Homomorphic Encryption

DSC 4110/6111: Modern Cryptography and AI/ML Security

Dr. Laltu Sardar

School of Data Science,
Indian Institute of Science Education and Research Thiruvananthapuram (IISER TVM)

August 14, 2026

Table of Content

Outsourcing Machine Learning Model

Motivation: Outsourcing a Machine Learning Model

Suppose an **Owner** has developed a valuable machine learning model using significant data, computation, and expertise.
As a simple example, consider the linear regression model

$$\boxed{y = mx + c}.$$

The Owner wants to provide prediction as a service without maintaining the computational infrastructure.

There are three entities:

- **Owner:** develops and owns the model (m, c).
- **Cloud:** provides computation and storage.
- **User:** wants to obtain predictions for input x.

The basic outsourcing workflow is

$$\boxed{\text{Owner} \xrightarrow{(m,c)} \text{Cloud}}$$

$$\boxed{\text{User} \xrightarrow{x} \text{Cloud} \xrightarrow{y=mx+c} \text{User}}$$

In this setting, the Cloud knows the model and performs the computation directly.

Why Is the Model Confidential?

The model parameters

$$\boxed{m,\ c}$$

may represent valuable intellectual property.
The Owner has invested in:

- collecting and preparing training data,
- developing and training the model,
- selecting the model and its parameters,
- computational resources and expertise.

If the Cloud receives the model in plaintext, it can obtain

$$m,\ c.$$

The Cloud could potentially copy the model and use it to provide prediction services to other users.

$$\boxed{\text{The Owner wants the Cloud to use the model, but not to learn the model.}}$$

This creates a fundamental tension:

$$\boxed{\text{Use the model} \quad \text{vs.} \quad \text{Hide the model}}$$

First Attempt: Encrypt the Model

The natural solution is to encrypt the model before sending it to the Cloud.

The Owner computes $C_m = \mathrm{Enc}(m), \qquad C_c = \mathrm{Enc}(c)$

and sends only $\boxed{C_m,\ C_c}$ to the Cloud.

The Cloud therefore cannot directly see

$$m,\ c.$$

The desired workflow becomes

$$\text{Owner} \xrightarrow{\mathrm{Enc}(m),\mathrm{Enc}(c)} \text{Cloud}$$

and for a user query x,

$$\text{User} \xrightarrow{x} \text{Cloud}.$$

The Cloud should somehow compute

$$\boxed{\mathrm{Enc}(mx + c)}$$

without knowing m and c.

The User can then decrypt the result:

$$\boxed{\mathrm{Dec}\left(\mathrm{Enc}(mx + c)\right) = mx + c.}$$

Why Ordinary Encryption Is Not Enough

Suppose the Owner uses a conventional symmetric encryption scheme, such as a block cipher or stream cipher. The Cloud receives

$$\mathrm{Enc}(m), \qquad \mathrm{Enc}(c).$$

However, ordinary encryption is designed to provide **confidentiality**, not algebraic computation.
In general, there is no efficient operation such that

$$\mathrm{Enc}(m) \square \mathrm{Enc}(c) = \mathrm{Enc}(m + c) \qquad \textit{or} \qquad \mathrm{Enc}(m) \square \mathrm{Enc}(x) = \mathrm{Enc}(mx).$$

The Cloud cannot simply perform

$$mx + c$$

on ciphertexts.

To perform the computation normally, the Cloud would need to obtain the plaintext model:

$$\mathrm{Enc}(m), \mathrm{Enc}(c) \xrightarrow{\mathrm{Dec}} m, c.$$

But then the model is revealed.

The Challenge

We need encryption that provides confidentiality **and** allows meaningful computation directly on ciphertexts.

The Required Cryptographic Property

For the linear regression model

$$\boxed{y = mx + c},$$

the Cloud must compute using the encrypted model parameters.
First,

$$\text{Enc}(m) \times x \longrightarrow \text{Enc}(mx).$$

Then,

$$\text{Enc}(mx) + \text{Enc}(c) \longrightarrow \text{Enc}(mx + c).$$

Thus, we need an encryption scheme that supports

$$\boxed{\text{Multiplication and Addition on encrypted data}}$$

while the input x can remain in plaintext.

Finally, the authorized party decrypts:

$$\boxed{\text{Dec}\left(\text{Enc}(mx + c)\right) = mx + c}$$

Key Requirement

The Cloud should be able to compute on the encrypted model without learning the model parameters m and c.

Hiding the Queries

What If the User's Query Is Also Sensitive?

So far, we considered only the confidentiality of the model.
But the user's input x may also contain sensitive information.
For example:

- medical measurements,
- financial information,
- personal information,
- confidential business data.

The User therefore may not want the Cloud to learn x either.

Now we want to protect **both sides**:

Model: (m, c)	Query: x

The desired setting is

$\mathrm{Enc}(m)$,	$\mathrm{Enc}(c)$,	$\mathrm{Enc}(x)$

and the Cloud should compute the prediction without seeing any of these plaintext values.

Computing on Both Encrypted Model and Query

The Owner encrypts the model:

$$C_m = \mathrm{Enc}(m), \qquad C_c = \mathrm{Enc}(c).$$

The User encrypts the query:

$$C_x = \mathrm{Enc}(x).$$

The Cloud receives only

$$\boxed{C_m,\ C_c,\ C_x}.$$

It should evaluate the model directly on ciphertexts:

$$\boxed{C_y = \mathrm{Eval}\left(y = mx + c,\ C_m, C_c, C_x\right)}$$

such that

$$\boxed{\mathrm{Dec}(C_y) = mx + c.}$$

Thus the Cloud performs the computation without learning

$$\boxed{m, \qquad c, \qquad x, \qquad y.}$$

Desired Privacy: Neither the proprietary model nor the user's private query needs to be revealed to the Cloud.

Complete Privacy-Preserving Prediction

The complete workflow becomes:

$$\boxed{\text{Owner} \xrightarrow{\text{Enc}(m),\text{Enc}(c)} \text{Cloud}}$$

$$\boxed{\text{User} \xrightarrow{\text{Enc}(x)} \text{Cloud}}$$

The Cloud evaluates

$$\boxed{\text{Enc}(y) = \text{Eval}\left(mx + c,\ \text{Enc}(m), \text{Enc}(x), \text{Enc}(c)\right)}$$

and returns the encrypted result:

$$\boxed{\text{Cloud} \xrightarrow{\text{Enc}(y)} \text{User}}$$

The User decrypts:

$$\boxed{y = \text{Dec}(\text{Enc}(y)).}$$

Therefore:

$$\boxed{\text{Encrypted Model} + \text{Encrypted Query} \xrightarrow{\text{Cloud Computation}} \text{Encrypted Prediction}}$$

What Does the Encryption Scheme Need to Support?

For $y = mx + c$, the Cloud must evaluate

$$\boxed{m \times x}$$

and then

$$\boxed{(mx) + c.}$$

When everything is encrypted, we need

$$\mathsf{Enc}(m) \times_{\mathsf{HE}} \mathsf{Enc}(x) = \mathsf{Enc}(mx)$$

followed by

$$\mathsf{Enc}(mx) +_{\mathsf{HE}} \mathsf{Enc}(c) = \mathsf{Enc}(mx + c).$$

Therefore, the encryption scheme must support both:

$$\boxed{\text{Homomorphic Addition}}$$

and

$$\boxed{\text{Homomorphic Multiplication}}.$$

This Leads to FHE: If the Cloud needs to evaluate arbitrary machine learning models, the encryption scheme must support sufficiently general computations over ciphertexts.

Homomorphic Encryptions

Homomorphic Encryption

We have reached the following requirement:

$$\boxed{\text{Encrypted Model} + \text{Encrypted Query} \xrightarrow{\text{Cloud Computation}} \text{Encrypted Prediction}}$$

The encryption scheme must allow the Cloud to perform computations directly on ciphertexts.

Homomorphic Encryption (HE) is an encryption technique that allows certain computations to be performed on encrypted data such that the decrypted result is the same as the result of performing the computation on the original plaintexts.
For a computation f,

$$\boxed{\text{Dec}\left(\text{Eval}\left(f, \text{Enc}(m)\right)\right) = f(m)}$$

Thus, the Cloud performs

$$\text{Enc}(m) \xrightarrow{\text{Eval}(f)} \text{Enc}(f(m))$$

without learning m.

For our linear regression example,

$$\boxed{\text{Enc}(m), \text{Enc}(c), \text{Enc}(x) \xrightarrow{\text{Eval}(mx+c)} \text{Enc}(mx + c)}$$

Somewhat Homomorphic Encryption

A **Somewhat Homomorphic Encryption (SHE)** scheme allows homomorphic evaluation of both addition and multiplication, but only for computations of **limited complexity**.

For ciphertexts $c_1 = \mathrm{Enc}(m_1)$ and $c_2 = \mathrm{Enc}(m_2)$,

$$\mathrm{Eval}(+,\ c_1, c_2) = \mathrm{Enc}(m_1 + m_2)$$

and

$$\mathrm{Eval}(\times,\ c_1, c_2) = \mathrm{Enc}(m_1 m_2).$$

However, repeated operations increase the internal **noise** in the ciphertext.
After a certain multiplicative depth, decryption may no longer recover the correct plaintext.

Addition + Multiplication but Limited Multiplicative Depth

Example

An SHE scheme may evaluate

$$f(x) = ax^2 + bx + c$$

but may fail for a computation requiring a much deeper circuit.

Fully Homomorphic Encryption

Fully Homomorphic Encryption (FHE) extends homomorphic evaluation to computations of **arbitrary depth**. It supports both fundamental operations:

$$\text{Eval}(+,\ c_1, c_2) = \text{Enc}(m_1 + m_2)$$

and

$$\text{Eval}(\times,\ c_1, c_2) = \text{Enc}(m_1 m_2).$$

These operations can be composed to evaluate an arbitrary computational circuit.

For any efficiently computable function f,

$$\boxed{\text{Dec}\left(\text{Eval}(f, \text{Enc}(m))\right) = f(m)}$$

The key challenge is to control the noise that grows during homomorphic computation.

$$\boxed{\text{Addition} + \text{Multiplication} + \text{Arbitrary Multiplicative Depth}}$$

Note

FHE allows a computation to be outsourced while keeping the underlying data encrypted throughout the computation.

Classical Homomorphic Encryption Schemes

RSA-based

Classical Homomorphic Encryption: RSA

We have already seen that textbook RSA has an important **homomorphic property**.
Recall RSA encryption:

$$\boxed{c = \mathsf{Enc}(m) = m^e \bmod N}$$

Suppose

$$c_1 = \mathsf{Enc}(m_1) = m_1^e \bmod N$$

and

$$c_2 = \mathsf{Enc}(m_2) = m_2^e \bmod N.$$

The Cloud can multiply the ciphertexts:

$$\begin{aligned} c_1 c_2 &= m_1^e m_2^e \bmod N \\ &= (m_1 m_2)^e \bmod N. \end{aligned}$$

Therefore,

$$\boxed{c_1 c_2 = \mathsf{Enc}(m_1 m_2)}$$

The Cloud has computed the product of the plaintexts without knowing m_1 or m_2.

RSA Homomorphism

Textbook RSA is **multiplicatively homomorphic**.

RSA as Homomorphic Encryption

The RSA homomorphic property can be summarized as

$$\boxed{\mathrm{Enc}(m_1) \times \mathrm{Enc}(m_2) = \mathrm{Enc}(m_1 m_2)}$$

Therefore, the Cloud can evaluate a multiplicative computation such as

$$m_1 m_2 m_3$$

without decrypting the data:

$$\mathrm{Enc}(m_1)\,\mathrm{Enc}(m_2)\,\mathrm{Enc}(m_3) = \mathrm{Enc}(m_1 m_2 m_3).$$

However, RSA does not naturally provide

$$\boxed{\mathrm{Enc}(m_1 + m_2)}$$

from

$$\mathrm{Enc}(m_1), \mathrm{Enc}(m_2).$$

Thus RSA supports only one useful algebraic operation:

$$\boxed{\text{RSA} \quad \longrightarrow \quad \text{Multiplicative Homomorphism}}$$

Classical Homomorphic Encryption Schemes

ElGamal-based

Classical Homomorphic Encryption: ElGamal

Recall ElGamal encryption over $\mathbb{Z}_p^*$:

$$c = (c_1, c_2) = (g^r,\ m\, y^r).$$

Suppose

$$c^{(1)} = (g^{r_1}, m_1 y^{r_1})$$

and

$$c^{(2)} = (g^{r_2}, m_2 y^{r_2}).$$

The Cloud can multiply the ciphertext components:

$$\begin{aligned} c^{(1)} c^{(2)} &= (g^{r_1 + r_2}, m_1 m_2 y^{r_1 + r_2}) \\ &= \mathrm{Enc}(m_1 m_2). \end{aligned}$$

Therefore,

$$\boxed{\mathrm{Enc}(m_1) \times \mathrm{Enc}(m_2) = \mathrm{Enc}(m_1 m_2)}$$

ElGamal Homomorphism

Standard ElGamal is **multiplicatively homomorphic**.

ElGamal as Homomorphic Encryption

The multiplicative homomorphism can be extended to a sequence of ciphertexts.

For $c_i = \mathrm{Enc}(m_i)$, the Cloud can compute

$$\prod_{i=1}^{t} c_i = \mathrm{Enc}\left(\prod_{i=1}^{t} m_i\right).$$

Thus, multiplication of ciphertexts produces multiplication of plaintexts:

$$\boxed{\prod_{i=1}^{t} \mathrm{Enc}(m_i) = \mathrm{Enc}\left(\prod_{i=1}^{t} m_i\right)}$$

However, from $\mathrm{Enc}(m_1), \mathrm{Enc}(m_2)$, standard ElGamal does not directly provide

$$\boxed{\mathrm{Enc}(m_1 + m_2)}$$

Thus,

$$\boxed{\text{ElGamal} \quad \longrightarrow \quad \text{Multiplicative Homomorphism}}$$

Limitation: Like textbook RSA, standard ElGamal alone cannot evaluate a general computation requiring both addition and multiplication.

Why RSA and ElGamal Are Not Enough

Both textbook RSA and standard ElGamal provide **multiplicative homomorphism**:

$$\boxed{\mathrm{Enc}(m_1)\,\mathrm{Enc}(m_2) = \mathrm{Enc}(m_1 m_2)}$$

However, they do not directly provide homomorphic addition:

$$\boxed{\mathrm{Enc}(m_1) + \mathrm{Enc}(m_2) \neq \mathrm{Enc}(m_1 + m_2)}$$

For the linear model

$$\boxed{y = mx + c},$$

multiplication alone gives

$$\mathrm{Enc}(m)\,\mathrm{Enc}(x) = \mathrm{Enc}(mx).$$

But we also need

$$\mathrm{Enc}(mx) + \mathrm{Enc}(c) = \mathrm{Enc}(mx + c).$$

Conclusion

RSA and ElGamal can handle the special case $c = 0$, but not the general case. We need **homomorphic addition as well**.

Key-Management Issue

Key-Management Challenge in Homomorphic Encryption

Homomorphic encryption allows the Cloud to compute on encrypted data.
But after computation, someone must decrypt the result.
Suppose the Owner encrypts using

$$C = \mathrm{Enc}_{pk}(m).$$

To recover the plaintext,

$$\boxed{m = \mathrm{Dec}_{sk}(C)}$$

Therefore, the party receiving the result must have access to the corresponding secret key sk.

This creates a problem in an outsourced system:

$$\boxed{\text{Who should possess the decryption key?}}$$

- The Cloud should not have it, otherwise it may decrypt the encrypted model or data.
- The User needs it to decrypt the prediction.
- The Owner does not necessarily want to distribute its secret key to every User.

Key Management for Multiple Users

Consider many Users:

$$U_1, U_2, \ldots, U_n.$$

If each User should independently decrypt its own results, a natural approach is to give each User a separate key pair:

$$(pk_1, sk_1),\ (pk_2, sk_2),\ \ldots,\ (pk_n, sk_n).$$

The Owner may encrypt the model separately for each User:

$$\mathrm{Enc}_{pk_1}(M),\ \mathrm{Enc}_{pk_2}(M),\ \ldots$$

This creates additional challenges:

- Key generation and distribution
- Key storage and protection
- User addition and removal
- Key revocation
- Access control

Challenge

Can we design the system so that the Owner can outsource computation without giving its secret key to the Cloud or distributing the same secret key to every User?

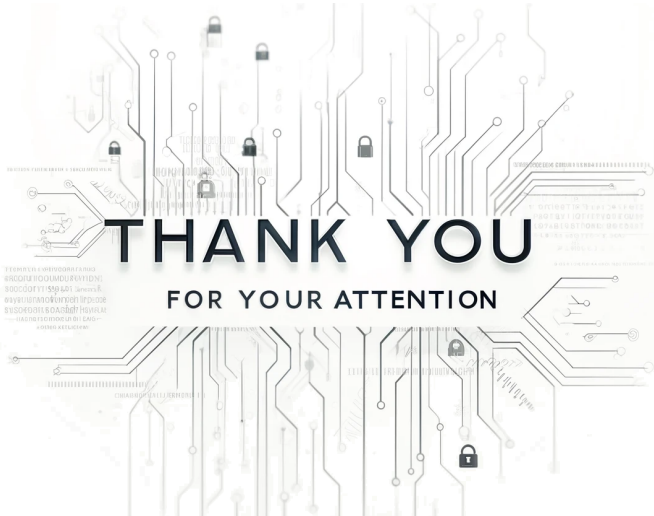
References

Textbook:

1. Douglas R. Stinson and Maura Paterson. *Cryptography: Theory and Practice*. 4th Edition. Chapman & Hall/CRC Press, 2019. ISBN: 9780367148591.

Acknowledgment:

- The figures in this work were generated with the assistance of ChatGPT (OpenAI).



Dr. Laltu Sardar, Assistant Professor, IISER Thiruvananthapuram
laltu.sardar@iisertvm.ac.in, laltu.sardar.crypto@gmail.com
Course webpage: https://laltu-sardar.github.io/courses/26_crypto.html.