Received September 25, 2020, accepted October 19, 2020, date of publication October 29, 2020,
date of current version November 11, 2020.

Digital Object Identifier 10.1109/ACCESS.2020.3034602

Privacy-Preserving Weighted Federated Learning Within the Secret Sharing Framework

HUAFEI ZHU[1], RICK SIOW MONG GOH[1], (Member, IEEE), AND WEE-KEONG NG[2]
[1]Institute for High Performance Computing, Agency for Science and Technology, Singapore 138632
[2]School of Computer Science and Engineering, Nanyang Technological University, Singapore 639798

Corresponding author: Huafei Zhu (zhu_huafei@ihpc.a-star.edu.sg)

ABSTRACT This paper studies privacy-preserving weighted federated learning within the secret sharing framework, where individual private data is split into random shares which are distributed among a set of pre-defined computing servers. The contribution of this paper mainly comprises the following four-fold:

- In the first fold, the relationship between federated learning (FL) and multi-party computation (MPC) as well as that of secure federated learning (SFL) and secure multi-party computation (SMPC) is investigated. We show that FL is a subset of MPC from the m-ary functionality point of view. Furthermore, if the underlying FL instance privately computes the defined m-ary functionality in the simulation-based framework, then the simulation-based FL solution is an instance of SMPC.
- In the second fold, a new notion which we call weighted federated learning (wFL) is introduced and formalized. Then an oracle-aided SMPC for computing wFL is presented and analysed by decoupling the security of FL from that of MPC. Our decoupling formulation of wFL benefits FL developers selecting their best security practices from the state-of-the-art security tools.
- In the third-fold, a concrete implementation of wFL leveraging the random splitting technique in the framework of the 3-party computation is presented and analysed. The security of our implementation is guaranteed by the security composition theorem within the secret share framework.
- In the fourth-fold, a complement to MASCOT is introduced and formalized in the framework of SPDZ, where a novel solution to the Beaver triple generator is constructed from the standard El Gamal encryption. Our solution is formalized as a three-party computation and a generation of the Beaver triple requires roughly 5 invocations of the El Gamal encryptions. We are able to show that the proposed implementation is secure against honest-but-curious adversary assuming that the underlying El Gamal encryption is semantically secure.

INDEX TERMS Beaver-triple, El Gamal encryption, privacy-preserving, secure multi-party computation, secret share, weighted federated learning.

I. INTRODUCTION

The concept of federated learning (FL) first introduced by McMahan *et al.* is a decoupling of model training from the need for direct access to the raw training data [1]. The definition of FL is in the evolution and a variation of FL definitions have been proposed (say, [2]–[4]). The datasets defined in the FL framework can be categorized as horizontal, vertical and hybrid types. Roughly speaking, in the horizontal FL, the feature spaces of datasets among different organizations (data owners) are same but not overlapped over the sample spaces [5]; in the vertical FL, the sample spaces of datasets among different organizations are same but not overlapped over the feature spaces [6], [7]; in the hybrid FL, both feature spaces and sample spaces of different organizations are overlapped [8], [9]. We refer to the reader [10]–[14] and the references therein for more details.

The associate editor coordinating the review of this manuscript and approving it for publication was Kim-Kwang Raymond Choo.

A. THE MOTIVATION PROBLEM

Going through the FederatedAveraging algorithm introduced in [1] that works over horizontal datasets, we know that each client k locally computes n_k data samples for the local model w^k_{t+1} at the $(t+1)$-round. The parameters n_k and w^k_{t+1} are then sent to the global FL server who in turn, computes the weighted average of the resulting model $w_{t+1} \leftarrow \sum_{k=1}^{K} \frac{n_k}{n} w^k_{t+1}$ where K is the number of clients and $n = n_1 + \cdots + n_K$.

1) WHAT ARE NOT ADDRESSED IN THIS PAPER

As discussed in [3], to incentivize individual data provider to participant an FL procedure, a reward will be allocated among the data providers based on the quality and quantity of their provided data. Hence n_k could be an incentive variant described in the FederatedAveraging algorithm. Generally, n_k can be evaluated by a trusted third party (say, a data quality and quantity evaluation service provider) or by the data owner himself/herself. Since n_k is a sensitive value, while n_k is verifiable,[1] it should be well protected. Since the research of incentive mechanism is a complex task, a comprehensive review of the incentive mechanism is out of the scope of this paper.

In this paper, we simply assume that n_k is evaluated by a trusted data quality and quantity service provider so that we can focus on our exploration of the relationship between FL and MPC as well as that of SFL and SMPC. Leveraging the explored results, we will introduce and formalize a new notion which we call weighted federated learning (wFL), and then provide implementations of wFL and prove the security of our implementations within the oracle-aided framework. Finally, we substitute the oracle that is aided to compute the wFL with a concrete Beaver triple generator in the framework of SPDZ [15]–[19].

2) WHAT ARE FOCUSED IN THIS PAPER

In this paper, a new notion which we call weighted federated learning (wFL) is introduced and formalized, where both n_k and w^k_{t+1} are encrypted. By $[n_k]$ (resp. $[w^k_{t+1}]$), we denote an encryption of n_k (resp. w^k_{t+1}). We remark that the selection of the underlying encryption scheme is flexible. It can be a secret sharing scheme based encryption or a partially (fully) homomorphic encryption that should be the suitable for the underlying use case. A wFL addresses the following problem: given K encrypted (weight, model) pairs $([n_k], [w^k_{t+1}])$ $(k = 1, \ldots, K)$, the global FL server wishes to compute an updated model $w_{t+1} \leftarrow \sum_{k=1}^{K} \frac{[n_k]}{n}[w^k_{t+1}]$, where K is the number of clients and $n = [n_1] + \cdots + [n_K]$. As stated in the footnote 1, the encrypted n_k will be served as a witness for the global FL server allocating the specified reward (for simplicity, we assume that the global FL server equally allocates the current round reward to the qualified data providers).

Since the theory of SMPC is well developed in cryptography [22]–[24], the understanding of the relationship between FL and MPC and that of secure FL and SMPC could largely benefit us to understand how SMPC mechanism can be applied to FL and SFL. To the best of our knowledge, this problem has not been addressed and thus leaves us the following interesting research problem:

Question 1: what is the relationship between FL and MPC as well as that of SFL and SMPC?

Leveraging the explored result (please refer to Section 2 for more details), we know that the state-of-the-art SMPC techniques such as the ring-based zero-splitting [29]–[31] and the SPDZ [15]–[19] can be applied to preserve the wFL data privacy. However, the evolution of security and privacy tools for securing FL systems increases the difficulty for researchers to evaluate the security of the underlying solutions. For example, in Sharemind, the multiplication operator based on the Du and Atallah's method [28] was replaced by the zero-splitting mechanism. Is the privacy of Sharemind working in the Du and Atallah model preserved in the zero-splitting mechanism? This leaves the following interesting research problem:

Question 2: how to decouple the security of a federated learning system from the underlying security and privacy tools that could largely benefit FL engineers selecting the best security tools for their practices?

The attractive feature of a ring-based zero-splitting based SMPC solution [29]–[31] is its efficiency. The scalability however, is problematic since it is designed for 3-party computation from the scratch. Note that the meaning of the ring-based zero-splitting and the secret sharing of private data, where individual private data is split into random shares which are distributed among a set of pre-defined computing servers, are same. As such as we do not distinguish the two notions and use them in interchange throughout the paper. Also please note that the secret sharing scheme is different from that of the Shamir threshold system [27], where the notion of security sharing emphasizes the random data splitting procedure while the threshold secret sharing system emphasizes on the recoverability of the shared data. There are many ways to resolve the scalability of the zero-splitting based MPC, for example:

- if the committee selection technique presented in [25], [26] is applied, then the committee selection policy should be integrated with Sharemind;
- if the Shamir threshold system [27] is applied to achieve the scalability, then the strategy for decreasing the degree of the resulting multiplicative polynomials should be integrated with Sharemind;
- if the SPDZ solution is applied, then the strategy for efficiently generating Beaver-triple is certainly welcome.

While Sharemind approach is efficient, the scalability is problematic. On the other hand, while SPDZ provides the high scalability, the efficiency of the cryptographic solution to generate Beaver triple is a challenging task. Each of the zero-splitting or SPDZ based solution has its own pros and cons. In this paper, we will follow the SPDZ solution to attain the security of wFL.

To the best of our knowledge, the most efficient solution working in the SPDZ framework is MASCOT where the notion of oblivious transfers extension (OT-extension) has been applied successfully [32]. MASCOT by its nature needs to invoke a number of field size related oblivious transfers (OTs). Namely, the number of invoked OTs is the size of the underlying finite field F where a secret sharing of

[1] For example, the global FL server could be convinced that $[n_k] \geq \tau$ by executing a zero-knowledge proof procedure [20], [21], where $[n_k]$ is an encryption of n_k, and τ is the pre-defined threshold by the global FL server.

multiplication $xy = a+b \in F$ is performed, $x \in F$ is a private input of Alice and $y \in F$ is a private input Bob; the output of Alice is $a \in F$ and the output of Bob is $b \in F$. To attain an acceptable security, MASCOT assumes that the size of field is 128-bit and hence 128 OTs will be invoked. We emphasize that the number of OTs used to generate the Beaver triple is fixed which equals to the size of underlying field even in case that only one Beaver triple is generated. Assuming that the computation complexity of each OT evocation is roughly same as that of an EL Gamal encryption,[2] and also assuming that there exists a solution such that λ invocations of El Gamal encryptions are used to generate a Beaver triple, then the proposed solution should be more efficient compared with MASCOT if less than $128/\lambda$ multiplications are required in an application (a Beaver-triple is used to assist a multiplication in the SPDZ framework and we expect more multiplications can be computed beyond the threshold $128/\lambda$, see Section 5 for more details). The proposed solution should be useful when the number of multiplications is small. Since the existence of such a solution is not addressed by MASCOT (in essence, it is a complement to MASCOT), we thus provide an interesting research problem below:

Question 3: how to construct an efficiet yet secure Beaver triple generator which invokes a constant number of the El Gamal encryptions only?

B. OUR CONTRIBUTION

Regarding the first question, we are able to show that:

- FL is a subset of MPC from the *m*-ary functionality point of view;
- If FL attains the security in the simulation-based framework, then the resulting SFL is a subset of SMPC. Since there are other known techniques such as the homomorphic encryption (HE) and the differential privacy (DP) to attain the privacy of federated learning procedures [6], [9] and we are not clear whether SFL is a subset of HE or DP within the correspondent framework, we thus leave these interesting problems to the research community.

Regarding the second question, our contribution is three-fold:

- in the first fold, a new notion which we call weighted federated learning (wFL) is introduced and formalized. The wFL concept formalized in this paper differs from FL introduced in the McMahan *et al.*'s pioneer work [1] *— both addition and multiplication operations are executed over the cipher space in the wFL model while these operations are executed over the plaintext space in the FL model*.
- in the second fold, an oracle-aided MPC solution for computing wFL that is formalized by decoupling the security of FL from that of underlying MPC. Our decoupling formulation may benefit machine learning developers selecting their best security practices from the state-of-the-art security tool sets;
- in the third fold, a concrete solution to the wFL is presented and analysed. The security of our implementation is guaranteed by the security composition theorem assuming that the underlying multiplication protocol is secure against honest-but-curious adversaries.

Regarding the third question, our contribution is two-fold:

- in the first fold, an efficient Beaver triple generator is proposed and analysed in the context of the El Gamal encryption scheme defined over Z_p^*. We follow the functionality of Beaver triple formalized as a three-party computation that is presented in [32]. At a high level, we allow Charlie generates a random value c which is encrypted under a combination of Alice and Charlie's public keys; The cipher is sent to Bob who in turn computes a randomized encryption of c/b. The cipher is then sent to Charlie who partially decrypts the received cipher and then randomizes the resulting cipher; The randomized cipher is then sent to Alice so that Alice can decrypt the randomized cipher to get value a which is c/b. As such, we are able to construct an efficient Beaver multiplication triple generator from the standard El Gamal encryption which invokes the constant number of encryptions (we refer to the reader Section 4 for more details).
- in the second fold, we are able to show that our solution to wFL is light-weight since only the standard El Gamal encryption is involved in our construction. We claim that our solution to wFL is highly scalable since our solution is based on the SPDZ framework which is highly scalable by the design; we also claim that our solution to wFL is efficiency since a generation of the Beaver triple only requires a constant number of invocations of the El Gamal encryption (roughly, 5 El Gamal encryptions are invoked to generated a Beaver triple). Hence the solution to wFL presented in this paper is not trivial at all.

C. THE ROAD-MAP

The rest of this paper is organized as follows: In section 2, the relationship between MPC and FL, and that of SMPC and SFL are investigated; The notion of wFL is formalized, implemented and analysed in Section 3. A complement of MASCOT is implemented and analysed in Section 4. We test the efficiency of our implementation in Section 5 and conclude our work in Section 6.

II. THE RELATIONSHIP BETWEEN MPC AND FL, AND SMPC AND SFL

In this section, we will discuss the relationship between MPC and FL as well as that of SMPC and SFL.

A. m-ARY FUNCTIONALITY, MPC AND SMPC

Recently, a new approach called SPDZ has been proposed [15]–[19]. The scalability of SPDZ framework benefits

[2]The assumption is based on the following the justification: if an OT is implemented by the Diffie-Hellman key exchange protocol [33], the computation of an El Gamal encryption [34] is roughly same as that of an OT invocation which is defined over the same field Z_p^*.

SPDZ are widely deployed in the FL community [3]. It is thus helpful to investigate the relationship between FL and MPC as well as SFL and SMPC. To convenient readers to understand the results, we briefly describe the notations and notions of m-ary functionality, MPC and SMPC below and refer to the reader [22]–[24] for more details.

1) m-ARY FUNCTIONALITY

An m-ary functionality, denoted by $f\colon (\{0,1\}^*)^m \rightarrow (\{0,1\}^*)^m$, is a random process mapping string sequences of the form $\overline{x} = (x_1, \cdots, x_m)$ into sequences of random variables, $f_1(\overline{x}), \cdots, f_m(\overline{x})$ such that, for every i, the ith party P_i who initially holds an input x_i, wishes to obtain the ith element in $f(x_1, \ldots, x_m)$ which is denoted by $f_i(x_1, \ldots, x_m)$.

2) MULTI-PARTY COMPUTATION

A multi-party computation (MPC) problem is casted by specifying an implementation of the defined m-ary functionality. Namely, an MPC protocol is a procedure computing the defined m-ary functionality. We emphasize that the notion of MPC does NOT guarantee the proposed MPC protocol securely computing the defined m-ary functionality throughout this paper. It is possible where no security is introduced in the MPC protocol at all.

3) SECURE MULTI-PARTY COMPUTATION

A multi-party computation securely computes an m-ary functionality (i.e., secure multi-party computation, SMPC) if the following simulation-based security definition is satisfied.

Let $[m] = \{1, \cdots, m\}$. For $I \in \{i_1, \ldots, i_t\} \subseteq [m]$, we let $f_I(x_1, \ldots, x_m)$ denote the subsequence $f_{i_1}(x_1, \ldots, x_m), \cdots, f_{i_t}(x_1, \ldots, x_m)$. Let Π be an m-party protocol for computing f. The view of the i-th party during an execution of Π on $\overline{x}$:= $(x_1, \ldots, x_m)$ is denoted by $\mathrm{View}_{\mathrm{i}}^{\Pi}(\overline{x})$. For $I = \{i_1, \ldots, i_t\}$, we let $\mathrm{View}_{\mathrm{I}}^{\Pi}(\overline{x})$:= $(I, \mathrm{View}_{\mathrm{i_1}}^{\Pi}(\overline{x}), \cdots, \mathrm{View}_{\mathrm{i_t}}^{\Pi}(\overline{x}))$.

- In case f is a deterministic m-ary functionality, we say Π privately computes f if there exists a probabilistic polynomial-time algorithm denoted S, such that for every $I \subseteq [m]$, it holds that $S(I, (x_{i_1}, \ldots, x_{i_t}), f_I(\overline{x}))$ is computationally indistinguishable with $\mathrm{View}_{\mathrm{I}}^{\Pi}(\overline{x})$.
- In general case, $S(I, (x_{i_1}, \ldots, x_{i_t}), f_I(\overline{x}), f(\overline{x}))$ is computationally indistinguishable with $\mathrm{View}_{\mathrm{I}}^{\Pi}$ $((\overline{x}), f(\overline{x}))$.

4) ORACLE-AIDED MULTI-PARTY COMPUTATION

An oracle-aided protocol is a protocol augmented by a pair of oracle types, per each party. An oracle-call step is defined as follows: a party writes an oracle request on its own oracle tape and then sends it to the other parties; in response, each of the other parties writes its query on its own oracle tape and responds to the first party with an oracle call message; at this point the oracle is invoked and the oracle answer is written by the oracle on the ready-only oracle tape of each party.

An oracle-aided protocol is said to privately reduce g to f if it securely computes g when using the oracle-functionality f. In such a case, we say that g is securely reducible to f.

TABLE 1. Federated learning procedure.

Federated learning procedure [3]
1) Client selection: The server specifies $(m-1)$-client who meet the eligibility requirements;
2) Broadcast: The selected clients download the current model weights and a training program from the server. The model weights and training program are part of system parameters;
3) Client computation: Each selected client locally computes an update to the model by executing the training program;
4) Aggregation: The server collects an aggregate of the clients' updates. If no privacy requirement is introduced, FL will be called plain-FL (or FL for short, in this short note).
5) Model update: The server locally updates the shared model based on the aggregated update computed from the clients that participated in the current round.

B. FEDERATED LEARNING PROCESS

The notion of FL presented in this section is due to [3]. In their definition, a federated learning server (FLS) orchestrates the training process, by repeating the following steps until training is stopped (see TABLE 1 for more details).

C. THE RELATIONSHIP

Theorem 1: 1) FL is a subset of MPC from the m-ary functionality point of view; and 2) if FL attains the security in the simulation-based framework, then SFL is a subset of SMPC.

Proof: For each iteration j defined in the FL procedure, we are able to define an m-ary functionality f_j for the current round. We know that the FL functionality f_{FL} can be defined as a composition of the round functionalities $f_n \circ f_{n-1} \circ \cdots \circ f_1$, where f_j is the m-ary functionality for the iteration j. As such, for each iteration j, if there exists an m-party protocol privately computing f_j, then by applying the SMPC composition theorem [22]–[24], we are able to show there is an m-party protocol privately computing f_{FL}.

The rest of the proof is to show that for each iteration, an m-ary functionality is defined accordingly. The details of the m-ary functionality are depicted in TABLE 2. One can verify that TABLE 2 defines the m-ary functionality derived from the FL procedure. Following the definitions of MPC and SMPC, the Theorem 1 is proved.

III. THE WEIGHTED FEDERATED LEARNING: SYNTAX AND SECURITY DEFINITION

In this section, we will provide a formal definition for weighted federated learning and then define the security of wFL within the oracle-aided multi-party computation framework.

A. SYNTAX OF WEIGHTED FEDERATED LEARNING

Definition 1: A weighted federated learning (wFL) consists of a group of clients $(c_1, \ldots, c_m)$, a global federated learning server and a group of MPC servers $P_1 \cdots, P_n$. Each

TABLE 2. An m-ary functionality derived from an iteration of the FL procedure.

For each iteration in FL, we define an m-ary functionality:

1) Define m-1 parties selected by the FL-server (FLS) as MPC participants. Including FLS itself, there are m parties;
2) Define the model weights and training program as system parameter ($sysparam$) to MPC;
3) Define initial $View_i$ as input D_i, randomness r_i and $sysparam$ for P_i. $View_i$ is append-only data type.
4) Define f_m: $View_{\text{FLS}} \rightarrow (sout_1, \cdots, sout_m)$, where $sout_i$ denotes server's output such that P_i gets its output $sout_i$ $(i = 1, \cdots, m-1)$. The output of FLS is $sout_m$.
5) Define m-ary $f = (f_1, \cdots, f_m)$, where f_i: $(view_1 \cdots view_m) \rightarrow sout_i$.

client c_i holds a pair of weight and feature $(x_i, y_i) \in Z_p^* \times Z_p^*$ (p is a prime number) which is additively shared among the MPC servers where P_j holds $(x_{i,j}, y_{i,j})$ such that $x_i = x_{i,1} + \cdots + x_{i,n}$ and $y_i = y_{i,1} + \cdots + y_{i,n}$. By $[x_i]$ (resp. $[y_i]$), we denote the random secret shares $(x_{i,1}, \ldots, x_{i,n})$ (resp. $(y_{i,1}, \ldots, y_{i,n})$) of $[x_i]$ (resp. $[y_i]$) among P_j $(j = 1, \ldots, n)$. The global federated learning server defines a random processing whose input is $([x_1], [y_1]), \cdots, ([x_n], [y_n])$ and output is $\sum_{k=1}^{K} [x_i] \times [y_i]$.

We remark that the separation between the clients and MPC servers is necessary since there could be such a case: for example, the first client is lack of ability to process the MPC task and hence its computation task will be outsourced to MPC servers.

B. SECURITY DEFINITION OF WEIGHTED FEDERATED LEARNING

The security of wFL protocol is formalized in the context of an oracle-aided SMPC which in essence, is a decoupling of machine learning algorithm from the need for MPC that may benefit machine learning developers to select their best security practices from the state-of-the-art security tool sets.

Definition 2: An multiplication-oracle aided *wFL* is privacy-preserving if *wFL* is privately reducible to the multiplication functionality.

Notice that the addition oracle is not needed to attain the defined privacy-preserving reduction since the underlying data sharing scheme is an additively secret sharing.

C. THE IMPLEMENTATION AND SECURITY PROOF

In this section, a concrete solution to the wFL based on the additive data sharing with the help of the zero-splitting technique defined over the three-server setting is presented and analysed. The security of our implementation is derived from the security composition theorem assuming that the underlying multiplication algorithm is secure against honest-but-curious adversaries.

D. THE IMPLEMENTATION

Following the state-of-the-art Sharemind framework [29], [30], we define following steps for our implementation: data splitting, resharing, addition and multiplication. Each of the steps is depicted in details below:

1) THE DATA SPLITTING

Suppose a wFL client Alice holds private data x and y locally. W.l.o.g., we assume that there are three MPC servers managed and maintained by independent computing service providers (say, FL auditor (P_1 plays the role of MPC server 1), FL insurance company P_2 plays the role of MPC server 2) and FL client association (P_3 plays the role of MPC server 3)). We assume that there is a secure (private and authenticated) channel between client Alice and each of MPC service providers. This assumption is standard and can be easily implemented under the standard PKI assumption. For simplicity, we assume that $x, y \in Z_p^*$, where p is a suitable large prime number (e.g., $|p| = 512$). The splitting procedure is defined below

- Alice selects $x_1, x_2 \in Z_p^*$ uniformly at random, and then sends x_1 to P_1, x_2 to P_2;
- Alice computes $x_3 = x - x_2 - x_1$ mod p and sends x_3 to P_3.

The splitting of the data x is defined by $[x] = (x_1, x_2, x_3)$ (as usual, a random split of data is called a secret share of that data). Similarly, a secret share of y is defined by $[y] = (y_1, y_2, y_3)$, where P_i holds y_i (i = 1,2,3).

2) THE RESHARING

A refreshing procedure is invoked before a multiplication operation is executed. The refreshing procedure is defined among P_1 (with input x_1), P_2 (with input x_2) and P_3 (with input x_3):

- P_1 selects $r_1 \in Z_p^*$ uniformly at random and sends r_1 to P_2 via a pre-defined secure channel;
- Similarly, P_2 (resp. P_3) selects $r_2 \in Z_p^*$ (resp. $r_3 \in_U Z_p^*$) uniformly at random and sends r_2 (resp. r_3) to P_3 (resp. P_1) via a pre-defined secure channel;
- P_1 locally computes $\sigma_1 = r_1$ - r_3 mod p and $x_1' = x_1 + \sigma_1$ mod p; P_2 locally computes $\sigma_2 = r_2$ - r_1 mod p and $x_2' = x_2 + \sigma_2$ mod p; P_3 locally computes $\sigma_3 = r_3$ - r_2 mod p and $x_3' = x_3 + \sigma_3$ mod p.

A refresh of $[x]$ is denoted by $[x]' = (x_1', x_2', x_3')$ such that $x_1' + x_2' + x_3'$ mod $p = x_1 + x_2 + x_3$ mod p.

3) THE ADDITION

Suppose P_i holds shares of x_i and y_i. P_i locally computes $z_i = x_i + y_i$ mod p and then sends z_i to the wFL global server who computes $z_1+z_2+z_3$ mod p and thus gets the value of addition $x + y$ mod p.

4) THE MULTIPLICATION

On input (x_i, y_i), each of participants P_i can jointly run the resharing protocol to get (x_i', y_i') $(i = 1, 2, 3)$. The role of resharing protocol plays a one-time padding of shares. P_i then sends its shares (x_i', y_i') to $P_{i \bmod 3+1}$. Finally, P_1 computes $z_1 = (x_1'y_1' + x_1'y_3' + x_3'y_1')$ mod p; P_2 computes

$z_2 = (x'_2y'_2 + x'_2y'_1 + x'_1y'_2) \bmod p$ and P_3 computes $z_3 = x'_3y'_3 + x'_3y'_2 + x'_2y'_3 \bmod p$. One can verify that $z_1 + z_2 + z_3 \bmod p = [x][y] \bmod p$.

5) PUTTING THINGS TOGETHER

Notice that $(n_{k,1}, w_{k,1})$ is a secret share held by P_1, $(n_{k,2}, w_{k,2})$ is a share held by P_2 and P_3 holds $(n_{k,3}, w_{k,3})$ for $k = 1, \ldots, K$ for the secret shares of the weight and feature $[n] = ([n_1], \ldots, [n_K])$ and $[w] = ([w_1], \ldots, [w_K])$, where $[n_k] = (n_{k,1}, n_{k,2}, n_{k,3})$ and $[w_k] = (w_{k,1}, w_{k,2}, w_{k,3})$. Applying the addition and multiplication operations described above, we are able to solve the wFL problem.

E. THE PROOF OF SECURITY

Theorem 2: Let g_{wFL} be a weighted Federated Learning functionality defined in the three-server framework. Let $\Pi^{g_{wFL}|f_{mult}}$ be an oracle-aided protocol that privately reduces g_{wFL} to f_{mult} and $\Pi^{f_{mult}}$ be a protocol privately computes f_{mult}. Suppose g_{wFL} is privately reducible to f_{mult} and that there exists a protocol for privately computing f_{mult}, then there exists a protocol for privately computing g_{wFL}.

Proof: We construct a protocol Π for computing g_{wFL}. That is, we replace each invocation of the oracle f_{mult} by an execution of protocol $\Pi^{f_{mult}}$. Note that in the semi-honest model, the steps executed $\Pi^{g_{wFL}|f_{mult}}$ inside Π are independent the actual execution of $\Pi^{f_{mult}}$ and depend only on the output of $\Pi^{f_{mult}}$.

For each $i = 1, 2, 3$, let $S_i^{g_{wFL}|f_{mult}}$ and $S_i^{f_{mult}}$ be the corresponding simulators for the view of party P_i. We construct a simulator S_i for the view of party P_i in Π. That is, we first run $S_i^{g_{wFL}|f_{mult}}$ and obtain the simulated view of party P_i in $\Pi^{g_{wFL}|f_{mult}}$. This simulated view includes queries made by P_i and the corresponding answers from the oracle. Invoking $S_i^{f_{mult}}$ on each of partial query-answer (q_i, a_i), we fill in the view of party P_i for each of these interaction of $S_i^{f_{mult}}$. The rest of the proof is to show that S_i indeed generates a distribution that is indistinguishable from the view of P_i in an actual execution of Π.

Let H_i be a hybrid distribution represents the view of P_i in an execution of $\Pi^{g_{wFL}|f_{mult}}$ that is augmented by the corresponding invocation of $S_i^{f_{mult}}$. That is, for each query-answer pair (q_i, a_i), we augment its view with $S_i^{f_{mult}}$. It follows that H_i represents the execution of protocol Π with the exception that $\Pi^{f_{mult}}$ is replaced by simulated transcripts. We will show that

- the distribution between H_i and Π are computationally indistinguishable: notice that the distributions of H_i and Π differ $\Pi^{f_{mult}}$ and $S_i^{f_{mult}}$ which is computationally indistinguishable assuming that $\Pi^{f_{mult}}$ securely computes f_{mult}.
- the distribution between H_i and S_i are computationally indistinguishable: notice that the distributions between $(\Pi^{g_{wFL}|f_{mult}}, S_i^{f_{mult}})$ is computationally indistinguishable from $(S_i^{g_{wFL}|f_{mult}}, S_i^{f_{mult}})$. The distribution $(S_i^{g_{wFL}|f_{mult}}, S_i^{f_{mult}})$ defines S_i. That means H_i and S_i are computationally indistinguishable.

TABLE 3. Beaver triple functionality.

The functionality of Beaver Triple $\mathcal{F}_{\text{Triple}}$
• The functionality maintains a dictionary, Val to keep track of assigned value, where entry of Val lies in a fixed field.   • On input (Triple, id_A,id_B, id_C) from all parties, sample two random values a, $b \leftarrow F$, and set [Val[id_A], Val[id_B], Val[id_C]] $\leftarrow (a, b, ab)$.

Corollary1: Assuming that the underlying multiplication algorithm presented in [29], [30] is secure against honest-but-curious adversary, our implementation is secure against the same adversarial type.

IV. A COMPLEMENT TO MASCOT

MASCOT provides an efficient solution to generate a large number (say, millions) of Beaver triples [32] in the framework of SPDZ [15], [16]. To deploy the MASCOT, a base oblivious transfer (OT) is invoked a number of times (say, 128-invocation called) and then an OT-extension procedure is applied to generate one-time symmetric encryption keys to mask the correlated input pairs. The number of base OTs invoked depend on the bit-length of the underlying field. With the help of the footnote 2, we assume that the computation complexity of an invocation of base OT is roughly same as that of an EL Gamal encryption. Since our solution roughly requires 5 ElGamal encryptions to be invoked for generating a Beaver triple, it follows that if the number of the Beaver triples to be generated are no more than $\lfloor 128/5 \rfloor$ $(= 25)$, then our solution is more efficient compared with the best known solution MASCOT [32]. That is, our solution can be viewed as a complement to MASCOT where a small number of multiplications are required.

A. BEAVER MULTIPLICATION TRIPLE GENERATOR BASED ON EL GAMAL ENCRYPTION

In this section, an efficient, light-weight, highly scalable secret-share based MPC within the SPDZ framework is presented and analysed.

1) BEAVER TRIPLE FUNCTIONALITY

We write $[x]$ to mean that each party P_i holds a random, additive sharing x_i of x such that $x = x_1 + \cdots + x_n$, where $i = 1, \ldots, n$. The values are stored in the dictionary Val defined in the functionality $\mathcal{F}_{Triple}$ [32]. Please refer to the table 3 for more details.

2) THE DESCRIPTION

Let p be a large safe prime number, $p = 2q + 1$, p and q are prime numbers. Let $G \subseteq Z_p^*$ be a cyclic group of order q and g be a generator of G. Let $h_i = g^{x_i} \bmod p$, where $x_i \in_R [1, q]$ is randomly generated by the Beaver multiplication triple generator G_i $(i = 1, 2, 3)$. The idea behind of our construction is that we allow G_1 to generate c, G_2 to generate b and G_3 to generate a such that $c = ab \bmod p$. Please refer to the table 4 for the details.

TABLE 4. An implementation of Beaver triple generator.

Beaver-triple generator based on El Gamal encryption
• G_1 selects $c \in Z_p^*$ uniformly at random and sends $c_1 := (u_1, v_1)$ to G_2, where $h = h_1 h_3$, $u_1 = g^{r_1}$ and $v_1 = c \times h^{r_1}$;
• Upon receiving c_1, G_2 selects $b \in Z_p^*$ uniformly at random and then computes $u_2 = u_1 g^{r_2}$, $v_2 = \frac{v_1}{b} h^{r_2}$ and then sends $c_2 := (u_2, v_2)$ to G_1;
• Upon receiving c_2 from G_2, G_1 partially decrypts (u_2, v_2) using its private key x_1 to get $c_3' := (u_3', v_3')$, where $u_3' = u_2$ and $v_3' = \frac{v_2}{u_2^{x_1}}$; G_1 selects $r_3 \in Z_p^*$ uniformly at random, computes $u_3 = u_3' g^{r_3}$ and $v_3 = v_3' h_3^{r_3}$ and then sends $c_3 := (u_3, v_3)$ to G_3;
• Upon receiving c_3 from G_1, G_3 decrypts (u_3, v_3) to get $a := c/b$.

Theorem 3: Assuming that the underlying El Gamal encryption is semantically secure, the proposed Beaver triple generator is secure against honest-but-curious adversary.

Proof: We consider the following three cases:

- Case 1: G_1 gets corrupted. Simulator S_1 receives secret parameter k and G_1's output c and then:
  1.1) S_1 chooses a randomness r_1 for G_1, and runs $G_1(r_1, 1^k)$ to generate $(x_1, (g, h_1))$ such that $h_1 = g^{x_1}$;
  1.2) S_1 invokes G_2 and G_3 to generate $(x_2, (g, h_2))$ and $(x_3, (g, h_3))$ such that $h_2 = g^{x_2}$ and $h_3 = g^{x_3}$. S_1 gets (g, h_2) and (g, h_3);
  1.3) S_1 computes c_1 exactly as that described in the real protocol and then sends it to G_2;
  1.4) S_1 generates an encryption c_2 of a random string in Z_p^* with public keys $(g, h = h_1\ h_3)$;
  1.5) Upon receiving c_2 from G_2, S_1 computes c_3 exactly as that described in the real protocol and then sends it to G_3.
  Notice that the only difference between the simulation and the real protocol is the Step 1.4. Namely, c_2 is an encryption of a random string in Z_p^* in the simulation while c_2 is an encryption of c/b in the real protocol. Since the underlying El Gamal encryption scheme is semantically secure, it follows that the view of the corrupted party G_1 is computationally indistinguishable from that the simulation described above.
- Case 2: G_2 gets corrupted. Simulator S_2 receives secret parameter k and G_2's output c and then:
  2.1) S_2 chooses a randomness r_2 for G_2, and runs $G_2(r_1, 1^k)$ to generate $(x_2(g, h_2))$ such that $h_2 = g^{x_2}$;
  2.2) S_2 invokes G_1 and G_3 to generate $(x_1, (g, h_1))$ and $(x_3, (g, h_3))$ such that $h_1 = g^{x_1}$ and $h_3 = g^{x_3}$. S_2 gets (g, h_1) and (g, h_3);
  2.3) S_2 generates a random encryption c_1 with public keys $(g, h = h_1\ h_3)$; and then generates c_2 exactly as that described in the real protocol.
  The simulation of the rest steps can be skipped since the adversary is NOT involved in the rest of the procedures. Notice that the only difference between the simulation described above and the real protocol is the generation of c_1 where c_1 is an encryption of a random string in Z_p^* defined by simulator. Since the underlying El Gamal encryption scheme is semantically secure, it follows that the view of the corrupted party G_2 is computationally indistinguishable from that the simulation described above.
- Case 3: G_3 gets corrupted. Simulator S_3 receives secret parameter k and G_3's output c and then
  3.1) S_3 chooses a randomness r_3 for G_3, and runs $G_3(r_1, 1^k)$ to generate $(x_3(g, h_3))$ such that $h_3 = g^{x_3}$;
  3.2) S_3 invokes G_1 and G_2 to generate $(x_1, (g, h_1))$ and $(x_2, (g, h_2))$ such that $h_1 = g^{x_1}$ and $h_2 = g^{x_3}$. S_3 gets (g, h_1) and (g, h_2);
  3.3) S_3 generates an encryption c_3 of $a \in Z_p^*$ with the help of the public keys (g, h_3);
  The simulation starts at the generation of c_3 step defined in the real protocol. Since G_3 gets output a, the simulator can simply define c_3 as an encryption of a. The adversary does not involved in the steps before c_3 is sent out. It follows that the view of the corrupted party G_3 is identical with that the simulation described above.

Combining the cases above, we know that the protocol is secure against honest-but-curious adversary assuming that the underlying El Gamal encryption scheme is semantically secure.

3) A LIGHT-WEIGHT AND HIGHLY SCALABLE SOLUTION FOR wFL

Throughout this section, we assume that there are three Beaver Triple generators G_1, G_2 and G_3 and n-MPC participants $P_1, \ldots, P_n$, where G_1 (resp., G_2 and G_3) holds a private value c (resp., b and a) such that $c = ab \bmod p$. In this solution, we make an explicit assumption that $G_i \notin \{P_1, \ldots, P_n\}$ and there is a secure (private and authenticated) channel between G_i and P_j which in turn, can be implemented under the standard PKI assumption.

Let G_1 be the first Beaver triple generator (say Charlie as above) with private input $c \in Z_p^*$. The splitting procedure is defined below

- G_1 selects $c_2, \ldots, c_n \in Z_p^*$ uniformly at random, and then sends c_2 to $P_2, \cdots, c_n$ to P_n via private channels shared between G_1 and $P_j, j = 1, \ldots, n$;
- G_1 computes $c_1 = c$ - c_2 - $\cdots$ - c_n mod p and sends c_1 to P_1.

A splitting of the private value c is defined by $[c] = (c_1, \ldots, c_n)$ (as usual, a random split of data is called a secret share of that data. The process can be viwed as a keyless encryption). Based on the data splitting procedure, one can define [b],[a] accordingly, where G_2 (resp., G_1) secretly shares b (resp., a) to all participants. We assume that as long as the secret sharing procedure ends, G_i erases his/her the generated randomness. For example, Charlie(G_1) will delete all internal randomness $(c, c_1, \ldots, c_n)$ as well as the generated index. The same erasure procedure applies to Bob (G_2) and Alice (G_3). Notice that this erasure assumption can be eliminated if an additively homomorphic encryption is applied to implement the secret sharing procedure in the context of Protocol reshare defined in [15].

The addition: Suppose P_i holds secret shares x_i and y_i of the secret values x and y. P_i locally computes $z_i = x_i + y_i \bmod p$ and then sends z_i to the wFL global server who computes $z_1 + \cdots + z_n \bmod p$ by invoking a resharing procedure defined below and then gets the value of addition $x + y \bmod p$.

The resharing: Usually, a refreshing procedure is called before an additive output is opened. The refreshing procedure is defined among $P_1, \ldots, P_n$. By $x_i \in Z_p$, we denote P_i's private input:

- P_1 selects $r_1 \in Z_p^*$ uniformly at random and sends r_1 to P_2 via a pre-defined secure channel;
- Similarly, P_2 selects $r_2 \in Z_p^*$ uniformly at random and sends r_2 to P_3 via a pre-defined secure channel;
- The similar procedure goes on and finally P_n selects $r_n \in Z_p^*$ uniformly at random and sends r_n to P_1 via a pre-defined secure channel.
- each party P_i locally computes $\sigma_1 = r_1$ - r_n and $\sigma_i = r_i$ - r_{i-1} for $i = 2, \ldots, n$ and then computes $x_i' = x_i + \sigma_i \bmod p$ for $i = 1, \ldots, n$.

A resharing of $[x]$ is denoted by $[x]' = (x_1', \ldots, x_n')$. One can verify that $x_1' + \cdots + x_n' \bmod p = x_1 + \cdots + x_n \bmod p$.

The multiplication: Assuming that P_i holds the private share x_i and y_i and n parties $P_1, \ldots, P_n$ wish to compute $[xy]$ collaboratively given $[x]$ and $[y]$. Borrowing the notation from SPDZ, by ρ, we denote an opening of $[x] - [a]$ and by ϵ, an opening of $[y]-[b]$. Given ρ and ϵ, each party can compute his secret share of $[xy] = (\rho + [a])(\epsilon + [b]) = [c] + \epsilon[a] + \rho[b] + \rho\epsilon$ locally.

V. EXPERIMENTS

In this section, we provide experiment results of SFL within the secret share framework. The Python 3.8.1 works in the Window 10 with processors: Intel(R) Core(TM) i7-8665U CPU 1.90GHz 2.11GHz; installed memory (RAM) 16.0GB (15.8 usable); and system type: 64-bit operating system, x64-based processor. The python crypto library called pyhf built by the first author that is used to test the efficiency of the design comprises the following four modules: data splitting, multiplication, relationship between the number of MPC servers and time costs on multiplications and Beaver triple generator constructed from 3-party computation based on the El Gamal cryptosystem. The details of each module are dipcted below.

A. THE EFFICIENCY OF DATA SPLITTING

Let $p = 2q + 1$ be a 512-bit safe prime number (p is fixed in the following experiments). We test 1 million and 10 millions data which are randomly selected in Z_p and split into three parties, where each party plays the role of an MPC server.

data size	1 million	10 million
time	4.9s	51s

We also test the efficiency of 1 million data split among different numbers of parties (MPC servers).

MPC-server size	3	4	10	20
time	4.8s	6.23s	16.8s	34.3s

That is, the efficiency of data splitting is related with the number of the parties (MPC servers).

B. THE EFFICIENCY OF MULTIPLICATION OVER Z_p

We test 1 million of multiplications $a \times b$ mod p along within Sharemind and SPDZ framework, where a and b are randomly selected in Z_p.

framework	multiplication size	time
Z_p^*	1 million	6.4s
Sharemind	1 million	49s
SPDZ	1 million	68s

where Beaver triples are generated by a trusted third party and distributed among 3-MPC servers.

C. RELATIONSHIP BETWEEN THE NUMBER OF MPC SERVERS AND TIME COSTS ON MULTIPLICATIONS

We test the relationship between number of MPC servers and the time costs on multiplications in the framework of SPDZ, where Beaver triples are generated by a trusted third party and distributed among k-MPC servers, $k = 3, 6, 12$.

number of MPC servers	time
3	68s
6	127s
12	236s

D. THE EFFICIENCY OF BEAVER TRIPLE GENERATOR

In the previous experiments, we assume that Beaver triples are generated by a trusted third party and distributed among 3-MPC servers. In this experiment, we eliminate this assumption and test the efficiency of our El Gmaml based solution.

number of Beaver triples	time
10,000	27s
100,000	276s
1 million	2667s

Our previous theoretical results show that if the number of multiplications are less than 25, then our solution is better than MASCOT. The expriment results above also show that our El Gamal based solution may be suitable for a small number of multiplications up to 10,000. Although the Beaver triple is generated off-line in the SPDZ framework, the time costs for 1 milliom multiplications is nearly 4.5 hours that should not be acceptable. As such, there is a big room for efficieny improvement to compete against MASCOT. At the current stage, we can only claim that our result should be viwed as a complement to MASCOT.

E. SUMMARY OF OUR SIMPLE EXPERIMENT

Our experiment shows that the time costs on multiplications depend on the number of MPC servers and the protocol used to define Beaver triple generator. Our El Gamal encryption based Beaver triple generator is suitable for the scenario where a small number of multiplications are required. As such, we successfully provide a complement to MASCOT.

VI. CONCLUSION

In this paper, the relationship between MPC and FL as well as that of SMPC and SFL has been investigated. A new notion which we call weighted federated learning problem has been then introduced and formalized in the context of MPC. The security of wFL is defined within the oracle-aided MPC framework and we are able to show that our implementation is secure against honest-but-curious adversary. Finally, we have proposed an efficient implementation of Beaver triple generator based on the El Gamal encryption scheme. Our solution is best suitable for the case where a small number of multiplications are required and hence a complement to MASCOT has been successfully implemented.

REFERENCES

[1] B. McMahan, "Communication-efficient learning of deep networks from decentralized data," in *Proc. 20th AISTATS*, Fort Lauderdale, FL, USA, 2017, pp. 1273–1282.

[2] Q. Yang, Y. Liu, T. Chen, and Y. Tong, "Federated machine learning: Concept and applications," *ACM Trans. Intell. Syst. Technol. (TIST)*, vol. 10, no. 2, pp. 1–19, 2019.

[3] P. Kairouz, "Advances and open problems in federated learning," 2019, *arXiv:1912.04977*. [Online]. Available: https://arxiv.org/abs/1912.04977

[4] Q. Li, Z. Wen, and B. He, "Federated learning systems: Vision, hype and reality for data privacy and protection," 2019, *arXiv:1907.09693*. [Online]. Available: https://arxiv.org/abs/1907.09693

[5] K. Bonawitz, V. Ivanov, B. Kreuter, A. Marcedone, H. B. Mcmahan, S. Patel, D. Ramage, A. Segal, and K. Seth, "Practical secure aggregation for privacy-preserving machine learning," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, Dallas, TX, USA, Oct. 2017, pp. 1175–1191.

[6] S. Hardy, "Private federated learning on vertically partitioned data via entity resolution and additively homomorphic encryption," 2017, *arXiv:1711.10677*. [Online]. Available: https://arxiv.org/abs/1711.10677

[7] R. Nock, S. Hardy, W. Henecka, H. Ivey-Law, G. Patrini, G. Smith, and B. Thorne, "Entity resolution and federated learning get a federated resolution," 2018, *arXiv:1803.04035*. [Online]. Available: https://arxiv.org/abs/1803.04035

[8] C. Zhang, "Efficient homomorphic encryption for cross-silo federated learning," in *Proc. USENIX Annu. Tech. Conf.*, Carlsbad, CA, USA, 2020, pp. 493–506.

[9] K. Wei, J. Li, M. Ding, C. Ma, H. H. Yang, F. Farokhi, S. Jin, T. Q. S. Quek, and H. Vincent Poor, "Federated learning with differential privacy: Algorithms and performance analysis," *IEEE Trans. Inf. Forensics Security*, vol. 15, pp. 3454–3469, 2020.

[10] K. Bonawitz, H. Eichner, W. Grieskamp, D. Huba, A. Ingerman, V. Ivanov, C. Kiddon, J. Konecný, S. Mazzocchi, H. B. McMahan, T. Van Overveldt, D. Petrou, D. Ramage, and J. Roselander, "Towards federated learning at scale: System design," 2019, *arXiv:1902.01046*. [Online]. Available: https://arxiv.org/abs/1902.01046

[11] A. Ratner *et al.*, "MLSys: The new frontier of machine learning systems," 2019, *arXiv:1904.03257*. https://arxiv.org/abs/1904.03257

[12] J. Read, A. Bifet, W. Fan, Q. Yang, and P. Yu, "Introduction to the special issue on big data, IoT streams and heterogeneous source mining," *Int. J. Data Sci. Anal.*, vol. 8, no. 3, pp. 221–222, Oct. 2019.

[13] A. Agrawal, R. Chatterjee, C. Curino, and A. Floratou, "Cloudy with high chance of DBMS: A 10-year prediction for Enterprise-Grade ML," in *Proc. 10th CIDR*, Amsterdam, The Netherlands, 2020, pp. 1–8.

[14] K. Karanasos, M. Interlandi, D. Xin, and F. Psallidas, "Extending relational query processing with ML inference," in *Proc. 10th CIDR*, Amsterdam, The Netherlands, 2020, pp. 1–8.

[15] I. Damgard, "Multiparty Computation from Somewhat Homomorphic Encryption," in *Proc. 32th CRYPTO*, Santa Barbara, CA, USA, 2012, pp. 643–662.

[16] I. Damgård, M. Keller, E. Larraia, and V. Pastro, "Practical covertly secure MPC for dishonest majority–or: Breaking the SPDZ limits," in *Proc. 18th ESORICS*, Egham, U.K., 2013, pp. 1–18.

[17] R. Cramer, I. Damgård, D. Escudero, P. Scholl, and C. Xing, "SPDZ_2^k: Efficient MPC mod 2^k for dishonest majority," in *Proc. 38th CRYPTO*, Santa Barbara, CA, USA, 2018, pp. 769–798.

[18] N. Smart and T. Tanguy, "TaaS: Commodity MPC via Triples-as-a-Service," in *Proc. ACM SIGSAC Conf. Cloud Comput. Secur. Workshop*, London, U.K., 2019, pp. 105–116.

[19] E. Orsini, N. Smart, and F. Vercauteren, "Overdrive2k: Efficient secure MPC over Z_2^k from somewhat homomorphic encryption," in *Proc. Cryptographers Track at RSA Conf.*, San Francisco, CA, USA, 2020, pp. 254–283.

[20] S. Goldwasser, S. Micali, and C. Rackoff, "The knowledge complexity of interactive proof-systems (extended abstract)," in *Proc. 17th STOC*, Providence, RI, USA, 1985, pp. 291–304.

[21] C. Schnorr, "Efficient identification and signatures for smart cards," in *Proc. 9th CRYPTO*, Santa Barbara, CA, USA, 1989, pp. 239–252.

[22] O. Goldreich, *The Foundations of Cryptography: Basic Techniques*, vol. 1. Cambridge, U.K.: Cambridge Univ. Press, 2001.

[23] O. Goldreich, *The Foundations of Cryptography, Basic Applications*, vol. 2. Cambridge, U.K.: Cambridge Univ. Press, 2004.

[24] Y. Lindell, "How To simulate it—A tutorial on the simulation proof technique," Int. Assoc. Cryptologic Res. (IACR), Bellevue, WA, USA, Tech. Rep. 216:46, 2016.

[25] J. Chen, "ALGORAND AGREEMENT: Super fast and partition resilient Byzantine agreement," Int. Assoc. Cryptologic Res. (IACR), Bellevue, WA, USA, Tech. Rep. 2018:377, 2018.

[26] J. Chen and S. Micali, "Algorand: A secure and efficient distributed ledger," *Theor. Comput. Sci.*, vol. 777, pp. 155–183, Jul. 2019.

[27] A. Shamir, "How to share a secret," *Commun. ACM*, vol. 22, no. 11, pp. 612–613, Nov. 1979.

[28] W. Du and J. Atallah, "Secure multi-party computation problems and their applications: A review and open problems," in *Proc. New Secur. Paradigms Workshop*, Cloudcroft, NM, USA, Sep. 2001, pp. 13–22.

[29] D. Bogdanov, M. Niitsoo, T. Toft, and J. Willemson, "High-performance secure multi-party computation for data mining applications," *Int. J. Inf. Secur.*, vol. 11, no. 6, pp. 403–418, Nov. 2012.

[30] D. Bogdanov, L. Kamm, B. Kubo, R. Rebane, V. Sokk, and R. Talviste, "Students and taxes: A privacy-preserving study using secure computation," *Proc. Privacy Enhancing Technol.*, vol. 2016, no. 3, pp. 117–135, Jul. 2016.

[31] T. Araki, J. Furukawa, Y. Lindell, A. Nof, and K. Ohara, "High-throughput semi-honest secure three-party computation with an honest majority," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, Vienna, Austria, Oct. 2016, pp. 805–817.

[32] M. Keller, E. Orsini, and P. Scholl, "MASCOT: Faster Malicious Arithmetic Secure Computation with Oblivious Transfer," in *Proc. 23rd ACM SIGSAC Conf. Comput. Commun. Secur.*, Vienna, Austria, Oct. 2016, pp. 830–842.

[33] T. Chou and C. Orlandi, "The simplest protocol for oblivious transfer," in *Proc. 4th Int. Conf. Cryptol. Inf. Secur. Latin Amer.*, Guadalajara, Mexico, Aug. 2015, pp. 40–58.

[34] T. ElGamal, "A public key cryptosystem and a signature scheme based on discrete logarithms," *IEEE Trans. Inf. Theory*, vol. 31, no. 4, pp. 469–472, Jul. 1985.

HUAFEI ZHU received the Ph.D. degree in cryptography from Xidian University, in 1997. From 1997 to 1999, he was an Assistant Professor with Zhejiang University, where he was an Associate Professor, from 1999 to 2003. Since 2003, he has been working with the Agency for Science and Technology (A*STAR), Singapore, where he is currently a Senior Research Scientist with the Institute of High Performance Computing. He has published more than 100-refereed journal and conference papers in the area of applied cryptography, information and network security, and privacy-preserving AI over protected data.

RICK SIOW MONG GOH (Member, IEEE) received the Ph.D. degree in electrical and computing engineering from the National University of Singapore, in 2006. He is currently the Director of the Computing and Intelligence (CI) Department, Institute of High Performance Computing (IHPC), a research institute under the Agency for Science, Technology and Research (A*STAR). He is the author or coauthor of more than 100-refereed publications. He has several patents filed and granted. He organized several international scientific conferences as a general co-chair.

WEE-KEONG NG received the Ph.D. degree from the University of Michigan at Ann Arbor, in 1996. He is currently an Associate Professor with the School of Computer Engineering, Nanyang Technological University, Singapore. He has published more than 200-refereed journal and conference papers in the area of data mining, web information systems, database and data warehousing, and data security and privacy.