Improving privacy of Federated Learning Generative Adversarial Networks using Intel SGX

Wouter Jehee
Supervisor(s): Kaitai Liang, Rui Wang
EEMCS, Delft University of Technology, The Netherlands

June 19, 2022

A Dissertation Submitted to EEMCS faculty Delft University of Technology,
In Partial Fulfilment of the Requirements
For the Bachelor of Computer Science and Engineering

Abstract

Federated learning (FL), although a major privacy improvement over centralized learning, is still vulnerable to privacy leaks. The research presented in this paper provides an analysis of the threats to FL Generative Adversarial Networks. Furthermore, an implementation is provided to better protect the data of the participants with Trusted Execution Environments (TEEs), using Intel Software Guard Extensions. Lastly, the viability of it's use in practice is evaluated and discussed. The results indicate that this approach protects the data, while not affecting the predicting capabilities of the model, with a noticeable but manageable impact on the training duration.

Keywords: Federated Learning, Privacy-preserving, Intel SGX, Trusted Execution Environments, Generative Adversarial Networks

1 Introduction

With machine learning becoming more widely used, more data is needed to train the models, this can come at the risk of violating the privacy of those from which the data is gathered. One way to circumvent this issue is using federated learning (FL) [1], which allows machine learning algorithms to train one model on the data of several devices, without those devices exchanging the data directly. This approach, contrary to centralized machine learning methods, limits the amount of possibly collected data and thus minimizes many privacy risks [2]. The research focuses specifically on Federated Learning using a Generative Adversarial Network (GAN), which are often used for the generation of fake data [3]. A GAN [4] is a machine learning algorithm using a generator, which tries to generate fake data, and a discriminator, which tries to differentiate the fake data generated by the generator from the real data samples. Even though FL already limits the direct leakage of private data, there are still ways for actors (i.e. the server or participating clients) to infer properties of the model and thus the data [5]. There are multiple proposed methods to counteract these types of attacks, one such study [3] focuses on the use of differential privacy in GANs to preserve the privacy of the clients. Other methods include the use of homomorphic encryption [6] and Secure Multi-party Computation (MPC) [7].

This research aims to analyze the use of Trusted Execution Environments (TEEs) [8], to improve the privacy of clients in a FL-GAN. TEEs provide a way to execute code on a remote machine without that machine being able to access the information being processed, this allows FL applications to run code privately on remote devices. There are several implementations of TEEs that currently exist, namely Arm's TrustZone, Sanctum for RISC-V and lastly, Intel's SGX [2], [9] which will be used for the purposes of this research. On it's own, the use of TEEs could remove the need for other privacy preserving techniques such as differential privacy or homomorphic encryption, possibly improving on the accuracy of the model or the performance respectfully. But it could also add to the idea of "Privacy in Depth" where multiple privacy-preserving techniques are applied as redundancies in case one or more (but not all) of them fail [2]. "It remains an open question how to partition federated learning functions across secure enclaves, cloud computing resources, and client devices" [2, p. 42]. Previous works [3], [5], [10] have proposed solutions to this problem, but GANs specifically have not been covered. This research serves as an addition to this subject of study in the aim to find a good solution to this problem, furthermore, this research analyzes the impact of relaxing some of the constraints on both the server and the clients.

Research Question

Formally, this paper aims to answer the question: how can Intel SGX be used to protect data in a Federated Learning Generative Adversarial Network? In order to do so, we aim to answer the following sub-questions.

- What is the attack surface in a FL-GAN (not using SGX) and what can malicious actors achieve?
- How can the data be protected using the secure enclaves implemented by SGX?
- What is the impact on the performance of the training algorithm?

In order to answer these questions, several research methods are used. First, a qualitative analysis of the actors in the form of a threat model is made to discover the risks that are present. Secondly, a possible solution with the goal to secure the data is described and discussed. Lastly, the impact on the run-time performance is empirically evaluated in order to verify the viability of the solution for use in production scenarios.

The structure of the paper is as follows. Section 2 explains the required theory and concepts. Section 3 gives a formal description of the problem. Section 4 covers the architecture and implementation of the system. Section 5 explains the setup of the experiment and discusses the results. Section 6 explains the relevant aspects of responsible research. Lastly, section 7 discusses the final conclusions and future research possibilities.

2 Preliminaries

This section covers the most important preliminary concepts to better understand the topic of the research.

2.1 Federated Learning (FL)

Federated learning is a machine learning method involving multiple remote devices with localized data that work together towards training a statistical model [1], [11]. Normally, all data could be collected on a central server and then trained, but this would allow the server to access all the data provided by the participating clients. The idea behind FL is that each client locally trains the model and only the model parameters are sent to the server (instead of the data). These parameters are then aggregated by the server using techniques such as FedAvg [12], AFL [13] or FedMA [14] to create a global model that is trained on the data of all clients.

2.2 Generative Adversarial Networks (GANs)

Generative Adversarial Networks are statistical models involving a pair of neural networks competing with each other [4]. This pair consists of a generator, which aims to generate fake data similar to the real data, and a discriminator which tries to distinguish the real data from the fake data generated by the generator.

2.3 Intel Software Guard Extensions (SGX)

Intel Software Guard Extensions provide developers a set of instructions to create so called secure enclaves in memory that are inaccessible by all other processes running on the system [10]. "The trusted hardware protects the data's confidentiality and integrity while the computation is being performed on it." [9, p. 1] Remote attestation allows external users to verify a proof that the party they are connected with are running in an enclave and that they are running the expected code [9], [10].

2.4 Attacks on FL systems

Two categories of attacks are considered in this paper, namely inference attacks and data poisoning attacks [15].

2.4.1 Inference attacks

Inference attacks try to leak information about clients or the global model that was not intended to be shared to the adversary. There are three main categories of inference attacks [5], [15], namely:

- Data Reconstruction Attack (DRA): tries to reconstruct the original input data
- Property Inference Attacks (PIA): tries to infer private properties in the input data
- Membership Inference Attacks (MIA): tries to infer the presence of certain instances of data in the training set.

2.4.2 Poisoning attacks

Poisoning attacks try to manipulate the training of the model in one of two ways. Either through random attacks, which aim to reduce the accuracy of the model or through targeted attacks which aim to influence the model to produce the desired output of the adversary [15].

3 Problem Description

In a FL environment, each client has it's own data which can be trained on, this data is considered to be private and therefore must not be shared in any way with the other participants. FL already mitigates the direct sharing of data as the training is done locally, but there are still several attacks (inference attacks) that can leak private information.

The system consists of an orchestration server that organizes the training and a set of clients which each have their own private data on which needs to be trained. Figure 1 shows how each of the participants interact with each other in the general FL setting. The clients and the server are assumed to all use the same algorithm and have the same training goal. Each client trains the model on the local data they possess before sending it to the orchestration server. The orchestration server is connected with each of the clients and receives the updated model and uses e.g. FedAvg [12] to combine the results into an aggregated model.

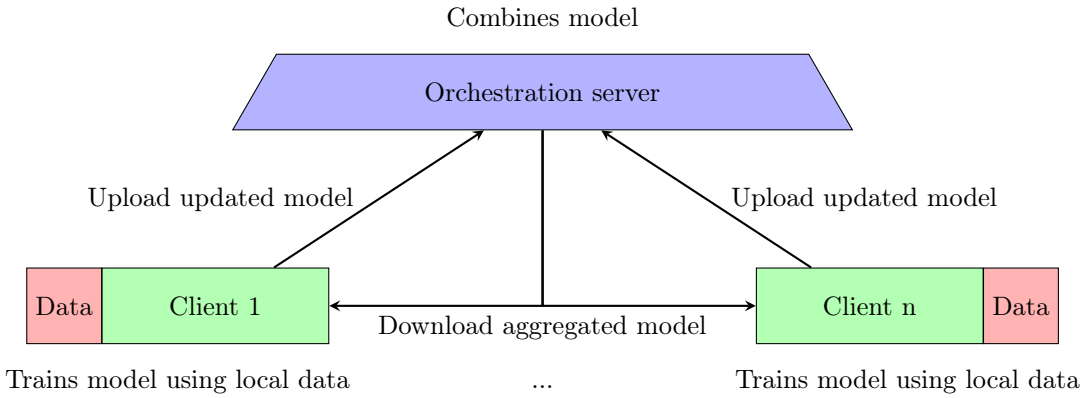

Figure 1: Overview of a generic Federated Learning architecture

Threat model

For the threat model we considered the two different roles of client and server and consider them one of two types of adversaries:

- Honest-but-curious: a fully cooperative participant in the network, but one that tries to gather as much data as possible from the legitimate messages.
- Malicious: a participant that can deviate from the standard protocol in order to achieve it's goals.

External actors, i.e. model consumers and eavesdroppers [16], are considered out of scope for this research. For the server side, only the honest-but-curious case is included in the evaluation, an honest-but-curious server tries to infer information about the model or the private data of the clients. The server has access to the intermediate updated models, but also to the individual contributions of each client, allowing it to infer properties about the private data of each client individually [16].

For clients we consider both a fully malicious client that tries to manipulate the model and an honest-but-curious client that tries to infer information. Clients can attempt to use the received global model updates to passively infer properties about the private data of the other clients. Secondly, a malicious client can purposely try to work against the training of the model. Lastly, a malicious client can also use active property inference, which updates the model in a specific way to infer certain properties. While the aim to leak private data [17], it can also alter the training of the model and affect the quality of the predictions.

4 Architecture of the system

The implementation [18] was done in Google's Go programming language [19] using the Gorgonia [20] library for implementing the GAN and using EGo [21] for SGX support. SGX provides the functionality for the three defining properties of a TEE, which consist of the following:

1. "Confidentiality: The state of the code's execution remains secret, unless the code explicitly publishes a message."[2, p. 41]
2. "Integrity: The code's execution cannot be affected, except by the code explicitly receiving an input."[2, p. 41]
3. "Measurement / Attestation: The TEE can prove to a remote party what code (binary) is executing and what its starting state was, defining the initial conditions for confidentiality and integrity."[2, p. 41]

The command-line tools provided by EGo allow for the compilation and signing of Go programs making them able to run in the SGX TEE, implicitly guaranteeing the first two properties. For the third property both the client and server use a quote provider that connects to the Provisioning Certificate Caching Service (PCCS) operated by Alibaba Cloud [22] in order to enable remote attestation as described above. The PCCS functions as the attestation service, which allows a program to verify the following proofs [23] that the code running in the enclave provides:

- The identity of the code / processor
- That it has not been tampered with
- That it is running on a genuine platform with Intel SGX enabled

The cryptographic keys used for the verification are generated after signing the compiled binary using the 'ego sign' command provided by EGo, creating a public-private key pair. The private key can then be used to sign the certificate and the public key can be used by the others to verify the certificate. Whenever the binary is ran inside an enclave the certificate will be made available by the PCCS such that other programs (i.e. the clients) can verify it.

The server serves an HTTPS endpoint that creates a websocket connection and the certificate is verified by the client using the PCCS to ensure that the expected code is running in the enclave of the server. Similarly the server verifies the client certificate to ensure no tampering from the client side. The websocket connection will be used for the training process that follows.

Initially the client uses the base model to perform the first round of local training. After each round of local training, the model weights and biases are encoded and the model is sent to the server. In future rounds the client receives the model, the model is decoded, the required amount of training steps are performed and the process is repeated, as can been seen in algorithm 1.

Algorithm 1 Client side implementation

```
procedure TRAINING ROUND
    epoch ← 0
    local_epochs ← 10 | 100                    ▷ Depending on parabola or iris data set
    total_epochs ← 100 | 10000                 ▷ Depending on parabola or iris data set
    model ← retrieve_model()
    while epoch <= total_epochs do
        epoch ← epoch + 1
        update_model(model)
        if epoch %local_epochs = 0 then
            send_model(encode(model))
            model ← decode(retrieve_model())   ▷ waits for server to send aggregated model
```

The server decodes the model and uses FedAvg to create the aggregated model, after which it encodes and sends the model to all connected clients as can be seen in algorithm 2.

Algorithm 2 Server side implementation

```
procedure RECEIVE MODEL(M)
    n ← n + 1                                  ▷ Keep track of number of models sent
    Add decode(m) to models                    ▷ m is the received model
    if n = total_clients then
        n ← 0
        model ← fedAvg(models)
        send_model(encode(model))
```

5 Results

Since the use of TEEs does not alter training algorithm or data in any way, it also does not affect the quality of the predictions of the model. Thus the experiments performed instead focus on the impact on the execution time of the algorithm. The execution time is reported using Go's built-in time library. All code was executed using Go version 1.16 and EGo version 0.5.1. All experiments were executed on a HP ZBook Studio G5 x360 with the Intel Core i7-8750H (hexa-core) Coffee Lake processor with 16GB DDR4 2666 MHz memory running on Ubuntu version 20.04.

Two different training goals were evaluated in the experiments, the total training time was tested using different numbers of clients and it was tested with and without SGX on both the server and client side. The first goal was to emulate the $y = x^2$ parabola function in 100 epochs of training while locally trained for 10 epochs, these results were averaged over 3 runs of the algorithm, with the size of the training set being 1024 samples. The results can be seen in figure 2 and the run-time overhead of using SGX can been seen in table 1.

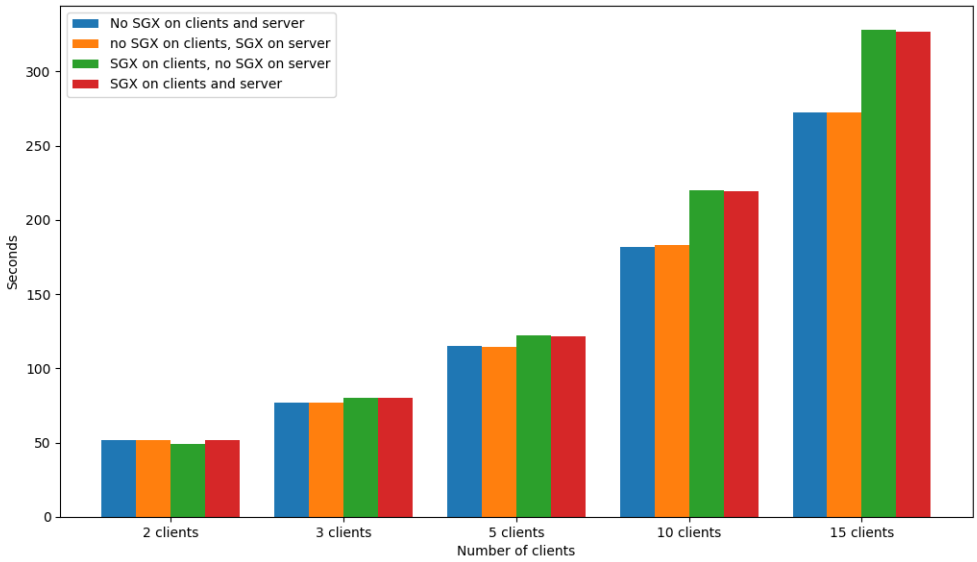

Figure 2: Performance of $y = x^2$ with and without SGX

	SGX server	SGX client	SGX both
2 clients	0.33%	-5.25%	-0.81%
3 clients	-0.03%	3.90%	3.85%
5 clients	-0.52%	6.23%	5.74%
10 clients	0.70%	20.62%	20.41%
15 clients	-0.07%	20.26%	19.77%

Table 1: Performance overhead using SGX with $y = x^2$

The second set of training was performed on the well known iris data set [24] in 10000 epochs of training, while being locally trained for 100 epochs, these results were averaged over 10 runs of the algorithm. The results can be seen in figure 3 and the run-time overhead of using SGX can been seen in table 2.

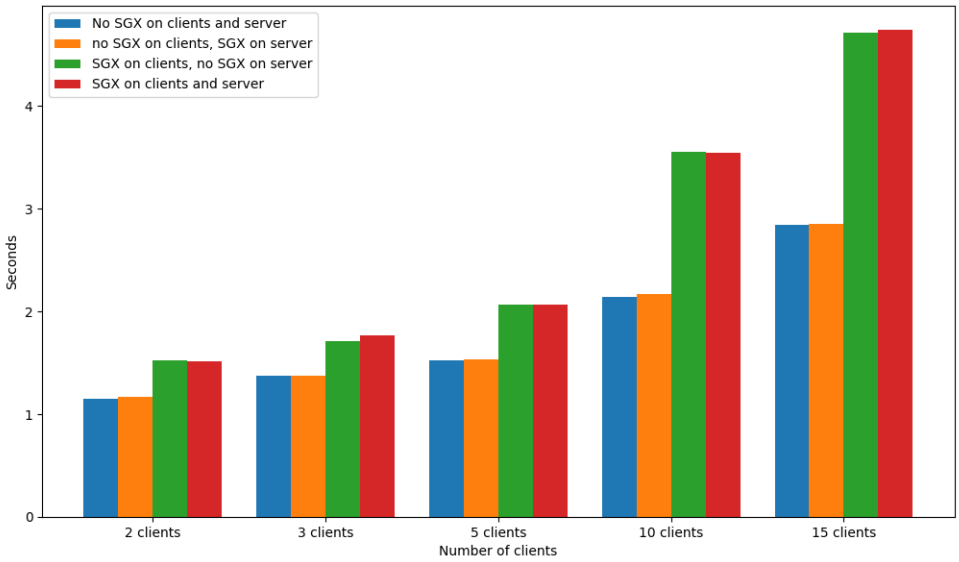

Figure 3: Performance on iris data set with and without SGX

	SGX server	SGX client	SGX both
2 clients	2.01%	32.46%	32.15%
3 clients	0.01%	24.57%	28.45%
5 clients	0.34%	35.47%	35.07%
10 clients	1.36%	65.76%	65.34%
15 clients	0.34%	65.71%	66.56%

Table 2: Performance overhead using SGX with the iris data set

The results from the first few experiments indicated a accelerating increase in execution time as the number of clients increased. The proportional increase could be caused by the memory limitations of SGX [5], requiring the operating system to page out memory, thus resulting in longer computation times. In order to investigate this, further experiments were performed with different sizes of training sets, as can be seen in table 3.

	1024 samples	2048 samples	4096 samples	8192 samples	16384 samples
Run 1	1.99%	8.07%	-5.46%	-4.93%	8.68%
Run 2	3.92%	4.86%	3.99%	2.03%	4.06%
Run 3	0.40%	-6.22%	-0.34%	0.26%	2.96%
Average	2.08%	1.98%	-0.68%	-0.98%	5.23%

Table 3: Performance overhead of SGX with different training set sizes

Discussion

Logically, using SGX introduces more computations for the CPU to perform, but some of the results show that using SGX on the server is faster than not using it. However, these differences are statistically insignificant as can be seen in tables 1 and 2, showing performance decreases of at most 1%. The results from both data sets indicate that the performance of using SGX on the server side is negligible. This is likely due to the small amount of actual computation being executed on the server side.

At first the results indicate that when the number of clients increase, the performance impact of using SGX also increases, as the computation time increases proportionally more compared to the clients not using SGX. We expected this trend to continue, yet the results seem to contradict this expectation. One can see in tables 1 and 2, that the number of clients increases from 10 to 15 but the computation time barely increases, while in the previous steps it did increase significantly.

The results in table 3 indicate that the computation time with SGX is higher with a larger numbers of samples, but only by a small amount, thus the size of the data does not influence the performance as much as expected. The performance impact on the iris data set is much higher as the number of clients increase than on the $y = x^2$ data set, while the iris data set has a lower amount of samples but a higher number of features per sample. Following from this, we estimate that the increase in computation time comes from the number of features instead of the number of samples.

6 Responsible Research

In order to make the research as reproducible as possible, all libraries used are open-source and the source code of the implementation as well as the scripts used to generate the results are publicly available [18]. Additionally, a detailed description of libraries, data and hardware used is provided such that the results can be verified by others. There are however a few things that cannot be guaranteed as the implementation relies on some external services, namely the Intel SGX attestation service and the Alibaba cloud PCCS used to receive the cached data from the Intel servers. Should the Intel Attestation servers be deprecated, this research will not be reproducible anymore, but under those circumstances Intel SGX would be deprecated in it's entirety. Should Alibaba cloud discontinue it's service, a different PCCS could be used as an alternative, this might alter the results, but the difference should not differ too much as the machine learning part of the algorithm is dominant in affecting the performance.

In order to maintain research integrity, none of the data was altered nor trimmed in order to produce "better" results. All data is presented as gathered during the experiments and anomalies in the data are explicitly discussed. Furthermore, all external information and tooling has been properly mentioned and documented in an effort to preserve authenticity. Lastly, there are no

conflicts of interest affecting the neutrality of the research nor was the study financed by any party.

7 Conclusions and Future Work

In this paper, a possible design for enabling FL-GAN using SGX was proposed to protect the data of clients and the training process from tampering based on the threat model. The performance impact of using SGX on both the server and client was empirically evaluated, indicating a negligible performance impact on the server side and a manageable impact on the client side. The use of SGX on the server side prevents the server from performing inference attacks while slightly impacting performance. Using SGX on the client side has more significant consequences regarding the performance but should be considered in privacy sensitive settings or situations prone to poisoning attacks in order to prevent attacks from clients. Those looking to implement SGX for machine learning do need to keep in mind the limited memory of SGX and adjust accordingly.

The implementation provided in this research continues to show that the use of TEEs to protect data in FL is a viable solution given the low performance overhead and because it does not impact the prediction capabilities of the model.

7.1 Evaluation

There are a few limitations to the provided results that are worth mentioning, they are the following:

- All computations where performed on a single machine, a more realistic scenario would consider different devices for clients and for the server.
- Due to the limited memory of SGX, only smaller data sets were used.
- Intel SGX does not support the use of GPU's, so training was only performed on the CPU.

7.2 Recommendations for future research

There are many possibilities for future research into this topic, a few proposals that could be considered include the following:

- Since SGX does not support the use of GPU's, future research could look into the performance difference between the use of SGX and a training algorithm that does use GPU acceleration for training.
- Current research has mainly focused on a central orchestration server, future research could look into using TEEs in a fully decentralized setting where each client can remotely attest each other client.

References

[1] T. Li, A. K. Sahu, A. Talwalkar, and V. Smith, "Federated learning: Challenges, methods, and future directions," *IEEE Signal Processing Magazine*, vol. 37, no. 3, pp. 50–60, 2020. DOI: 10.1109/MSP.2020.2975749.

[2] P. Kairouz, H. B. McMahan, B. Avent, *et al.*, "Advances and open problems in federated learning," *Foundations and Trends in Machine Learning*, vol. 14, 1-2 2021, ISSN: 19358245. DOI: 10.1561/2200000083.

[3] B. Xin, Y. Geng, T. Hu, *et al.*, "Federated synthetic data generation with differential privacy," *Neurocomputing*, vol. 468, 2022, ISSN: 18728286. DOI: 10.1016/j.neucom.2021.10.027.

[4] A. Creswell, T. White, V. Dumoulin, K. Arulkumaran, B. Sengupta, and A. A. Bharath, "Generative adversarial networks: An overview," *IEEE Signal Processing Magazine*, vol. 35, no. 1, pp. 53–65, 2018. DOI: 10.1109/MSP.2017.2765202.

[5] F. Mo, H. Haddadi, K. Katevas, E. Marin, D. Perino, and N. Kourtellis, "Ppfl: Privacy-preserving federated learning with trusted execution environments," in *Proceedings of the 19th Annual International Conference on Mobile Systems, Applications, and Services*, 2021, pp. 94–108.

[6] A. Acar, H. Aksu, A. S. Uluagac, and M. Conti, "A survey on homomorphic encryption schemes: Theory and implementation," *ACM Computing Surveys (Csur)*, vol. 51, no. 4, pp. 1–35, 2018.

[7] W. Du and M. J. Atallah, "Secure multi-party computation problems and their applications: A review and open problems," in *Proceedings of the 2001 workshop on New security paradigms*, 2001, pp. 13–22.

[8] M. Sabt, M. Achemlal, and A. Bouabdallah, "Trusted execution environment: What it is, and what it is not," in *2015 IEEE Trustcom/BigDataSE/ISPA*, IEEE, vol. 1, 2015, pp. 57–64.

[9] V. Costan and S. Devadas, "Intel sgx explained," *Cryptology ePrint Archive*, 2016.

[10] O. Ohrimenko, F. Schuster, C. Fournet, *et al.*, "Oblivious {multi-party} machine learning on trusted processors," in *25th USENIX Security Symposium (USENIX Security 16)*, 2016, pp. 619–636.

[11] T. Li, A. K. Sahu, A. Talwalkar, and V. Smith, "Federated learning: Challenges, methods, and future directions," *IEEE Signal Processing Magazine*, vol. 37, no. 3, pp. 50–60, 2020.

[12] B. 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*, A. Singh and J. Zhu, Eds., ser. Proceedings of Machine Learning Research, vol. 54, PMLR, 2017, pp. 1273–1282. [Online]. Available: https://proceedings.mlr.press/v54/mcmahan17a.html.

[13] M. Mohri, G. Sivek, and A. T. Suresh, "Agnostic federated learning," in *International Conference on Machine Learning*, PMLR, 2019, pp. 4615–4625.

[14] H. Wang, M. Yurochkin, Y. Sun, D. Papailiopoulos, and Y. Khazaeni, "Federated learning with matched averaging," *arXiv preprint arXiv:2002.06440*, 2020.

[15] L. Lyu, H. Yu, and Q. Yang, "Threats to federated learning: A survey," *arXiv preprint arXiv:2003.02133*, 2020.

[16] X. Yin, Y. Zhu, and J. Hu, "A comprehensive survey of privacy-preserving federated learning: A taxonomy, review, and future directions," *ACM Computing Surveys (CSUR)*, vol. 54, no. 6, pp. 1–36, 2021.

[17] L. Melis, C. Song, E. De Cristofaro, and V. Shmatikov, "Exploiting unintended feature leakage in collaborative learning," in *2019 IEEE Symposium on Security and Privacy (SP)*, IEEE, 2019, pp. 691–706.

[18] "Cse3000 intel sgx based fl-gan." (2022), [Online]. Available: `https://github.com/WJehee/CSE3000`.

[19] "Go." (2022), [Online]. Available: `https://go.dev/`.

[20] "Gorgonia." (2022), [Online]. Available: `https://github.com/gorgonia/gorgonia`.

[21] "Ego." (2022), [Online]. Available: `https://github.com/edgelesssys/ego`.

[22] "Remote attestation." (2022), [Online]. Available: `https://docs.edgeless.systems/ego/#/reference/attest`.

[23] "Provisioning secrets with remote attestation." (2022), [Online]. Available: `https://www.intel.com/content/www/us/en/developer/articles/code-sample/software-guard-extensions-remote-attestation-end-to-end-example.html`.

[24] R. A. Fisher, "The use of multiple measurements in taxonomic problems," *Annals of eugenics*, vol. 7, no. 2, pp. 179–188, 1936.