Published in Towards Data Science

Daniel Huynh
Jun 19, 2020 · 11 min read · Listen

Homomorphic Encryption intro: Part 1: Overview and use cases

Source: Broadcom

Homomorphic encryption intro:

Part 1: Overview and use cases

Part 2: HE landscape and CKKS

Part 3: Encoding and decoding in CKKS

Introduction

Advancements in machine learning algorithms have resulted in a widespread adoption across industries. Nonetheless, areas dealing with sensitive and private data, like healthcare or finance, have lagged behind due to regulatory constraints to protect users' data.

With the emergence of Machine Learning as a Service, entities are providing model inference as a service. We can distinguish three parties in such scenarios, a model owner such as a hospital who has trained a model, a host such as a cloud provider providing computing power, and a client wanting to benefit from the service. A model owner can also be a host in some scenarios. Trust must be established between those parties as the client does not want her data to be leaked, and the model owner wants to protect her model.

In this article, we will have a quick look at Machine Learning, and the advances it brought. Then we will see how the data dependence of Machine Learning makes it unsuitable for some sensitive use cases, and how new solutions can make training and inference of models possible on encrypted data. Finally we will have a focus on Homomorphic Encryption, and see what use cases it can cover.

This article is non-technical and is aimed at a broad audience. Following articles will dig deeper into the technicalities of Homomorphic Encryption, with both the theory and a Python implementation of an Homomorphic Encryption scheme.

I. The privacy concerns around data exploitation

Machine Learning, especially Deep Learning, which is the sub-field of Machine Learning focusing on Deep Neural Networks (DNNs), have proved to advance the state of the art on several various tasks. Diverse domains such as image recognition with ResNet, text processing with BERT, or even speech generation with WaveNet, have all seen massive improvements using Deep Learning, while other models failed behind by a considerable margin.

The underlying principle of Deep Learning is to train models on a massive amount of data, and by optimizing the loss of the model. By doing so, Deep learning has managed to get human-like performances, as it is able to find complex patterns that would be invisible to the human eye.

Therefore, Machine Learning seems to pave the way for new technological leaps in the future, but relies heavily on the use of data, be it for its training, or for inference.

Most of the data used in those Deep Learning models often came from public, and non-personnal data, such as Wikitext, or Imagenet. Nonetheless other scenarios might require more sensible training data. For instance, a speech-to-text model might require people to record their voices, a diagnosis tool will require private health data to be sent, or a credit analysis tool might need to have a look at financial information.

While we saw how powerful Machine Learning is, we see here that some sensitive use cases can not be directly answered by such approach, as the data is too sensitive to be shared, either for training or for inference. Therefore, there seems to be a trade off between data privacy, and data efficiency, in the sense that one can uphold better confidentiality of private data, by imposing strict processes where data is only seen by a trusted human expert, at the cost of making this process long and expensive, or use trained models that can have excellent performances and scalability, but at the expense of needing to see data during training and inference.

However, several techniques have emerged in the past years that allow to reconcile privacy and efficiency. Among them, three seem to be the most promising :

- **Homomorphic Encryption** (HE) is a public key cryptographic scheme. The user creates a pair of secret and public key, uses the public one to encrypt her data, before sending it to a third party which will perform computations on the encrypted data. Because of the homomorphic properties of the encryption and decryption, the user can get the encrypted result and decode it with her own key to see the output of the computation on her data, without having shown it once in clear to the third party.
- **Secure Multi Party Computation** (SMPC) is a different paradigm which relies more on communication between the participants. Data can be split, as well as the model, and each actor only sends a few shares of her data, so that others can not reconstruct the initial data, but can participate, and do some computation on shares of data. Then once each party has finished, everything can be aggregated and the result of the output is known to each party.
- **Trusted Execution Environments** (TEE) enable the development of software thanks to hardware guarantees of privacy. Intel's SGX technology provides an implementation of such system. The enclave technology allows programs to be executed in isolation of other programs. All data inbound and outbound is encrypted, and computation in clear only happens within the enclave. The enclave code and integrity can then be checked externally.

SMPC provides interesting features, such as the possibility to enable different parties to collaborate while hiding each party's data. One of the main strengths of SMPC is its ability to compute complex operations, such as comparisons, or argmax, while these are not possible in HE. Moreover, it is easier to evaluate deep models with SMPC compared to HE, due to the leveled nature of most HE schemes.

However SMPC requires each party to follow the protocol. Moreover, it needs a trusted third party not to collaborate with one of the parties. If so, data of one or several parties could be compromised. Finally, because SMPC relies on communication and computation between parties, this requires more computation and more bandwidth which does not necessarily fit the usual client/server paradigm, where the server performs most of the operations with little communication.

TEE provides hardware guarantees that sensible data is processed in a controlled environment. In this secured space, called enclave, data is decrypted and is used in plain for computation. Outside of the enclave, data remains encrypted. This allows for much more efficiency, as we do not need to encrypt data before computation, but we can directly compute on plaintext and encrypt the output before it leaves the enclave.

Nonetheless writing secure code for TEEs is a difficult task, and several attacks have been found on Intel SGX. "A Survey of Published Attacks on Intel SGX" covers several types of attack on Intel SGX, such as side channel attacks where the OS-Kernel leverages its near total control over the platform to construct an attack on the enclave which relies on the kernel's services. Other attacks such as cache attacks, branch attacks, or speculative action attacks exist, but we let the interested reader learn more about them in the above paper.

HE enables almost all of the computation to be externalized to the server once encryption has been done. Using homomorphic properties, computations can be done on the encrypted data without any leak of privacy, and the output can be safely decrypted by the user. Moreover, HE does not need as much communication as SMPC, and does not need any special hardware compared to TEEs.

While HE is relatively simple, and has little dependence compared to the other methods, HE cannot efficiently perform arbitrary many multiplications and HE can only perform addition and multiplication, which can make some computation more difficult to be performed.

While each method has its own set of pros and cons, one desirable property that these techniques share is that they can allow both the model and data to be encrypted. This could open the way to new practices of "Private Machine Learning", where users would be able to share private data, such as medical or financial records, in order to benefit from Machine Learning services, without having to actually show their data.

For the rest of this paper we will focus on Homomorphic Encryption, as it is the simplest to deploy in practice among the three methods. Indeed, SMPC and TEEs require extra infrastructure deployment, as TEEs need special hardware to be used, and SMPC relies on a trusted third party (which could be a TEE), while HE is pretty straightforward and can be implemented without any special dependency.

II. Privacy friendly use cases with Homomorphic Encryption

We will now see how those concepts can be applied in practice, and what could be the different use cases that such privacy friendly methods make possible.

Throughout this section, we will explore the different uses cases that HE open. As we will see in another article, the HE scheme enables a user to encrypt a plaintext message into a ciphertext, which can be used for computation. Operations can be done between a ciphertext and a ciphertext, or between a ciphertext and a plaintext. Finally, only the user who encrypted the data can decrypt the output.

Therefore HE schemes can allow the model and the data to be encrypted and the result must always come back to the owner of the secret key to get the final result. Using these assumptions we will see how different scenarios can be covered.

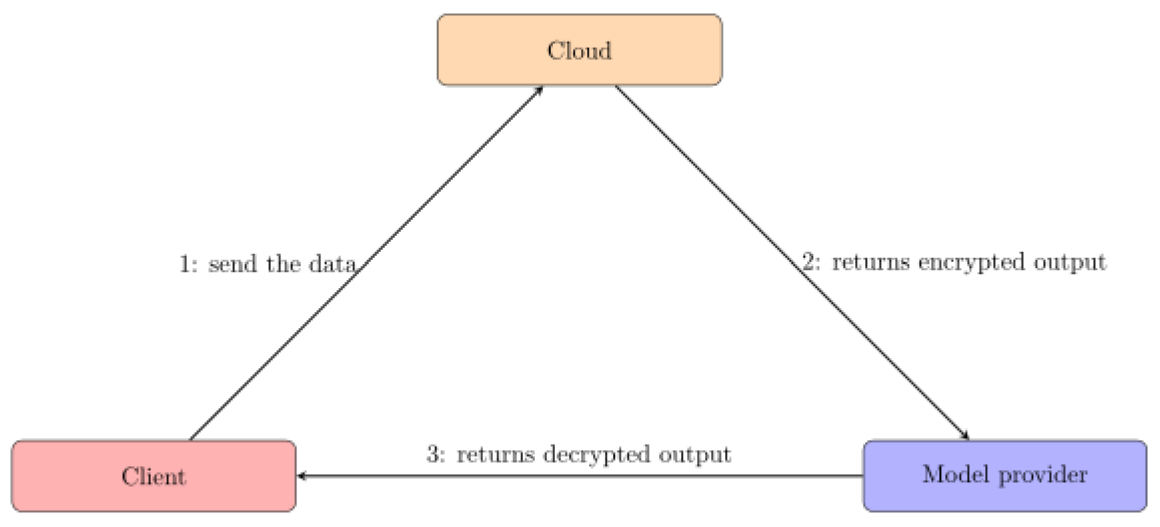
A. Model confidentiality : financial indicator from public data

learning has benefited from a very active research and open source community. This has enabled many state of the art models to become available to non researchers. For instance, **Fastai** a well known Deep Learning framework has enabled the democratize powerful models such as ResNet for image recognition, or ULMFiT for text classification.

Therefore, the value of such models does not reside in the architecture, which are often known to most, but in the weights obtained after training, therefore in the training data as well.

Here we will explore the scenario where an user has managed to collect, clean and pre-process a data set which has been used for the training of a given model. While the data used in training can be safely stored, if the user wants to deploy the model on Cloud platforms, she will have to trust the Cloud provider not to steal the weights, nor leak the model, be it intentionally or unintentionally.

Thus, one can imagine that by encrypting the weights, the Cloud host will not be able to gain any insight on the model, and it becomes safe for the user to deploy her model on the Cloud.



Overview of CKKS (Source: Daniel Huynh)

The figure above provides an illustration of this. The scenario would be the following:

- The client sends her data unencrypted to the model hosted on the Cloud provider.
- The Cloud host performs computation using the encrypted model, and the unencrypted data, then sends the result to the model provider.
- The model provider decrypts the result, then sends it to the client.

Because the model provider still neeeds to decrypt the result, this would be more interesting in a case of hybrid cloud, where the non critical parts of the infrastructure are hosted on a public cloud, while the more critical parts are isolated on a private cloud. Here one could imagine that the public cloud part takes care of most of the computation and manages load balancing, while the private part would only decrypt the result and send it to the client (eventually through the public cloud).

One example of such scenario would be for instance a company which analyzes public market to indicators and aggregated data to its client. Then it would make sense to host the bulk of the computation on a public cloud, but then protect all the algorithms by encrypting them before sending it to the cloud. The results can then be decrypted by this company and then be sent to their client.

B. Private inference : Private Set Intersection

In the private inference setting, we will assume now the opposite, where the data held by the user is private, but the model held by the model provider does not need encryption. This can be the case for instance in a setup where the model owner and the Cloud provider are either the same entity, or that they trust each for this scenario.

In that case, the goal is then to provide a scalable and privacy friendly solution, so that users can query a server with private data without revealing it to the server.

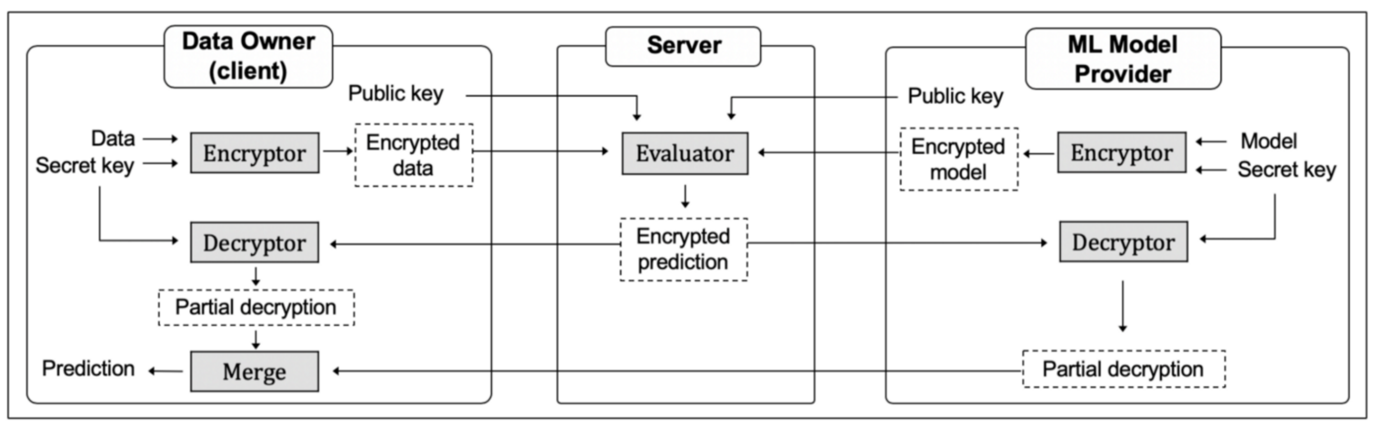
For instance, Private Set Intersection (PSI) is the scenario where one user holds private information, but want to check whether or not her data intersects with a larger database of the same nature. A server will then operate this database, and the server's data needs not necessarily be encrypted. HE offers the possibility to do this while preserving the user's data privacy. "Fast Private Set Intersection from Homomorphic Encryption" explores this scenario and shows one can use PSI to do contact discovery. In that case, a service provider such as Whatsapp has large databases about who uses their service, and a new client with low ressources would like to know who among their contact uses Whatsapp. Then this client simply needs to ask the server to privately intersects her list of contact with all users, and decrypt the result to know who among her contacts uses Whatsapp.

As another example, in the case of the recent crisis around the Covid19, one could check if contact has been made between the user and a known group of infected patients. This could be done in a client server scenario where the user shares her location data, and the server could perform computation on it to output to the client the eventual patients the client might have encountered.

C. Three-party confidential computation : hospital model hosted on the Cloud

The last scenario we will cover is the case where both data and model are encrypted. This could be the case in a three party setting, where we have a user wanting to have her data analyzed, a model owner who wants to provide her model, and a Cloud provider provides the infrastructure to host this service.

In the standard HE scheme, only ciphertexts generated by the same public key can be used together for addtion and multiplication. This would make it impossible to compute anything that was generated by two different clients, therefore makes HE rather limited. However "Efficient Multi-Key Homomorphic Encryption with Packed Ciphertexts with Application to Oblivious Neural Network Inference" provided a way to have computation done on several inputs from different owners.



Multiparty HE overview (source: Efficient Multi-Key Homomorphic Encryption with Packed Ciphertexts with Application to Oblivious Neural Network Inference)

The figure above provides an overview of the process. The main idea is that both the client and the model owner can encrypt their data before sending it to the server. This one will be able to do computation using those two inputs, then send partial results to each party. The model owner will then remove part of the noise that he introduced using her private key, send it to the client, who will remove the remaining noise and get the output.

This scenario is particularly interesting for medical scenarios. The client might be a patient providing her data, while the model owner can be an hospital which has trained a model. In order to serve as best the patients, the model can then be safely deployed on a public cloud, while preserving the weights trained by the hospital. At inference, the data will be encrypted by the patient, and decrypted in the two step process described above.

Conclusion

We have seen throughout this article why we need more privacy friendly solutions, and what are the current approaches with HE, SMPC and TEE. Then we saw what could be the different use cases that HE could address.

HE seems to be a very promising field in a context of GDPR and Covid19 where we need both to advance fast but with caution with respect to people's data.

We will cover more technical parts in the next articles, where we will see how to build our own HE scheme from scratch.

So I hope you enjoyed reading this article, would be great to have your feedback and I will try to post more articles in the future.

If you have any question, do not hesitate to contact me on Linkedin , you can also find me on Twitter !

117

Get unlimited access

Search

Daniel Huynh
169 Followers

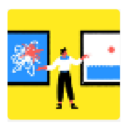
CEO at Mithril Security

Follow

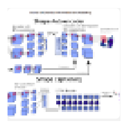
More from Medium

Aliso... in Toward...
Managing Language Model Risks; Dogs as...

Savoga
Explain any rule-based model using game...



Utkarsh Parashar
Generating similar types of molecules, give...



Fan Yang
Difference of batch Normalization...