ORNL/TM-2020/1760

Considerations for using Privacy Preserving Machine Learning Techniques for Safeguards

Nathan Martindale
Scott L. Stewart
Mark Adams
Greg Westphal

December 1, 2020

OAK RIDGE NATIONAL LABORATORY
MANAGED BY UT-BATTELLE FOR THE US DEPARTMENT OF ENERGY

ORNL/TM-2020/1760

Nuclear Nonproliferation Division

CONSIDERATIONS FOR USING PRIVACY PRESERVING MACHINE LEARNING TECHNIQUES FOR SAFEGUARDS

Nathan Martindale
Scott L. Stewart
Mark Adams
Greg Westphal

Date Published: December 1, 2020

Prepared by
OAK RIDGE NATIONAL LABORATORY
Oak Ridge, TN 37831-6283
managed by
UT-Battelle, LLC
for the
US DEPARTMENT OF ENERGY
under contract DE-AC05-00OR22725

CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACRONYMS

FHE	fully homomorphic encryption
HE	homomorphic encryption
IAEA	International Atomic Energy Agency
LHE	leveled homomorphic encryption
PPML	privacy-preserving machine learning
RLWE	ring learning with errors
SHE	somewhat homomorphic encryption
SMPC	secure multiparty computation
ZKP	zero-knowledge proof

1. EXECUTIVE SUMMARY

In international nuclear safeguards, the International Atomic Energy Agency (IAEA) is tasked with inspecting and verifying nuclear facilities and their activities. Data analytics and machine learning to support inspections require large amounts of data that nuclear facility operators may consider proprietary or sensitive, so the IAEA may not have full access. Allowing computation over private data without compromising its security therefore has value for safeguards inspections and analysis.

Privacy-preserving machine learning (PPML) consists of security-focused techniques that allow data analytics and machine learning algorithms to run on sensitive data without revealing it. This includes ideas like homomorphic encryption (HE), secure multiparty computation (SMPC), and secure enclaves. HE allows algorithms and mathematical operations to be conducted directly on the encrypted data instead of first decrypting it. With SMPC, multiple entities collaboratively compute over distributed data such that no party is able to directly view any others' original data. Secure enclaves allow computation to take place in a separate and heavily blocked-off section of a CPU.

Techniques like these allow for several potential use cases in which the security of data is essential. With SMPC, machine learning models can be trained over the input data from multiple entities, resulting in a model that all users can benefit from without leaking the input data from any particular entity. With SMPC or a zero-knowledge proof (ZKP), an algorithm returning some single answer or truth value can be run on someone else's data without ever needing to see that data, potentially allowing for verification or proof of some underlying question. HE can allow for outsourcing computation on data to a hostile or untrusted environment.

Although most of the research in this field resides within the health and financial domains, tools from PPML may have similar applications in nuclear safeguards. Allowing the IAEA to compute over proprietary information, such as process models and raw sensor data using PPML techniques, provides the baseline for running complex analytics without needing direct unencrypted access to the underlying data, maintaining its privacy.

Important limitations to consider for these techniques include the efficiency and level of security required. The security of HE and SMPC come at the cost of speed—the significant amount of overhead means that algorithms implemented in these protocols and encryption schemes are slower than when run on plaintext. Additionally, several important parameters determine what techniques or protocols are used based on the security requirements. SMPC protocols may need to be selected for resistance against a party that attempts to deviate from the protocol to distort the result or gain access to additional information, and a protocol secure against these attacks may further increase the overhead of the algorithm.

2. INTRODUCTION

In recent years, the International Atomic Energy Agency (IAEA) has shifted toward information-driven inspections [1, 2] for supporting its role in safeguards. The IAEA's long-term research and development plan emphasizes better usage and security of data via points 8 and 9 [3]. The ability to utilize sensitive information from nuclear facility operators in analytics for verification without needing direct access to the raw unencrypted data would support these objectives.

PPML is a field of study about the protection of data used either in training a machine-learning model or that is used to make a secure inference with a trained machine learning model. Data privacy has become increasingly important, and this capability has applications in many domains, including the medical field [4], where data might be protected under HIPAA, or the financial field [5], where data may be considered business secrets. Moreover, this has potential in nuclear safeguards, where facility operator data may be highly sensitive but simultaneously valuable to IAEA inspections. Several techniques support maintaining model and data privacy, and these can broadly be categorized into the areas of homomorphic encryption (HE), secure multiparty computation (SMPC), and secure enclaves.

Homomorphic encryption refers to encryption schemes that can apply certain operations to ciphertexts that, once decrypted, yield the same results as if those operations had been applied to the plaintext messages [6]. For example, encrypting two numbers, 2 and 3, and adding the ciphertexts should result in 5 when decrypted. The base operations usually discussed under HE are addition and multiplication. Many of the operations involved in machine learning (ML) algorithms such as neural networks are either made up of or can be sufficiently approximated with only addition and multiplication. A simple example of how this could be used involves two parties: (1) a server wishing to keep a neural network model private and (2) one or more clients wishing to use this model without revealing their data or predictions to anyone else. This is achieved by encrypting both the model parameters as well as the data with an HE scheme. The model operations are then carried out solely on the respective ciphertexts, and the resulting prediction is decryptable by the party with the encrypted data. This process was demonstrated with CryptoNets [7].

Secure multiparty computation refers to protocols that allow parties to evaluate a publicly known function on each other's private data without revealing that data. The classic example of this in the original proposition for SMPC is the "millionaire's problem." In this scenario, two millionaires wish to determine who is richer without revealing how much money they have [8]. Two core techniques generally used to resolve this problem: garbled circuits and secret sharing. Garbled circuits, which are used for secure two party computation and were initially proposed by Yao [9], encrypts a Boolean circuit and sends it to another party to determine the output with similarly encrypted inputs from both parties. In additive secret sharing, generally involving three or more parties, a number is split into a "share" for each party so that the sum of all of the shares equals the original number, but no individual party has enough information to reconstruct it. Two other works look at the application of SMPC techniques in the nuclear safeguards space [10, 11].

Secure enclaves refer to special hardware that provides protected or isolated regions of memory and computation that other programs—and even the operating system—do not have access to [12]. These can be used to set up trusted execution environments or allow for certain security guarantees even on compromised hosts. Protocols designed for these enclaves can run PPML algorithms such that data uploaded by different parties is encrypted, and cannot be seen outside of the enclave.

3. BACKGROUND

3.1 HOMOMORPHIC ENCRYPTION

Distinguishing the different levels of homomorphic encryption is important. Although schemes that are partially homomorphic have existed for a long time, supporting either addition or multiplication, the first fully homomorphic encryption (FHE) scheme, supporting arbitrary depth, or arbitrarily many multiplication and addition operations was proposed by Gentry [13]. HE schemes introduce random noise into the result of each operation, and when chained in a long enough series of operations, can produce an incorrect answer. Gentry's solution involved "bootstrapping," which is a technique to reset the noise level. Although this operation is expensive, the noise management allows arithmetic circuits of arbitrary depth. In practice, less computationally intensive approaches are used more often. A leveled homomorphic encryption (LHE) scheme, such as used in CryptoNets [7], allows arithmetic circuits up to an arbitrary specified depth or level, where the level must be known beforehand. This level is generally based on the number of multiplications needed because that is the more expensive operation. Another approach is to use a somewhat homomorphic encryption (SHE) scheme, where only some subset of possible arithmetic circuits are feasible. This is used, for example, in the SPDZ protcol defined in [14], referencing the BV SHE scheme [15], allowing circuits with many additions and a single multiplication operation.

One example of how some HE schemes work are those based on polynomial rings. Encryption is a mapping from $R_t \rightarrow R_q \times R_q$, where R_t is the polynomial ring encoding of the message, meaning the message integer is encoded into the coefficients of a large polynomial, and the cipher text space is the Cartesian product of R_q polynomial rings. Notably, the cipher text is much larger than the original message. Because a cipher text is a vector of two ring polynomials, $\vec{c} = (c_1, c_2)$, addition and multiplication operations between cipher texts are carried out based on these cipher text polynomial rings. This example shows addition [6]:

$$\vec{c_1} + \vec{c_2} = ([c_{11} + c_{21}]_q, [c_{12} + c_{22}]_q)$$

Note, HE is generally asymmetric, with a separate public key for encryption and private key for decryption. The security of schemes using this is based on the ring learning with errors (RLWE) problem [6, 16].

Because it supports addition and multiplication, an FHE scheme can compute arbitrary polynomials. However, HE schemes can also compute Boolean circuits by using a binary message space, where addition is equivalent to a logical XOR gate and multiplication is equivalent to a logical AND gate [6].

Although many works in this space tend to use a combination of HE and secure multiparty computation, a few solely utilize HE. CryptoNets [7], as discussed before, is a framework for running trained networks on data encrypted with an HE scheme. Specifically, CryptoNets rely on an LHE scheme called YASHE [17]. This framework assumes a model that was trained offline and unencrypted. Training an encrypted network is possible, but it is at least an order of magnitude slower. Even the encrypted prediction process alone is expensive and time-consuming. A single application of the network took 250 seconds. However, their approach allows batching inputs, and in their evaluation on MNIST it was capable of making approximately 59,000 predictions per hour. This approach was improved in a recent work with low latency, or LoLa, CryptoNets [18]. LoLa Cryptonets achieve a 10-fold speed increase through encoding networks on a per-layer basis rather than a per-node basis. Each layer then can potentially use a different representation more suited to the computations carried out in that layer. Higher overall efficiency is then achieved by switching between representations throughout the network layers.

Other ML algorithms besides neural networks have been implemented via HE. Confidential versions of two binary classifiers, linear means and Fisher's linear discriminant were implemented in [19], as well as a protocol to run them.

Naive Bayes, decision trees, and AdaBoost ensembles are implemented with HE in [20]. They design a set of composable building blocks for the algorithms, such as comparison and argmax, which are chainable through allowing conversion between encryption schemes. They use three different additively homomorphic schemes, meaning they are only capable of addition operations. These are the Paillier cryptosystem [21], the quadratic residuosity cryptosystem [22], and a leveled homomorphic encryption scheme based on the HELib implementation. As with the previous HE works mentioned above, an emphasis is placed on privacy-preserving classification, or keeping both input and output prediction data secret.

3.1.1 LIMITATIONS

Homomorphic encryption has several noteworthy limitations. First is that the size of the allowable message is limited by the message space, which depends on the encryption scheme [6]. This means that for an integer message, that integer may be bounded. However, a work-around is to use the Chinese remainder theorem, which can either allow a larger integer to be represented by multiple smaller ones, or can allow multiple "batched" values to be represented in a single cipher text, which is how CryptoNets batch prediction inputs. Another potential issue is the expanded size of the cipher text. One example is that given an integer scheme that uses more than 4,000 coefficients, a 1 MB set of plaintext data can take upward of 16 GB after encryption [6], although this will of course depend on the encryption scheme used. One of the most prevalent issues in relation to using HE for ML is the much greater computational cost. A simple multiplication operation on cipher texts can take significantly longer than if it were carried out on the plaintext messages. This contributes to the longer processing time of CryptoNets and is the reason why training with encrypted data is harder and more compute intensive. This problem is in part solved with some of the secure multiparty computation protocols discussed below in that they limit the depth of HE circuits needed. As already discussed, the operations allowed, only addition and multiplication, and depth of allowed operations due to cumulative noise are concerns. Finally, an important limitation is that these encryption schemes cannot operate on floating point data—all floating point numbers need to be encoded into integers, such as by scaling to some fixed precision [7].

There is an ongoing effort to standardize homomorphic encryption [23]. This standards document describes the security of existing homomorphic encryption schemes and makes parameter recommendations.

3.2 SECURE MULTIPARTY COMPUTATION

3.2.1 GARBLED CIRCUITS AND OBLIVIOUS TRANSFER

As mentioned before, Yao's garbled circuits [9] provide a protocol for two parties to securely compute a publicly known function over their privately held data. This function must be a Boolean function consisting of AND and XOR gates [24]. One party, the "garbler," creates a random label for each possible value, a 0 or 1, on each wire in the circuit. The truth table outputs for each gate are replaced with these values by the garbler, and then each row output is encrypted via encryption keys made up of the two corresponding inputs for that row. The table rows are then shuffled and sent to the other party to evaluate along with that party's private input (hidden by the value labels.) The evaluator then needs to obtain their private input, also replaced with the value labels determined by the garbler, and so these labels are received from the garbler via a technique known as "oblivious transfer" [25]. In 1-out-of-2 oblivious transfer, a sender offers two hidden messages to a receiver, the receiver picks and encrypts a number representing which message they want to read, and passes this back to the sender. The sender creates two keys based on the encrypted selection, and encrypts the messages with these keys. The sender sends the encrypted messages, and the receiver's decryption only produces a correct value for the message they selected. The result is that the receiver cannot see the value of the message he did not select, and the sender cannot see which value the receiver selected. A more in-depth explanation and proof of security of garbled circuits can be found in [26].

Input 1	Input 2	Output	Encrypted Output
0 **abc**	0 **lmn**	0 **xyz**	$\text{ENC}_{\textbf{abc}}(ENC_{\textbf{lmn}}(\textbf{xyz}))$
0 **abc**	1 **qrs**	1 **jkl**	$\text{ENC}_{\textbf{abc}}(ENC_{\textbf{qrs}}(\textbf{jkl}))$
1 **efg**	0 **lmn**	1 **jkl**	$\text{ENC}_{\textbf{efg}}(ENC_{\textbf{lmn}}(\textbf{jkl}))$
1 **efg**	1 **qrs**	0 **xyz**	$\text{ENC}_{\textbf{efg}}(ENC_{\textbf{qrs}}(\textbf{xyz}))$

Table 1. Garbled table example for a XOR gate.

To give a more concrete visualization, take the following example along with Table 1. Two parties, Alice and Bob, want to evaluate a Boolean circuit containing a XOR gate. We list the procedure in the following steps:

1. Alice, the garbler, generates different sets of random labels to represent the possible values on each of the three wires: the two inputs and the output. In the table, the labels Alice generated for the first input are **abc** for 0 and **efg** for 1, for the second input **lmn** for 0 and **qrs** for 1, and for the output **xyz** for 0 and **jkl** for 1. These labels and the values associated with them are known only to the garbler, Alice.
2. Each of the output labels are doubly encrypted by Alice, using that row's two input labels as the keys (shown in the Encrypted Output column of Table 1).
3. Alice shuffles the rows of this encrypted column, and only this column (now known as the garbled table) is sent to Bob, the evaluator.
4. For this example, assume that the respective private data is 0 for Alice and 1 for Bob. Bob, as the evaluator, must determine the gate output, and so needs keys or labels associated with both his and Alice's data. Alice sends him the label for her data, **abc**, and Bob does not know if this represents a 1 or a 0.
5. Bob then needs the label for his own data, which he receives from Alice via oblivious transfer—Bob sees **qrs**, but he does not know that the other label is **lmn**, and Alice does not know which label he received.
6. Bob tests the two input keys (**abc**, **qrs**) against the encrypted table until he successfully decrypts an output label, **jkl**. As before, Bob does not know whether this label represents a 0 or 1.
7. A Boolean circuit in a nontrivial application presumably contains more than a single XOR gate, so this output label and any further inputs needed from either party are processed similarly through all other gates in the circuit (which Alice would have similarly garbled, using output labels of a previous gate as input labels for the next where needed,) repeating steps 1–6, until a final circuit output label is obtained by Bob.
8. This final output label is sent to Alice who knows the associated value, which is finally communicated back to Bob. Only this final output can be known by either party as Alice never sees which intermediate output labels are reached, and Bob does not know what the labels represent and cannot decrypt any remaining rows of the garbled tables (as he does not know the other possible labels.)

3.2.2 SECRET SHARING

Secret sharing is a method of splitting up a piece of data into "secret shares," such that recombining them in some way reconstructs the original piece of data. Secret shares are distributed among different parties, and no individual party should be capable of inferring information about the original data or the other shares

without the cooperation of all parties. One example of this frequently used in SMPC protocols is additive secret sharing [24, 27]. In additive secret sharing, a number is split into random numbers in an integer ring such that the sum of those numbers modulo the ring is equivalent to the number. Some mathematical operations can be run on these distributed shares, such as addition and multiplication.

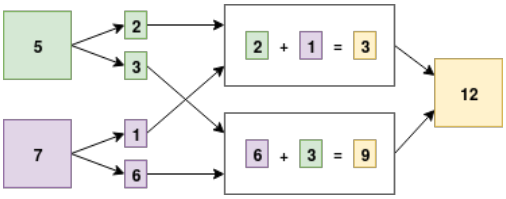

Figure 1. Additive secret sharing addition example.

A simplified example of running a mathematical operation (addition) on two pieces of data that are secret shared is shown in Figure 1. Note, secret sharing generally requires three parties to be secure, although two party variants have been used [24].

3.2.3 PROTOCOLS

Over the past few years, the number of secure multiparty computation protocols specifically for supporting ML and deep learning has experienced explosive growth. Many of these protocols are hybrid protocols, some using HE for certain operations and switching back and forth between using garbled circuits and secret sharing.

One of the first major SMPC protocols was SPDZ [14]. A major drawback of HE schemes is the computational overhead involved, and this work proposed an efficient way of using SHE for multiparty computation. This is done by splitting computation into an online and offline phase, with the offline phase used solely to generate supporting material for online multiplication operations. This material is made up of randomized "Beaver triplets" [28], a collection of three numbers defined by $<a>$, $<b>$, $<c>$ such that $c = ab$, where the syntax $<a>$ represents a collection of secret shares that recombine to create a. These numbers are encrypted with the BV SHE scheme, and because only a single multiplication step is needed, the overall depth is low and thus quick to run. All values are additively secret shared in this protocol, as is the private decryption key, meaning no individual can decrypt the data without the collaboration of all parties.

SecureML [27] is a protcol similar to SPDZ, but one of the first more oriented toward ML. It still relies on an offline phase to generate triplets for online multiplication, and because it is only intended for two party computation, garbled circuits are used for part of the algorithms. Notably they switch between different mechanisms: arithmetic, Boolean, and Yao. The work proposes a new activation function that is a cheaper approximation of a logistic and is much more efficient to implement than previous approximations, needing only small garbled circuits. It uses oblivious transfer, but minimizes the amount of overall communications needed. Similarly, they implement a cheaper softmax approximation.

The ABY [29] protocol is an acronym for arithmetic, Boolean, and Yao (garbled circuits), as their framework efficiently switches between these three different protocols, similarly to SecureML, but for three-party computation. Neither ABY nor DeepSecure [30] use HE, with DeepSecure relying exclusively on garbled circuits. DeepSecure specifically optimizes some of the garbled circuit steps needed by doing some of the work during preprocessing.

Chameleon [24] is an extension of ABY that switches between additive secret sharing and garbled circuits to support both linear and nonlinear functions. Gazelle [31] implements a protocol that switches between using HE encryption schemes and garbled circuits. SecureNN [32], one of the most recent protocols, claims the

highest efficiency of the SMPC protocols for neural networks, achieved by eliminating the need for garbled circuits and oblivious transfer, translation protocols, and reducing the overall amount of communication overhead.

3.3 ZERO-KNOWLEDGE PROOFS

Zero-knowledge proofs are an important concept explicitly tied into many secure multiparty computation protocols. At its core, a zero-knowledge proof allows for one party (the prover) to assert some statement such that some other party (the verifier) is convinced of the assertion's truth without needing any information other than the statement itself revealed [33, 34, 35].

A zero-knowledge proof has three behaviors or properties: (1) for true statements, an honest verifier will be convinced of its truth (completeness); (2) for false statements, no deviation from the protocol by the prover will convince the verifier that it is true within some tolerance probability (soundness); and finally, the verifier does not learn anything beyond whether the statement is true or false (zero-knowledge).

Zero-knowledge proofs can be considered a subset or special case of protocols [34]. More importantly, they can be used to create a malicious-secure SMPC protocol [33]. SPDZ is an example of this, implementing zero-knowledge proofs to ensure that the ciphertexts used in the beaver triplets are correct and can be decrypted [14]. Other, older works have even established the ability to turn arbitrary semihonest secure protocols into malicious-secure protocols with zero-knowledge proofs by requiring a proof on every message [36, 33].

3.4 MODEL SECURITY

Although techniques such as homomorphic encryption and secure multiparty computation help protect the processes of training and inference themselves, there are various kinds of attacks that can be used against trained ML models, even if the only access an adversary has is an inference API end point to a trained model [37]. The three main types of attacks are causitive, evasion, and exploratory. Causitive attacks pertain to poisoning the training process, such as the data. Evasion attacks try to find inputs that cause an incorrect output from a trained model. Finally, exploratory attacks try to gather data about trained models. PPML tries to keep data and trained models private, and so exploratory attacks are the most relevant type to explain.

Subcategories of exploratory attacks include model inversion, model extraction, and membership inference attacks. Model inversion attacks use outputs to infer broader features or characterizations about the inputs that were used during training. Membership inference attacks [38] can be used to distinguish whether a particular input was used to train a model or not. These were demonstrated even in a "black box" setting, where an adversary only has access to prediction outputs of the model. With only this API-like access, a generic technique capable of determining training set membership was created [38]. Model extraction is an attack that attempts to recover or replicate the parameters and weights of a private trained model [39]. Many attacks in this area are based on ML as a service systems that provide more information beyond just the output prediction, such as confidence vectors, but purely black box attacks against SVM's trained and protected with SMPC protocols have been demonstrated [37].

In PPML, one type of countermeasure in particular has received a lot of attention: differential privacy [40]. Differential privacy is about maintaining high statistical accuracy across a set of input data and not revealing information about the individuals within that data. This is directly to address model membership inference—with high differential privacy, it should not be feasible to determine if a model trained with a given input or not. This has been implemented into deep learning in practice through a differentially private stochastic gradient descent algorithm (DP-SGD) [41, 42]. DP-SGD is a loss function, an altered version of SGD such that after the gradient is computed, a small amount of Gaussian noise is added. This noise is intended to

prevent the network from "memorizing" any individual training instances it sees. [42] demonstrates this not only provides protection against a black box adversary, but it also protects against adversaries with direct access to the model parameters.

Note, however, that even with differential privacy employed, it is still possible for exploratory attacks to reveal and leak information about a model's training data [43]. Whether the type of data that can be leaked from these attacks is a threat or not is problem specific and a consideration to be taken into account based on the configuration of the threat model. A malicious server hosting a homomorphically encrypted model cannot gain information about that model, but a malicious entity who has been given access to the model endpoint for use may be able to gain information about the original training data.

3.5 SECURE ENCLAVES

Secure enclaves are a partially hardware-based approach to allowing private computation in untrusted or unsecure environments. An enclave is a trusted execution environment where both memory and computation are isolated from the rest of the system, including privileged processes such as the operating system. This is achieved by encrypting all memory that the enclave uses. Another important component is hardware and software verification, which is done via a process called remote attestation. Remote attestation is a way of creating a certificate, signature, or hash of the hardware and code that is running within the enclave, which a client can use to verify both the enclave and its contents [12]. There are several existing implementations of secure enclaves, namely Intel SGX (Software Guard eXtensions), ARM TrustZone, and Sanctum [44].

Despite the encryption of enclave memory, multiple side and controlled-channel attacks are known to work against Intel's SGX [45], including page faults, cache timing, address bus monitoring, and processor monitoring. Spectre in particular was demonstrated to work against SGX [46]. Data obliviousness is one ideal property of algorithms discussed in [12] that can partially mitigate data leakage. The resulting sequence of memory and disk accesses from a data oblivious algorithm does not depend on the data that the algorithm is acting on. Although data obliviousness helps mitigate some side-channel attacks regarding memory traces, it does not necessarily solve timing or related attacks [12].

Using a secure enclave can be significantly faster than either homomorphic encryption or secure multiparty computation, but it still has its own set of performance penalties and overhead [47], caused for example by a frequent need to continually encrypt and decrypt memory. Various works have proposed partial solutions to this for implementing efficient ML algorithms. The Slalom framework [47] allows for portions of neural network operations to be outsourced to a faster untrusted processor, such as a GPU. Myelin [48] uses small modular libraries that fit entirely in enclave memory. Chiron [49] allows parallel computation between multiple enclaves.

4. COMMON USE CASES

4.1 OUTSOURCING COMPUTATION WITH HOMOMORPHIC ENCRYPTION

Figure 2. Outsourcing computation with HE.

The simplest use case for HE is to support outsourcing computation over private data, as described in [6]. For example, a model owner wants to keep a neural network model private, and a client would like to use the neural network model without revealing their data or analysis results to anyone else. Using HE, the client could ensure their data is protected by encrypting it before sending it to the model owner. The model owner would then operate on the encrypted data using their model to generate an encrypted result. The encrypted result would then be sent back to the client for decryption and review. In this scenario, the client's information is protected because they are the only one with the key to encrypt or decrypt the data and results. The parameters of the model owner's model would also be protected because they could keep the model on servers that they own. Microsoft Research demonstrated this process with CryptoNets [7].

We diagram this scenario in Figure 2. There are two parties, a client (blue) and a server (purple) to which the computation is outsourced. The client has some set of private data, and the server has a model or algorithm to run. This scenario could fit multiple situations, where either the client uploads that model or algorithm, or if some other party or the server itself offers the private algorithm/model as a type of service (MLaaS). The client wants both input and output to remain private.

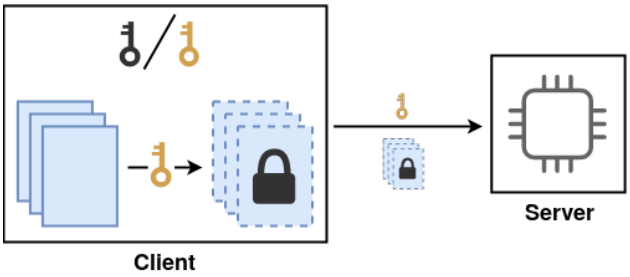

Figure 3. Client encrypts data with public key and sends to server.

The client first encrypts their local data and uploads it to the server, as shown in Figure 3. Most HE schemes are asymmetric, so the client generates a public/private key pair and encrypts all local data with the public key. The client then sends both the encrypted data as well as the public encryption key to the server.

In the next step, shown in Figure 4, the server processes this data by encrypting any algorithm or model parameters/weights as needed and then running the encrypted data through the now-encrypted model. As all operations are run on the ciphertexts, the output results are likewise encrypted.

Finally, in Figure 5, the server sends the encrypted output back to the client. The client can then use their private key to decrypt this data to see the final plaintext results. Throughout this use case, the server has never seen any plaintext data, and the client would not directly observe any model parameters.

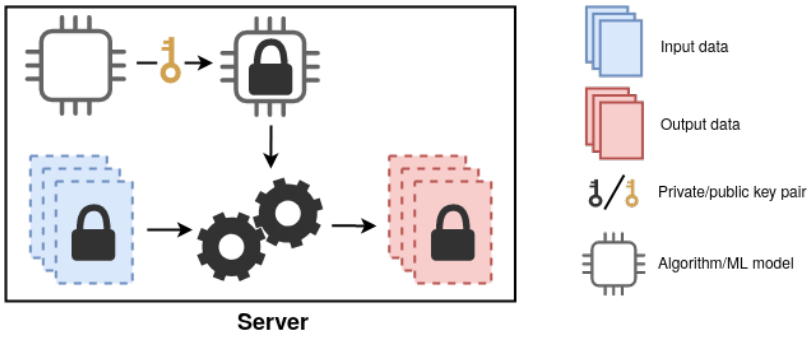

Figure 4. Server computes model output over encrypted input.

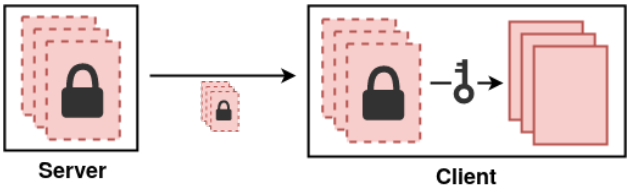

Figure 5. Server sends encrypted output to client for decryption with private key.

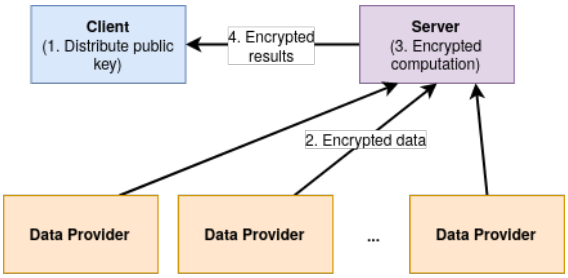

Figure 6. Multiple private data providers with homomorphic encryption.

4.2 USING HOMOMORPHIC ENCRYPTION WITH MULTIPLE DATA PROVIDERS

Our second use case is based on discussions in [6, 19], which expands on the previous example by allowing for multiple data providers to contribute private data. There are three different entities: a client who wishes to collect output for a set of data providers, a server containing a model or algorithm, and a collection of data providers. A potential application of this setup would be a research institute that wants to train a model based on the private data from many hospitals, represented by the data providers.

In the first step, shown in Figure 7, the client generates a public/private key pair and publishes (or directly sends) the public encryption key to the data providers and the server. This allows the data providers to each homomorphically encrypt their data using the same key without the ability to decrypt anything.

Once the data providers receive the public key from the client, they encrypt their data and send it to the server (Figure 8). The server also encrypts any necessary model or algorithm parameters with the same public key.

The server runs the encrypted model over the aggregated encrypted inputs, shown in Figure 9, and the output is still homomorphically encrypted. The server then sends the results to the client, who decrypts them with their private decryption key. Throughout this use case, neither the server nor any data provider views anything in plaintext beyond their original privately held data. As noted in [6], there may need to be a

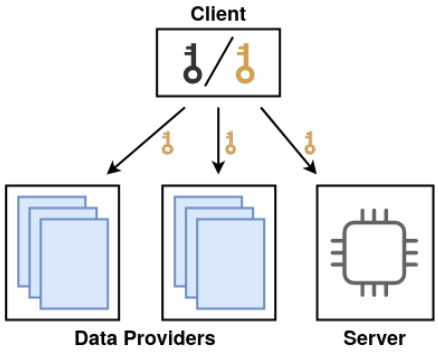

Figure 7. Client shares public key for encryption with data providers and the server.

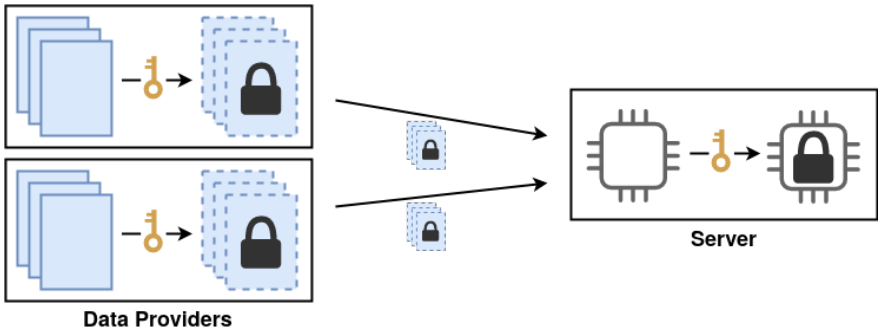

Figure 8. Data providers encrypt local data and send to server.

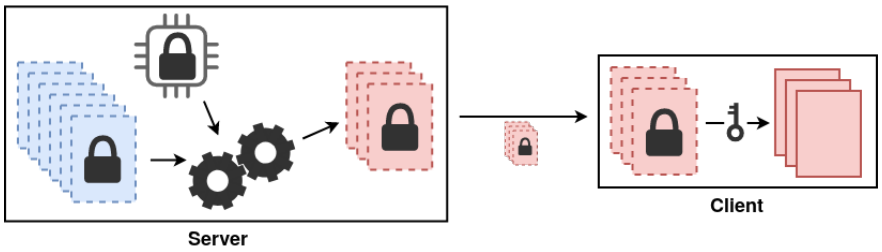

Figure 9. Server computes model output over encrypted input and sends encrypted output to client for decryption with private key.

contractual assumption between the client and server that the server does not send any of the encrypted data providers' data to the client, as the client is capable of decrypting it with the private key.

4.3 SERVING TRAINED MODELS WITH SECURE MULTIPARTY COMPUTATION

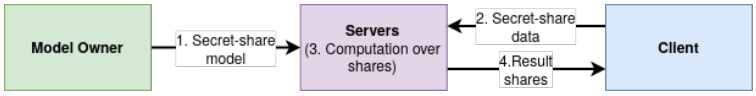

Figure 10. Serving a trained model with SMPC.

The next example uses SMPC in a similar manner to [32]. This example follows the same basic flow as the previous HE outsourcing use case but with a SMPC protocol instead. Three different entities are involved: a model owner who is offering their model as a service but wishes to keep it private, a client with private

data wishing to run the model on that data, and a collection of servers to use for the computation. For the purposes of this example, the model has already been trained offline.

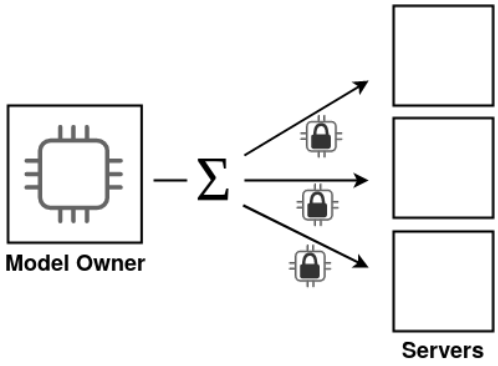

Figure 11. Model owner secret shares the model with the servers.

In the first step, shown in Figure 11, the model owner secret shares their model, dividing it into shares and sending one to each server. This keeps the model private because neither the client nor any individual server will be able to infer the original model without all of the shares.

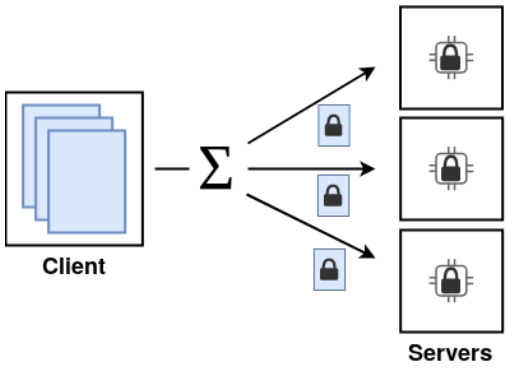

Figure 12. Client (data owner) secret shares the data with the servers.

In the next step, Figure 12, the client secret shares its private data with the servers.

Next, in Figure 13, the servers run the distributed model over the distributed shares, and the results remain secret shared across all of the servers. The client receives the result shares and reconstructs them to obtain the plaintext results. Throughout this use case, the client never accesses the model, and the model owner never accesses the input or output data.

Note, depending on the security model of the protocol, these servers are assumed to be noncolluding. For instance, the servers could be distributed among the model owner and client. Different protocols support different configurations, as some are capable of two party computation and others require three or more.

4.4 ONLINE TRAINING WITH SECURE MULTIPARTY COMPUTATION

The next example is drawn from [32], presenting a scenario building off of the previous one. In it, the model is trained online and can be trained on private data from multiple different data providers. There are three entities in a similar setup as the second HE example in Section 4.2: a collection of data providers who wish to contribute data but have it remain private, a collection of servers on which to run the computation, and a client who wishes to use the trained model.

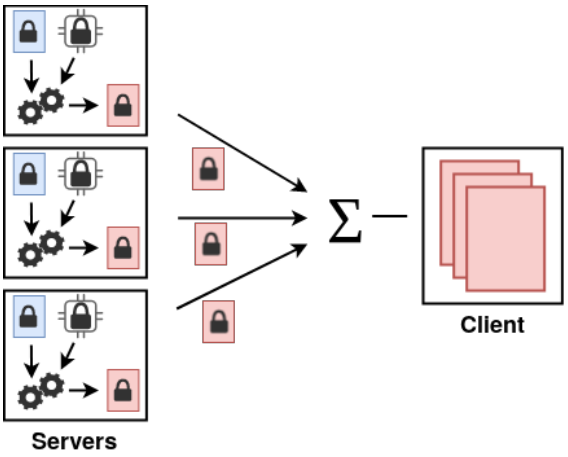

Figure 13. The servers compute the model over the data and the client receives reconstructed output shares.

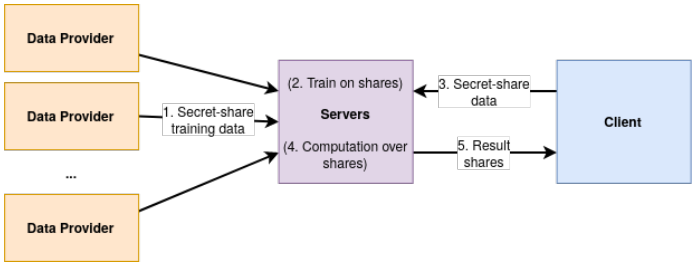

Figure 14. Training a model online with SMPC.

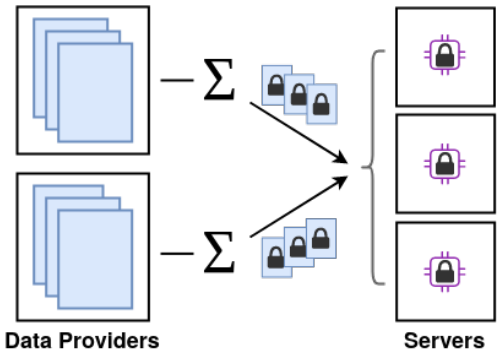

Figure 15. Data providers secret share their data with the servers.

Initially, all of the computation servers begin with a secret-shared untrained model. Each data provider secret shares their private data with the servers, shown in Figure 15. Importantly, no data provider is able to see any other party's data.

In the next step (Figure 16), the servers iteratively train their model shares over their data shares which results in a trained model still secret-shared across the servers. Note, in both of the presented SMPC examples, the server computation stage shown in these figures is a heavily simplified version of the actual process. Most operations involved in SMPC require frequent communication between the different servers, and this communication overhead (and sometimes the offline phases required by the various SMPC proto-

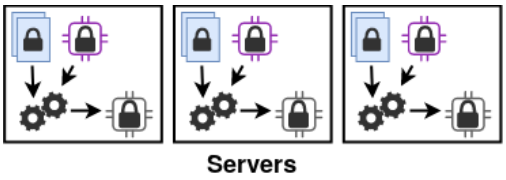

Figure 16. Private model trains over private data.

cols) contributes to the significantly greater run time compared to training the same model offline [32].

From this point, the trained model can either be left secret-shared and offered as a service to clients, or the model can be reconstructed and published or returned to the data providers. In the former case, a client could make secure inferences by secret-sharing their input data with the servers and recombining the output, following Figures 12 and 13.

4.5 FEDERATED LEARNING

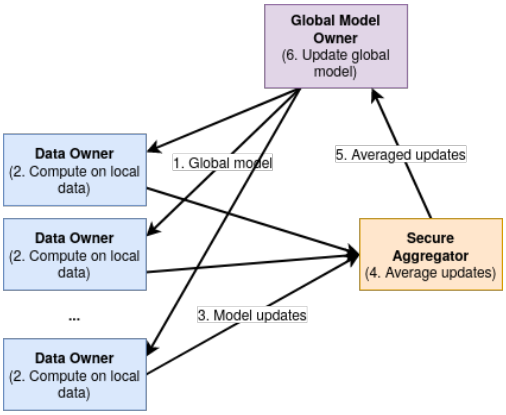

Figure 17. Federated learning.

Federated learning [50, 51, 52] is a technique for maintaining data privacy by keeping all data local to its owner and distributing the model training among many data owners. The local training updates are then aggregated or compiled into a final update that is applied to the global model. Similar to the HE and SMPC multiple data provider scenarios, this allows potentially sensitive data from multiple owners to be used while keeping it private from both the model owner and the other data owners. Although there are different ways to architect federated learning, this use case (as described in [51, 52]) shown in Figure 17 involves a global model owner, the data providers, and a separate aggregator that provides an additional layer of security and privacy between the model owner and the data providers. An applied example of this is given in [51], which discusses a Google research project about the use of federated learning to train an Android keyboard text prediction model. Predictive text is locally trained instead of centralizing the data from mobile devices that could contain sensitive information.

Initially, the global model owner has an untrained model. As shown in Figure 18, the model owner sends a copy of the model to each individual data owner or device.

In the next step, Figure 19, each data owner/device locally computes a model update (represented by the up arrow) on their local data. For instance, this could include stochastic gradient descent (SGD) updates, or the weight gradients calculated from running SGD on multiple batches of local data [51].

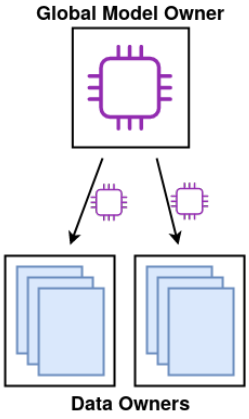

Figure 18. Global model owner shares model with all devices/data owners.

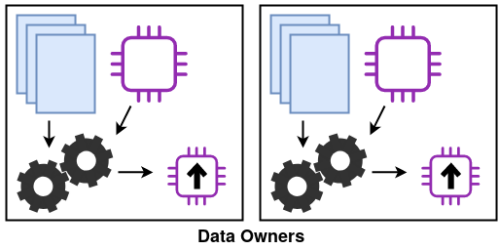

Figure 19. Devices calculate model update from local data.

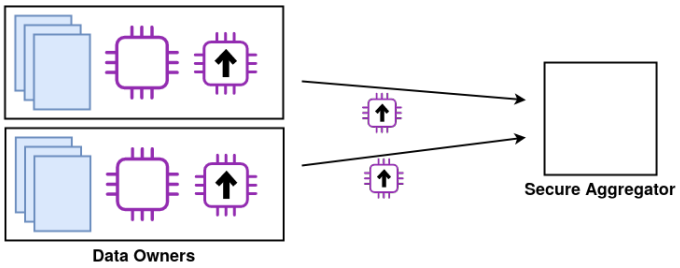

Figure 20. Devices send model updates to secure aggregator.

As the data owners calculate local weight updates, they are sent to a secure aggregator (Figure 20). SMPC protocols can be employed on these updates to ensure the aggregator cannot read them until a large number of data owners have participated. This is to prevent the global model owner from getting updates from only a few data owners and inferring anything about the original data.

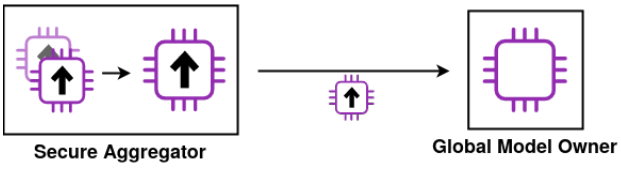

Figure 21. Secure aggregator averages model updates and sends to global model owner.

Next, in Figure 21, the secure aggregator averages all of the individual weight updates obtained from the data owners.

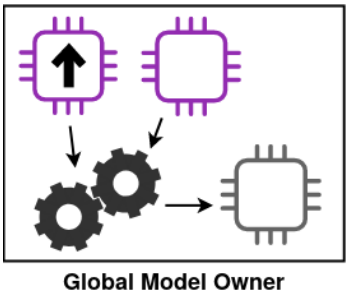

Figure 22. Global model owner updates model.

Finally, once the global model owner receives the averaged weight update, they apply it to the global model as shown in Figure 22. This process can be repeated for multiple epochs to iteratively train the public model. Data owner privacy is maintained because their individual data never leaves their local devices. Note, federated learning is demonstrably susceptible to attacks via a GAN approach [43], and it is currently infeasible for record-level differential privacy alone to protect against this type of attack.

4.6 SECURE ENCLAVE MODEL TRAINING

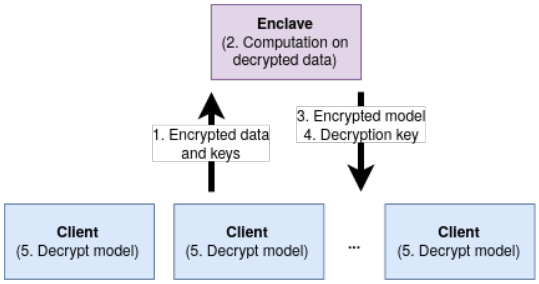

Figure 23. Training inside a secure enclave.

The last use case is based on a secure enclave, potentially hosted by a cloud vendor, and is discussed in [12]. The scenario presented is similar to the SMPC online training situation, as well as the multiple data provider HE scenario, in that private data from multiple data owners or clients can be used to train a model. Additionally this use case guarantees fairness by ensuring that all participating clients get a copy of the final trained model.

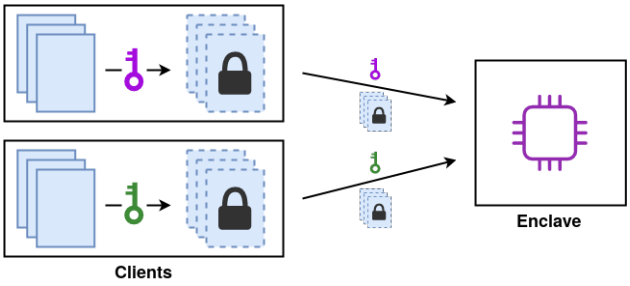

Figure 24. Clients encrypt data with different symmetric keys and send key and encrypted data to secure enclave.

In the first step (Figure 24), all clients use a symmetric encryption scheme to encrypt their local data. (Note, this is not a HE scheme. In the example given in [12], AES-GCM is used.) The clients then send this

encrypted data to the enclave with the key, which the enclave needs to locally decrypt and work with the client data.

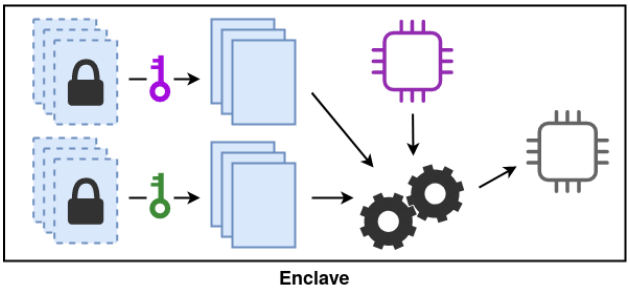

Figure 25. Enclave decrypts private data and trains model over it.

The secure enclave initially has an untrained model, and the clients can agree upon and verify via the code remote attestation. The enclave then uses each client key to decrypt the individual data sets and train the model over this data, shown in Figure 25.

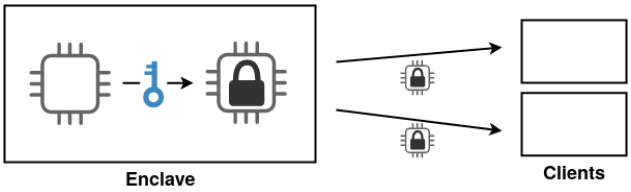

Figure 26. Enclave encrypts trained model with new symmetric key.

Once the model has been trained, the secure enclave generates a new symmetric key and uses it to encrypt the final model, in Figure 27. This encrypted model is then sent to each of the clients. To ensure fair availability, the enclave waits until receipt of the encrypted model is confirmed by all clients before releasing the key.

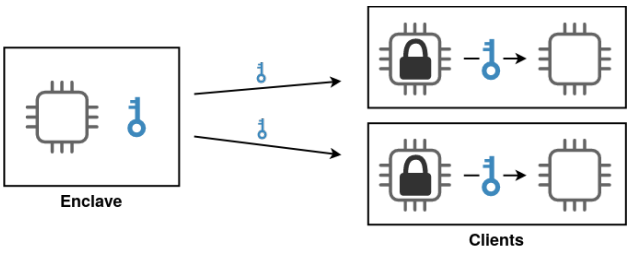

Figure 27. Once model receipt is acknowledged by all clients, the enclave releases the key.

Finally, when all clients have confirmed receipt of the model, the enclave releases the key in Figure 27 (potentially publishing it to a public third party, again to support fair availability) to each client. The clients then use the key to decrypt their local models. No clients ever see another clients' data, and although the enclave locally decrypts the data to compute over it, the enclave protects this from the environment, defending against a potentially malicious cloud vendor attempting to steal it. As mentioned in Section 3.4, there are still security concerns in this scenario, as with direct access to the model, white-box attacks could be conducted in an attempt to infer original training data.

5. ADDITIONAL CONSIDERATIONS

5.1 ALGORITHM SECURITY

Two fundamental differences exist between techniques like HE and SMPC—the adversary the technique is protecting against, and the basis of the security guarantees. In the former, we distinguish between a technique that protects against parties outside of the system and a technique that protects parties inside the system. Homomorphic encryption, similar to a conventional encryption technique like advanced encryption standard (AES), protects data in transit or at rest from an outside eavesdropper. Without the key, anyone who observes the encrypted data is unable to read it. Secure multiparty computation itself has no inherent mechanism for protection from eavesdropping, but the parties themselves are unable to learn anything about the others' data from what they explicitly receive.

The algorithms also carry different computational hardness assumptions. The security of HE is generally based on the ring learning with errors problem, meaning that similar to RSA, (the security of which is based on the difficulty of factoring large numbers), the amount of computational resources required to break the algorithm is significantly higher than what is available in practice. In contrast, ZKP and SMPC are information-theoretic secure—without some specific set of information, breaking the cryptosystem is impossible, regardless of compute capability.

SMPC has several additional security characteristics, and protocols are generally designed to address different security models, or to what degree of party dishonesty the protocol will protect against. These frequently fall into either semihonest (passive) security or malicious (active) security. For both cases, we will visualize the scenario that the security model protects against, and for simplicity, we assume only one party is secret sharing their data with the others as shown in Figure 28.

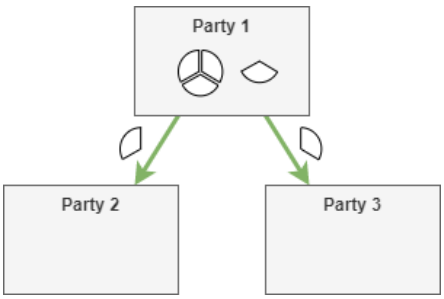

Figure 28. One party secret shares their data with two other parties.

In the first case, the model assumes semihonest but curious adversaries, meaning they do not deviate from the protocol but passively observe everything they can or collude in an attempt to collect more information than they should have. Figure 29 shows the two semihonest adversaries highlighted in yellow. All parties correctly compute over their data shares, following the protocol without deviation, and obtain valid result shares.

In Figure 30, parties 2 and 3 send their result shares to party 1 to be reconstructed into a final output, but also collude and combine the input data shares they received in an attempt to reconstruct the original data themselves.

If the protocol does not provide passive security, Figure 31 may result, in which parties 2 and 3 are able to infer the final missing data share (the dotted red segment) based on their combined shares. Input data privacy cannot be guaranteed in this situation.

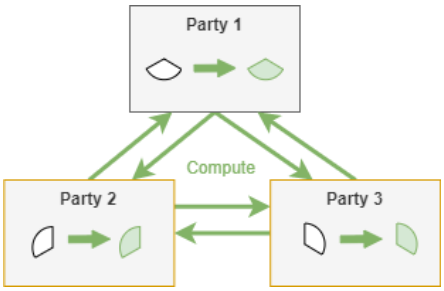

Figure 29. All parties follow protocol to compute a result.

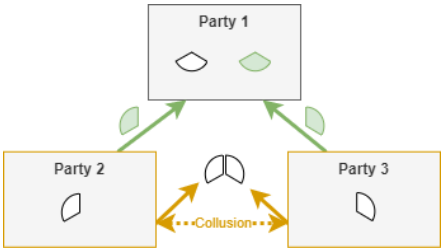

Figure 30. Parties 2 and 3 collude, combining their input data shares.

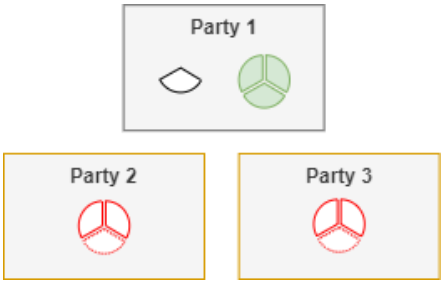

Figure 31. Original data is inferred, visualized by the dotted segment.

Active security, or security against malicious adversaries, is harder to achieve. This scenario assumes the malicious adversary is willing to arbitrarily deviate from the given protocol, either to corrupt the computation or to obtain private data. This is visualized in Figure 32 with the red computation and communication arrows from party 2, the malicious adversary. In this situation, the adversary may intentionally compute their result share incorrectly, corrupting the end result, or they may deviate such that the other parties inadvertently send enough information that the malicious adversary is able to infer the missing data, shown with the dotted red segments.

Figure 33 shows the potential result without active security. Party 1 may end up with an incorrect output, and the malicious adversary may end up with access to the original private input data.

Although it is feasible to make any arbitrary protocol secure against malicious adversaries [36], protocols that are malicious-secure frequently add a great deal of overhead. Another security model sometimes targeted is covert security [53], which relaxes some of these constraints. In covert security, an adversary may still arbitrarily deviate, but has an incentive to not get caught cheating. Protocols that provide covert security need to ensure a high probability that deviation will be detected.

Additionally, another parameter generally discussed in relation to security models is the number or ratio of parties that may be dishonest—a protocol that is malicious-secure against a dishonest majority may be much

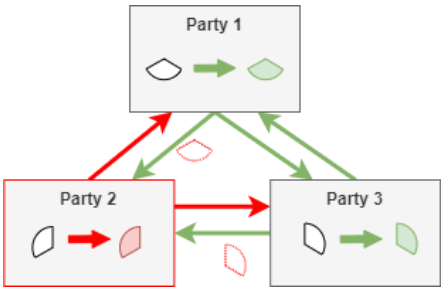

Figure 32. A malicious adversary deviates from the protocol, highlighted in red.

Figure 33. A malicious adversary deviates from the protocol, highlighted in red.

more computationally expensive than one secure against an honest majority.

Finally, it is worth reiterating that although HE and SMPC are strong techniques for evaluating functions over data while maintaining its security, machine learning models themselves can still be attacked via the methods referenced in Section 3.4 to gain some level of information about the training data. Other techniques such as differential privacy may need to be incorporated to help mitigate this risk. Notably, federated learning does not maintain security even with differential privacy in use.

5.2 FRAMEWORK MATURITY

A final consideration is the relative maturity of existing implementations of many of these techniques. The number of libraries and protocols has exploded in recent years, but the majority of them are intended solely for research purposes and are unsuitable for production use. Older and more generic libraries such as HELib and SEAL [54] are more likely to be production ready, but lack many of the higher level features for enabling faster machine learning algorithms without implementing them from scratch.

REFERENCES

[1] MD Laughter, JM Whitaker, and D Lockwood. Information-driven inspections. 2010.

[2] JM Whitney, S LaMontagne, A Sunshine, D Lockwood, D Peranteau, and G Dupuy. Next generation safeguards initiative: 2010 and beyond. Technical report, 2010.

[3] International Atomic Energy Agency. IAEA Department of Safeguards long-term R&D plan, 2012–2023. 2013.

[4] M Barni, P Failla, R Lazzeretti, A-R Sadeghi, and T Schneider. Privacy-preserving ECG classification with branching programs and neural networks. *IEEE Transactions on Information Forensics and Security*, 6(2):452–468, 2011.

[5] D Bogdanov, R Talviste, and J Willemson. Deploying secure multi-party computation for financial data analysis. In *International Conference on Financial Cryptography and Data Security*, pages 57–64. Springer, 2012.

[6] LJM Aslett, PM Esperança, and CC Holmes. A review of homomorphic encryption and software tools for encrypted statistical machine learning. *arXiv preprint arXiv:1508.06574*, 2015.

[7] 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 *International Conference on Machine Learning*, pages 201–210, 2016.

[8] AC Yao. Protocols for secure computations. In *23rd annual symposium on foundations of computer science (sfcs 1982)*, pages 160–164. IEEE, 1982.

[9] AC-C Yao. How to generate and exchange secrets. In *27th Annual Symposium on Foundations of Computer Science (sfcs 1986)*, pages 162–167. IEEE, 1986.

[10] AA Solodov, DR Farley, C Brif, and ND Pattengale. Development of novel approaches to anomaly detection and surety for safeguards data. Technical report, Sandia National Laboratory, Albuquerque, NM, 2019.

[11] DR Farley, MG Negus, and RN Slaybaugh. Industrial internet-of-things and data analytics for nuclear power and safeguards. Technical report, Sandia National Laboratories , Livermore, CA, 2018.

[12] O Ohrimenko, F Schuster, C Fournet, A Mehta, S Nowozin, K Vaswani, and M Costa. Oblivious multi-party machine learning on trusted processors. In *25th {USENIX} Security Symposium ({USENIX} Security 16)*, pages 619–636, 2016.

[13] C Gentry and D Boneh. *A Fully Homomorphic Encryption Scheme*, volume 20. Stanford University, 2009.

[14] I Damgård, V Pastro, N Smart, and S Zakarias. Multiparty computation from somewhat homomorphic encryption. In *Annual Cryptology Conference*, pages 643–662. Springer, 2012.

[15] Z Brakerski and V Vaikuntanathan. Fully homomorphic encryption from ring-LWE and security for key dependent messages. In *Annual Cryptology Conference*, pages 505–524. Springer, 2011.

[16] V Lyubashevsky, C Peikert, and O Regev. On ideal lattices and learning with errors over rings. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 1–23. Springer, 2010.

[17] JW Bos, K Lauter, J Loftus, and M Naehrig. Improved security for a ring-based fully homomorphic encryption scheme. In *IMA International Conference on Cryptography and Coding*, pages 45–64. Springer, 2013.

[18] A Brutzkus, R Gilad-Bachrach, and O Elisha. Low latency privacy preserving inference. In *International Conference on Machine Learning*, pages 812–821, 2019.

[19] T Graepel, K Lauter, and M Naehrig. Ml confidential: Machine learning on encrypted data. In *International Conference on Information Security and Cryptology*, pages 1–21. Springer, 2012.

[20] R Bost, RA Popa, S Tu, and S Goldwasser. Machine learning classification over encrypted data. In *NDSS*, volume 4324, page 4325, 2015.

[21] P Paillier. Public-key cryptosystems based on composite degree residuosity classes. In *International Conference on the Theory and Applications of Cryptographic Techniques*, pages 223–238. Springer, 1999.

[22] S Goldwasser and S Micali. Probabilistic encryption and how to play mental poker keeping secret all partial information. In *Proceedings of the Fourteenth annual ACM Symposium on Theory of Computing*, pages 365–377. ACM, 1982.

[23] M Albrecht, M Chase, H Chen, D Ding, S Goldwasser, S Gorbunov, S Halevi, J Hoffstein, K Laine, K Lauter, S Lokam, D Micciancio, D Moody, T Morrison, A Sahai, and V Vaikuntanathan. Homomorphic encryption security standard. Technical report, HomomorphicEncryption.org, Toronto, Canada, November 2018.

[24] MS Riazi, C Weinert, O Tkachenko, EM Songhori, T Schneider, and F Koushanfar. Chameleon: A hybrid secure computation framework for machine learning applications. In *Proceedings of the 2018 on Asia Conference on Computer and Communications Security*, pages 707–721. ACM, 2018.

[25] S Even, O Goldreich, and A Lempel. A randomized protocol for signing contracts. *Communications of the ACM*, 28(6):637–647, 1985.

[26] Y Lindell and B Pinkas. A proof of security of Yao's protocol for two-party computation. *Journal of Cryptology*, 22(2):161–188, 2009.

[27] P Mohassel and Y Zhang. Secureml: A system for scalable privacy-preserving machine learning. In *2017 IEEE Symposium on Security and Privacy (SP)*, pages 19–38. IEEE, 2017.

[28] D Beaver. Efficient multiparty protocols using circuit randomization. In *Annual International Cryptology Conference*, pages 420–432. Springer, 1991.

[29] P Mohassel and P Rindal. ABY 3:A mixed protocol framework for machine learning. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, pages 35–52. ACM, 2018.

[30] BD Rouhani, MS Riazi, and F Koushanfar. Deepsecure: Scalable provably-secure deep learning. In *Proceedings of the 55th Annual Design Automation Conference*, page 2. ACM.

[31] C Juvekar, V Vaikuntanathan, and A Chandrakasan. {GAZELLE}: A low latency framework for secure neural network inference. In *27th {USENIX} Security Symposium ({USENIX} Security 18)*, pages 1651–1669, 2018.

[32] S Wagh, D Gupta, and N Chandran. SecureNN: 3-party secure computation for neural network training. *Proceedings on Privacy Enhancing Technologies*, 1:24, 2019.

[33] D Evans, V Kolesnikov, and M Rosulek. A pragmatic introduction to secure multi-party computation. *Foundations and Trends® in Privacy and Security*, 2(2–3), 2017.

[34] Y Ishai, E Kushilevitz, R Ostrovsky, and A Sahai. Zero-knowledge proofs from secure multiparty computation. *SIAM Journal on Computing*, 39(3):1121–1152, 2009.

[35] M Petkus. Why and how zk-SNARK works. *arXiv preprint arXiv:1906.07221*, 2019.

[36] O Goldreich, S Micali, and A Wigderson. How to play ANY mental game. In *Proceedings of the Nineteenth Annual ACM Symposium on Theory of Computing*, STOC '87, pages 218–229, New York, NY, 1987. Association for Computing Machinery.

[37] RN Reith, T Schneider, and O Tkachenko. Efficiently stealing your machine learning models. In *Proceedings of the 18th ACM Workshop on Privacy in the Electronic Society*, pages 198–210. ACM, 2019.

[38] R Shokri, M Stronati, C Song, and V Shmatikov. Membership inference attacks against machine learning models. In *2017 IEEE Symposium on Security and Privacy (SP)*, pages 3–18. IEEE, 2017.

[39] F Tramèr, F Zhang, A Juels, MK Reiter, and T Ristenpart. Stealing machine learning models via prediction apis. In *25th {USENIX} Security Symposium ({USENIX} Security 16)*, pages 601–618, 2016.

[40] C Dwork. Differential privacy. *Encyclopedia of Cryptography and Security*, pages 338–340, 2011.

[41] HB McMahan, G Andrew, U Erlingsson, S Chien, I Mironov, N Papernot, and P Kairouz. A general approach to adding differential privacy to iterative training procedures. *arXiv preprint arXiv:1812.06210*, 2018.

[42] M Abadi, A Chu, I Goodfellow, HB McMahan, I Mironov, K Talwar, and L Zhang. Deep learning with differential privacy. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, pages 308–318. ACM, 2016.

[43] B Hitaj, G Ateniese, and F Perez-Cruz. Deep models under the GAN: Information leakage from collaborative deep learning. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, pages 603–618, 2017.

[44] V Costan, I Lebedev, and S Devadas. Sanctum: Minimal hardware extensions for strong software isolation. In *25th {USENIX} Security Symposium ({USENIX} Security 16)*, pages 857–874, 2016.

[45] T Hunt, Z Zhu, Y Xu, S Peter, and E Witchel. Ryoan: A distributed sandbox for untrusted computation on secret data. In *12th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 16)*, pages 533–549, 2016.

[46] G Chen, S Chen, Y Xiao, Y Zhang, Z Lin, and TH Lai. Sgxpectre attacks: Stealing intel secrets from sgx enclaves via speculative execution. *arXiv preprint arXiv:1802.09085*, 2018.

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

[48] N Hynes, R Cheng, and D Song. Efficient deep learning on multi-source private data. *arXiv preprint arXiv:1807.06689*, 2018.

[49] T Hunt, C Song, R Shokri, V Shmatikov, and E Witchel. Chiron: Privacy-preserving machine learning as a service. *arXiv preprint arXiv:1803.05961*, 2018.

[50] S Truex, N Baracaldo, A Anwar, T Steinke, H Ludwig, R Zhang, and Y Zhou. A hybrid approach to privacy-preserving federated learning. In *Proceedings of the 12th ACM Workshop on Artificial Intelligence and Security*, pages 1–11. ACM, 2019.

[51] A Hard, K Rao, R Mathews, S Ramaswamy, F Beaufays, S Augenstein, H Eichner, C Kiddon, and D Ramage. Federated learning for mobile keyboard prediction. *arXiv preprint arXiv:1811.03604*, 2018.

[52] K Bonawitz, V Ivanov, B Kreuter, A Marcedone, HB McMahan, S Patel, D Ramage, A Segal, and K Seth. Practical secure aggregation for privacy-preserving machine learning. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, pages 1175–1191. ACM, 2017.

[53] Y Aumann and Y Lindell. Security against covert adversaries: Efficient protocols for realistic adversaries. In *Theory of Cryptography Conference*, pages 137–156. Springer, 2007.

[54] Microsoft SEAL (release 3.4). https://github.com/Microsoft/SEAL, October 2019. Microsoft Research, Redmond, WA.