NTNU | Norwegian University of Science and Technology

SMILE – Homomorphic Encryption

Dr. Ferhat Ozgur Catak

19.06.2020

Outline

- What is Homomorphic Encryption
- Use-Cases for Homomorphic Encryption Including Border Control
- Types of Homomorphic Encryption Crypto Schemes
- Open Source Libraries Based Solutions for the Untrusted Cloud Systems

What is Homomorphic Encryption?

- **Purpose**: Computation on encrypted data
  - data can remain **confidential** while it is processed in **untrusted environments**.
  - **Homomorphism**: is a structure-preserving map between two algebraic structures
    - describes the transformation of one data set into another while preserving relationships between elements in both sets.
  - Greek words for **"same structure."**
  - **Traditional Encryption vs Homomorphic Encryption**
    - **Homomorphic encryption** allows computation to be performed directly on **encrypted** data without requiring access to a secret key.
    - The result of such a computation remains in **encrypted** form, **and** can at a later point be revealed by the owner of the secret key.

What is Homomorphic Encryption?

No Trusted Third-Parties

- Data remains secure and private in untrusted environments
- The data stays encrypted at all times, which minimizes the likelihood that sensitive information ever gets compromised.

Tradeoff between data usability and privacy

- There is no need to mask or drop any features in order to preserve the privacy of data.
- All features may be used in an analysis, without compromising privacy.

Quantum-safe

- Fully homomorphic encryption schemes are resilient against quantum attacks

What is Homomorphic Encryption?

Limitations

Poor performance

- Between slow computation speed, fully homomorphic encryption remains problematic for computationally-heavy applications

Large Memory Consumption

- Compared to plaintext operations making them sometimes impractical for the database queries

What is Homomorphic Encryption?

operation(plain)

Public key: (23,143)
Private key: (47,143)

Private What is the area? Width:7 height:3

Public

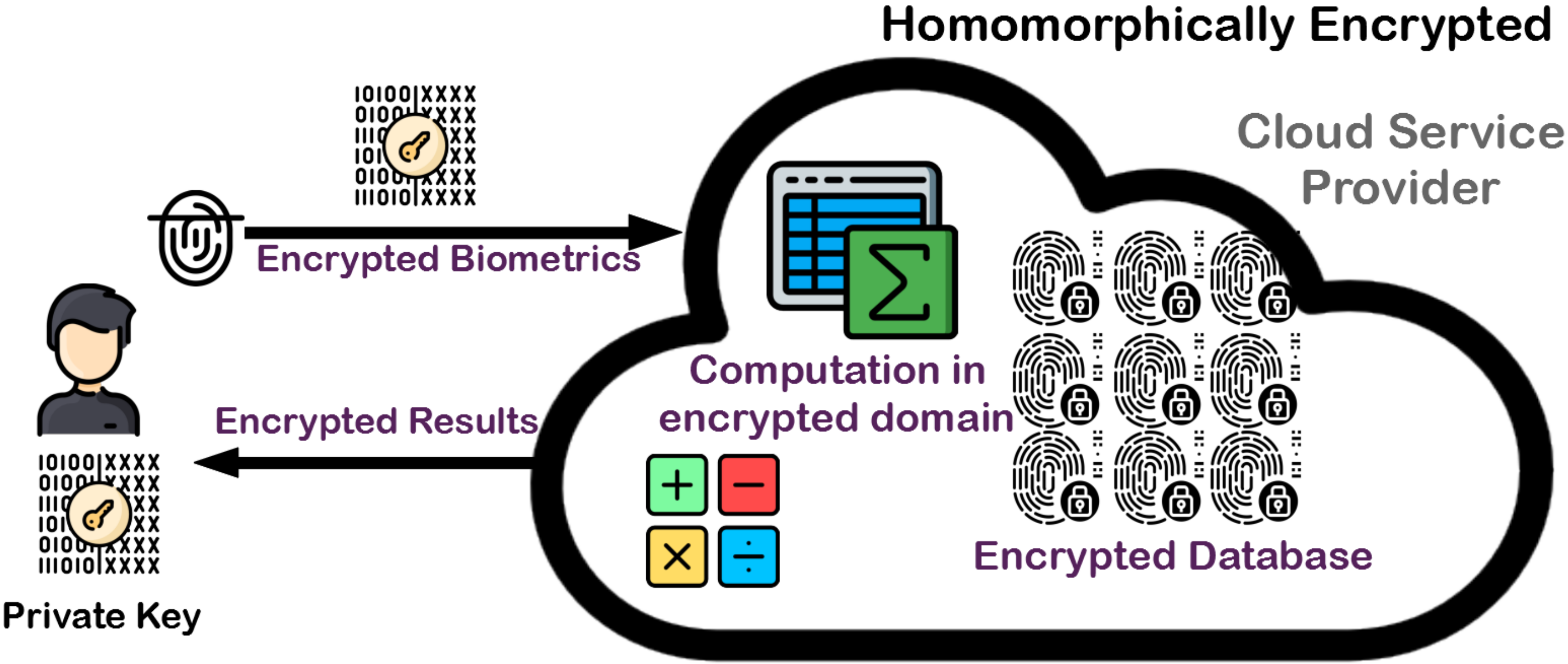
2 × 126 = 252

Enc(width) = width^{e} mod N
Enc(width) = 7^{23} mod 143
Enc(width) = 2

Enc(height) = height^{e} mod N
Enc(height) = 3^{23} mod 143
Enc(height) = 126

area = cipher^{d} mod N
area = 252^{47} mod 123
area = 21
7 × 3 = 21

What is Homomorphic Encryption?



Use-Cases

Medical Records

- Analyze disease/treatment without disclosing
- Search DNA markers without revealing DNA

SPAM Filtering

- Blacklisting Encrypted mails

Biometric Matching

- Encrypted distance metrics
- Similarity

Privacy Preserving Biometric Authentication

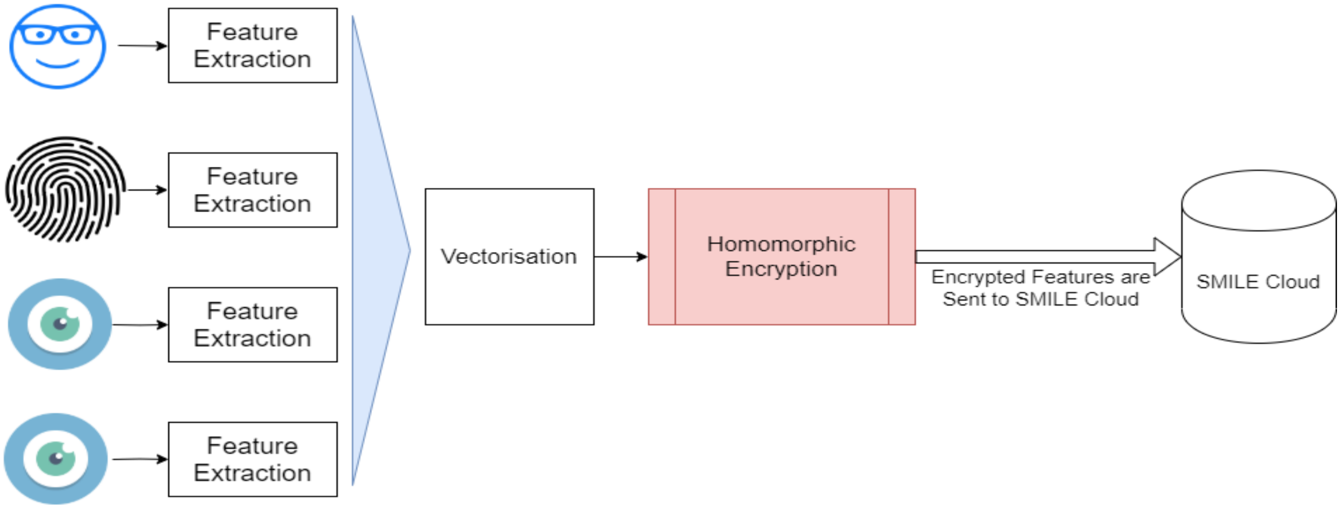
- Strong Authentication Method: Biometrics
  - Represents who you are
  - Unique, Universal, Permanent, and Collectable
- Security Concerns
  - Biometric identity stored at multiple service providers
  - Different proprietary protocols
  - Revealing Biometric ID during authentication

Privacy Preserving Biometric Authentication

- Stolen biometrics
  - Cannot be revoked easily
- Being compromised
  - Captured, cloned or forged
  - Identity theft or individual profiling

Privacy Preserving Biometric Authentication

- Face biometrics, fingerprint biometrics, and iris biometrics are protected by the **homomorphic encryption technique**. As shown in the figure below, feature vectors are extracted using smartphone technology and feature extraction algorithm.
- The homomorphic encryption is an additional layer of security to protect the privacy on top of the transport layer encryption such as SSL and TLS.



Types of Homomorphic Encryption

- Partially Homomorphic
- Somewhat Homomorphic
- Fully Homomorphic

Partially Homomorphic Encryption

- When you can only perform certain mathematical operations on the ciphertext but not others
  - **RSA cryptosystem**: partially homomorphic with respect to **multiplication**
    - [a] x [b] OK
    - [a] + [b] Not Ok
  - **Caesar Cipher**: partially homomorphic with respect to **addition**
    - [a] + [b] OK
    - [a] x [b] Not OK
  - **Paillier:** partially homomorphic with respect to **addition**
    - [a] + [b] OK
    - [a] x b Ok (Encrypted x Plain)
    - [a] x [b] Not OK

Somewhat Homomorphic Encryption

- SHE is more general than PHE in the sense that it supports homomorphic operations with additions and multiplications. The drawback is that you can perform only a limited number homomorphic operations.

Fully Homomorphic Encryption

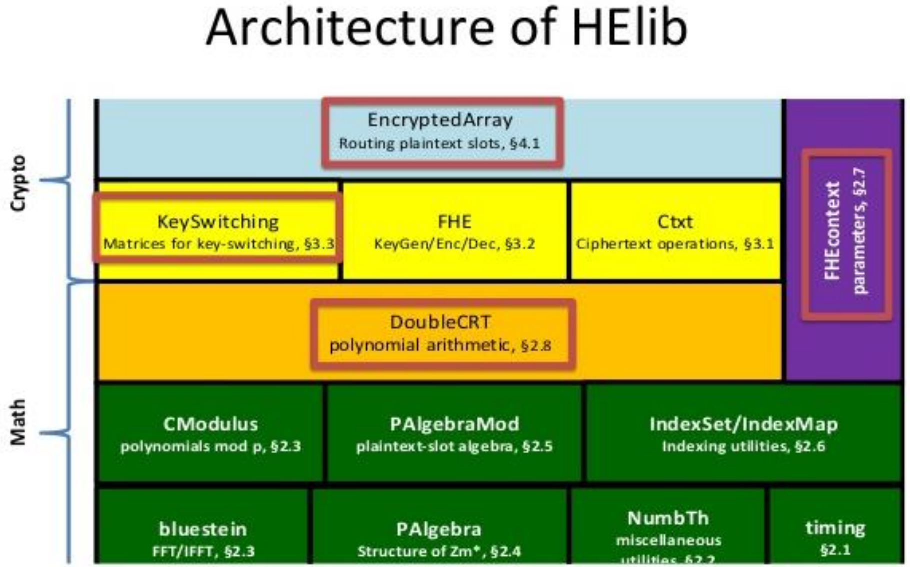
- When you can perform mathematical operations on the ciphertext
  - **BGV** and **BFV**
    - **(Encrypted with Encrypted)**
      - **[a] + [b] OK**
      - **[a] – [b] Ok**
      - **[a] x [b] OK**
    - **(Encrypted with Plain)**
      - **[a] + b OK**
      - **[a] – b Ok**
      - **[a] x b OK**

Open Source Libraries for HE

Microsoft SEAL

Simple Encrypted Arithmetic Library

PyFHEL
- SEAL
- HeLib
- PALISADE



Privacy-Preserving Area Solver

```python
from Pyfhel import Pyfhel

print("1. Creating Context and KeyGen in a Pyfhel Object ")
HE = Pyfhel()                                           # Creating empty Pyfhel object
HE.contextGen(p=65537, m=2048, flagBatching=True)   # Generating context.
HE.keyGen()                                             # Key Generation.

print("2. Encrypting integers")
integer1 = 7
integer2 = 3
ctxt1 = HE.encryptInt(integer1) # Encryption makes use of the public key
ctxt2 = HE.encryptInt(integer2) # For integers, encryptInt function is used.

print("3. Operating with encrypted integers")
ctxtSum = ctxt1 + ctxt2         # `ctxt1 += ctxt2` for quicker inplace operation
ctxtSub = ctxt1 - ctxt2         # `ctxt1 -= ctxt2` for quicker inplace operation
ctxtMul = ctxt1 * ctxt2         # `ctxt1 *= ctxt2` for quicker inplace operation

print("4. Decrypting result:")
resSum = HE.decryptInt(ctxtSum) # Decryption must use the corresponding function
                                #  decryptInt.
resSub = HE.decryptInt(ctxtSub)
resMul = HE.decryptInt(ctxtMul)
print("\taddition:       decrypt(ctxt1 + ctxt2) = ", resSum)
print("\tsubstraction:   decrypt(ctxt1 - ctxt2) =  ", resSub)
print("\tmultiplication: decrypt(ctxt1 * ctxt2) = ", resMul)
```

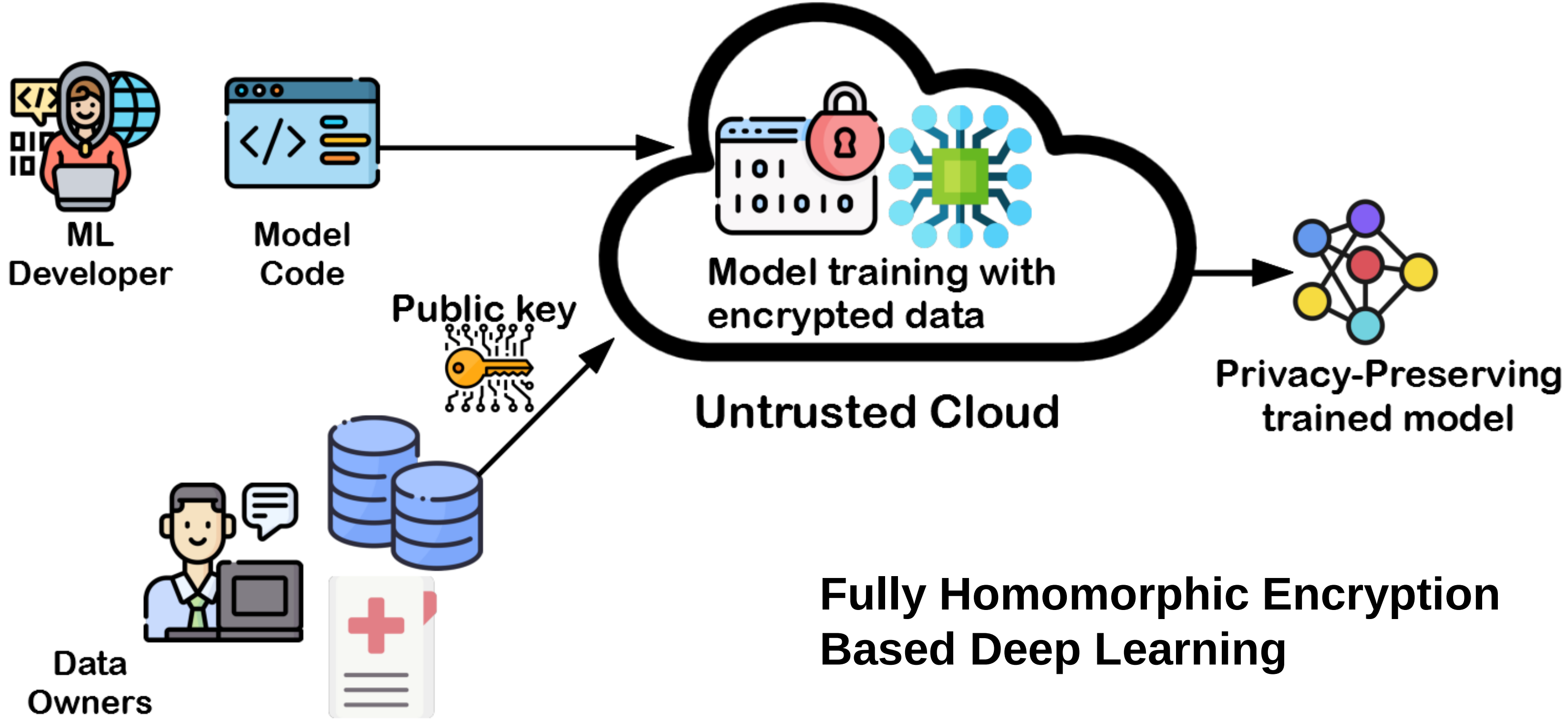
```
1. Creating Context and KeyGen in a Pyfhel Object
2. Encrypting integers
3. Operating with encrypted integers
4. Decrypting result:
    addition:       decrypt(ctxt1 + ctxt2) =  10
    substraction:   decrypt(ctxt1 - ctxt2) =   4
    multiplication: decrypt(ctxt1 * ctxt2) =  21
```

Homomorphic Encryption Applications

- Privacy-Preserving Machine Learning

Homomorphic Encryption + **Secure Machine Learning** = **Privacy Preserving Machine Learning**

Homomorphic Encryption Applications



Fully Homomorphic Encryption Based Deep Learning

Contact Information

Dr. Ferhat Ozgur Catak
Norwegian University of Science and Technology
email: ferhat.o.catak@ntnu.no

Thank You!