Multi-party Computation for Privacy and Security in Machine Learning: a Practical Review

Multi-party Computation for Privacy and Security in Machine Learning: a practical review

Alessandro Bellini
Mathema s.r.l.
Florence, Italy
Email:abel@mathema.com

Emanuele Bellini
LOGOS-RI
Florence, Italy
Email: emanuele.bellini@logos-ri.eu
ORCID:0000-0002-7878-8710

Massimo Bertini
Mathema s.r.l.
Florence, Italy
Email: stefano.cuomo@mathema.com

Doaa Almhaithawi
Department of Control and Computer Engineering
Politecnico di Torino
Torino, Italy
Email: Doaa.Almhaithawi@polito.it

Stefano Cuomo
Mathema s.r.l.
Florence, Italy
Email: stafano.cuomo@mathema.com

***Abstract*—Machine Learning, particularly Deep Learning, is transforming society in any of its fundamental domains - healthcare, culture, finance, transportation, education, just to mention a few. However Machine Learning suffers from serious weaknesses in privacy and security due to the large amount of data (datasets for training and parameters in trained models) and the probabilistic approximation inherent in any ML function. Multi-Party Computation (MPC) is a family of techniques and tactic with a sound scientific and operative base that can be applied to mitigate some relevant weaknesses of ML. In particular, privacy in training may be assured by MPC with federated learning techniques (these may be considered particular interpretations and implementation of a general MPC method) and also security in training and inference may be enforced by continuous model testing using MPC is a technique that allows multiple parties to evaluate a machine learning model on their private data without revealing it to each other. This brief paper is a practical and essential review on how to use MPC to mitigate privacy and security issues in ML.**

I. Introduction

Machine Learning (ML), particularly Deep Learning, has an impressive and an unprecedented success in a wide range of applications encompassing any crucial domain of human activity such as healthcare [1], , finance [2], retail [3], transportation [4], education [5], entertainment [6], manufacturing [7]. However ML poses a serious security menace along the whole life-cycle from dataset, through training, to inference. Substantially, ML consists of computing the parameters (weights) of an approximate function by an optimization process on a dataset. So evident potential surface of attack is represented by:

- The approximate nature of ML functions can be intrinsically fooled (it is an approximation);
- Data are pervasive (dataset and parameters) posing both privacy and security issues;
- The optimization process if only slightly altered may compromise the generation of the ML approximate function.

A. Types of Attack on Machine Learning

ML is subjected to several attacks in the training and in the inference phases. Among the most relevant are:

- Poisoning: Poisoning attacks involve injecting malicious data into the training dataset in order to cause the ML model to make incorrect predictions. This can be done by an attacker who has access to the training data or by an attacker who is able to influence the data that is collected by the ML system. This attack is struck against the dataset to be use for training.
- Adversarial examples: Adversarial examples are inputs that have been manipulated to cause an ML model to make incorrect predictions. They can be crafted to mislead the model, by adding small perturbations to the input that are imperceptible to humans but cause the model to fail. This is an inference phase attack.
- Model inversion: Model inversion attacks involve inferring sensitive information about the training data based on the predictions made by an ML model. This is an attack executed in inference phase but the object of the attack is to steal the model parameters computed in the training phase.

  Poisoning may be avoided inhibiting access to the dataset and/or encrypting the dataset, for instance using HE. The other two attacks – adversarial examples and model inversion – are very hard to be prevented or totally neutralized but Multipart Computation (MPC) may mitigate them quite effectively.

A perfect technique or a set of techniques to solve all the privacy and security issues inherent to ML is not currently available but Multi-Party Computing (MPC) seems to be a general method that pragmatically may mitigate almost any ML weaknesses. This paper presents how this mitigation can be attained applying MPC to the training and inference of ML model. In fact, Multi-party computation (MPC) can be

considered a promising approach to secure ML by enabling multiple parties to collaborate on a computation without revealing their private data to each other. MPC allows parties to jointly compute a function on their private inputs while preserving the confidentiality of those inputs. This technique is particularly useful in scenarios where sensitive data is involved. In this context, this technology can enable a range of new and valuable services without requiring parties to trust each other with sensitive data. With the increasing use of ML and the growing need for secure data sharing, MPC is poised to become an essential tool in the development of secure ML based applications.

II. Multi-Party Computing

Multi-Party Computing (MPC) is a method of securely computing a function or sharing data among multiple parties without revealing any information about the inputs of the other parties [8][9] .

This is typically achieved through the use of cryptographic techniques such as secret sharing [10] and secure multiparty computation protocols. MPC is typically useful in scenarios where multiple parties have private data and want to compute a function on that data without revealing the private data to any other party. It can be used for a wide range of applications, such as secure voting systems and secure financial transactions.

Multi-Party Computing (MPC) can be used also in Machine Learning to enable multiple parties to collaboratively train a machine learning model without revealing their private data to each other.

A. MPC process

The basic process of MPC can be broken down into a few key steps:

- Input preparation: Each party starts by preparing their private input, which they want to use as input to the function. This input may be encrypted using a secure encryption scheme such as Paillier (partially homorphic encryption scheme) [11] or ElGamal (asymmetric key encryption algorithm) [12].
- Secret sharing: The parties then share their inputs using a technique called secret sharing[10]. Secret sharing allows a value to be split into multiple shares, such that a certain threshold of shares is required to reconstruct the value. This allows the parties to share their inputs without revealing them to the other parties [13].
- Function evaluation: The parties then jointly evaluate the function on their shared inputs. This is done by first reconstructing the inputs from the shares, and then applying the function. The function evaluation is performed in a way that preserves the privacy of the inputs, by using techniques such as garbled circuits or Yao's protocol.
- Output reconstruction: The parties then reconstruct the output of the function from the shares. This is done by combining the shares in a way that allows the output to be reconstructed without revealing anything beyond the output.
- Decryption: The final output of the function is then decrypted by each party to reveal the final result

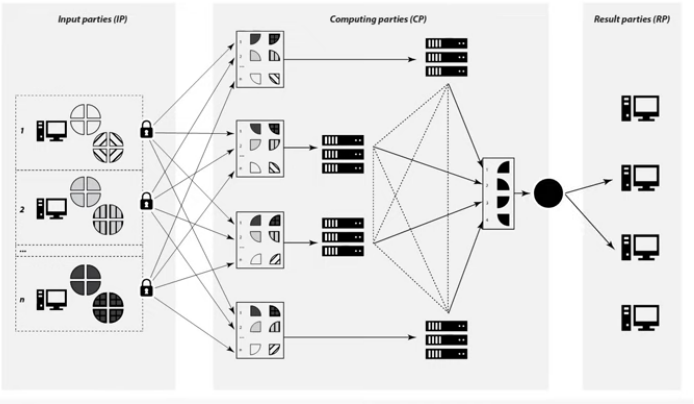

Fig. 1. General Architecture of the MPC method

In general, an MPC is composed of (Fig.1):

- Input Parties (IP) that own their private data and they do not trust other IPs;
- Computing Parties (CP) that run their private computations and interact with other CPs to compute a function and
- Result Parties that are the recipients of the results computed by CPs and that, in general, are different by IPs.

B. The MPC Protocols

A MPC protocol is the set of messages and data exchanged among CPs to collaboratively compute a function. MPC protocols may be very complex and of very different types [14]. Some of the most common ones include:

- Secret Sharing: this protocol allows multiple parties to share a secret value without revealing it to any individual party. It can be used for MPC tasks such as secure function evaluation and secure voting [13].
- Secure Multi-Party Computation (sMPC): this protocol allows multiple parties to jointly compute a function over their private inputs without revealing them to each other. It can be used for tasks such as privacy-preserving data analysis, secure multi-party decision making and secure federated learning. In particular, secure federate learning locally trains a model on their individual data, then client send the parameters(weights) of their local model to the server using an encryption scheme which still allows the server to perform computations on the encrypted data; the server will compute a weighted average of all the encrypted weights received from clients, but cannot discover the original weights for any client. Even if MPC reveals information only about the final result, in some scenarios, knowing just the result can be enough to reveal information about the inputs. One solution is to applying differential privacy on top of MPC. In the solution which combines MPC and distributed differential privacy [15][16] [16], the noisy weights sent to the server,

are also encrypted such that the server can only calculate the result, and cannot infer anything about even the noisy weights of any particular user.
- Secure Multi-Party Protocols (sMPP): this protocol allows multiple parties to jointly compute a function over their private inputs with the help of a third party. It can be used for tasks such as secure data aggregation, secure multiparty key generation and secure multiparty machine learning.

According to [17] there are two practical limitations: (i) standard protocols require many rounds of communication over private channels between the parties, which makes them inadequate for low-end devices and unreliable networks. (ii) current approaches require a per-party communication that increases linearly in the circuit size (that increases at least linearly in the number of parties). Hence, this quadratic factor quickly becomes a bottleneck for large numbers of parties. Homomorphic encryption (HE) techniques are well-known for reducing the communication complexity of MPC[18], especially in their various threshold and multi-key variants that we generally refer to as multiparty-HE. Thus, the aodpion of HE allowed the implementaton of two more protocols:

- Secure Multi-Party Computation with Homomorphic Encryption (HE-MPC): this protocol uses Homomorphic Encryption (HE) to allow computations to be performed on ciphertext, which encrypts data, rather than on plaintext. It can be used for tasks such as privacy-preserving data analysis, secure machine learning, and secure multiparty decision making.
- Fully Homomorphic Encryption (FHE): this protocol allows arbitrary computations to be performed on ciphertext without the need for decryption. It is considered the gold standard for privacy-preserving computation, but currently, it is less practical for real-world use due to its high computational cost.

III. Homomorphic encryption

Homomorphic encryption allows computations to be performed on encrypted data without the need to decrypt it first. This means that sensitive data can be processed without being exposed, which mitigates the risk of data leakage or theft. By using homomorphic encryption, parties can perform computations on encrypted data while preserving its confidentiality, which is particularly useful in scenarios where data privacy is a critical concern.

Homomorphic encryption is a form of encryption that allows computations to be performed on ciphertext, which are encrypted data, and produce an encrypted result that can be decrypted to match the result of the same computations performed on the plaintext, which is unencrypted data. This is achieved through the use of mathematical operations that are homomorphic with respect to the encryption scheme being used.

One common example of homomorphic encryption is using a scheme based on the RSA algorithm. The RSA algorithm uses the properties of large prime numbers to encrypt and decrypt data, and it has a homomorphic property with respect to multiplication. This means that if two ciphertexts are multiplied together, the result can be decrypted to reveal the product of the corresponding plaintexts.

Another example of homomorphic encryption is the Paillier cryptosystem [19], based on the difficulty of factoring composite integers. The Paillier cryptosystem has a homomorphic property with respect to addition, meaning that if two ciphertexts are added together, the result can be decrypted to reveal the sum of the corresponding plaintexts.

In general, there are two main types of homomorphic encryption:

- partially homomorphic
- fully homomorphic encryption.

Partially homomorphic encryption allows the execution of a limited set of operations (e.g. only addition or only multiplication) on the ciphertext and get a correct result after decrypting it.

Fully homomorphic encryption [20] allows the execution of any computation on the ciphertext and get a correct result after decrypting it. Fully homomorphic encryption is more powerful than partially homomorphic encryption but also more computationally expensive [21].

A. Fully homomorphic encryption

Fully homomorphic encryption (FHE) is a type of encryption that allows for arbitrary computations to be performed on ciphertext, producing an encrypted result that can be decrypted to match the result of the same computations performed on the plaintext. This is a very powerful property, as it allows for sensitive data to be processed and analyzed without the need to decrypt it first. One example of a fully homomorphic encryption scheme is the fully homomorphic encryption (FHE) scheme based on lattices, like the Braverman-Gentry-Vaikuntanathan (BGV) scheme. This scheme uses the properties of lattices, which are mathematical structures composed of discrete points, to encrypt and decrypt data. The encryption algorithm encodes the plaintext into a lattice point, and the decryption algorithm extracts the plaintext from a lattice point. The FHE scheme allows for any computation to be performed on the encrypted data, as long as the computation can be represented as a sequence of linear operations on the lattice points. Another example of FHE is the scheme based on circuit, like the Fully Homomorphic Encryption (FHE) scheme of Gentry-Halevi-Smart (GHS)[22]. This scheme allows for any circuit to be evaluated on the ciphertext, which is represented as a set of gates on the plaintext, and it is decrypted to the result of the circuit.

IV. Attack mitigation with MPC

In general Multi-Party Computation (MPC) and Homomorphic Encryption (HE) [23] is a combination that can be used together to protect the privacy of data in machine learning tasks. Some of the ways MPC and HE can be used together in machine learning include:

- Secure Multi-Party Computation with Homomorphic Encryption (HE-MPC): This approach allows multiple parties to jointly compute a machine learning model over their private data without revealing their data to each other. The data is encrypted using an HE scheme, and the computations are performed on the ciphertext using an MPC protocol.
- Secure Federated Learning with HE: In this approach, multiple parties encrypt their data using an HE scheme and then train a machine learning model locally on the encrypted data. The parties then share the trained models with each other without revealing their data.
- Secure Distributed Training with HE: This approach allows multiple parties to jointly train a machine learning model on their private data without revealing the data to each other, by using secure multi-party computation protocols and HE.
- Secure Data Aggregation with HE: This approach allows multiple parties to share their encrypted data with a trusted third party, who then aggregates the data and trains a machine learning model on the aggregate data, without revealing the individual data of the parties.
- Model inversion may also benefit from a differential privacy technique, where the parameter values optimized in the training process (gradient descent) are obfuscated with Laplacian or Gaussian noise. This is a sort of soft encryption, weaker but less intrusive than HE, that is effective at providing a certain degree of data protection. In order to test the effectiveness of differential privacy tests are conducted on the trained model by independent multi-party actors with their "secret recipes" of techniques and use cases to test if it is possible to infer sensitive information about the training data based.
- MPC may also be used to make the model more robust to the adversarial examples attack. In fact robust model testing using MPC is a technique that allows multiple parties to evaluate a machine learning model on their private data without revealing it to each other. This helps to ensure that the model is accurate and secure, and reduces the risk of data breaches or malicious actors manipulating the model's outputs.

V. Secure MPC Implementation

There are several librarises and tools available for implementing multi-party computation to secure artificial intelligence. These platforms use a range of cryptographic techniques, including secret-sharing, HE, and garbled circuits, to ensure that computations are performed securely and without revealing sensitive data. Even if there are sweveral tools available online, we avoided to include ptrojects that are officially disconinuetd as SCALE-MAMBA or

- HElib [24]: HElib is an open-source library for HE, that provides efficient and optimized implementations of various homomorphic encryption schemes, such as BGV, Gentry's and CKKS. It is written in C++ and provides a high-level API for users to write their own protocols. HElib is widely used in both academia and industry and has been used in several research projects and real-world applications.
- SEAL[25]: Simple Encrypted Arithmetic Library, is an open-source library for HE, that provides efficient and optimized implementations of various HE schemes, such as BFV. It is written in C++ and provides a high-level API for users to write their own protocols. SEAL is widely used in industry and in research projects.
- PALISADE[26]: is another open-source library for HE that provides implementations of various homomorphic encryption schemes, such as BFV, CKKS and BGV. It is written in C++ and provides a high-level API for users to write their own protocols.
- Sharemind [27]: Sharemind uses a secret-sharing scheme to distribute data across multiple parties, ensuring that no single party has access to the full data set. Sharemind also provides a programming language for implementing computations in a secure and efficient manner.
- SecureML: SecureML is an MPC platform developed by Google. It uses a combination of HE and secret-sharing to enable secure collaborative training of machine learning models. SecureML is built on top of TensorFlow, a popular machine learning framework, and provides an API that allows developers to easily implement secure computations.
- HE-Transformer [28]: HE-Transformer is an MPC platform developed by Intel. It uses HE to enable secure computation of machine learning models. HE-Transformer is designed to be compatible with popular machine learning frameworks such as TensorFlow and PyTorch, making it easy to integrate into existing workflows.
- ABY [29]: ABY is an MPC platform developed by the University of California, Berkeley. It uses a combination of secret-sharing and garbled circuits to enable secure computation of arbitrary functions. ABY is designed to be highly scalable, making it suitable for large-scale computations.
- PySyft[30]: This is a library that provides a framework for secure and private machine learning. It implements a wide range of MPC protocols, including secret sharing and homomorphic encryption, and it can be used in conjunction with popular machine learning libraries such as PyTorch and TensorFlow.
- SecureNN[31]: This is a library that provides an implementation of secure multi-party computation (MPC) for neural networks. It uses secure aggregation and secure function evaluation techniques to preserve the privacy of the inputs and intermediate computations.
- PyMPC: This is a library that provides an implementation of various MPC protocols, including secret sharing and homomorphic encryption. It is built on top of the PyTorch library and allows for the evaluation of circuits on shared inputs.
- MPyC[32]: This is a library that provides a wide range of MPC protocols, including secret sharing, homomorphic

encryption, and secure function evaluation. It is written in Python and it is based on the GMP library for big number arithmetic. MPyC utilizes secret sharing-based protocols,including Shamir's secret sharing, to enable secure computations among multiple parties.
- EMP-Toolkit[33]: The Efficient Multi-Party Computation (EMP) Toolkit is a library that implements secure multi-party computation protocols. It supports both arithmetic and Boolean circuits and provides efficient implementations of various cryptographic protocols. The EMP-Toolkit is designed for high-performance MPC applications and provides flexible APIs for defining secure computations.

VI. Discussions

From the analysis in I it is possible to highlight some relevant trends. In particular it is worth to notice that C++ is mostly used for cryptography. The reasons why modules for Cryptography as HElib and SEAL are often written in C++ resepct to the modules for MPC that are written in Python, can be here summarised:

- Performance: C++ is known for its efficiency and low-level control, making it suitable for implementing computationally intensive tasks like cryptography. C++ allows developers to fine-tune memory management and optimize code execution, resulting in high-performance cryptographic operations sidorov2022comprehensive.
- Portability: C++ is a widely supported language across various platforms and architectures. Cryptographic libraries need to be compatible with different operating systems and hardware configurations. By using C++, developers can achieve better portability and reach a broader user base.
- Existing Ecosystem: C++ has a rich ecosystem of libraries, tools, and frameworks that can aid in cryptographic development. Many cryptographic algorithms and protocols have existing C++ implementations, allowing developers to leverage and integrate them into their modules easily.
- Low-level Control: Cryptographic algorithms often require low-level control over memory, data structures, and hardware features. C++ provides this level of control through features like manual memory management, direct hardware access, and inline assembly code, which can be crucial for implementing secure and efficient cryptographic operations.
- Legacy Considerations: Some cryptographic libraries have been in development for many years and may have started in C or C++. Over time, they have evolved and maintained compatibility with existing codebases, making it more practical to continue using C++ for consistency and backward compatibility.

Another major trend identified is related to the open source approach. Software modules implementing multi-party computation (MPC) are often open source. Apart the typical reasons that inspire open source initiatives as fostering collaboration, innovation and community creation to ensure ongoing maintenance and improvement of the software over time, other specific reasons can be identified as:

- Transparency: MPC protocols are used to enable secure computations among multiple parties while preserving privacy. Open source software allows interested parties to review the source code and understand how the cryptographic protocols are implemented. This transparency helps build trust by allowing the community to verify the correctness and security of the implementation.
- Peer Review: Open source projects encourage collaboration and peer review from a wide range of experts. By making the source code accessible, developers and researchers worldwide can contribute improvements, identify vulnerabilities, and propose fixes. The collective effort of the open source community can lead to higher-quality code and stronger security.
- Adoption and Standardization: By providing open source implementations, MPC protocols can gain wider adoption and become de facto standards. This allows different parties to use compatible software, promoting interoperability and enabling secure computations across different systems and platforms.

Moreover, all the modules implement APIs with security and privacy features with secret sharing schemes to protect the privacy of individual inputs by distributing them across multiple parties.

VII. Conclusion

Making ML models that are completely secure is difficult, as they are designed to process and store large amounts of data and make predictions based on that data.

This means that there is always some risk of data breaches or malicious actors manipulating the model's inputs or outputs.

However, it is possible to reduce these risks by implementing best practices in data protection, such as encryption, secure storage, access controls and MPC.

MPC enforces the use of secure training procedures and frequent testing that can help ensure that the model behaves as intended and is resistant to tampering or attacks.

MPC is still a relatively new area of research, and its full potential for improving machine learning has yet to be realized. However, it has the potential to address some of the key challenges facing ML, such as data privacy and security, and to enable new applications that were previously impossible due to data sharing and privacy concerns.

A. Limitations and Challenges

While MPC offers new possibilities for securing AI, it also has some limitations and challenges. MPC can be computationally intensive and may require significant resources to implement. Additionally, the performance of MPC can be affected by the number of parties involved and the complexity of the function being computed. Finally, ensuring the security

TABLE I
LIBRARIES EVALUATION

Tool	Purpose	Domain	Programming Language	API Available	Licence
HElib	Homomorphic Encryption	Cryptography	C++	Yes	Open-Source
SEAL	Homomorphic Encryption	Cryptography	C++	Yes	Open-Source
PALISADE	Homomorphic Encryption	Cryptography	C++	Yes	Open-Source
HE-Transformer	Homomorphic Encryption	Cryptography	Python	Yes	Open-Source
Sharemind	Secure Multi-Party Computation	Privacy-Preserving Computation	Java, C++	No	Proprietary (Cybernetica)
SecureML	Machine Learning with Privacy	Privacy-Preserving Computation Machine Learning	Python	Yes	Open-Source
ABY	Secure Multi-Party Computation	Privacy-Preserving Computation	C++	Yes	Open-Source
PySyft	Privacy-Preserving Machine Learning, Federated Learning	Privacy-Preserving Computation Machine Learning	Python	Yes	Open-Source
SecureNN	Privacy-Preserving Machine Learning, Neural Networks	Privacy-Preserving Computation, Machine Learning	Python	Yes	Open-Source
PyMPC	Secure Multi-Party Computation	Privacy-Preserving Computation	Python	Yes	Open-Source
MPyC	Secure Multi-Party Computation	Privacy-Preserving Computation	Python	Yes	Open-Source
EMP-Toolkit	Secure Multi-Party Computation	Privacy-Preserving Computation	Python	Yes	Open-Source

of the MPC protocol itself is essential, as any vulnerabilities could compromise the privacy of the data being processed.

ACKNOWLEDGMENT

This research has received funding from the European Union's Horizon Europe EXTRACT (GA 101093110)

REFERENCES

[1] L. X. L. Q. Yang S, Zhu F and Z. P, "Intelligent health care: Applications of deep learning in computational medicine," *Frontiers in Genetics*, 2021.

[2] M. Dixon, *Machine Learning in Finance From Theory to Practice*. springer, 2020.

[3] V. B. Edouard de Mézerac, Emmanuel Ladoux and A. Ilin, "Machine learning for retail. using machine learning algorithms to predict sales on promotions," 2023.

[4] R. Ma, Z. Zhang, Y. Dong, and Y. Pan, "Deep learning based vehicle detection and classification methodology using strain sensors under bridge deck," *Sensors*, vol. 20, no. 18, 2020.

[5] I. T. Sanusi, S. S. Oyelere, H. Vartiainen, J. Suhonen, and M. Tukiainen, "A systematic review of teaching and learning machine learning in k-12 education," *Education and Information Technologies*, 2022.

[6] J. Wilson, "Artificial intelligence, machine learning, and the future of entertainment," *Forbes*, 2022.

[7] T. Wuest, D. Weimer, C. Irgens, and K.-D. Thoben, "Machine learning in manufacturing: advantages, challenges, and applications," *Production & Manufacturing Research*, vol. 4, no. 1, pp. 23–45, 2016.

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

[9] D. Evans, V. Kolesnikov, and M. Rosulek. 2018.

[10] A. Shamir, "How to share a secret," *Commun. ACM*, vol. 22, p. 612–613, nov 1979.

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

[12] T. Elgamal, "A public key cryptosystem and a signature scheme based on discrete logarithms," *IEEE Transactions on Information Theory*, vol. 31, no. 4, pp. 469–472, 1985.

[13] M. Ito, A. Saito, and T. Nishizeki, "Secret sharing scheme realizing general access structure," *Electronics and Communications in Japan (Part III: Fundamental Electronic Science)*, vol. 72, no. 9, pp. 56–64, 1989.

[14] T. Dugan and X. Zou, "A survey of secure multiparty computation protocols for privacy preserving genetic tests," in *2016 IEEE First International Conference on Connected Health: Applications, Systems and Engineering Technologies (CHASE)*, pp. 173–182, 2016.

[15] E. Shi, T.-H. H. Chan, E. Rieffel, and D. Song, "Distributed private data analysis: Lower bounds and practical constructions," *ACM Trans. Algorithms*, vol. 13, dec 2017.

[16] D. Byrd and A. Polychroniadou, "Differentially private secure multi-party computation for federated learning in financial applications," in *Proceedings of the First ACM International Conference on AI in Finance*, ICAIF '20, (New York, NY, USA), Association for Computing Machinery, 2021.

[17] V. Bindschaedler, S. Rane, A. E. Brito, V. Rao, and E. Uzun, "Achieving differential privacy in secure multiparty data aggregation protocols on star networks," in *Proceedings of the Seventh ACM on Conference on Data and Application Security and Privacy*, CODASPY '17, (New York, NY, USA), p. 115–125, Association for Computing Machinery, 2017.

[18] R. Cramer, I. Damgård, and J. B. Nielsen, "Multiparty computation from threshold homomorphic encryption," in *Advances in Cryptology — EUROCRYPT 2001* (B. Pfitzmann, ed.), (Berlin, Heidelberg), pp. 280–300, Springer Berlin Heidelberg, 2001.

[19] P. Paillier, "Public-key cryptosystems based on composite degree residuosity classes," in *Advances in Cryptology — EUROCRYPT '99* (J. Stern, ed.), (Berlin, Heidelberg), pp. 223–238, Springer Berlin Heidelberg, 1999.

[20] C. Marcolla, V. Sucasas, M. Manzano, R. Bassoli, F. H. P. Fitzek, and N. Aaraj, "Survey on fully homomorphic encryption, theory, and applications," *Proceedings of the IEEE*, vol. 110, no. 10, pp. 1572–1609, 2022.

[21] S. Marrone, A. Tortora, E. Bellini, A. Maione, and M. Raimondo, "Development of a testbed for fully homomorphic encryption solutions," in *2021 IEEE International Conference on Cyber Security and Resilience (CSR)*, pp. 206–211, 2021.

[22] I. Damgård, V. Pastro, N. Smart, and S. Zakarias, "Multiparty computation from somewhat homomorphic encryption," in *Advances in Cryptology – CRYPTO 2012* (R. Safavi-Naini and R. Canetti, eds.), (Berlin, Heidelberg), pp. 643–662, Springer Berlin Heidelberg, 2012.

[23] I. Damgård, V. Pastro, N. Smart, e.-N. R. Zakarias, Sarah", and R. Canetti, "Multiparty computation from somewhat homomorphic encryption," in *Advances in Cryptology – CRYPTO 2012*, (Berlin, Heidelberg), pp. 643–662, Springer Berlin Heidelberg, 2012.

[24] "Helib." https://github.com/homenc/HElib. Accessed: 2023-02-01.

[25] "Seal." https://github.com/microsoft/SEAL. Accessed: 2023-02-01.

[26] "Palisade." https://palisade-crypto.org/. Accessed: 2023-02-01.

[27] "Sharemind." https://sharemind.cyber.ee/. Accessed: 2023-02-01.

[28] "he-transformer." https://github.com/IntelAI/he-transformer. Accessed: 2023-02-01.

[29] "Aby." https://github.com/encryptogroup/ABY. Accessed: 2023-02-01.

[30] "Pysyft." https://github.com/OpenMined/PySyft. Accessed: 2023-02-01.

[31] "Securenn." https://github.com/snwagh/securenn-public. Accessed: 2023-02-01.

[32] "Mpyc." https://github.com/lschoe/mpyc. Accessed: 2023-02-01.

[33] "Emp-toolkit." https://github.com/emp-toolkit. Accessed: 2023-02-01.