IEICE TRANS. INF. & SYST., VOL.E103–D, NO.7 JULY 2020

PAPER Special Section on Information and Communication System Security

# Online-Efficient Interval Test via Secure Empty-Set Check**

**Katsunari SHISHIDO**[†*a)], *Nonmember and* **Atsuko MIYAJI**[†,††b)], *Member*

**SUMMARY** In the age of information and communications technology (ICT), not only collecting data but also using such data is provided in various services. It is necessary to ensure data privacy in such services while providing efficient computation and communication complexity. In this paper, we propose the first interval test designed according to the notion of online and offline phases by executing our new empty-set check. Our protocol is proved to ensure both server and client privacy. Furthermore, neither the computational complexity of a client in the online phase nor the communicational complexity from a server to a client depends on the size of the set. As a result, even in a practical situation in which one server receives requests from numerous clients, the waiting time for a client to obtain the result of an interval test can be minimized.

***key words:*** *secure interval test, private set intersection, empty-set check*

## 1. Introduction

In this age of information and communications technology (ICT), important data are sent to servers for various applications such as automobile insurance, healthcare records, and school records. Both the collection and use of data is also provided by various services. In such services, it is necessary to protect data privacy. One of the effective privacy-preserving methods of utilizing data is a private set intersection (PSI) [3], [6]. PSI is executed by two parties, a client and a server, in which both jointly compute the intersection of their private sets and, at the end, only the client learns the intersection and the server learns nothing. PSI is extended to the multiparty private set intersection (MPSI) protocol [2], [5], [6], [9], [14]. MPSI is executed by multiple parties who jointly compute the intersection of their private datasets, and ultimately, only the designated parties learn the intersection while others do not have access.

An empty-set check is a special case of PSI. An empty-set check has an important role in an interval test. A interval test is that in which a client possesses a value $x$, a server possesses a range $I = [a, b] = \{x | a \le x \le b\}$, and the client wants to check whether $x \in I$ while protecting client value $x$ to a server and server's interval $I$ to a client. This problem is a variant of the famous *Yao's Millionaires' problem* in which one determines who is richer between two parties such that no information about a party's amount of assets is leaked to the other party [17]. To solve Yao's Millionaires' problem using PSI, the 0/1 encoding technique was introduced in [7]. To solve an interval test regarding whether $x \in I$ is transformed to an empty-set check whether $S_x \cap S_I = \phi$— that is,

$$x \in I \Longleftrightarrow S_x \cap S_I = \phi, \tag{1}$$

where $S_x$ and $S_I$ are sets corresponding to $x$ and $I$, respectively. Thus, if we construct an efficient empty-set check protocol compared with an ordinary PSI, then we can also provide an efficient interval test.

In this paper, we construct an interval test between a client and a server by executing our new empty-set check to sets induced by Eq. (1). We note that all features satisfied with our interval test hold on our empty-set check. Let us explain our design idea of an interval test and an empty-set check. The server's processing power or network bandwidth is limited. As a result, the fulfillment of clients' requests is delayed. Our main idea is to reduce the time it takes to obtain a result. For this purpose, we divide a protocol into two phases, the *offline* phase and *online* phase. In the *offline* phase, a client computes data sent to a server beforehand; in the *online* phase, a server gets data from a client and computes data sent back to a client. A client obtains data from a server and computes a result. In general, the offline phase can be conducted beforehand; thus, client's waiting times depend on the online phase. Our protocol focuses on the online phase and reduces the computational complexity of the online phase. Furthermore, we also focus on communicational complexity from a server to a client. In fact, communicational complexity from a server to a client is more critical than that from a client to a server since a server must reply to requests from many clients.

To design the interval test/empty-set-check protocol in two phases, we start with MPSI [9] based on the Bloom filter and reconstruct the server computation phase in such a way to combine all arrays in Bloom Filter into one array. Our protocol ensures both server and client privacy; after executing the protocol, a server cannot obtain any information on client data by utilizing the knowledge of a server, and a client cannot get any information on server data by utilizing the knowledge of a client, except as a result of an interval test/empty-set check. Our protocol has the following advantage: the computational complexity of a client in the online

Manuscript received August 31, 2019.
Manuscript revised January 8, 2020.
Manuscript publicized May 14, 2020.

[†]The authors are with Graduate School of Engineering, Osaka University, Suita-shi, 565–0871 Japan.
[††]The author is with Japan Advanced Institute of Science and Technology, Nomi-shi, 923–1211 Japan.
[*]Presently, with the FUJITSU LABORATORIES LTD.

[**]The part of this paper was presented at ASIAJCIS 2019 [15].

a) E-mail: shishido@cy2sec.comm.eng.osaka-u.ac.jp
b) E-mail: miyaji@comm.eng.osaka-u.ac.jp

DOI: 10.1587/transinf.2019ICP0014

phase is just one field exponentiation $E$ that does not depend on the size of set. The computational complexity of a server in online phase is at most $2(mM + E)$ in which $m$ is the size of the Bloom filter and $M$ is the computational complexity of one field multiplication. The communicational complexity from a server to client is just two field elements $|\mathbb{F}_p|$, which does not depend on the size of set. Our protocol reduces the computational complexity of a client in the online phase thanks to that of server. Considering computational complexity of server, our protocol is suitable for a small range test such as an income, an academic record of the grade point average (GPA), the TOEFL score, etc. We remark that this is the first interval test/empty-set-check protocol designed using the notion of online and offline phases that achieves total computational complexity of a client in the online phase is independent to the size of interval test.

This paper is organized as follows. Building blocks used to develop the proposed protocol are summarized in Sect. 2. Section 3 introduces several related studies on interval tests. We propose the new interval test in Sect. 4. We note that our interval test is achieved by executing our empty-set check on sets induced by Eq. (1), and, thus only interval test are presented. We explain how appropriate parameters of our protocol are chosen in Sect. 5 and evaluate complexity in our protocol in Sect. 6. Finally, we conclude our work in Sect. 7.

## 2. Preliminary

### 2.1 Decisional Diffie–Hellman Assumption

Let $\mathbb{G}$ be a finite field and $g \in \mathbb{G}$ be a basepoint whose order is a prime $q$. The decisional Diffie–Hellman (DDH) assumption is for any probabilistic polynomial-time algorithm $\mathcal{A}$ to be impossible to distinguish the DDH tuple $(g, g^\alpha, g^\beta, g^{\alpha\beta})$ from the non-DDH tuple $(g, g^\alpha, g^\beta, g^\gamma)$, where $\alpha, \beta, \gamma \leftarrow \mathbb{Z}_q^*$. Precisely, for any probabilistic polynomial-time algorithm $\mathcal{A}$, the following formula holds: $|\mathsf{Pr}[\mathcal{A}(g, g^\alpha, g^\beta, g^{\alpha\beta}) \rightarrow 1 | \alpha, \beta \leftarrow \mathbb{Z}_q^*] - \mathsf{Pr}[\mathcal{A}(g, g^\alpha, g^\beta, g^\gamma) \rightarrow 1 | \alpha, \beta, \gamma \leftarrow \mathbb{Z}_q^*]| \leq negl(\lambda)$, where $\lambda$ is security parameter and $negl(\lambda)$ is the negligible function of $\lambda$.

### 2.2 The 0/1 Encoding

Lin and Tzeng proposed the 0/1 encoding [7], which reduces the Millionaire's problem [16] to find the intersection of two sets. Precisely, for two integers $a, b \in \{0,1\}^\ell$, the 0/1 encoding generates two binary string sets $S_a^1$ and $S_b^0$, respectively, and then we compute $S_a^1 \cap S_b^0$. If $S_a^1 \cap S_b^0 \neq \phi$, then it means $a > b$. Otherwise, it means $a \leq b$. Hereafter, we describe how to solve the greater-than problem by using the 0/1 encoding. Let $s = s_1 s_2 \cdots s_\ell \in \{0,1\}^\ell$ be a set of binary digits of length $\ell$.

**Definition 2.1** (The prefix string set $P_s$): The prefix string set $P_s$ is the set of binary strings such that $P_s = \{s_1 s_2 \cdots s_h | 1 \leq h \leq \ell\}$.

**Definition 2.2** (The 0–encoding set $S_s^0$): The 0–encoding set $S_s^0$ is the set of binary strings such that $S_s^0 = \{s_1 s_2 \cdots s_{h-1} 1 \mid s_h = 0, 1 \leq h \leq \ell\}$.

**Definition 2.3** (The 1–encoding set $S_s^1$): The 1–encoding $S_s^1$ is the set of binary strings such that $S_s^1 = \{s_1 s_2 \cdots s_{h-1} s_h \mid s_h = 1, 1 \leq h \leq \ell\}$.

**Theorem 2.1:** If we encode $a$ into its 1–encoding $S_a^1$ and $b$ into its 0–encoding $S_b^0$, we can see that $a > b \Longleftrightarrow S_a^1 \cap S_b^0 \neq \phi$.

As a proof of Theorem 2.1, please refer to the paper [7]. We define the new 0–encoding set and the new 1–encoding set [4] for the Lemma 2.1 in below.

**Definition 2.4** (The new 0–encoding set $\tilde{S}_s^0$): The new 0–encoding set $\tilde{S}_s^0$ is the set of binary strings such that $\tilde{S}_s^0 = \{s_1 s_2 \cdots s_{h-1} 0 \mid s_h = 0, 1 \leq h \leq \ell\}$.

**Definition 2.5** (The new 1–encoding set $\tilde{S}_s^1$): The new 1–encoding set of s is the set $\tilde{S}_s^1$ of binary strings such that $\tilde{S}_s^1 = \{s_1 s_2 \cdots s_{h-1} 0 \mid s_h = 1, 1 \leq h \leq \ell\}$.

**Lemma 2.1:** Let $d$ be an integer. We denote $P_d$ as the prefix string set of $d$.

If we encode $a$ into the new 1–encoding $\tilde{S}_a^1$ and $b$ into the 0–encoding $S_b^0$, we can see that

$$\begin{aligned} d \notin [a,b] &\Longleftrightarrow d < a \text{ or } b < d \\ &\Longleftrightarrow P_d \cap \tilde{S}_a^1 \neq \phi \text{ or } P_d \cap S_b^0 \neq \phi. \end{aligned}$$

**Proof 2.1:** We know that $d \notin [a,b]$ if and only if $d < a$ or $b < d$ obviously. So we prove that $d < a$ or $b < d \Longleftrightarrow P_d \cap \tilde{S}_a^1 \neq \phi$ or $P_d \cap S_b^0 \neq \phi$. Firstly, we assume that $d < a$ or $b < d$. According to the Theorem 2.1, $d < a \Longrightarrow S_a^1 \cap S_d^0 \neq \phi$. This condition indicates that there exists a binary string such that $a_1 a_2 \cdots a_{h-1} 1|_{a_h=1} = d_1 d_2 \cdots d_{h-1} 1|_{d_h=0}$ for $1 \leq h \leq \ell$. This is equivalent to $a_1 a_2 \cdots a_{h-1} 0|_{a_h=1} = b_1 b_2 \cdots b_{h-1} 0|_{b_h=0}$, that is, $S_a^1 \cap S_d^0 = \tilde{S}_a^1 \cap \tilde{S}_d^0$. Since $\tilde{S}_d^0 \subseteq P_d$, $d < a \Longrightarrow \tilde{S}_a^1 \cap P_d \neq \phi$. Similarly, owing to the $b < d \Longrightarrow S_d^1 \cap S_b^0 \neq \phi$ and $S_d^1 \subseteq P_d$, $b < d \Longrightarrow P_d \cap S_b^0 \neq \phi$. Lastly, we assume that $P_d \cap \tilde{S}_a^1 \neq \phi$ or $P_d \cap S_b^0 \neq \phi$. When $\tilde{S}_a^1 \cap P_d \neq \phi$, their exists common elements denoted as $a_1 a_2 \cdots a_{h-1} 0|_{a_h=1} = d_1 d_2 \cdots d_{h-1} d_h$. These common elements indicate that $a_i = d_i$ for $1 \leq \forall i \leq h-1$ and $a_h > d_h$. Thus, $\tilde{S}_a^1 \cap P_d \neq \phi \Longrightarrow a > d$. When $P_d \cap S_b^0 \neq \phi$, there exist binary strings such that $d_1 d_2 \cdots d_{h-1} d_h = b_1 b_2 \cdots b_{h-1} 1|_{b_h=0}$, which means that $d_i = b_i$ for for $1 \leq \forall i \leq h-1$ and $d_h > b_h$. Therefore, $d > b$. □

### 2.3 Bloom Filter

H. Bloom proposed a probabilistic data structure called a Bloom filter [1] that can check whether an element $x$ is contained in a set $S$. It consists of an array of length $m$ and $k$ hash functions. There are two algorithms denoted by const.BF and check.BF. The const.BF takes a set $S$ as an input and generates a Bloom filter $\mathsf{BF}_S$. The check.BF takes a Bloom filter $\mathsf{BF}_S$ and an element $x$ as input and

**Algorithm 1** const.BF($S$)

**Require:** A set $S$, $k$ hash functions $\mathcal{H} = \{h_1, \cdots, h_k\}$
**Ensure:** A Bloom filter $\mathsf{BF}_S$

```
for i = 0 to m − 1 do
    BF_S[i] ⟵ 0
end for
for all x ∈ S do
    for i = 0 to k − 1 do
        j = h_i(x)
        if BF_S[j] = 0 then
            BF_S[j] ⟵ 1
        end if
    end for
end for
```

**Algorithm 2** check.BF($\mathsf{BF}_S, x$)

**Require:** A Bloom filter $\mathsf{BF}_S$, $x$, $k$ hash function $\mathcal{H} = \{h_1, \cdots, h_k\}$
**Ensure:** True if $x \in S$, False otherwise

```
for i = 0 to k − 1 do
    j = h_i(x)
    if BF_S[j] = 0 then
        return False
    end if
end for
return True
```

then outputs $x \in S$ or $x \notin S$. A false positive occurs when $\mathsf{check.BF}_S$ outputs $x \in S$ probabilistically, even if $x \notin S$. On the other hand, check.BF always outputs $x \in S$ if $x \in S$—that is, a false negative does not occur.

Hereafter, we explain how to generate the $\mathsf{BF}_S$ representing a set $S$. The algorithm 1 shows the procedure of generating a Bloom filter. It takes a set $S$ as an input and outputs the Bloom filter $\mathsf{BF}_S$. Initially, all bits in an array are set to 0. To insert an element $x \in S$ into the filter, the element is hashed using $k$ hash functions to obtain $k$ index numbers. The bits at these indexes are set to 1—that is, we set $\mathsf{BF}_S[h_i(x)] = 1$ for $1 \le i \le k$.

Next, we explain how to check whether an element is contained in a set. Algorithm 2 shows the procedure of checking whether an element is contained in a set $S$. It takes the Bloom filter $\mathsf{BF}_S$ and an element $x$ as inputs, and outputs $x \in S$ or $x \notin S$. To check if the element $x \in S$, $x$ is hashed using the $k$ hash functions; all locations at which $x$ is hashed are checked. The element $x$ is considered to not be in $S$ if one place at checked locations is 0; otherwise, $x$ is probably in $S$. Some false positive matches may occur—that is, it is possible for all $\mathsf{BF}_S[h_i(y)]$ to be set to 1 because of some collisions of hash values on the Bloom filter even if $y \notin S$.

The false positive rate (FPR) denoted by $\sigma$ is $\sigma = (1-(1-\frac{1}{m})^{kw})^w \approx (1-e^{kw/m})^k$ [8]. Given a set $S$ with at most $w$ elements and a length of $\mathsf{BF}_S$ denoted by $m$, the number of hash functions denoted by $k$ that minimizes FPR is $k = (m/w)\ln 2$. When $e^{-(kw/m)} = 1/2$, $\sigma = (1/2)^k \approx (0.6185)^k$. However, false negatives are not possible; thus, Bloom filters have a 100% recall rate. Given a FPR, denoted by $\sigma$, and the cardinality of a set, denoted by $w$, a length of $\mathsf{BF}_S$ is $m = -(w \cdot \ln\sigma)/(\ln^2\sigma)$.

### 2.4 Exponential ElGamal Encryption

Exponential ElGamal encryption is a public key encryption based on the discrete logarithm problem, which achieves IND-CPA security under the DDH assumption. The system parameter is denoted by $params = (\mathbb{F}_p, g, q)$, where $\mathbb{F}_p$ is a finite field, $g \in \mathbb{F}_p$ is a basepoint, and $q$ is a prime order of $g$. There are three algorithms that are called key generation, encryption, and decryption.

- Key generation: $\mathsf{KeyGen}(params) \to (x, y)$
  A key generation algorithm takes $params$ as an input. It samples a random integer $x \xleftarrow{U} \mathbb{Z}_q^*$[†] and then computes $y = g^x \bmod p$. Finally, it outputs $x$ as a secret key and $y$ as a public key.
- Encryption: $\mathsf{Enc}(y, d) \to (u, v)$
  An encryption algorithm takes a public key $y$ and a message $d$ as inputs. It samples a random integer $r \xleftarrow{U} \mathbb{Z}_q^*$ and then computes $u = g^r \bmod p$ and $v = g^d \cdot y^r \bmod p$. Finally, it outputs $(u, v)$ as a ciphertext of the message $d$.
- Decryption: $\mathsf{Dec}(x, (u, v)) \to g^d \bmod p$
  A decryption algorithm takes a secret key $x$ and a ciphertext $(u, v)$ as inputs and then computes $g^d = v \cdot u^{-x} \bmod p$. Finally, it outputs $g^d \bmod p$ as a result of decryption.

Exponential ElGamal encryption has the following properties: Let $s$, $t$, and $c$ be integers in $\mathbb{Z}_q$. The encryption algorithm satisfies the following formulas: $\mathsf{Enc}(y, s+t) = \mathsf{Enc}(y, s) \cdot \mathsf{Enc}(y, t)$ and $\mathsf{Enc}(y, c \cdot s) = \mathsf{Enc}(y, s)^c$.

Exponential ElGamal encryption is referred to as additively homomorphic encryption, which has the aforementioned properties. Another famous additively homomorphic encryption is Paillier encryption [12]. Exponential ElGamal encryption is more efficient than Paillier encryption. However, the decryption algorithm outputs $g^d \bmod p$. Since the discrete logarithm problem is hard for any probabilistic polynomial-time algorithm, it is usually impossible to decrypt $d$ from $g^d \bmod p$. In [13], it is necessary to learn the original message $d$ using the decryption algorithm. Thus, Peng et.al should employ Paillier encryption. In our scheme, we use a improved Exponential ElGamal encryption that generates the public key such that $y = g^{-x} \bmod p$. Therefore, it is not necessary for decryption to calculate a field inverse—that is, the decryption algorithm is denoted by $\mathsf{Dec}(x, (u, v)) = v \cdot u^x \equiv g^d \bmod p$ instead of $v \cdot u^{-x} \equiv g^d \bmod p$ in the original Exponential ElGamal encryption.

### 2.5 Secure Two-Party Interval Test

We begin by defining the following interval test and its security model. If given two sets instead of the prefix encoding

[†] $x \xleftarrow{U} \mathbb{Z}_q^*$ represents that a integer $x$ is sampled uniformly at random from a set $\mathbb{Z}_q^*$.

set and the 0/1 encoding set, the notions are changed to the case of the empty-set check.

**Definition 2.6** (Secure two-party interval test): A secure two-party interval test assumes that there are two parties—a client and a server. The client has a secret integer $d \in \{0,1\}^\ell$ and the server has a secret interval $I = [a,b] \in \{0,1\}^\ell \times \{0,1\}^\ell$, where $\ell$ is the number of binary digits in the binary string of integers. The secure two-party interval test does not allow the client to learn any information except for $d \in I$ or $d \notin I$. During and after executing the test, the server does not learn any information.

Our scheme aims to preserve both the client's secret integer $d$ and the server's secret interval $I = [a,b]$ during and after executing the scheme. We define a security model for the secure two-party interval test.

**Definition 2.7** (Security model for the secure interval test): We define a scheme of the secure two-party interval test as being secure if it achieves the following properties.

- Client privacy I (CPI): Any Probabilistic Polynomial-Time Algorithm (PPTA) cannot distinguish the data strings that the client sends to the server from random strings.
- Client privacy II (CPII): Server cannot distinguish received data strings from random strings even if the server utilizes own knowledge.
- Server privacy I (SPII): Any PPTA cannot distinguish the data strings that the server sends to the client from random strings.
- Server privacy II(SPII): Client cannot learn any information except for $d \in I$ or $d \notin I$ even if the client utilizes own knowledge—that is, client cannot learn the server's secret interval $I = [a,b]$.

## 3. Related Work

Secure interval test protocols [7], [10], [11] have been studied as well as the secure comparison protocol ever since Yao proposed the milionaire's problem [17]. Not only the secret sharing schemes, but also the (fully) homomorphic encryption can be applied to constructing the secure multi–party computation. Nishide and Ohta [11] proposed a secret sharing based constant-round secure interval test protocol. Based on their idea, Morita and Attrapadung [10] proposed a secure interval test protocol employing client–aid model, in which clients not only provide input but also can generate and secret–share correlated randomness to server. Such correlated randomness is used by $N$ servers to make secure computation more efficient. However, this model restricts that any server is not allowed to collude with any client. Lin and Tzeng [7] proposed a homomorphic encryption based secure comparison protocol employing the 0/1 encoding. Since our protocol employs the homomorphic encryption, the rest of this section summarizes the Lin and Tzeng [7]'s protocol in detail.

### 3.1 Secure Comparison Protocol Proposed by Lin et al.

Lin and Tzeng proposed a two-round protocol for solving the Millionaires' problem in the presence of honest–but–curious adversaries. By using the 0/1 encoding, the protocol turns data comparison to the problem of finding the intersection of two sets. Let Alice and Bob be parties in the protocol. Alice has an integer $a \in \{0,1\}^\ell$ and Bob has an integer $b \in \{0,1\}^\ell$. The protocol does not allow Alice to learn any information except for $a > b$ or $a \leq b$. Furthermore, Bob cannot learn any information from the protocol. We now explain the procedure of the protocol briefly. Initially, Alice first sets up Exponential ElGamal encryption and obtains a secret key $x$ and public key $y$; then, Alice executes the 1–encoding of $a$ and obtains the 1–encoding set $S_a^1$. Similarly, Bob executes the 0–encoding of $b$ and obtains the 0–encoding set $S_b^0$. Let $a_i \in \{0,1\}$ be the $i$–th digit of the binary string $a = a_1 a_2 \cdots a_\ell \in \{0,1\}^\ell$. We define $\overline{a}_i = 1 - a_i$. Alice constructs a $2 \times \ell$ table $T[i,j]$ for $i \in \{0,1\}$ and $1 \leq j \leq \ell$ as follows: she samples $\ell$ random integers $[r_1, \cdots, r_\ell] \xleftarrow{U} (\mathbb{Z}_q^*)^\ell$, and computes $\ell$ $\mathsf{Enc}(y,0)$ and $\mathsf{Enc}(y,r_j)$ for $1 \leq j \leq \ell$. Finally, she executes $T[a_j, j] \leftarrow \mathsf{Enc}(y,0)$ and $T[\overline{a}_j, j] \leftarrow \mathsf{Enc}(y,r_j)$ for $1 \leq j \leq \ell$.

She sends the $2 \times \ell$ table $T[i,j]$ to Bob. Bob executes $c_\beta = T[t_1,1] \cdot T[t_2,2] \cdots T[t_\beta,\beta]$ for all $t_\beta \in S_b^0$. Let $n$ be $n = \ell - |S_b^0|$. Bob samples $n$ random integers $[r'_1, \cdots, r'_n] \xleftarrow{U} (\mathbb{Z}_q^*)^n$ and generates $n$ ciphertexts $c'_1, \cdots, c'_n$, in which $c'_1 \leftarrow \mathsf{Enc}(y, r'_1), \cdots, c'_n \leftarrow \mathsf{Enc}(y, r'_n)$, and then sends $\ell$ ciphertexts denoted by $c_1, \cdots, c_{|S_b^0|}, c'_1, \cdots, c'_n$ to Alice. Finally, Alice decrypts all received ciphertexts and obtains plaintexts denoted by $d_0, d_1, \cdots, d_\ell$. By Theorem 2.1, if $g^0 \bmod p \in [d_0, d_1, \cdots, d_\ell]$, then Alice gets $a > b$. Otherwise, Alice gets $a \leq b$. This protocol cannot be expanded to a secure interval test easily because it is impossible to check whether $a \leq d$ and $d \leq b$ simultaneously. Repeating a secure comparison protocol from 2 times discloses information about a secret range.

## 4. Our Secure Two–Party Interval Test

We present our secure two-party interval test in this section. First, we apply a private set intersection proposed by Miyaji et.al [9] to an interval test. Then, we present a new interval test by introducing a method of checking whether an intersection of two sets is empty. We assume that a client has a secret integer $d \in \{0,1\}^\ell$ and a server has a secret interval $I = [a,b] \in \{0,1\}^\ell \times \{0,1\}^\ell$, and $\ell$ denotes the length of binary digits of integers.

### 4.1 Application of PSI to Interval Test

We present how to apply PSI to a two-party interval testing. As the Lemma 2.1 shows, $P_d \cap S_I \neq \phi$ if and only if $d \in [a,b]$. Note that $S_I$ denotes as $\tilde{S}_a^1 \cup S_b^0$ in this paper. So if

we compute two-party private set intersection [9], then we can check whether $P_d \cap S_I \neq \phi$ and learn the result of the interval test such as $d \notin I$ or $d \in I$. This protocol consists of three phases, which are initialization, online phase, and offline phase. The procedure of this protocol is as follows.

- Initialization:
  1. Client sets up Exponential ElGamal encryption and obtains a system parameter, secret key, and public key such as $(params = (\mathbb{F}_p, g, q), x, y)$.
  2. Client gets the prefix string set $P_d$ of the integer $d$. We define $C$ as $P_d$.
  3. Server applies the 0/1 encoding into its own secret interval $I = [a, b]$ and obtains the 0–encoding set $S_b^0$ of $b$ and new 1–encoding set $\tilde{S}_a^1$ of $a$. We define $S$ as $S_I$.
- Offline phase:
  1. Client and server execute $\mathsf{BF}_C \leftarrow \mathsf{const.BF}(C)$ and $\mathsf{BF}_S \leftarrow \mathsf{const.BF}(S)$, respectively.
  2. Client runs the encryption algorithm $\mathsf{Enc}(y, 1 - \mathsf{BF}_C) = [\mathsf{Enc}(y, 1 - \mathsf{BF}_C[0]), \cdots, \mathsf{Enc}(y, 1 - \mathsf{BF}_C[m-1])]$, and sends it to the server.
- Online phase: Server computation
  1. Server runs the encryption algorithm $\mathsf{Enc}(y, 1 - \mathsf{BF}_S) = [\mathsf{Enc}(y, 1 - \mathsf{BF}_S[0]), \cdots, \mathsf{Enc}(y, 1 - \mathsf{BF}_S[m-1])]$.
  2. Server samples $m$ random integers $r_0, \cdots, r_{m-1} \xleftarrow{U} (\mathbb{Z}_q^*)^m$ and calculates $\mathsf{Enc}(y, \mathsf{IBF}) = [(\mathsf{Enc}(y, 1 - \mathsf{BF}_C[0]) \cdot \mathsf{Enc}(y, 1 - \mathsf{BF}_S[0]))^{r_0}, \cdots, (\mathsf{Enc}(y, 1 - \mathsf{BF}_C[m-1])] \cdot \mathsf{Enc}(y, 1 - \mathsf{BF}_S[m-1]))^{r_{m-1}}]$. Finally, the server sends $\mathsf{Enc}(y, \mathsf{IBF})$ to client.
  3. Client executes the decryption algorithm $\mathsf{IBF} \leftarrow \mathsf{Dec}(x, \mathsf{Enc}(y, \mathsf{IBF}))$.
- Online phase: Client Test:
  1. For each $e \in C$, client executes $result \leftarrow \mathsf{check.BF}(\mathsf{IBF}, e)$ and outputs $d \notin I$ if $result = True$. For any $e \in C$, it outputs $d \in I$ if any $result = False$.

The correctness of this protocol is trivial because two-party PSI is able to find $C \cap S$. $C \cap S \neq \phi$ if at least one element $e \in C$ is contained in $S$. It means $d \notin I$. This protocol computes $P_d \cap S_I \neq \phi$ which is more information than necessary since we just need whether $P_d \cap S_I \neq \phi$. We polish the PSI to check whether $P_d \cap S_I \neq \phi$ and reduce to the total computational and communicational compplexity.

### 4.2 Our Protocol

We now present our secure two-party interval test, which is an extension of PSI-based interval test in Sect. 4.1. To reduce the redundant computation, we the following Lemma 4.1.

**Lemma 4.1:** Let $S_1$ and $S_2$ be sets such that $S_1 \cap S_2 \neq \phi$. We set $\mathsf{BF}_{S_1} \leftarrow \mathsf{const.BF}(S_1)$ and $\mathsf{BF}_{S_2} \leftarrow \mathsf{const.BF}(S_2)$. Then, the following feature holds: $S_1 \cap S_2 \neq \phi \Longrightarrow \exists i \in [0, m)$ $s.t.$ $\mathsf{BF}_{S_1}[i] \wedge \mathsf{BF}_{S_2}[i] = 1$.

According to the Lemma 4.1, we only check whether $\exists i \in [0, m)$ $s.t.$ $\mathsf{BF}_{P_d}[i] \wedge \mathsf{BF}_{S_I}[i] = 1$ and then output $d \notin I$ if there exists $i \in [0, m)$. Otherwise, we output $d \in I$. However, this method may output the wrong results, that is, it outputs $d \notin I$ even if $d \in I$. After we explain the construction of our protocol, we prove that our protocol achieves the correctness and defined security goals in Sect. 2.7. In Sect. 5, we consider how to set up an optimized Bloom filter for our protocol. Our protocol uses the same the initialization as PSI-based interval test in Sect. 4.1. Thus, we omit the initialization. The construction is as follows:

- Offline phase:
  1. Client and server execute $\mathsf{BF}_C \leftarrow \mathsf{const.BF}(C)$ and $\mathsf{BF}_S \leftarrow \mathsf{const.BF}(S)$, respectively.
  2. Client runs the encryption algorithm $\mathsf{Enc}(y, \mathsf{BF}_C) = [\mathsf{Enc}(y, \mathsf{BF}_C[0]), \cdots, \mathsf{Enc}(y, \mathsf{BF}_C[m-1])]$ and sends it to the server.
- Online phase: Server computation
  1. Server calculates $\mathsf{Enc}(y, \sum_{i=0}^{m-1} \mathsf{BF}_C[i] \cdot \mathsf{BF}_S[i]) = \prod_{i=0}^{m-1} \mathsf{Enc}(y, \mathsf{BF}_C[i])^{\mathsf{BF}_S[i]}$.
  2. Server samples random integer $r \xleftarrow{U} \mathbb{Z}_q^*$, and calculates $\mathsf{Enc}(y, r \cdot \sum_{i=0}^{m-1} \mathsf{BF}_C[i] \cdot \mathsf{BF}_S[i]) = \mathsf{Enc}(y, \sum_{i=0}^{m-1} \mathsf{BF}_C[i] \cdot \mathsf{BF}_S[i])^r$.
  3. Server sends $\mathsf{Enc}(y, r \cdot \sum_{i=0}^{m-1} \mathsf{BF}_C[i] \cdot \mathsf{BF}_S[i])$ to the client.
  4. Client runs the decryption algorithm $\xi \leftarrow \mathsf{Dec}\Big(x, \mathsf{Enc}(y, \mathsf{Enc}(y, r \cdot \sum_{i=0}^{m-1} \mathsf{BF}_C[i] \cdot \mathsf{BF}_S[i]))\Big)$.
- Online phase: Client Test:
  1. Client learns $d \notin I$ if $\xi \neq g^0 \bmod p$. Otherwise, it learns $d \in I$.

### 4.3 Correctness and Security

We show that our protocol exhibits correctness and security as follows. Firstly, we prove that our protocol is accurate.

**Theorem 4.1** (Correctness): Our protocol outputs $d \notin I$ if for any $d \notin I$. It means that the following formula holds: $\mathsf{Pr}[\xi \neq g^0 \bmod p | d \notin I] = 1$.

**Proof 4.1:** We assume that $d \notin I = [a, b]$, which means that $C \cap S \neq \phi$ by Lemma 2.1. Client finally gets $\xi$ by decrypting $\mathsf{Enc}(y, \mathsf{Enc}(y, r \cdot \sum_{i=0}^{m-1} \mathsf{BF}_C[i] \cdot \mathsf{BF}_S[i]))$— that is, $\xi = g^{r \cdot \sum_{i=0}^{m-1} \mathsf{BF}_C[i] \cdot \mathsf{BF}_S[i]} \bmod p$. Since this assumption supposes that there exists at least one common element in $C$ and $S$, $i \in [0, m)$ must exist such that $\mathsf{BF}_C[i] = 1$ and $\mathsf{BF}_S[i] = 1$. Therefore, $\xi \neq g^0 \bmod p$ if $d \notin I$. Our protocol holds $\mathsf{Pr}[\xi \neq g^0 \bmod p | d \notin I] = 1$. □

Secondly, we analyze the security of our protocol to prove that it achieves the defined security goals.

**Theorem 4.2:** Our protocol achieves the defined client privacy I and client privacy II against honest-but-curious adversaries under the DDH assumption.

**Proof 4.2:** We use the contraposition to prove the Theorem 4.2. Supposing that there exists a distinguisher $D$ that can distinguish the data strings that the client sends to the server from random strings with probability $\epsilon$, then there exists a polynomial-time adversary $\mathcal{A}$ that can solve the DDH problem with non-negligible probability. We construct the polynomial-time adversary $\mathcal{A}$, which uses the distinguisher $D$ as a subroutine to solve the DDH problem.

Firstly, the adversary $\mathcal{A}$ receives a tuple $(g, g^\alpha, g^\beta, g^\zeta)$ and samples $m$ random integers $r_0, \cdots, r_{m-1} \xleftarrow{U} (\mathbb{Z}_q^*)^m$ and $m-1$ random integers $\gamma_1, \cdots, \gamma_{m-1} \xleftarrow{U} (\mathbb{Z}_q^*)^{m-1}$. After, the adversary generates $\delta$ such that $\delta = ((g^\alpha, r_0 \cdot g^\zeta), (g^\alpha)^{\gamma_1}, r_1 \cdot (g^\zeta)^{\gamma_1}, \cdots, (g^\alpha)^{\gamma_{m-1}}, r_{m-1} \cdot (g^\zeta)^{\gamma_{m-1}})$ and inputs $\delta$ in the distinguisher $D$. Since $\delta$ is same distribution as the data strings that the client sends to the server in our protocol, the distinguisher $D$ outputs 1 with probability of $\epsilon$ if $\zeta = \alpha \cdot \beta$—that is, the tuple is the DDH tuple $(g, g^\alpha, g^\beta, g^{\alpha\cdot\beta})$. On the other hand, since $\delta$ is not same distribution in our protocol, the distinguisher $D$ outputs 1 with a probability of 1/2 if the tuple is the non-DDH tuple. Thus, the adversary $\mathcal{A}$ can solve the DDH problem with a probability of $1/2 + \epsilon$, which means the adversary can solve the DDH problem with non-negligible probability. By using the contraposition, for any PPTA, the data that the client sends to the server cannot be distinguished from random strings under the DDH assumption. Therefore, our protocol achieves client privacy I. In addition, the server cannot decrypt all received data from client, so that server cannot learn any information about $d$. Thus, our protocol achieves client privacy II. □

**Theorem 4.3:** Our protocol achieves server privacy I and server privacy II against honest-but-curious adversaries.

**Proof 4.3:** Supposing that there exists a distinguisher $D$ that can distinguish the data strings that the server sends to the client from random strings with probability $\epsilon$, then there exists a polynomial-time adversary $\mathcal{A}$ that can solve the DDH problem with non-negligible probability. In the same way of the Theorem 4.2, we can show that for any PPTA, the data the the server sends to the client cannot be distinguished from random strings under the DDH assumption. Therefore, server privacy I is achieved. In online phase 4) of our protocol, the client learns $\xi$ such that $\xi = g^{r\cdot\sum_{i=0}^{m-1} \mathsf{BF}_C[i]\cdot\mathsf{BF}_S[i]}$. According to our protocol, $\xi = g^0 \bmod p$ means $d \in I$ and $\xi \neq g^0 \bmod p$ means $d \notin I$. In the case of $d \in I$, client gets only $\mathsf{BF}_S[i] = 0$ for all $i$ such that $\mathsf{BF}_C[i] = 1$. In the case of $d \notin I$, the client learns nothing because $\xi$ is a random integer depending on $r$. Therefore, our protocol achieves server privacy II. □

In summary, based on the Definition 2.7, we conclude that our protocol is secure against honest-but-curious adversaries under the DDH assumption.

## 5. Analysis

### 5.1 Bloom Filter for the Empty-Set Check

It is clear that our protocol in Sect. 4.2 employs the Bloom filter to execute the empty-set check of the set intersection of two sets. We use Lemma 4.1 and the 0/1 encoding to construct our protocol. Some false positive matches may occur—that is, it is possible for arrays $\mathsf{BF}_C[i]$ and $\mathsf{BF}_S[i]$ to be set to 1 even if $C \cap S = \phi$. We have analyzed the probability of false positive matches occurring in our protocol as follows.

In Sect. 2.3, the probability of obtaining a false positive for the Bloom filter $\mathsf{BF}_S$ is $\sigma = (1-(1-\frac{1}{m})^{kw})^w \approx (1-e^{kw/m})^k$, where $m$, $k$, and $w$ denote a size of Bloom filter, the number of hash functions and a cardinality of set $S$, respectively. It means that the probability of a false positive match occurring for an element $y \notin S$. Let $S_1$ and $S_2$ be sets such that $S_1 \cap S_2 = \phi$, $|S_1| = w_1$, and $|S_2| = w_2$. Considering that we run $\mathsf{check.BF}(y, \mathsf{BF}_{S_1})$ for all $y \in S_2$, we evaluate the probability that $\mathsf{check.BF}(y, \mathsf{BF}_{S_1})$ outputs $True$ for at least one element $y \in S_2$. The probability that a false positive match does not occur for all $y \in S_2$ is $(1-\sigma)^{w_2}$. Therefore, by considering complementary events, the probability of false positive matches $\mu$ is $\mu = 1-(1-\sigma)^{w_2} = 1-\left(1-(1-(1-\frac{1}{m})^{kw_1})^k\right)^{w_2}$. As the number of hash functions increases, FPR becomes higher. In order to achieve a lower FPR, we set $k = 1$. In this case, the probability of false positive matches $\mu$ is $\mu = 1-(1-\frac{1}{m})^{w_1 w_2}$. Therefore, by rearranging the formula above, the optimized size of the Bloom filter is $m = 1/(1-(1-\mu)^{1/(w_1 w_2)})$.

We verify the validity of the formulas $\mu$ by developing the following experiment. In the experiment, given $\ell = 16, 32, 64$, we sample $d \in \{0,1\}^\ell$, $a \in \{0,1\}^\ell$, and $b \in \{0,1\}^\ell$ such that $d \in [a, b]$ randomly for each $\ell$ and construct the prefix encoding set $P_d$ of $d$, the new 1–encoding set $\tilde{S}_a^1$ of $a$ and the 0–encoding set $S_b^0$ of $b$. We take $\mu$, $w_1 = |P_d|$ and $w_2 = |S_I|$ as inputs and decide the size of the Bloom filter by calculating $m = 1/(1-(1-\mu)^{1/(|P_d|\cdot|S_I|)})$. We check whether $P_d \cap S_I = \phi$ by using two Bloom filters $\mathsf{BF}_{P_d}$ and $\mathsf{BF}_{S_I}$, the sizes of which are $m$. We repeat this experiment $10^4$ times and compute the probability of obtaining a false positive. Table 1 indicates that the comparison between the theoretical FPR $\mu$ and experimental false positive probabilities. Table 1 shows the error rates of the theoretical FPR and experimental false positive probabilities are less than 5%. These results validate our claims.

**Table 1** Comparison with FPR with $w_2 = |S_I|$

| FPR | Bit length | | |
|---|---|---|---|
| | 16 | 32 | 64 |
| 0.2 | 0.195 | 0.196 | 0.192 |
| 0.1 | 0.104 | 0.105 | 0.098 |
| 0.05 | 0.052 | 0.052 | 0.052 |

### 5.2 Optimized Bloom Filter for the Proposed Protocol

In order to execute our protocol, we initially decide the size of Bloom filters $m$. It relies on the FPR $\mu$, the size of a prefix string $|P_d|$, and the size of the union of the 0–encoding set and a new 1–encoding set, which denotes $S_I = \tilde{S}_a^1 \cup S_b^0$. However, clients do not learn $|S_I|$ because our protocol does not allow the server to convey this to clients for achieving security. Thus, clients need to get the size of Bloom filter $m$ even if they do not know $|S_I|$. In this section, we discuss how to construct an optimized Bloom filter even if the client does not know $|S_I|$.

**Lemma 5.1:** Given $\ell$, the size of a prefix string set of $d$ is $|P_d| = \ell$.

By the Lemma 5.1, we set $\omega_1 = \ell$ in our protocol. On the contrary, a set $S_I$ such as $\omega_2 = |S_I|$ is not determined uniquely. Consequently, even given $|S_I|$, we cannot specify a set $S_I$ such that $|S_I|$. Since $|S_I|$ has affected the FPR $\mu$, we theoretically discuss how to construct an optimized Bloom filter. There are two ways to determine $\omega_2$: one in which the server determines $\omega_2$ by considering the $S_I$ and another in which the client determines $\omega_2$ independent on the $S_I$.

**Lemma 5.2:** Given $\ell$, let $w_2$ be $|S_I|$, that is, $w_2 = |S_I|$.

- In the case of $\omega_2 = 2\ell - 2$, the number of possible intervals is 1 such that $I = [\sum_{i=0}^{\ell-2} 2^i, 2^{\ell-1}]$.
- In the case of $\omega_2 = 1$, the number of possible intervals is $2\ell$ such that $I = [0, \sum_{i=0}^{\ell-2} 2^i], [0, \sum_{i=0}^{\ell-3} 2^i + 2^{\ell-1}], \cdots, [0, \sum_{i=1}^{\ell-1} 2^i]$, $[2^0, 2^\ell - 1], [2^1, 2^\ell - 1], \cdots, [2^{\ell-1}, 2^\ell - 1]$.
- In the case of $\omega_2 = \ell$, the number of possible intervals is $\sum_{i=0}^{\ell} \binom{\ell}{i}!$.

By the Lemma 5.2, the probability of $|S_I| = 1$ or $|S_I| = 2\ell-2$ is lesser than the probability of $|S_I| = \ell$. Therefore, we establish the way in which $\omega_2 = \ell$. We verify how much this way has affected the FPR by developing an experiment. This experiment is almost same as the previous experiment. The difference is to calculate $m = 1/(1 - (1 - \mu)^{1/\ell^2})$ to decide the size of the Bloom filter. Table 2 indicates that for each $\ell = 16, 32, 64$, the comparison between the theoretical FPR $\mu$ and experimental false positive probabilities and the concrete size of the Bloom filter. Table 2 shows the error rates of the theoretical FPR and experimental false positive probabilities are less than 10%. Therefore, our protocol uses this method to initially construct the Bloom filter.

**Table 2** The size of Bloom filter $m$ and FPR with $w_2 = \ell$

| $\ell$ | 16 | | 32 | | 64 | |
|---|---|---|---|---|---|---|
| FPR | $m$ | FPR | $m$ | FPR | $m$ | FPR |
| 0.2 | 1148 | 0.185 | 4589 | 0.202 | 18356 | 0.192 |
| 0.1 | 2430 | 0.095 | 9720 | 0.098 | 38877 | 0.095 |
| 0.05 | 4991 | 0.056 | 19964 | 0.045 | 79855 | 0.048 |

## 6. Evaluation on Practical Situation

In this section we show the experimental result of our protocol in some practical situations. Our protocol is suitable for a small range test. For example, it can be used to check whether an annual income is contained in a certain interval given by an insurance company when we contract an insurance. The annual income decides a benefit of the insurance product. Typically, 27-bit length is enough for representing the annual income: $2^{27} = 134217728$. 32-bit binary representation is enough for handing practical problems: Internet Protocol version 4 (IPv4) uses 32-bit addresses which limits the address space. As for other small range cases, it can be used to check an academic record: the grade point average (GPA), the TOEFL score, etc. In these cases, 8-bit binary representation is enough. Here, we evaluate our protocol in such small ranges equal or less than 32 bits.

We implemented a prototype in Python3 using the PyCryptodome (PyCrypt) library (version 3.8.2). PyCrypt is used for parameters generation and random number generation in the Exponential ElGamal Encryption. To instantiate a hash function for the Bloom filter, we used SHA–1 in PyCrypt.

The most expensive process of our protocol is Bloom filter encryption. In this paper, we divide our protocol into the offline phase and online phase. Bloom filter encryption is executed in the offline phase, which means that we can execute Bloom filter encryption before the interaction between a client and a server. Our aim is to reduce the time clients take to obtain a result.

All experiments were performed on a single server machine and a single laptop machine. We used the Ubuntu 18.04 LTS operating system with Intel(R) Xeon(R) Gold 6130 2.10 GHz CPU and 200 GB memory as a single server machine. We also used a Windows 10 Pro Education operating system with Intel(R) Core(TM) i7-3770 3.40GHz CPU and 8GB memory as a single laptop machine. We measured the performance for $\ell \in \{4, 8, 16, 24, 32\}$ bits integers. The time required for whole server processing and whole client processing in the online phase was measured. We set the security parameter to $\lambda = 80$, and group size $|\mathbb{F}_p| = 2048$ and sub group size $|\mathbb{F}_q| = 160$ were used in reference to the NIST guidelines for key management. We evaluated our protocol with false positive rates FPR $\in \{0.2, 0.1, 0.05, 0.01, 0.001\}$.

### 6.1 Comparison of Theoretical Complexity

We compare our protocol with Lin and Tzeng's secure comparison protocol [7] and an application in Sect. 4.1. Table 3 indicates that the comparison of the computational and communicational theoretical complexity among these protocols. All protocols are secure against honest-but-curious adversaries, and they employ Exponential ElGamal encryption. In this paper, we separate processing in protocols into offline and online phases and evaluate these protocols. The offline phase is all processing before the server receives encryption

**Table 3** The comparison of theoretical complexity

| | | Computational complexity Client | Computational complexity Server | Comm. comp. |
|---|---|---|---|---|
| Lin and Tzeng's secure comparison protocol [7] | Offline | $6\ell E$ | - | Upload: $4\ell\|\mathbb{F}_p\|$ |
| | Online | $\ell(E+M)$ | $3(\ell - \|S_b^0\|)E + \sum_{t_\beta \in S_b^0} \|t_\beta\| M$ | Download: $2\ell\|\mathbb{F}_p\|$ |
| An application of PSI to Interval test in Sect. 4.1 | Offline | $2mE$ | - | Upload: $2m\|\mathbb{F}_p\|$ |
| | Online | $mE$ | $5mE + 2mM$ | Download: $2m\|\mathbb{F}_p\|$ |
| Our (Interval test based on Empty-set check) | Offline | $2mE$ | - | Upload: $2m\|\mathbb{F}_p\|$ |
| | Online | $E + M$ | $2(E + mM)$ | Download: $2\|\mathbb{F}_p\|$ |

**Table 4** Practical communication cost of upload in our protocol (80-bits security) (MB)

| FPR | Bit length 4 | 8 | 16 | 24 | 32 |
|---|---|---|---|---|---|
| 0.2 | 0.035 | 0.140 | 0.561 | 1.261 | 2.241 |
| 0.1 | 0.074 | 0.297 | 1.187 | 2.669 | 4.741 |
| 0.05 | 0.152 | 0.609 | 2.437 | 5.483 | 9.748 |
| 0.01 | 0.777 | 3.109 | 12.44 | 27.98 | 49.75 |
| 0.001 | 7.809 | 31.23 | 124.9 | 281.1 | 499.8 |

of the Bloom filter $\mathsf{Enc}(y, \mathsf{BF}_C)$ from the client. The online phase is all processing after the server receives encryption of the Bloom filter $\mathsf{Enc}(y, \mathsf{BF}_C)$ from the client. The computational complexity of a field multiplication, a field exponentiation and a field inverse are denoted by $M, E$ and $I$, respectively. Even though field multiplications are used in these protocols, its computational complexity is much lower than the computational complexity of a field exponentiation. The communicational complexity is denoted by the number of field elements in $\mathbb{F}_p$. Similarly, we separate communication into upload and download to evaluate these protocols. Note that the size of the Bloom filter $m$ is larger than $\ell$. Table 3 shows that the comparison of complexity between our protocol and other protocols. In [7] and an application in Sect. 4.1, the computational complexity of online phase and the communicational complexity of download depend on $\ell$ and the size of the Bloom filter $m$, respectively. On the other hand, the computational and communicational complexity of our protocol don't depend on any parameters—that is, the client receives only two field elements and requires just one decryption. However, the computational and communicational complexity of our protocol is larger than other protocols because the complexities depend on $m$. We conclude that our protocol needs to receive only two field elements, and the client executes just one decryption. Therefore, our protocol is faster than other protocols in online phase.

## 6.2 Comparison of Practical Complexity

We report the practical communication cost in Tables 4, 5, and 6. The communication cost of our protocol is bigger than the others. Precisely, our communication cost of upload is approximately 1000 times of the others communication cost. In regard in download, the server send only one ciphertext to the client in any FPR. Therefore, communication cost is the size of ciphertext. Namely, $2|\mathbb{F}_p|$ bits is the communication cost in download. On the other hand, the others' communication costs are almost same as the communication cost of upload.

**Table 5** Practical communication cost of upload and download in Lin and Tzeng's protocol [7] (80-bits security) (KB)

| | Bit length 4 | 8 | 16 | 24 | 32 |
|---|---|---|---|---|---|
| Upload | 4.0 | 8.0 | 16.0 | 24.0 | 32.0 |
| Download | 2.0 | 4.0 | 8.0 | 12.0 | 16.0 |

**Table 6** Practical communication cost of upload and download of an application in Sect. 4.1 (80-bits security) (KB)

| FPR | Bit length 4 | 8 | 16 | 24 | 32 |
|---|---|---|---|---|---|
| 0.2 | 14.0 | 27.0 | 54.0 | 81.0 | 108.0 |
| 0.1 | 20.0 | 39.0 | 77.0 | 116.0 | 154.0 |
| 0.05 | 25.0 | 50.0 | 100.0 | 150.0 | 200.0 |
| 0.01 | 39.0 | 77.0 | 154.0 | 231.0 | 307.0 |
| 0.001 | 58.0 | 116.0 | 231.0 | 346.0 | 461.0 |

**Table 7** Server's average runtime in our protocol (ms)

| FPR | Bit length 4 | 8 | 16 | 24 | 32 |
|---|---|---|---|---|---|
| 0.2 | 5.866 | 6.167 | 6.481 | 7.344 | 8.250 |
| 0.1 | 6.241 | 7.370 | 6.207 | 9.146 | 11.81 |
| 0.05 | 6.723 | 6.977 | 7.510 | 12.96 | 21.13 |
| 0.01 | 3.091 | 7.827 | 27.84 | 59.89 | 104.9 |
| 0.001 | 17.93 | 65.82 | 274.5 | 593.2 | 1078 |

**Table 8** Server's average runtime in Lin and Tzeng's protocol [7] (ms)

| Bit length 4 | 8 | 16 | 24 | 32 |
|---|---|---|---|---|
| 15.19 | 20.57 | 27.54 | 36.64 | 46.69 |

**Table 9** Server's average runtime of an application of PSI in Sect. 4.1 (ms)

| FPR | Bit length 4 | 8 | 16 | 24 | 32 |
|---|---|---|---|---|---|
| 0.2 | 49.70 | 86.24 | 162.5 | 237.9 | 313.0 |
| 0.1 | 66.94 | 120.3 | 226.0 | 335.8 | 443.6 |
| 0.05 | 81.06 | 151.5 | 292.0 | 431.8 | 573.1 |
| 0.01 | 119.9 | 226.8 | 444.3 | 659.5 | 873.2 |
| 0.001 | 174.0 | 336.1 | 660.6 | 982.9 | 1307 |

We compared our protocol with Lin and Tzeng's secure comparison protocol [7] and an application in Sect. 4.1. We implemented them in Python3 with Pycrypt library for the comparison. All measurements were conducted in the sequential processing mode and repeated these protocols 10 times and computed average runtime. The results are presented in Tables 7, 8, 9, 10, 11, and 12 (Figs. 1 and 2).

**Table 10** Client's average runtime in our protocol (ms)

| FPR | Bit length | | | | |
|---|---|---|---|---|---|
| | 4 | 8 | 16 | 24 | 32 |
| 0.2 | 1.967 | 1.852 | 3.183 | 6.072 | 3.589 |
| 0.1 | 3.937 | 2.458 | 1.923 | 3.512 | 2.361 |
| 0.05 | 3.049 | 5.423 | 2.515 | 2.469 | 3.899 |
| 0.01 | 4.456 | 1.834 | 1.872 | 1.769 | 1.757 |
| 0.001 | 2.212 | 1.756 | 1.313 | 1.849 | 2.635 |

**Table 11** Client's average runtime in Lin and Tzeng's protocol [7] (ms)

| Bit length | | | | |
|---|---|---|---|---|
| 4 | 8 | 16 | 24 | 32 |
| 7.227 | 15.54 | 28.64 | 34.39 | 43.90 |

**Table 12** Client's average runtime of an application in Sect. 4.1 (ms)

| FPR | Bit length | | | | |
|---|---|---|---|---|---|
| | 4 | 8 | 16 | 24 | 32 |
| 0.2 | 13.19 | 37.84 | 58.81 | 85.03 | 103.6 |
| 0.1 | 36.68 | 50.22 | 81.09 | 109.9 | 141.5 |
| 0.05 | 39.48 | 57.07 | 99.99 | 135.5 | 176.8 |
| 0.01 | 50.33 | 80.48 | 143.1 | 201.1 | 259.6 |
| 0.001 | 64.09 | 111.2 | 198.4 | 292.3 | 377.8 |

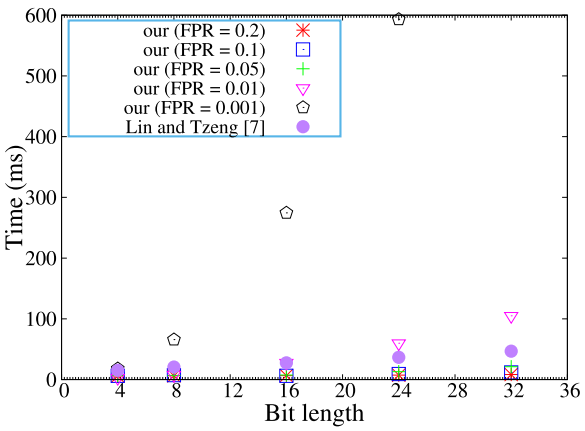


**Fig. 1** Server average runtime in online

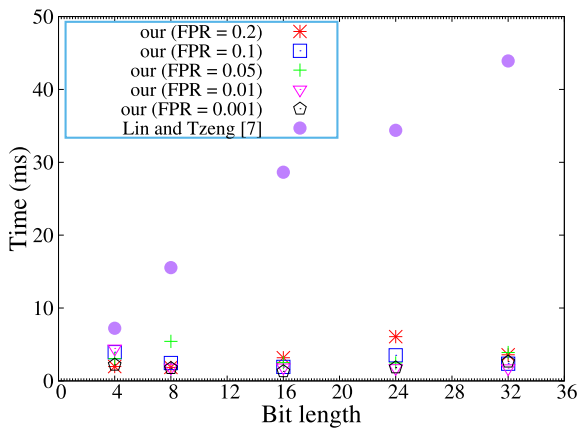


**Fig. 2** Client average runtime in online

Figure 1 shows that server's average runtime of our protocol is faster than Lin and Tzeng's secure comparison protocol except when FPR is less than 0.05, and an application in Sect. 4.1. Especially, Fig. 2 shows that client's average runtime is almost fixed time independent on the bit length of input in our protocol and faster than the others in any cases.

As we discussed the false positive probability of our protocol in Sect. 5.1, it relies on the size of a prefix string $w_1 = |P_d|$, and the size of the union of the 0–encoding set and a new 1–encoding set $w_2 = |\tilde{S}_a^1 \cup S_b^0|$. We should choose an appropriate FPR according to the $|P_d|$ and $|\tilde{S}_a^1 \cup S_b^0|$. As we discussed optimized Bloom filter for our protocol in Sect. 5.2, $|P_d|$ and $|\tilde{S}_a^1 \cup S_b^0|$ are set to $\ell$, where $\ell$ is bit length of integers. When $\ell = i$, the number of combination is $2^i$. This number is the same as the number of candidate element of set $P_d$ and $\tilde{S}_a^1 \cup S_b^0$. So it is desirable that the appropriate FPR is $2^{-\ell}$. In the cases of 8 bits or less, we can choose the appropriate FPR: FPR $= 2^{-4} = 0.0625$ when $\ell = 4$ and FPR $= 2^{-8} \approx 0.0039$ when $\ell = 8$. In these cases, our protocol is better than the others on the practical situations. On the other hand, the others is faster than our protocol when bit length is 16 or more because we cannot choose the appropriate FPR. However, our evaluation indicates that it is faster than the others for rough situations, in which the protocol is allowed to go wrong every several executions in average. Our protocol is available for rough cases even if bit length is 16 or more.

## 7. Conclusion

We proposed our new empty-set check, and designed a secure two-party interval test by using our empty-set check. Our scheme solves the problem of client and server complexity. To compare two integers, our scheme adopts the 0/1 encoding technique, and constructs a secure interval test. To avoid increasing complexity after a client sends encryption of a Bloom filter, our scheme has introduced a novel method of the empty-set check based on Bloom filter. We formally analyze security in our scheme and prove that it is secure against honest-but-curious adversaries under the DDH assumption. We also formally analyze the false positive probability and how to theoretically generate an optimized Bloom filter for our scheme, which shows that the error rate between the theoretical false positive probability and the experimental false positive probability is less than 10% if our scheme adopts $w_2 = \ell$. The theoretical comparison of results indicates that the complexity of our scheme after the client sends encryption of a Bloom filter is less than the others. We evaluated our protocol on practical situation; then the practical comparison of results indicates that the runtimes of a client and a server are faster than the others in online phase if the situation, that go wrong every several executions in average, is allowed. Thus, our scheme allows the client to quickly obtain the result of an interval test after sending data to the server.

## Acknowledgments

This work is partially supported by CREST (JPMJCR1404)

at Japan Science and Technology Agency, Project for Establishing a Nationwide Practical Education Network for IT Human Resources Development, Education Network for Practical Information Technologies and Innovation Platform for Society 5.0 at MEXT.

**References**

[1] B.H. Bloom, "Space/time trade-offs in hash coding with allowable errors," Commun. ACM, vol.13, no.7, pp.422–426, 1970.
[2] M. Burkhart and X. Dimitropoulos, "Fast private set operations with sepia," Tech. Rep. 345, 2012.
[3] E. De Cristofaro, J. Kim, and G. Tsudik, "Linear-complexity private set intersection protocols secure in malicious model," In Advances in Cryptology - ASIACRYPT 2010 - 16th International Conference on the Theory and Application of Cryptology and Information Security, Singapore, Dec. 5-9, 2010. Proceedings, vol.6477, pp.213–231, 2010.
[4] Y. Dou, H.C.B. Chan, M.H. Au, and Y. Mu, "Order-hiding range query over encrypted data without search pattern leakage," Comput. J., vol.61, no.12, pp.1806–1824, 2018.
[5] R. Egert, M. Fischlin, D. Gens, S. Jacob, M. Senker, and J. Tillmanns, "Privately computing set-union and set-intersection cardinality via bloom filters," In Information Security and Privacy - 20th Australasian Conference, ACISP 2015, Brisbane, QLD, Australia, June 29 - July 1, 2015, Proceedings, vol.9144, pp.413–430, 2015.
[6] L. Kissner and D.X. Song, "Privacy-preserving set operations," In Advances in Cryptology - CRYPTO 2005: 25th Annual International Cryptology Conference, Santa Barbara, California, USA, August 14-18, 2005, Proceedings, vol.3621, pp.241–257, 2005.
[7] H.-Y.Lin and W.-G. Tzeng, "An efficient solution to the millionaires' problem based on homomorphic encryption," In Applied Cryptography and Network Security, Third International Conference, ACNS 2005, New York, NY, USA, June 7-10, 2005, Proceedings, vol.3531, pp.456–466, 2005.
[8] M. Mitzenmacher and E. Upfal, "Probability and Computing: Randomized Algorithms and Probabilistic Analysis," Cambridge University Press, 2005.
[9] A. Miyaji, K. Nakasho, and S. Nishida, "Privacy-preserving integration of medical data - A practical multiparty private set intersection," J. Medical Systems, vol.41, no.3, pp.37:1–37:10, 2017.
[10] H. Morita and N. Attrapadung, "Client-aided two-party secure interval test protocol," In Cryptology and Network Security - 18th International Conference, CANS 2019, Fuzhou, China, Oct. 25-27, 2019, Proceedings, vol.11829, pp.328–343, 2019.
[11] T. Nishide and K. Ohta, "Constant-round multiparty computation for interval test, equality test, and comparison," IEICE Transactions, vol.90-A, no.5, pp.960–968, 2007.
[12] P. Paillier, "Public-key cryptosystems based on composite degree residuosity classes," In Advances in Cryptology - EUROCRYPT '99, International Conference on the Theory and Application of Cryptographic Techniques, Prague, Czech Republic, May 2-6, 1999, Proceeding, vol.1592, pp.223–238, 1999.
[13] K. Peng, F. Bao, and E. Dawson, "Correct, private, flexible and efficient range test," Journal of Research and Practice in Information Technology, vol.40, no.4, pp.275–289, 2008.
[14] Y. Sang and H. Shen, "Efficient and secure protocols for privacy-preserving set operations," ACM Trans. Inf. Syst. Secur., vol.13, no.1, pp.9:1–9:35, 2009.
[15] K. Shishido and A. Miyaji, "Secure online-efficient interval test based on empty-set check," In 14th Asia Joint Conference on Information Security, AsiaJCIS 2019, Kobe, Japan, Aug. 1-2, 2019, pp.56–63, 2019.
[16] A.C.-C. Yao, "Protocols for secure computations (extended abstract)," In 23rd Annual Symposium on Foundations of Computer Science, Chicago, Illinois, USA, 3-5 Nov. 1982, pp.160–164, 1982.
[17] A.C.-C. Yao, "How to generate and exchange secrets (extended abstract)," In 27th Annual Symposium on Foundations of Computer Science, Toronto, Canada, 27-29 Oct. 1986, pp.162–167, 1986.

**Katsunari Shishido** received the B.E. degree from the National Institute of Technology, Advanced course, Ibaraki College in 2016 and the M.S. degree from Osaka University in 2018. Since 2019, he has worked for the FUJITSU LABORATORIES LTD., Japan. His current research interests include information security and AI security. He received the 14th Asia Joint Conference on Information Security (AsiaJCIS 2019) Best Paper Award.

**Atsuko Miyaji** received the B.Sc., the M.Sc., and the Dr. Sci. degrees in mathematics from Osaka University, in 1988, 1990, and 1997 respectively. She joined Panasonic Co., LTD from 1990 to 1998 and engaged in research and development for secure communication. She was an associate professor at the Japan Advanced Institute of Science and Technology (JAIST) in 1998. She joined the computer science department of the University of California, Davis from 2002 to 2003. She has been a professor at Japan Advanced Institute of Science and Technology (JAIST) since 2007. She has been a professor at Graduate School of Engineering, Osaka University since 2015. Her research interests include the application of number theory into cryptography and information security. She received Young Paper Award of SCIS'93 in 1993, Notable Invention Award of the Science and Technology Agency in 1997, the IPSJ Sakai Special Researcher Award in 2002, the Standardization Contribution Award in 2003, the AWARD for the contribution to CULTURE of SECURITY in 2007, the Director-General of Industrial Science and Technology Policy and EnvironmentBureau Award in 2007, DoCoMo Mobile Science Awards in 2008, Advanced Data Mining and Applications (ADMA 2010) Best Paper Award, Prizes for Science and Technology, the Commendation for Science and Technology by the Minister of Education, Culture, Sports, Science and Technology, International Conference on Applications and Technologies in Information Security (ATIS 2016) Best Paper Award, the 16th IEEE Trustocm 2017 Best Paper Award, IEICE milestone certification in 2017, and the 14th Asia Joint Conference on Information Security (AsiaJCIS 2019) Best Paper Award. She is a member of the International Association for Cryptologic Research, the Institute of Electrical and Electronics Engineers, the Institute of Electronics, Information and Communication Engineers, the Information Processing Society of Japan, and the Mathematical Society of Japan.