A Secure Federated Learning framework using Homomorphic Encryption and Verifiable Computing

Abbass Madi, Oana Stan, Aurélien Mayoue, Arnaud Grivet Sébert, Cedric Gouy-Pailler, Renaud Sirdey

A Secure Federated Learning framework using Homomorphic Encryption and Verifiable Computing

Abbass Madi, Oana Stan, Aurélien Mayoue, Arnaud Grivet-Sébert, Cédric Gouy-Pailler and Renaud Sirdey

Université Paris-Saclay, CEA, List, F-91120, Palaiseau, France
name.surname@cea.fr

***Abstract*—In this paper, we present the first Federated Learning (FL) framework which is secure against both confidentiality and integrity threats from the aggregation server, in the case where the resulting model is not disclosed to the latter. We do so by combining Homomorphic Encryption (HE) and Verifiable Computing (VC) techniques in order to perform a Federated Averaging operator directly in the encrypted domain (by means of HE) and produce formal proofs that the operator was correctly applied (by means of VC). Due to the simplicity of the aggregation function, we are able to ground our approach in additive HE techniques which are highly mature in terms of security and decently efficient. We also introduce a number of optimizations which allows to reach practical execution performances on the larger deep learning models end of the spectrum. The paper also provides extensive experimental results on the FEMNIST dataset demonstrating that the approach preserves the quality of the resulting models at the cost of practically meaningful computing and communication overheads, at least in the cross-silo setting for which higher-end machines can be involved on both the client and server sides.**

***Index Terms*—Homomorphic Encryption, Verifiable Computation, Federated Learning**

I. Introduction

In recent years, machine learning solutions and in particular deep learning ones are widely employed in different domains such as healthcare, autonomous driving, finance and computer vision. However, this raises several security and privacy issues since the learning step requires access to massive amounts of heterogeneous data, part of which may be sensitive or private information.

Federated Learning (FL), an emerging recent training setting, allows to collaboratively train a model under the coordination of a central server or service provider without data outsourcing. As such, the data of each client remain stored locally without being shared and only the successive models are disclosed [1]. This is interesting for various reasons such as coping with the sensitive nature of training data, privacy laws and data regulations (*e.g.* GDPR [2], HIPAA [3], etc.) or business requirements. Even if this paradigm offers privacy improvements over a traditional centralized training model, recent research shows that attackers can indirectly retrieve private client data (based on the shared model updates, see [4], [5], [6] for more details). Moreover, there are cases in which the model itself is sensitive (*e.g.* due to proprietary reasons) or subject to attacks from/on the coordination server in order to alter it or modify the resulting inference capabilities.

In this context, there have been recent proposals to improve data privacy and model confidentiality through emerging cryptographic techniques such as Homomorphic Encryption [7], [8] or Multi-Party Computation [9], [10]. However, none of these approaches address integrity issues with respect to the computations performed to build the global model or to the clients' behavior. To the best of our knowledge, VerifyNet [11] is the only work proposing a privacy-preserving training for "cross-device" FL (the clients are a very large number of mobile or IoT devices) with guarantees of integrity of the global model, using a double-masking protocol and a homomorphic hash function.

In this paper, we complete the picture by proposing a novel secure approach for Federated Learning guaranteeing privacy protection for the training data and integrity for the overall model, using Homomorphic Encryption and Verifiable Computation. Our solution, which addresses threats coming from the server, is intended mainly for the "cross-silo" FL setting [12] in which the training involves only a small number of clients (that we suppose reliable), such as several organizations (e.g. medical, financial, insurance) collaborating to train a model.

In this work, we make the following main contributions:

- We present a secure framework for privacy-preserving and verifiable FL relying on Homomorphic Encryption and Verifiable Computation. Besides the general architecture, we also give some examples of practical use cases for our secure federated learning solution.
- We concretely instantiate our framework using the Paillier additive homomorphic scheme associated to the LePCoV primitive [13] which is a linearly homomorphic Authenticated Encryption with Public Verifiability scheme derived from [14]. It allows us to authenticate the computation of the global training model over the homomorphically encrypted data, with the additional guarantee that the correctness of the result can be publicly verified.
- In order to accelerate the performances of the system, we also propose a batching approach for the homomorphically encrypted data. We then give extensive performances results on the FEMNIST dataset and we study the accuracy of the resulting models and the overhead induced by using our homomorphic encryption and authentication primitives. All these results have been obtained using an efficient C/C++ prototype implementation of the framework written as part of this work.
- We provide a security and threat analysis for our FL approach. As such, under the security hypothesis we made, we ensure that both the clients local data and the updates to the gradients remain undisclosed to the server and that a malicious server cannot tamper or misuse the model while training.

Paper organization. The reminder of this paper is organized as follows. After presenting the related approaches in section II and introducing the necessary background in section III, we present our general framework, the associated threat analysis and the underlying cryptographic tools in section IV. The experimental results and related performances are shown in section V.

978-1-7281-6937-8/20/$31.00 ©2020 IEEE

II. Related work

A. Secure Federated Learning and Homomorphic Encryption

Most of the work consisting in applying homomorphic encryption to machine learning models concentrate on making the inference on private encrypted data (*e.g.* CryptoNets [15], TAPAS [16], NED [17]) and not so much on the training.

The first works addressing the problem of privacy-preserving machine learning training concentrated on a centralized setting where all the data are outsourced and the models are only linear [18], [19]. As for the few approaches proposing a complete centralized training of neural networks on homomorphic encrypted data, they have quite impractical performances or huge cryptographic parameters ([20]).

Other works propose solutions in the case of multi-servers either for clustering or regression. A lot of recent approaches employing homomorphic encryption are proposed for a collaborative distributed learning in which there is no central server mostly for linear models [21], [22] and, more recently, for neural networks [23].

As for the case of cross-silo FL, there are only a few recent papers, proposing the use of homomorphic encryption (usually additive) to ensure the secure computation of the global model ([7], [8], [24]). The first two approaches are only theoretical and the third one uses different datasets for the validation. Morever, all the above approaches are under the classical hypothesis of an honest but curious central server, without making use of any verifiable computing protocols to ensure the global model integrity.

B. Secure Federated Learning and Verifiable computation

In order to solve the problem of privacy protection and verifiability in deep learning system, several works have been proposed, like SafetyNets [25], and Slalom [26]. However, these schemes either support a small variety of activation functions like in [25] or require additional hardware assistance as in [26].

As for applying the verifiable computing protocols for the FL setting (i.e. verify the integrity of the aggregated results returned from the central server), to the best of our knowledge, there is only a recent work on this subject. Xu et al. [11] introduce the VerifyNet architecture as a solution for cross-device FL preserving the integrity and the confidentiality of the model with regards to the global server. The verification of the server calculation results (i.e. the aggregated model) is realized using the homomorphic hashes and pseudorandom functions. The privacy of user's gradients is guaranteed via a double-masking protocol, having the inconvenient that it requires multiple exchanges between the users.

As seen in this section and, to the best of our knowledge, so far there are no approaches for a secure FL which supports the integrity of the server calculation results while guaranteeing the model privacy by means of verifiable computation and homomorphic encryption.

C. Secure Federated Learning and other Multi-Party Computation

Multi-party computation (MPC) protocols allow several parties to collaborate in order to compute a function on their private data such that each party knows only its input and output. There are several FL approaches using MPC [9], [10] but due to the high-communication costs of the multi-party computation and the inherent distributed nature of FL, it is difficult to implement efficient methods.

D. Secure Federated Learning and Differential Privacy

A few works [27]–[30] have implemented differential privacy to protect clients' data from other clients or end-users in a FL context. These works suggest that differential privacy is more appropriate for cross-device FL applications. Indeed, a high number of clients is required to simultaneously allow that

- the ratio of participants per round is low, thus limiting the probability that a given client participates and therefore (indirectly) releases any information about his training data in the considered round
- the absolute number of participants per round is high, reducing the sensitivity of the model updates in a client-level differential privacy point of view

This hypothesis is not reasonable in the context of the present paper where the number of clients does not exceed a few hundreds. That is why we focus in this work on scenarios in which we consider that the recipients of the final model (clients and end-users) do not perform attacks like membership inference [31], [32] or model inversion [4], [6] on the model updates or the final model. Properly implementing differential privacy would need further experiments and we wish to design such a framework in a future work.

III. Preliminaries

A. Federated Learning (FL)

FL is a decentralized framework that enables multiple clients to collaboratively train a shared global model under the orchestration of a central server while keeping the training data distributed on the client devices. As a starting point, the server initializes a global model randomly and then the FL process consists of multiple rounds. At the beginning of each round, the server selects a subset of clients that take part in training and sends to them the current global model. Next, each selected client trains the model locally on his own data and communicates only the model updates back to the server. Finally, the server aggregates these updates before accumulating them into the global model thereby concluding the round. The most common approach to optimization for FL is the Federated Averaging algorithm [1]. Here, each client runs several epochs of minibatch stochastic gradient descent (SGD) minimizing a local loss function, and then the central server performs a weighted averaging of the updated local models to form the updated global model. Pseudocode is given in Algorithm 1. By removing the need to aggregate all data on a central server, FL helps to ensure data privacy and reduces communication costs. As a result, FL applies best in situations where data are privacy-sensitive or large in such a way that it is undesirable or infeasible to transmit them to the server.

B. Homomorphic Encryption (HE)

Homomorphic Encryption (HE) schemes allow to perform computations directly over encrypted data without decrypting it first. That is, with a Fully homomorphic Encryption scheme E, we can compute $\mathsf{E}(\mathsf{m}_1+\mathsf{m}_2)$ and $\mathsf{E}(\mathsf{m}_1\times\mathsf{m}_2)$ from Encrypted messages $\mathsf{E}(\mathsf{m}_1)$ and $\mathsf{E}(\mathsf{m}_2)$. Thus homomorphic encryption provides a way to outsource computations to the cloud while protecting the data confidentiality. Moreover, a simple additive homomorphic cryptosystem (*i.e.* allowing to obtain only the encryption of the addition of two messages) is enough to perform secure federated averaging. Let us recall the general principles of the **Paillier cryptosystem**, a well-known and popular additive homomorphic scheme [33].

KeyGen$(sz)\rightarrow(pk,sk)$: It generates the keys for the cryptosystem taking as input the number of bits sz of the modulus.

Algorithm 1 Federated Averaging M is the total number of clients; K is the number of participants per round t; the selected clients are indexed by k with $\mathbf{D_k}$ the training set of data points on client k and $n_k = |\mathbf{D_k}|$; B is the local minibatch size; E is the number of local epochs; μ is the learning rate; w are model parameters and l is the local loss function.

Server executes:

initialize w_0
for each round t **do**
  $K_t \leftarrow$ random set of $K \leq M$ clients
  for each client $k \in K_t$ in parallel **do**
    $(w_{t+1}^k, n_k) \leftarrow$ ClientUpdate(k, w_t)
  $w_{t+1} \leftarrow \sum_{k=1}^{K} \frac{n_k}{n} w_{t+1}^k$ where $n = \sum_{k=1}^{K} n_k$

ClientUpdate(k,w)**:**

initialize $w_k = w$
$\mathcal{B} \leftarrow$ split n_k samples of $\mathbf{D_k}$ into batches of size B
for each round epoch from 1 to E **do**
  for each batch $b \in \mathcal{B}$ **do**
    $w_k \leftarrow w_k - \mu \bigtriangledown l(w_k; b)$
return (w_k, n_k)

Choose two large prime numbers p_E and q_E such that $\lambda = \text{lcm}(p_E - 1, q_E - 1)$, and set $N_E = p_E q_E$. We note that the cleartext domain is $\mathbb{Z}_{N_E}$ and the ciphertext domain is $\mathbb{Z}_{N_E^2}$.

Select a random $g < N_E^2$ such that $\gcd(L(g^\lambda \bmod N_E^2), N_E) = 1$,with $L(u) = \frac{u-1}{N_E}$.

Set $pk = (N_E, g)$ and $sk = (p_E, q_E)$.

Enc$_{pk}(m) \rightarrow c$: It produces a ciphertext c using the public key pk by computing $c = g^m r^{N_E} \bmod N_E^2$, where $m < N_E$ is the message and r is uniformly chosen in $\mathbb{Z}_{N_E}$.

Dec$_{sk}(c) \rightarrow m$: It computes the plaintext m from the ciphertext c, using the private key sk.

Letting $D = L(g^\lambda \bmod N_E^2)$ and D^{-1} its multiplicative inverse in $\mathbb{Z}_{N_E}$, the decryption is performed by evaluating

$$m = \text{Dec}(c) = L(c^\lambda \bmod N_E^2) \times D^{-1} \bmod N_E.$$

More importantly, for the present purpose, this cryptosystem has the following homomorphic properties:

1) $\text{Dec}(\text{Enc}(m_1)\text{Enc}(m_2)) \bmod N_E^2 = m_1 + m_2 \bmod N_E$ (addition of two encrypted messages).
2) $\text{Dec}(\text{Enc}(m) g^k) \bmod N_E^2 = m + k \bmod N_E$, for all $k \in \mathbb{Z}_{N_E}$ (addition of an encrypted message to a clear integer).
3) $\text{Dec}(\text{Enc}(m)^k) \bmod N_E^2 = km \bmod N_E$, for all $k \in \mathbb{Z}_{N_E}$ (multiplication of an encrypted message by a clear integer).

C. Batching for Paillier

We can batch several plaintext messages m_i in a same Paillier ciphertext, each one represented on t bits. Let b be the size of a batch, i.e. the maximum number of positive messages we can encrypt in a same Paillier ciphertext $c_p = g^{m_1 + m_2 + ... + m_b} * r^{N_E} \bmod N_E^2$ or written differently $Enc(m_1|m_2|...|m_b)$. To decrypt correctly c_p it follows that $|m_1 + m_2 + ... m_b| \leq N_E$ and $b \leq log(N_E)/t$ (i.e. with a modulus of 2048 bits and messages of $t = 64$ bits, it is possible to pack together max 32 messages).

However, if we want to perform addition on these packed ciphertexts, one must take this into account to set up the dimension of the initial batch to avoid an overflow. Let nb_{add} the number of additions we want to perform on these packed messages. The padding (i.e. the number of zero bits) that has to be added to each slot is equal to nb_{add}.

As such, one has to encrypt $Enc(m_1 0...0 | m_2 0...0 | ... | m_b 0...0)$. The size of the batch will then have to be at most: $b = \left\lfloor \frac{log_2(N_E)}{t + nb_{add}} \right\rfloor$.

Let us give a short example. Suppose each message we want to encrypt is a real positive number between 0 and U, with $U = 100$ and m_i having 4 representative digits after the decimal point. As such to represent these messages, one must have at least $\lceil log_2(U * 10^4) \rceil$, i.e. 20 bits. To perform two additions on ciphertexts batching these type of messages, for a modulus of 2048 bits, one can pack $2048/(20+2) = 93$ messages in a single ciphertext. For 100 additions on the ciphertexts of the same format, one can have at most 17 slots/ciphertext.

D. Verifiable Computation

Verifiable Computation (or Verifiable Computing) VC is a cryptography tool meant to secure the integrity of computations on authenticated data. It enables a client to delegate to another entity (in most cases a server) the computation of a function. The other entity evaluates the function and returns the result with a proof that the computation of the function was carried out correctly.

Now, we present the VC for Paillier cryptosystem [13]. It is a Linearly Homomorphic Authenticated Encryption scheme with Public Verifiability (LAEPuV) and provable correctness called LEPCoV and it allows the public verifiability of data returned by the server. This scheme improves Catalano et al.'s instantiated scheme [14] by avoiding false negatives during the verification step.

Let $S = (KeyGenS, Sign, Verify)$ be a signature scheme.

- $AkeyGen(sz, I)$: takes as input a prime size sz (in number of bits) and an integer I representing the upper bound for the number of messages encrypted in each dataset. It calculates the secure sk and public pk parameters as follows: sample four (safe) primes p_E, q_E, p_S, q_S of size $sz/2$, such that $N_E = p_E \cdot q_E$ and $N_S = p_S \cdot q_S$ it holds that $\varphi(N_S) = (p_S - 1)(q_S - 1)$, the group elements g_0, g_1, h_1,..., $h_I \in \mathbb{Z}_{N_s}^*$ and $g \in \mathbb{Z}_{N_E^2}^*$ of order N_E, and picks a hash function H. Finally, it runs $KeyGenS(sz)$ to obtain the secure and private signature key (sk_S, pk_S). Returns the key pair (sk, pk), where $pk = (N_E, g, N_S, g_0, g_1, h_1, ..., h_I, H, pk_S)$ and $sk = (p_E, q_E, p_S, q_S, sk_S)$.
- $AEncrypt(sk, \tau, i, m)$: probabilistic algorithm that takes as input the secret parameter, a message $m \in M$, a dataset identifier τ, and an index $i \in \{1,...,I\}$ to calculate the ciphertext c containing the encryption of the message with the tag of verification. Thus, it computes the Paillier encryption C of the message m, $R = H(\tau||i)$, and a tuple $(a,b) \in \mathbb{Z}_{N_E} \times \mathbb{Z}_{N_E}^*$ such that $g^a b^{N_E} = CR \bmod N_E^2$ (using the factorisation of N_E). In addition, if τ is used for the first time, it chooses a not yet used prime e of length $l \leq sz/2$ such that $gcd(eN_E, \varphi(N_S)) = 1$, it computes its inverse $e^{-1} \bmod \varphi(N_S)$, and its signature $\mu_e = Sign_{sk_S}(\tau||e)$ and it stores (τ, e, e^{-1}, μ_e) in the list L. Otherwise, it takes (τ, e, e^{-1}, μ_e) from the list L. Then, it chooses an element s uniformly at random from $\mathbb{Z}_{eN_E}$ and it computes x using the p_S and q_S such that $x^{eN_E} = g_0^s h_i g_1^a \bmod N_S$. It returns $c = (C, e_i, e_i^{-1}, \mu_e, \tau, \sigma)$, where $\sigma = (a, b, s, x)$ is the verification tag.
- $AEval(pk, \tau, f, \{c_i\}_{i \in [I]})$: takes as input the public key pk, a dataset identifier τ, a linear function $f = (f_i)_{i \in I}$ and I ciphers $\{c_i\}_{i \in [I]} = (C_i, e_i, e_i^{-1}, \tau_i, \sigma_i)$. The output is a cipher c. The algorithm checks if there exists an index $l \in [I]$ such that $\tau \neq \tau_l$, or that the signature $(Verify(pk_S, \tau||e_l, \mu_{e_l}) = 0)$. Furthermore,

the algorithm checks if there are two indexes $i \neq j \in [I]$ such that $e_i \neq e_j$. If one of the checks is true, the algorithm aborts. Otherwise, the algorithm sets $e = e_1, e^{-1} = e_1^{-1}$, $\mu_e = \mu_{e_1}$ and evaluates f over ciphertext as: $C = \prod_{i=1}^{I} C_i^{f_i} \mod N_E^2$. It also evaluates f over the tag to obtain a new tag (a,b,s,x) as follows: $a = \sum_{i=1}^{I} f_i a_i \mod N_E$, $b = \prod_{i=1}^{I} b_i^{f_i} \mod N_E^2$, $s = \sum_{i=1}^{I} f_i s_i \mod eN_E$, $s' = \left(\sum_{i=1}^{I} f_i s_i - s\right)/(eN_E)$, $a' = \left(\sum_{i=1}^{I} f_i a_i - a\right)/N_E$, and $x = \frac{\prod_{i=1}^{I} x_i^{f_i}}{g_0^{s'} g_1^{a' e^{-1}}} \mod N_S$.
It returns the cipher $c = (C, e, e^{-1}, \mu_e, \sigma)$.

- $AVerify(pk, \tau, c, f)$: takes as input the public key pk, a dataset identifier τ, a cipher $c = (C, a, b, e, s, \tau, x)$, and a linear function $f = (f_i)_{i \in [I]}$, to detect if c is a valid or invalid cipher. For this goal, the algorithm checks that:
$$\begin{cases} Verify(pk_S, \tau || e, \sigma_e) = 1; \\ a, s \in \mathbb{Z}_{eN_E}; \\ x^{eN_E} = g_0^s \prod_{i=1}^{I} h_i^{f_i} g_1^a \mod N_S; \\ g^a b^{N_E} = C \prod_{i=1}^{I} H(\tau || i)^{f_i} \mod N_E^2; \end{cases}$$
If all checks pass, it outputs 1 (i.e c is a valid cipher), else it outputs 0, (i.e. c is an invalid cipher).
- $ADecrypt(sk, \tau, c, f)$: Taking as input the secret parameter sk, a data set identifier, a cipher c, and a linear function $f = (f_i)_{i \in [I]}$, it calculates the decryption of c or $\perp$ (if c is invalid cipher). Then it verifies if c is a valid cipher by running $AVerify\ (pk, \tau, c, f)$. If passed, the algorithm returns the message m obtained by $Dec(c)$. Otherwise, it returns $\perp$.

For lack of space, we refer the reader to [14] for the correctness and the security proofs for LAEPuV scheme and to [13] for the security and correctness proofs of the LEPCoV scheme.

IV. A Secure Framework for Confidential and Verifiable Federated Learning

A. Overview of the architecture

Figure 1 shows the high-level view of the FL framework while training with a total of M clients in the case of a malicious central server. In our approach, the confidentiality of the model is ensured by homomorphic encryption and the integrity for the computation of the global model by the central server is guaranteed by public verifiability.

Before the training actually begins, the central server shares with the M clients the global architecture of the neural network that will be trained. Also, once the enrollment of the clients participating to the training is completed, the keys necessary for the homomorphic encryption and the signatures are generated by one of the clients. This client responsible for the key generation can be randomly selected by the server or be the result of a leadership election protocol. All the clients will then share the same pair of (sk, pk) keys, with the sk key necessary for the decryption and signing and the public key pk for evaluation and verification. The central server holds only the pk required for the homomorphic evaluation and signature of the global model.

At each round of the training which is an iterative process, the server randomly selects K out of the M clients. Each client sends its local updates of the weights homomorphically encrypted together with an authentication tag. The server updates the global model by computing a homomorphic aggregation on the weights. In the same time, it computes the signature tag associated with the global model. The homomorphic result and the signature tag are sent back to the K clients. Each of them will verify that the global model was computed correctly and if so they will decrypt the received global weights. This concludes the current round and a new iteration can begin. If the verification fails, the clients will take appropriate actions, outside the scope of this paper.

Once the training is finished, the parameters of the final global model are sent back to the M clients which decrypt them and explore them internally for prediction on their local datasets.

It is noted that the same technique can be extended with minor modifications to the case we consider gradients instead of weights as local updates sent by the clients.

As example of a relevant application of our secure FL framework, one can cite the training of a federated model across multiple medical institutions without sharing the patient data. This allows to collaborate and build relevant models while keeping the patient data in local, at the hospital and thus reduce the risk of data leakage while respecting the health regulations. Another example of use case is the training of a common model by the partners of a common defence alliance (*e.g.* NATO) without sharing their sensitive military data.

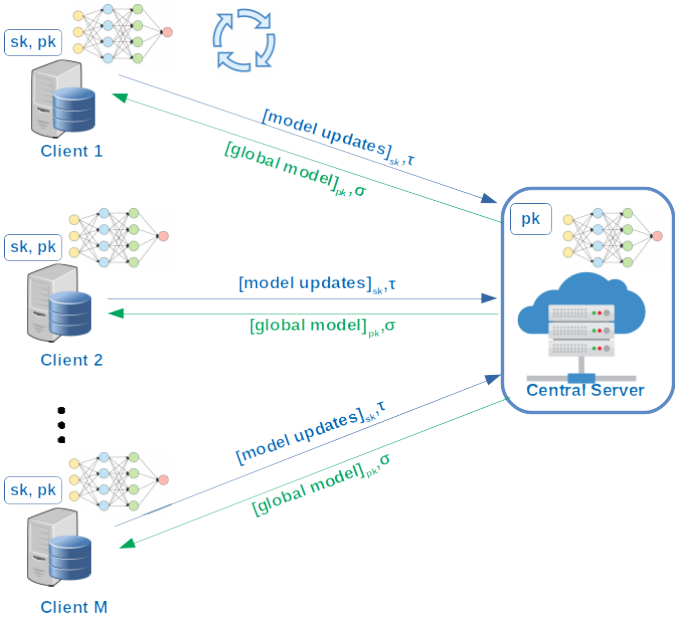

Fig. 1. Cross-silo federated learning architecture with Homomorphic Encryption and public Verifiability

B. Threat and security analysis

Following common cybersecurity practices, this section summarizes the security properties of our framework in terms of assets, threats and countermeasures. In our setup we have two assets: the training data, owned by the clients, and the model they collectively build with the help of the aggregation server. First, the confidentiality of the training data of a given client must be guaranteed with respect to threats coming from both the server as well as (ideally) the other clients. Additionally, the confidentiality of the model (or, rather the successive models) must be guaranteed with respect to threats from the server (as all the clients are granted access to the succession of models, there is no confidentiality requirements on the models w. r. t. the clients in our protocol). The integrity of the model should also be guaranteed against threats coming from the server and, ideally, from the clients. In this setup, the framework presented in this paper encompasses two complementary countermeasures in order to address the above server threats: Fully Homomorphic Encryption and public Verifiable

Computing. Since FHE allows the server to perform its aggregation function by working directly over encrypted model data, it addresses confidentiality threats on the model data coming from the server and, as a by-product that the server works in the encrypted-domain, on the clients' training data as well. Now, turning to integrity, our Verifiable Computing-approach allows all clients to formally establish that the server correctly performs its aggregation function (albeit on encrypted data) and, as such, mitigates integrity threats from the server. At present, client threats are not (yet) covered by our framework. In particular Differential Privacy, by adding an appropriate noise to the successive models coefficients (thus preventing model inversion attacks and the like by the clients which receive the successive models), can help mitigating confidentiality threats on the clients training data with respect to one another. Still, properly introducing DP within a FL framework is easier said than done and will be the focus of another paper. The last kind of threats that our framework does not fully cover (and which would also stay uncovered even when bringing DP into the picture) concerns integrity threats on the model coming from the clients [4]. It should however be emphasized that these threats are very hard to mitigate as malicious clients can misbehave in many different and possibly harmful ways (*e.g.* from lying on their training set sizes to using false or even misleading training data). Still, some form of integrity on the final model can be checked a posteriori, for example by having each client running the model on its own private test set and measuring the resulting classification rate. Let us also note that our security model is valid under the hypothesis of non collusion between the server and any user participating in the FL protocol.

C. Cryptographic tools and optimizations

As detailed in the section III-C, one can batch several messages into the same Paillier ciphertext. In the context of our FL approach, the weights updates by client as well as the overall global model parameters are quite important in terms of size. By batching the weights local updates and the weights of the global model one can diminish the bandwidth requirements and also the evaluation time on the server side.

Let us now give the example on how the batching technique applies on federated averaging. For the federated averaging, on the central server side, one must compute in the encrypted domain: $w_{t+1} \longleftarrow \sum_{k=1}^{K} \frac{n_k}{n} * w_{t+1}^k$ based on the encrypted updates of the weights received from the clients k.

Of course $0 < n_k/n \leq 1$ with r representative digits after the decimal point. The weights are usually real numbers for which we will keep only p representative digits of precision. For simplicity reason, we suppose that all weights are translated into the positive domain. In this case, one must have for each slot in the packed message extra space for the term n_k/n. As such, each slot i for a packed ciphertext will be in the form

$$[\underbrace{n_k/n}_{\lceil log_2(10^r) \rfloor} | \underbrace{w_{t+1,i}^k}_{\lceil log_2(10^p) \rfloor} | \underbrace{0...0}_{K}].$$

Therefore, one can pack at most b weights in a single ciphertext with

$$b = \Big\lfloor \frac{log_2(n)}{log_2(10^r) + log_2(10^p) + K} \Big\rfloor.$$

More concretely, let us suppose $n_k/n = (0, r_1 r_2 r_3)$, each weight is in the form $(0, p_1 p_2 p_3 p_4)$ and $K = 10$. It follows that we can have at most $\lfloor 2048/(log_2(10^3 * 10^4) + K) \rfloor = 61$ packed messages in a single ciphertext.

Let us now go into more details on the training protocol when using the above specified cryptographic primitives.

At each round of the training, each client sends the result of $AEncrypt$ over the local updates of the weights: $c = (C, e, e^{-1}, \mu_e, \sigma)$ containing the ciphers $(C,\ e,\ e^{-1},\ \mu_e)$ concatenated with the corresponding tag σ. The server updates the global model by calling the $AEval$ algorithm over the received messages c and using f, where f is the aggregation function. The output of $AEval$ is sent back to the K clients. Each of them runs the $AVerify$ algorithm to verify that the global model was computed correctly. If so they will evaluate the $ADecrypt$ over the message received to decrypt this message and obtain the global weights. As mentioned in the section III-D, the VC scheme for Paillier encryption (LEPCoV) verifies the outsourced computation of any linear function over any messages in $\mathbb{Z}_{N_E}$. Then to adapt the LEPCoV for the Paillier batching encryption of weights it is sufficient to modify only the message weights w_k to w'_k, the packed version of the weights. Then each user runs $AEncrypt(sk, \tau, i, w'_k)$ instead of $AEncrypt(sk, \tau, i, w_k)$ and the cipher becomes $c = (C', e, e^{-1}, \mu_e, \sigma)$, where $C' = Enc(w'_k)$ as illustrated above. The remaining of the framework is completed like before. Finally, we note that in the evaluation algorithm, we can evaluate C, b, and x in parallel (i.e. the runtime of $AEval = max(AEval(C), AEval(b), AEval(s) + AEval(a) + AEval(x))$). Each user can run $AEncrypt$ and respectively $AVerify$ in parallel over the batches of uploaded and respectively downloaded weights so the associated evaluation times can also be reduced.

V. Experimental results

We use the Federated Extended MNIST (FEMNIST) dataset[1] as experimental setup. The extended version of MNIST contains 62 classes (digits, upper and lower letters) and comes with the writer id in such a way that its federated version was built by partitioning the data based on the writer [34]. Among the 3,596 writers contained in the original dataset, we keep the 500 users with the most data. Each selected user's dataset has a train/test data split for a total of 165,050 train and 27,433 test images.

Evaluations were done with a standard CNN composed of two convolution layers (the first with $5*5$ kernel size and 128 channels, the second with $3*3$ kernel size and 64 channels, each followed with $2*2$ max pooling), a fully connected layer with 512 units and ReLu activation, and a final softmax output layer (486,654 total parameters). The evaluation metric was the accuracy on the test sets and, for each experimentation, 200 learning rounds were done.

We have led two distinct sets of experimentation. On one hand, we aimed at finding the best hyperparameters (K, B, E)[2] to improve the speedup of the learning process and thus decrease the communication cost. On the other hand, our concern was about privacy and our goal was to secure the FL process without degrading its performance.

A. Setting FL hyperparameters

To evaluate the speedup of the learning process relative to the FL hyperparameters, we report the number of communication rounds to reach a decent target accuracy of 80%. We first experiment with the number K of participants per round, which controls the amount of multi-client parallelism. Setting $B = 5$ and $E = 10$, we show the impact of varying K (see Fig.2). Above $K = 10$, there is only a small advantage in increasing the client fraction. Thus, for the remainder

[1]Dataset available at https://www.nist.gov/itl/products-and-services/emnist-dataset

[2]Notations are those introduced in Algorithm 1

of our experiments we fix $K = 10$, which strikes a good balance between computational efficiency and convergence rate.

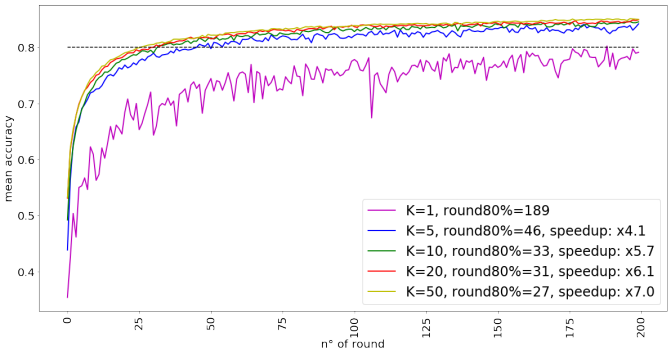

Fig. 2. Test set accuracy vs. communication rounds varying the number of participants (B=5 and E=10)

Our second experiment aims at choosing B and E, which control the number of local calculations for each client. Starting from $B = 50$ and $E = 1$, we add more computation per client on each round, either decreasing B, increasing E, or both. Table I demonstrates that adding more local SGD updates per round can produce a dramatic decrease in communication costs. In the following, we use the parameters B=5 with E=10 which give the best convergence rate for the FL process.

		E			
		1	5	10	20
B	50	-	195	149	152
	10	155	50	48	44
	5	87	40	**33**	34

TABLE I
NUMBER OF COMMUNICATION ROUNDS TO REACH A TARGET ACCURACY OF 80% VARYING BOTH B AND E (WHILE K=10)

B. Quantization vs. utility

Since the encryption quantifies the clear messages, we conducted experiments to analyze the impact of this quantization on the utility of the model. Contrary to Section V-A, we fixed the number of learning rounds to 200 and reported in Table II the accuracy varying the precision on both w_k and $\frac{n_k}{n}$ for each participant k. We observe that a 10^4 precision is required not to degrade performances. In Figure 3, we focus on performance deterioration due to precision on w_k by showing accuracy for each round considering different weights precisions (while precision on $\frac{n_k}{n}$ is float32).

		precision on w_k			
		float32	10^4	10^3	10^2
precision on $\frac{n_k}{n}$	float32	**84.6%**	**84.2%**	82.6%	75.4%
	10^4	**84.5%**	**84.6%**	82.8%	75.1%
	10^3	83.5%	83.3%	82.0%	75.4%
	10^2	78.0%	78.0%	77.2%	73.9%

TABLE II
ACCURACY AFTER 200 LEARNING ROUNDS DEPENDING ON PRECISION ON BOTH w_k AND $\frac{n_k}{n}$

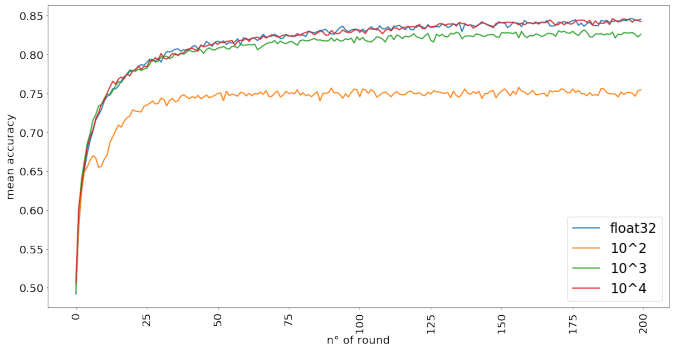

Fig. 3. Test set accuracy vs. communication rounds varying the precision on weights w_k (while precision on $\frac{n_k}{n}$ is float32)

C. Performance evaluation of LEPCoV scheme

In the beginning, let us specify that all tests presented in this section were performed on a 2016 DELL PC(Genuine-Intel Core $i7-6600U$, 4 cores at $2.60GHz$ with $16GB$ RAM at $2.13GHz$), on Ubuntu (Linux kernel $4.15.0-91-generic$, with the architecture $x86-64$) as an operating system. Finally we used the C++ language to implement the LEPCoV scheme.

Table III summarizes the average runtimes of $AKeyGen$, $AEncrypt$, $AVerify$, $ADecrypt$, $AEval$ and the detailed evaluation, over the tag $\sigma = (a,b,s,x)$, and over the ciphers C. Note that the evaluation time of $AEncrypt$ and $ADecrypt$ depends only on the size of the cryptosystem parameters and the evaluation time of $AKeyGen$, $AEval$, $AVerify$ further depends on the number of participants K. We note that the $AKeyGen$ is performed once. The time presented in this table for $AEncrypt$ is the time evaluation for the encryption and tag generation for one message (weight). We recall that we can evaluate f over C, b, and x in parallel (i.e. the runtime of $AEval = max(AEval(C), AEval(b), AEval(s) + AEval(a) + AEval(x))$). We also remark that the verification time is rather fast (e.g 28,87ms for $K = 10$ and a 2048 bits modulus).

Table IV shows the batching characteristics one can use to encrypt with Paillier cryptosystem the clients updates for the CNN model described earlier with 486,654 parameters. As such, we report the number of slots (column "#slots") and the number of ciphertexts (column "#ctxts") per participant in one round in function of the number of participants (K), the precision of the weights w_k and of the term n_k/n as well as the modulus size. The number of slots (and implicitly the number of encrypted messages per round) depends on the number of bits of the modulus, the number of participants K and the precision of both n_k/n and w_k.

Table V shows the bandwidth size in one round between a client and the central server on upload and download. On the upload (direction client-server) this message is the output of AEncrypt algorithm, it thus contains the cipher c and the tag of verification σ. On the download, it contains the result of $AEval$ algorithm that runs on the central server. For example, for a modulus size of 2048 bits, 10^4 of precision for both n_k/n and w_k and $K = 10$, each user (out of 10) sends $486{,}654 \times 5.6\text{KB} \simeq 2.5\text{GB}$ to central server without batching or it sends $8848 \times 5.6\text{KB} \simeq 48.3\text{MB}$ to the central server with batching. Each client receives the same message from the server of the size 2.5 GB in the case without batching or 49.5 MB with batching. The 5.6 KB mentioned above is the size for one message, containing the size of the ciphers $C = (c, e, e^{-1}, \mu_e, \tau,)$ auditioned with the size of the tag $\sigma = (a,b,x,s)$, both of equal sizes.

Modulus size	1024			2048			3072		
K	10	20	50	10	20	50	10	20	50
$AkeyGen$	19.98	20.07	21.04	163.9571	167.89	170.84	638.79	644.5	645.44
$AEncrypt$	5.17	5.23	5.36	37.12	37.05	37.12	124.07	117.39	118.6
$AVerify$	4.36	4.54	5.93	28.87	30.3	34.14	96.54	96.21	101.35
$Decrypt$	1.71	1.26	1.21	8.6	9.01	8.69	28.18	27.8	28.34
$AEval$	0.38	0.46	1.16	2.76	1.57	4.16	8.58	8.17	9.31
$AEval(c)$	0.23	0.46	1.14	0.8	1.57	4.08	1.77	3.82	9.31
$AEval(a)$	0.0006	0.0008	0.001	0.0008	0.001	0.002	0.001	0.001	0.003
$AEval(b)$	0.23	0.43	1.16	0.77	1.55	4.16	1.75	3.55	8.53
$AEval(s)$	0.0009	0.001	0.004	0.001	0.001	0.003	0.001	0.002	0.004
$AEval(x)$	0.38	0.37	0.42	2.76	2.73	2.5	8.58	8.17	7.52

TABLE III

AVERAGE RUNTIMES (IN MS) OF **AKEYGEN,AENCRYPT,ADECRYPT** AND **AEVAL**, FOR DIFFERENT MODULUS AND DATA SIZE K. WITH THE FUNCTION $f(x)=\sum_{i=0}^{k} n_k/n\ w_i$ WHERE n_k/n AND w_k OF SIZE 10^4

		Modulus size											
		1024				2048				3072			
	n_k/n precision	10^4		10^3		10^4		10^3		10^4		10^3	
K	w_k precision	#slots	#ctxt	#slots	#ctxt	#slots	#ctxt	#slots	#ctxt	#slots	#ctxt	#slots	#ctxt
10	10^4	27	18024	30	16221	55	8848	61	7977	83	5863	92	5289
	10^3	30	16221	34	14313	61	7977	68	7156	92	5289	102	4771
	10^2	34	14313	38	12806	68	7156	76	6403	102	4771	115	4231
20	10^4	21	23174	23	21158	43	11317	47	10354	65	7486	71	6854
	10^3	23	21158	25	19466	47	10354	51	9542	71	6854	76	6403
	10^2	25	19466	27	18024	51	9542	55	8848	76	6403	83	5863
50	10^4	13	37434	13	37434	26	18717	27	18024	40	12166	41	11869
	10^3	13	37434	14	34761	27	18024	29	16781	41	11869	43	11317
	10^2	14	34761	15	32443	29	16781	30	16221	43	11317	46	10579

TABLE IV

PAILLIER BATCHING REQUIREMENTS IN FUNCTION OF THE PRECISION AND NUMBER OF PARTICIPANTS

		without batching	with batching		
	K / mod	10,20,50	10	20	50
Client, Server	1024	1200	49.2	63.3	102.3
Client, Server	2048	2500	48.3	61.8	102.3
Client, Server	3072	3800	47.5	60.6	98.6

TABLE V

SIZE OF BANDWIDTH (IN MB) BETWEEN ONE CLIENT AND THE CENTRAL SERVER IN ONE ROUND, WITH 10^4 PRECISION OF BOTH n_k/n AND w_k.

	K=10		K=20		K=50	
	unit	total	unit	total	unit	total
AEncrypt	0.73	6469.7	0.756	8552.49	0.741	13865.18
AEval	0.03	244.82	0.06	650.50	0.14	2619.63
AEval(c)	0.028	244.82	0.06	650.50	0.14	2619.63
AEval(a)	≈0	0.02	≈0	0.06	≈0	0.18
AEval(b)	0.03	243.94	0.06	636.35	0.14	2619.63
AEval(s)	≈0	0.04	≈0	0.09	≈0	0.19
AEval(x)	≈0	36.45	≈0	46.62	≈0	78.61
AVerify	0.07	579.54	0.10	1144.15	0.21	4004.69
ADecrypt	0.01	76.09	0.01	97.89	0.01	160.97

TABLE VI

Sequential RUNTIMES (IN SECONDS) FOR 10^4 PRECISION OF BOTH n_k/n AND w_k AND A MODULUS SIZE OF 2048 BITS

Table VI shows the *sequential* evaluation times for the different cryptographic primitives with batching, for one round of the training, for a precision level of 10^4 for both n_k/n and w_k and a modulus size of 2048 bits. The experiments were performed for a modulus on 2048 bits since in terms of security, a modulus size of 2048 bits provides long-term security guarantees following common practice, 1024 bits is considered insufficient and 3078 bits is reserved for "beyond 30 years" confidentiality requirements. Columns "unit" report the times per unitary encrypted messages while columns "total" report the times spent to execute the model with all the 486 654 parameters but without any parallelization. Let us however emphasize that the execution times especially for $AEncrypt$ and $AVerify$ can be further improved by involving simple multi-core parallelization, which seems a viable option in the cross-silo FL context to which this work applies. Indeed, on the client side, each ciphertext (which contains several weights) can be prepared independently (and furthermore the Paillier encryption function can be split in a message independent part, which can be precomputed, and an online message dependent part in order to reduce latency). Similarly, on the server side, the averages can also be computed independently. So due to the "embarrassingly parallel" nature of both, the computing times in Table VI can easily (*e.g.* by an OpenMP parallel-for) be reduced by one or two orders of magnitude depending on the number of cores of the machines involved in the protocol. Again, involving high-end machines on both client and server sides seems realistic in the cross-silo setting.

VI. CONCLUSION AND PERSPECTIVES

The framework presented in this paper addresses both confidentiality and integrity threats, on both the training data and model, coming from the aggregation server by means of HE and VC techniques. On top of providing strong cryptographic security guarantees, and despite the far from negligible overhead induced by these techniques, we claim that our framework achieves practical performances at least in cross-silo setting when the participants are willing to deploy high-end machines (between 10 to 100 cores) to decrease the overall protocol latency to a sustainable level.

As such, this framework is a significant step towards private-by-design federated learning. However, it is also desirable to cover a wider security model by also countering threats from the end-users thus extending our solution to a fully secure framework applicable in a context where the recipients of the final model may not be trusted (to some extent). Notably, this will require to bring differential privacy (DP) into the picture in order to prevent indirect leakage of sensitive information on the training data from the successive models built (and disclosed to the clients) in the protocol. In a scenario in which the clients or the end-users of the final model may be malicious, DP would indeed protect the model updates or the model itself from attacks like membership inference [31], [32] or model inversion [4],

[6]. Yet, there are a number of subtleties in doing so, in particular with respect to distributed noise generation or interferences between HE and VC on one hand and DP on the other hand.

ACKNOWLEDGMENTS

The research leading to these results has received funding from the European Union's Preparatory Action on Defence Research (PADR-FDDT-OPEN-03-2019) under grant agreement GA 886854-2 (PRIVILEGE— PRIVacy and homomorphIc encryption for artificiaL intElliGence(under preparation)). This paper reflects only the authors' views and the Commission is not liable for any use that may be made of the information contained therein. The experiments were performed using the FactoryIA supercomputer, financially supported by the Ile-de-France Regional Council.

REFERENCES

[1] H. McMahan, E. Moore, D. Ramage, S. Hampson, and B. A. y Arcas, "Communication-efficient learning of deep networks from decentralized data," in *Proceedings of the 20th International Conference on Artificial Intelligence and Statistics*, 2017, pp. 1273–1282.

[2] (2018) GDPR 2018 reform of eu data protection rules. European Commission. [Online]. Available: https://ec.europa.eu/commission/sites/beta-political/files/data-protection-factsheet-changes_en.pdf

[3] Centers for Medicare & Medicaid Services, "The Health Insurance Portability and Accountability Act of 1996 (HIPAA)," Online at http://www.cms.hhs.gov/hipaa/, 1996.

[4] B. Hitaj, G. Ateniese, and F. Ferez-Cruz, "Deep models under the GAN: Information leakage from collaborative deep learning," 2017.

[5] L. Melis, C. Song, E. De Cristofaro, and V. Shmatikov, "Exploiting unintended feature leakage in collaborative learning," in *IEEE SP*, 2019, pp. 691–706.

[6] M. Fredrikson, S. Jha, and T. Ristenpart, "Model inversion attacks that exploit confidence information and basic countermeasures," in *ACM SIGSAC*, 2015, pp. 1322–1333.

[7] L. T. Phong, Y. Aono, T. Hayashi, L. Wang, and S. Moriai, "Privacy-preserving deep learning via additively homomorphic encryption," Cryptology ePrint Archive, Report 2017/715, 2017, https://eprint.iacr.org/2017/715.

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

[9] 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 *ACM SIGSAC*, 2017, p. 1175–1191.

[10] V. Mugunthan and A. Polychroniadou, "SMPAI: Secure multi-party computation for federated learning," 2019.

[11] G. Xu, H. Li, S. Liu, K. Yang, and X. Lin, "Verifynet: Secure and verifiable federated learning," *IEEE Transactions on Information Forensics and Security*, vol. 15, pp. 911–926, 2019.

[12] P. Kairouz *et al.*, "Advances and open problems in federated learning," 2019, arXiv preprint 1912.04977.

[13] P. Struck, L. Schabhüser, D. Demirel, and J. Buchmann, "Linearly homomorphic authenticated encryption with provable correctness and public verifiability," in *International Conference on Codes, Cryptology, and Information Security*. Springer, 2017, pp. 142–160.

[14] D. Catalano, A. Marcedone, and O. Puglisi, "Authenticating computation on groups: New homomorphic primitives and applications," in *ASIACRYPT 2014*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2014, pp. 193–212.

[15] R. Gilad-Bachrach, N. Dowlin, K. Laine, K. Lauter, M. Naehrig, and J. Wernsing, "CryptoNets: Applying neural networks to encrypted data with high throughput and accuracy," in *ICML*, 2016, pp. 201–210.

[16] A. Sanyal, M. Kusner, A. Gascón, and V. Kanade, "TAPAS: Tricks to accelerate (encrypted) prediction as a service," in *ICML*, 06 2018.

[17] E. Hesamifard, H. Takabi, and M. Ghasemi, "Deep neural networks classification over encrypted data," in *ACM CODASPY*, 2019, p. 97–108.

[18] F. Bergamaschi, S. Halevi, T. T. Halevi, and H. Hunt, "Homomorphic training of 30,000 logistic regression models," in *International Conference on Applied Cryptography and Network Security*. Springer, 2019, pp. 592–611.

[19] S. Carpov, N. Gama, M. Georgieva, and J. R. Troncoso-Pastoriza, "Privacy-preserving semi-parallel logistic regression training with fully homomorphic encryption," Cryptology ePrint Archive, Report 2019/101, 2019, https://eprint.iacr.org/2019/101.

[20] Q. Lou, B. Feng, G. C. Fox, and L. Jiang, "Glyph: Fast and accurately training deep neural networks on encrypted data," *arXiv preprint arXiv:1911.07101*, 2019.

[21] W. Zheng, R. A. Popa, J. E. Gonzalez, and I. Stoica, "Helen: Maliciously secure coopetitive learning for linear models," in *2019 IEEE SP)*. IEEE, 2019, pp. 724–738.

[22] J. Li and H. Huang, "Faster secure data mining via distributed homomorphic encryption," in *ACM SIGKDD*, 2020, pp. 2706–2714.

[23] S. Sav, A. Pyrgelis, J. R. Troncoso-Pastoriza, D. Froelicher, J.-P. Bossuat, J. S. Sousa, and J.-P. Hubaux, "POSEIDON: Privacy-preserving federated neural network learning," *arXiv preprint arXiv:2009.00349*, 2020.

[24] C. Zhang, S. Li, J. Xia, W. Wang, F. Yan, and Y. Liu, "Batchcrypt: Efficient homomorphic encryption for cross-silo federated learning," in *2020 {USENIX} Annual Technical Conference ({USENIX}{ATC} 20)*, 2020, pp. 493–506.

[25] Z. Ghodsi, T. Gu, and S. Garg, "Safetynets: Verifiable execution of deep neural networks on an untrusted cloud," in *Advances in Neural Information Processing Systems*, 2017, pp. 4672–4681.

[26] F. Tramer and D. Boneh, "Slalom: Fast, verifiable and private execution of neural networks in trusted hardware," *arXiv preprint arXiv:1806.03287*, 2018.

[27] H. B. McMahan, D. Ramage, K. Talwar, and L. Zhang, "Learning differentially private language models without losing accuracy," *arXiv preprint arXiv:1710.06963*, 2017.

[28] R. C. Geyer, T. Klein, and M. Nabi, "Differentially private federated learning: A client level perspective," *arXiv preprint arXiv:1712.07557*, 2017.

[29] B. Jayaraman, L. Wang, D. Evans, and Q. Gu, "Distributed learning without distress: Privacy-preserving empirical risk minimization," *NeurIPS*, vol. 31, pp. 6343–6354, 2018.

[30] N. Agarwal, A. T. Suresh, F. X. X. Yu, S. Kumar, and B. McMahan, "cpsgd: Communication-efficient and differentially-private distributed sgd," *NeurIPS*, vol. 31, pp. 7564–7575, 2018.

[31] R. Shokri, M. Stronati, C. Song, and V. Shmatikov, "Membership inference attacks against machine learning models," in *IEEE SP*. IEEE, 2017, pp. 3–18.

[32] A. Sablayrolles, M. Douze, C. Schmid, Y. Ollivier, and H. Jégou, "White-box vs black-box: Bayes optimal strategies for membership inference," in *ICML*, 2019, pp. 5558–5567.

[33] P. Paillier, "Public-key cryptosystems based on composite degree residuosity classes," in *International conference on the theory and applications of cryptographic techniques*. Springer, 1999, pp. 223–238.

[34] S. Caldas, S. K. Duddu, P. Wu, T. Li, J. Konečný, H. McMahan, V. Smith, and A. Talwalkar, "Leaf: A benchmark for federated settings," 2019, arXiv preprint 1812.01097.