol of Type 3 using inputs x' (from Alice) and y' (from Bob). More precisely, this subroutine can be specified as the following table.

Type 3 (used as a subroutine)			
	Alice	Bob	Constraints
Input	x'	y'	
Output	sk	$E_{pk}(\delta')$	$\delta' = (x' \geq y')$

   - As a result of this sub-routine, Bob will obtain

$$E_{pk}(\delta') = E_{pk}(1 - \beta_{a,b,n}).$$

   This equation holds since

$$\begin{aligned}\delta' = 1 &\Leftrightarrow x' \geq y' \Leftrightarrow (a+b) \bmod n \geq b \bmod n \\ &\Leftrightarrow \beta_{a,b,n} = 0\end{aligned}$$

   where the first equivalence is due to the correctness of the Type 3 protocol, the second is due to the definition of x', y', and third is from the definition of $\beta_{a,b,n}$.
4) **Bob**:
   - Compute $-E_{pk}(\beta_{a,b,n}) = E_{pk}(1 - \beta_{a,b,n}) - E_{pk}(1)$.
   - Compute $E_{pk}(a \operatorname{div} n)$ as

$$E_{pk}((a+b) \operatorname{div} n) - E_{pk}(b \operatorname{div} n) - E_{pk}(\beta_{a,b,n}).$$

   - From Lemma 4 and Eq.(1), we have $E_{pk}(a \operatorname{div} n) = E_{pk}(\delta)$.

The intuition is given above, while more details can be found in [20], [10]. We write this result into the following lemma:

Lemma 5. *PN-NP-SE* $\Rightarrow$ *SE-SE-SE.*

D. Combinations

In the next section, we will describe efficient constructions for type PN-NP-SS (or Type 4), and type PN-NP-NP (or Type 5). From these two protocols, together with conversions from the above lemmata in this section (Lemma 2,3,5), we can construct any of 174 non-trivial protocol types.

V. Previous Specific Protocols

In this section we recapture slight variants of the protocols by Nergiz et al. [9]. We (implicitly) uses a classical method called *dyadic range* which expresses a range (i.e., a set of consecutive integers) efficiently using nodes in a complete binary tree.[5]

We describe some intuition first. A naive way to compare two integers is to represent both values as binary strings and use the standard Boolean circuit for comparison, which firstly compares the most significant bit, and if equal, simply compares the next bit, and so on. This means that it requires to execute on each bit in a *sequential* manner. On the other hand, in the dyadic range method, two integers are represented in such a way that their comparison can be done by comparing each bit in an *independent* manner (and simply OR all the results). Contrasting the two methods, the latter has much shallower circuit; this makes it more efficient than the former. Moreover, in the latter method, only secure equality checking will be required as sub-routine, and we can straightforwardly use additive homomorphic encryption to implement a secure protocol for it.

A. Notations for Tree-based Structure

We firstly describe our own terminology and notations for tree-based structure.

Let $\mathbb{T}_n$ be the complete binary tree that has leaves corresponding to each index in $[1, n]$. Let $\mathbb{S}_n$ be the set of all nodes in $\mathbb{T}_n$ that are labeled in a systematic way. For a node $w \in \mathbb{S}_n$, let parent(w) denote its parent node in $\mathbb{T}_n$. Consider node $w, y, z \in \mathbb{S}_n$; z is an *ancestor* of w if z is on the path from w to the root (including w); y is a *descendant* of w if y is on a path from w moving away from the root (including w). For any node w, we define its *layer* as the distance from its leaves. (Hence, in particular, the layer of any leaf is 0, and the layer of the root is $\log_2 n$.) We label each node as a pair (i, j) where i is its layer and j is its index in that layer from the left of tree (starting from 1). See how we label each node in e.g., Fig 1, where we omit the comma in the figure.

We let

$$\begin{aligned}\mathcal{D}_n &:= \{\, [u, v] \mid 1 \leq u \leq v \leq n \,\}, \\ \mathcal{L}_n &:= \{\, [1, v] \mid 1 \leq v \leq n \,\}, \\ \mathcal{R}_n &:= \{\, [u, n] \mid 1 \leq u \leq n \,\}.\end{aligned}$$

That is, $\mathcal{D}_n$ is the set of all ranges, while $\mathcal{L}_n$ and $\mathcal{R}_n$ fix the start point and end point to 1 and n, respectively. For any range $R \in \mathcal{D}_n$, a node $w \in \mathbb{S}_n$ is called a *cover node* of R, and we write $w \in \text{cover}(R)$, if all the leaves that are descendants of w are in R. Let $2^{\mathbb{S}_n}$ be the collection of all subsets of $\mathbb{S}_n$.

B. Range and Point Encodings

We define two encoding functions:

[5] It is not clear to us who firstly proposed this method, which is referred variously as *dyadic range* in [24], [25], or *segment tree* in [26], [27], and has been rediscovered in [28], [9], while also has been used in [29], [30].

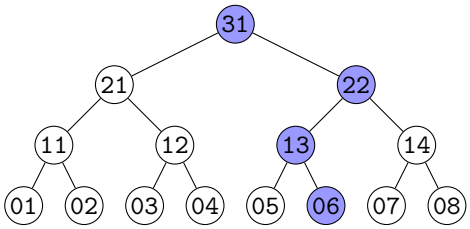

Fig. 1. Example of Point Encoding, for $x = 6$. Here, $\mathsf{pointEnc}(6) = \{ (0,6), (1,3), (2,2), (3,1) \}$.

- **Range Encoding.** $\mathsf{rangeEnc} : \mathcal{D}_n \to 2^{\mathbb{S}_n}$. For $R \in \mathcal{D}_n$, define

$$\mathsf{rangeEnc}(R) := \{ w \in \mathbb{S}_n \mid w \in \mathrm{cover}(R), \\ \mathrm{parent}(w) \notin \mathrm{cover}(R) \}. \quad (2)$$

- **Point Encoding.** $\mathsf{pointEnc} : [1,n] \to 2^{\mathbb{S}_n}$. For $x \in [1,n]$, define $\mathsf{pointEnc}(x)$ as the set of all ancestors of x in $\mathbb{T}_n$.

Lemma 6. *For $R \in \mathcal{L}_n \cup \mathcal{R}_n$, we have that* $\mathsf{rangeEnc}(R)$ *contains* at most one node *from each layer.*

Lemma 7. *For $x \in [1,n]$, we have that* $\mathsf{pointEnc}(x)$ *contains* exactly one node *from each layer.*

From these lemmata, we have that we can write

$$\mathsf{rangeEnc}(R) = \{ (i, a_i) \mid i \in W_R \}, \quad (3)$$
$$\mathsf{pointEnc}(x) = \{ (i, b_i) \mid i \in [1, \log n] \}, \quad (4)$$

for some a_i, b_i, where we let $W_R \subseteq [0, \log n]$ be the set of layers in which there exists a node in $\mathsf{rangeEnc}(R)$. That is

$$W_R := \{ i \mid \exists a \text{ s.t. } (i,a) \in \mathsf{rangeEnc}(R) \} \quad (5)$$

Lemma 8. *For any $R \in \mathcal{L}_n \cup \mathcal{R}_n$, and any $x \in [1,n]$,*

$$|\mathsf{rangeEnc}(R) \cap \mathsf{pointEnc}(x)| = \begin{cases} 1 & \text{if } x \in R \\ 0 & \text{if } x \notin R \end{cases}.$$

From this lemma, Eq.(3), and Eq.(4), we have the following corollary.

Corollary 9. *For any $R \in \mathcal{L}_n \cup \mathcal{R}_n$, and any $x \in [1,n]$,*

$$x \in R \;\Leftrightarrow\; \text{There exists a unique } i \in W_R \text{ s.t. } a_i = b_i.$$

where W_R, a_i, b_i are defined in Eq.(3),(4).

Example. Let $n = 8$. Consider $x = 6$, $R = [4,8] \in \mathcal{R}$, and $L = [1,3] \in \mathcal{L}$. We have

$$\mathsf{pointEnc}(6) = \{ (0,6), (1,3), (2,2), (3,1) \},$$
$$\mathsf{rangeEnc}([4,8]) = \{ (0,4), (2,2) \},$$
$$\mathsf{rangeEnc}([1,3]) = \{ (0,3), (1,1) \},$$

and $W_{[4,8]} = \{0,2\}$, $W_{[1,3]} = \{0,1\}$. Now since $6 \in [4,8]$, we have that Lemma 8 holds with an intersection node $(2,2)$, while Corollary 9 holds at $i = 2$, and $a_2 = b_2 = 2$. On the other hand, since $6 \notin [1,3]$, we can verify that the intersection of their encodings is empty.

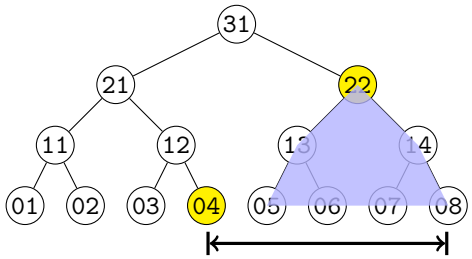

Fig. 2. Example of Range Encoding, for $R_1 = [4,8]$. Here, $\mathsf{rangeEnc}([4,8]) = \{ (0,4), (2,2) \}$.

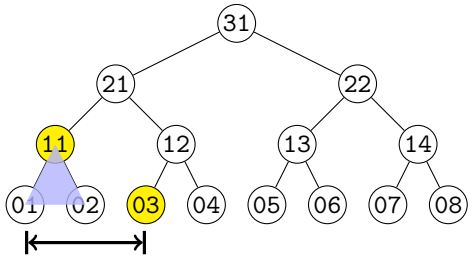

Fig. 3. Example of Range Encoding, for $R_2 = [1,3]$. Here, $\mathsf{rangeEnc}([1,3]) = \{ (0,3), (1,1) \}$.

C. Tree-based Comparison Protocols

We are now ready to describe the protocol of Type 5 and 4 by Nergiz et al. [9]. Let $H : \{0,1\}^* \to \mathbb{Z}_N$ be an injective function (or collision-resistant hash).

Protocol 1 (for Type 5) [9].

1) **Alice**:
   - Compute $E_{pk}(H(i,b_i))$ for all $(i,b_i) \in \mathsf{pointEnc}(x)$ and send to Bob, together with pk.
2) **Bob**:
   - Set $R = [y,n]$. (Hence, $R \in \mathcal{R}_n$.)
   - Compute $\mathsf{rangeEnc}(R)$ as Eq.(2) and W_R as Eq.(5).
   - For all $i \in [0, \log n]$, randomly choose $r_i \in \mathbb{Z}_N^*$ and compute

$$V_i := \begin{cases} E_{pk}\Big(r_i\big(H(i,b_i) - H(i,a_i)\big)\Big) & \text{if } i \in W_R \\ E_{pk}(r_i) & \text{if } i \notin W_R \end{cases} \quad (6)$$

   - Randomly shuffle all V_i to V_i', and send all to Alice.
3) **Alice**:
   - Decrypt all V_i' and output $\delta = 1$ (meaning $x \geq y$) if there is exactly one plaintext being zero. Otherwise, output $\delta = 0$ (meaning $x < y$).

We sketch the proof for its correctness. Suppose $x \geq y$. We have $x \in [y,n]$. Hence, due to Corollary 9, there is exactly one layer where the encrypted node label is the same from Alice's point encoding and Bob's range encoding. Therefore, exactly one ciphertext in the list of V_i' will be 0. The value r_i is used for randomizing each non-zero value to a *random* non-zero element. The shuffling is for preventing Alice to know which layer is matched.

Protocol 2 (for Type 4) [9].

1) **Alice**:
   - Compute $E_{pk}(H(i, b_i))$ for all $(i, b_i) \in \mathsf{pointEnc}(x)$ and send to Bob, together with pk.
2) **Bob**:
   - Randomly choose $\delta^B \in \{0, 1\}$ and set
   $$R = \begin{cases} [1, y-1] & \text{if } \delta^B = 1 \\ [y, n] & \text{if } \delta^B = 0 \end{cases}.$$
   (Hence, $R \in \mathcal{L}_n \cup \mathcal{R}_n$.)
   - Compute $\mathsf{rangeEnc}(R)$ as Eq.(3).
   - For all $i \in [0, \log n]$, randomly choose $r_i \in \mathbb{Z}_N^*$ and compute
   $$V_i := \begin{cases} E_{pk}\Big(r_i\big(H(i, b_i) - H(i, a_i)\big)\Big) & \text{if } i \in W_R \\ E_{pk}(r_i) & \text{if } i \notin W_R \end{cases} \tag{7}$$
   - Randomly shuffle all V_i to V_i', and send all to Alice.
3) **Alice**:
   - Decrypt all V_i' and output $\delta^A = 1$ if there is exactly one plaintext being zero. Otherwise, output $\delta^A = 0$.

We can verify that $\delta^A \oplus \delta^B = \delta$ (which is the bit indicating that $x \geq y$). The bit δ^B is used to hide which of the two relations, $x \in [y, n]$ or $x \notin [y, n]$, is being tested. The latter is equivalent to $x \in [1, y-1]$, which is R for the case $\delta^B = 1$ in the protocol.

Improvement. We provide a further improvement for Protocol 1 and 2 above. This reduces the communication cost by one ciphertext for each of Step 1 and 2 (hence, two ciphertexts overall). This can be done by first observing that since the root node, namely, the node with label $(\log n, 1)$, is always in the point encoding of *any* point in $[1, n]$, Alice can omit sending $E_{pk}(H(\log n, 1))$ to Bob in the first pass (in Step 1), and simply letting Bob compute by himself. This already reduces one ciphertext in Step 1. The next observation is that $V_{\log n}$ encrypts 0 if and only if $y = 1$. (Since, $\log n \in W_R$ iff $R = [1, n]$, the full range.) Hence, we just treat only the case of $y = 1$ specifically while simply omitting $V_{\log n}$ altogether. This will reduce one ciphertext in Step 2. To enable this, in Step 2, we can let Bob compute as follows.

- If $y > 1$, then compute V_i as usual (Eq.(6),(7)), albeit for only $i \in [0, \log n - 1]$. (That is, $V_{\log n}$ will not be used.) Shuffle V_i to V_i' for all $i \in [0, \log n - 1]$ and send back to Alice these $\log n$ ciphertexts.
- If $y = 1$, then simply generate $\log n$ ciphertexts with only one message being 0 and the others are random.

In Step 3, Alice does as usual, i.e., to check if one plaintext is zero, albeit among all the $\log n$ ciphertexts (instead of $\log n + 1$ ciphertexts as before).

Parallel Time Complexity. One of the main reason that we choose the tree-based protocols from [9] is that all the computationally-heavier procedures, such as encryption, decryption, homomorphic valuation, can be run in *parallel*. In particular, in Step 2, all the calculation for V_i can be done independently for each i, which means that we can compute them in parallel. The same can be said for Step 1 and 3. Hence the parallel time complexity is almost *constant* regardless of the number of layers, $\ell + 1$, where we recall that $n = 2^\ell$. (ℓ is the bit length of the compared numbers x, y.) It is *almost* constant since Range encoding and Point encoding will depend on ℓ, but these computations are much lower since they are operated on plain data and are comparable to bit decomposition (of a plain value). Indeed, Point encoding is exactly the bit decomposition, phrased in term of tree-based structure. We remark that this almost-constant-parallel-time property has not been observed in the original paper [9], and to the best of our knowledge, we are the first to observe so.

VI. Our Comparison Protocol of Type 1

A. Construction

We construct a new comparison protocol of Type 1 (shared inputs/shared output), which is the type that can be used as a sub-protocol in generic two-party computation. Our protocol is simply a combination of previous results: we base on Protocol 2 (of Type 4), described in Sect. V, and use necessary conversions, described in Sect. IV, to convert it to Type 1.

Our Protocol (for Type 1).

1) **Alice** and **Bob**:
   - Run Conversion 1 to convert shared inputs to encrypted inputs (for both data: x and y, and in parallel). As a result, Alice obtains sk, while Bob obtains $E_{pk}(x), E_{pk}(y)$.
2) **Alice** and **Bob**:
   - Run Conversion 5 (for protocol Type 2), which we recall its description as the following table.

Type 2

	Alice	Bob	Constraints
Input	sk	$E_{pk}(x), E_{pk}(y)$	
Output		$E_{pk}(\delta)$	$\delta = (x \geq y)$

   - Inside Conversion 5, we must specify the Type-3 protocol that it uses as a subroutine. Recall that the Type-3 protocol has the following generic description.

Type 3

	Alice	Bob	Constraints
Input	x'	y'	
Output	sk	$E_{pk}(\delta')$	$\delta' = (x' \geq y')$

   We use the following specific protocol for implementing this Type-3 protocol.
   - First, run Protocol 2 (for Type 4) on input x' and y'. This enables us to compute as in the following table.

Type 4			
	Alice	Bob	Constraints
Input	x'	y'	
Output	$(\delta')^A$	$(\delta')^B$	$(\delta')^A \oplus (\delta')^B = \delta'$, $\delta' = (x' \geq y')$

As a result, Alice obtains $(\delta')^A$ and Bob obtains $(\delta')^B$, which are the shares of $\delta' = (x' \geq y')$.

– Second, run Conversion 1 to convert Shares of δ' to encrypted forms. As a result, Alice has sk, while Bob obtains $E_{pk}(\delta')$.

3) **Alice** and **Bob**:

- Run Conversion 2 to convert encrypted outputs to shared outputs. As a result, Alice obtains δ^A, while Bob obtains δ^B.

B. Experiment

We implemented our above Type-1 protocol. In this subsection, we analyze its performance. We compared our protocol to the state-of-the-art Type-1 protocol where an offline phase is not permitted, which is the DGK protocol [14] (we actually use the improved version in [31], [10]). We also included two other protocols in our comparison, namely, the Type-1 protocol by Bunn and Ostrovsky (BO) [3] (used as a part of their secure k-means clustering protocol) and the more recent protocol by Gentry et al. (GHJR) [15] (used as a part of their ORAM protocol), converted to Type 1.

For our protocol, we used the DGK encryption scheme [14] for the Type-3 protocol (which is a subroutine in Step 2 of our protocol), and the Paillier encryption scheme [32] for the other steps (Step 1 and 3). Similarly, for the improved DGK protocol of [31], [10], we used the DGK encryption scheme for the core part (which is a Type-3 protocol), while the conversions to Type 1 were done via the Pailler encryption scheme. For the BO protocol, we used the Paillier encryption scheme. For the GHJR protocol, we used the HElib library as in their paper [15]. We implemented these protocols on a PC with an Intel Core i7-4790 3.6 GHz CPU and 16.0 GB of memory.

The total computation time and the communication cost of our protocol are shown in Table I. Our protocol consists of a few sub-protocols, for which we show the computation times separately in Table II. In the experiment, we varied the input size to 5, 10, 25, 50, and 100 bits, ran each experiment 10 times, and averaged the various results.

Fig. 4 shows the comparison results for computation time. We observe that in our protocol, the computation time does not vary much even when the bit size increases. This is thanks to the parallel-processing property of [9] that we observe and implement. In contrast, the other protocols require time proportional to the bit size with larger proportional factors. We also note that the DGK protocol cannot be executed in a parallel manner as ours, as it involves computing sums of ℓ elements, where parallel time complexity is $O(\log \ell)$, at best.

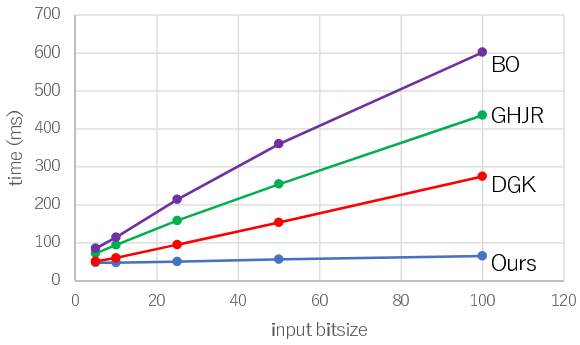

Fig. 4. Computation time

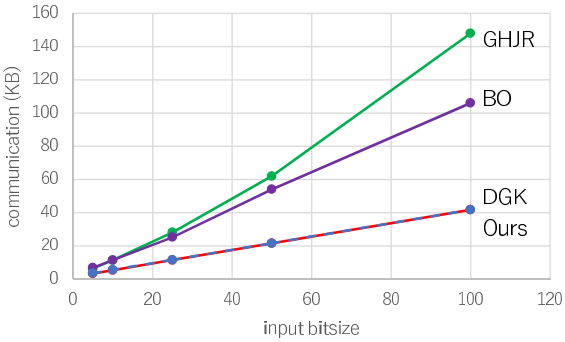

Fig. 5. Communication cost

Regarding the round complexity, our protocol and the DGK protocol work in constant rounds (more precisely, 6 rounds). The GHJR protocol requires $\log n$ rounds.

Fig. 5 shows the comparison for communication cost. All the protocols require communication proportional to the bit size. The communication cost of our protocol and the DGK protocol are almost the same[6] and are more efficient than the other two protocols.

Acknowledgement. A part of this work is supported by JST CREST grant number JPMJCR1688.

References

[1] A. C.-C. Yao, "How to generate and exchange secrets," in *Foundations of Computer Science, 1986., 27th Annual Symposium on*, pp. 162–167, IEEE, 1986.

[2] O. Goldreich, S. Micali, and A. Wigderson, "How to play any mental game," in *Proceedings of the nineteenth annual ACM symposium on Theory of computing*, pp. 218–229, ACM, 1987.

[3] P. Bunn and R. Ostrovsky, "Secure two-party k-means clustering," in *Proceedings of the 14th ACM conference on Computer and communications security*, pp. 486–497, ACM, 2007.

[4] R. Ostrovsky, Y. Rabani, L. J. Schulman, and C. Swamy, "The effectiveness of lloyd-type methods for the k-means problem.," *J. ACM*, vol. 59, no. 6, p. 28, 2012.

[5] R. Bost, R. A. Popa, S. Tu, and S. Goldwasser, "Machine learning classification over encrypted data.," in *NDSS*, 2015.

[6] Indeed, their plots in Fig 5 can be seen completely on the same line in this scale.

TABLE I
PERFORMANCE OF OUR PROTOCOL

Cost	5 bit	10 bit	25 bit	50 bit	100 bit
Computation (ms)	47.731	47.700	50.301	56.419	65.459
Communication (KB)	5.294	8.399	17.715	33.243	64.296

TABLE II
COMPUTATION TIME (MS)

Protocol	5 bit	10 bit	25 bit	50 bit	100 bit
Step 1 (Conversion 1)	0.011	0.011	0.011	0.012	0.012
Step 2 (Conversion 5 including subroutines)	47.350	47.679	50.279	56.396	65.435
Step 2 (Type-3 protocol as a subroutine)	38.626	38.678	38.220	38.030	35.000
Step 2 (Conversion 5 except subroutine)	8.724	9.001	12.059	18.366	30.435
Step 3 (Conversion 2)	0.010	0.010	0.011	0.011	0.011

[6] Z. Erkin, M. Franz, J. Guajardo, S. Katzenbeisser, I. Lagendijk, and T. Toft, "Privacy-preserving face recognition," in *International Symposium on Privacy Enhancing Technologies Symposium*, pp. 235–253, Springer, 2009.

[7] M. Barni, P. Failla, R. Lazzeretti, A.-R. Sadeghi, and T. Schneider, "Privacy-preserving ecg classification with branching programs and neural networks," *IEEE Transactions on Information Forensics and Security*, vol. 6, no. 2, pp. 452–468, 2011.

[8] Z. Erkin, T. Veugen, T. Toft, and R. L. Lagendijk, "Generating private recommendations efficiently using homomorphic encryption and data packing," *IEEE transactions on information forensics and security*, vol. 7, no. 3, pp. 1053–1066, 2012.

[9] A. E. Nergiz, M. E. Nergiz, T. Pedersen, and C. Clifton, "Practical and secure integer comparison and interval check," in *Social Computing (SocialCom), 2010 IEEE Second International Conference on*, pp. 791–799, IEEE, 2010.

[10] T. Veugen, F. Blom, S. J. de Hoogh, and Z. Erkin, "Secure comparison protocols in the semi-honest model," *IEEE Journal of Selected Topics in Signal Processing*, vol. 9, no. 7, pp. 1217–1228, 2015.

[11] T. Nishide and K. Ohta, "Multiparty computation for interval, equality, and comparison without bit-decomposition protocol," in *International Workshop on Public Key Cryptography*, pp. 343–360, Springer, 2007.

[12] O. Catrina and S. de Hoogh, "Improved primitives for secure multiparty integer computation," in *Security and Cryptography for Networks, 7th International Conference, SCN 2010, Amalfi, Italy, September 13-15, 2010. Proceedings*, pp. 182–199, 2010.

[13] J. A. Garay, B. Schoenmakers, and J. Villegas, "Practical and secure solutions for integer comparison," in *Public Key Cryptography - PKC 2007, 10th International Conference on Practice and Theory in Public-Key Cryptography, Beijing, China, April 16-20, 2007, Proceedings*, pp. 330–342, 2007.

[14] I. Damgård, M. Geisler, and M. Krøigaard, "Homomorphic encryption and secure comparison," *IJACT*, vol. 1, no. 1, pp. 22–31, 2008.

[15] C. Gentry, S. Halevi, C. Jutla, and M. Raykova, "Private database access with he-over-oram architecture," in *International Conference on Applied Cryptography and Network Security*, pp. 172–191, Springer, 2015.

[16] B. Schoenmakers and P. Tuyls, "Practical two-party computation based on the conditional gate," in *International conference on the theory and application of cryptology and information security*, pp. 119–136, Springer, 2004.

[17] I. F. Blake and V. Kolesnikov, "Strong conditional oblivious transfer and computing on intervals," in *International Conference on the Theory and Application of Cryptology and Information Security*, pp. 515–529, Springer, 2004.

[18] I. F. Blake and V. Kolesnikov, "Conditional encrypted mapping and comparing encrypted numbers," in *International Conference on Financial Cryptography and Data Security*, pp. 206–220, Springer, 2006.

[19] I. Damgard, M. Geisler, and M. Kroigard, "A correction to'efficient and secure comparison for on-line auctions'," *International Journal of Applied Cryptography*, vol. 1, no. 4, pp. 323–324, 2009.

[20] T. Veugen, "Encrypted integer division and secure comparison," *Int. J. Appl. Cryptol.*, vol. 3, pp. 166–180, June 2014.

[21] O. Goldreich, *Foundations of Cryptography: Volume 2, Basic Applications*. New York, NY, USA: Cambridge University Press, 2004.

[22] A. C. Yao, "Protocols for secure computations," *1982 IEEE 54th Annual Symposium on Foundations of Computer Science*, vol. 00, pp. 160–164, 1982.

[23] F. Kerschbaum, T. Schneider, and A. Schröpfer, "Automatic protocol selection in secure two-party computations," in *Applied Cryptography and Network Security: 12th International Conference, ACNS 2014*, pp. 566–584, Springer, 2014.

[24] A. C. Gilbert, Y. Kotidis, S. Muthukrishnan, and M. J. Strauss, "How to summarize the universe: Dynamic maintenance of quantiles," in *Proceedings of the 28th International Conference on Very Large Data Bases*, VLDB '02, pp. 454–465, VLDB Endowment, 2002.

[25] G. Cormode and S. Muthukrishnan, "An improved data stream summary: The count-min sketch and its applications," *J. Algorithms*, vol. 55, pp. 58–75, Apr. 2005.

[26] J. L. Bentley, "Solutions to klee's rectangle problems," *Tech. report, Carnegie- Mellon Univ.*, 1977.

[27] M. de Berg, M. van Kreveld, M. Overmars, and O. Schwarzkopf, *Computational Geometry: Algorithms and Applications*. Secaucus, NJ, USA: Springer-Verlag New York, Inc., 1997.

[28] E. Shi, J. Bethencourt, H. T. Chan, D. X. Song, and A. Perrig, "Multi-dimensional range query over encrypted data," in *2007 IEEE Symposium on Security and Privacy (S&P 2007), 20-23 May 2007, Oakland, California, USA*, pp. 350–364, 2007.

[29] R. Gay, P. Méaux, and H. Wee, "Predicate encryption for multi-dimensional range queries from lattices," in *Public-Key Cryptography - PKC 2015 - 18th IACR International Conference on Practice and Theory in Public-Key Cryptography, Gaithersburg, MD, USA, March 30 - April 1, 2015, Proceedings*, pp. 752–776, 2015.

[30] N. Attrapadung, G. Hanaoka, K. Ogawa, G. Ohtake, H. Watanabe, and S. Yamada, "Attribute-based encryption for range attributes," in *International Conference on Security and Cryptography for Networks*, pp. 42–61, Springer, 2016.

[31] T. Veugen, "Improving the DGK comparison protocol," *WIFS 2012 - Proceedings of the 2012 IEEE International Workshop on Information Forensics and Security*, pp. 49–54, 2012.

[32] P. Paillier, "Public-key cryptosystems based on composite degree residuosity classes," in *Proceedings of the 17th International Conference on Theory and Application of Cryptographic Techniques*, EUROCRYPT'99, (Berlin, Heidelberg), pp. 223–238, Springer-Verlag, 1999.