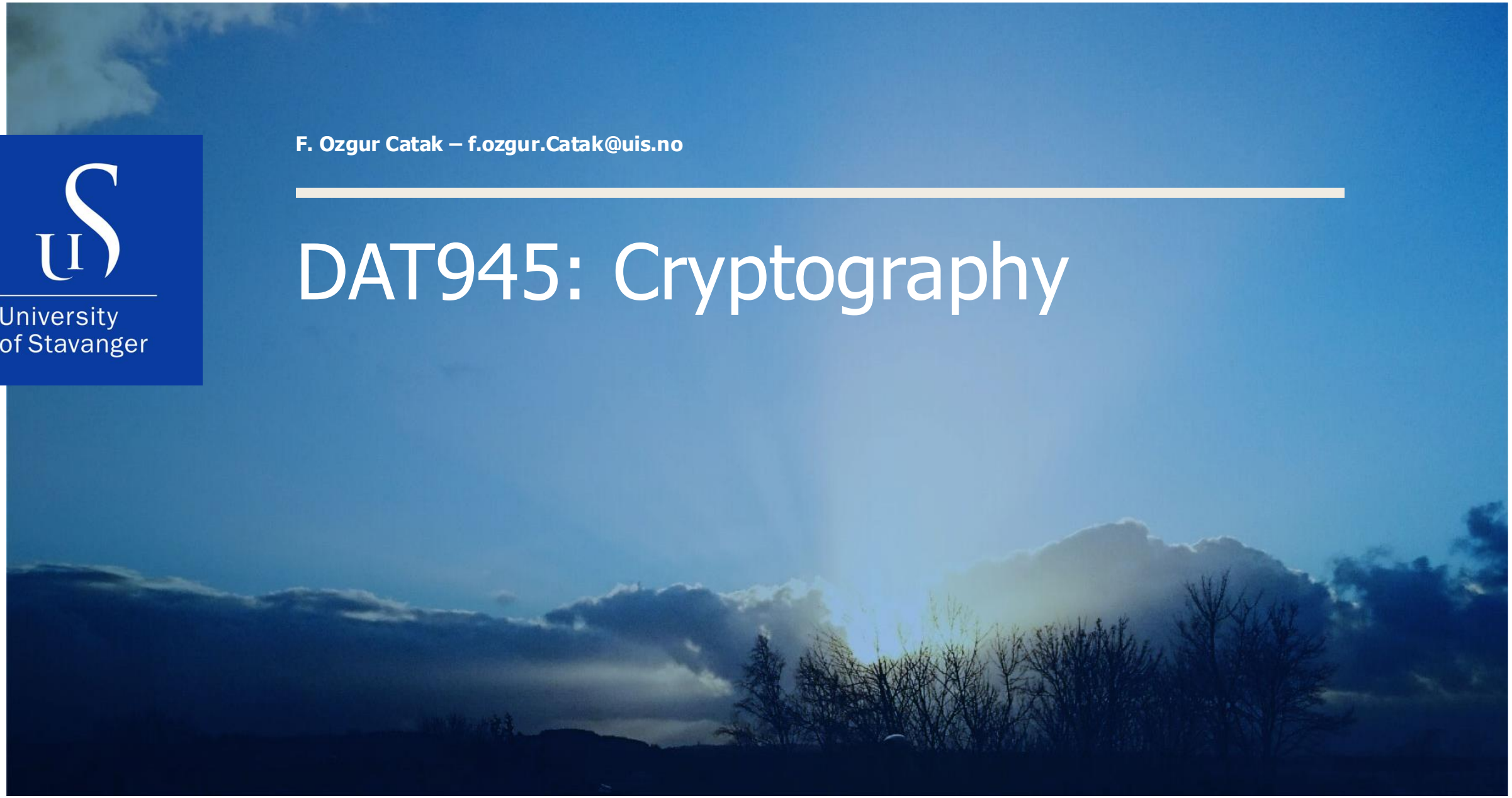
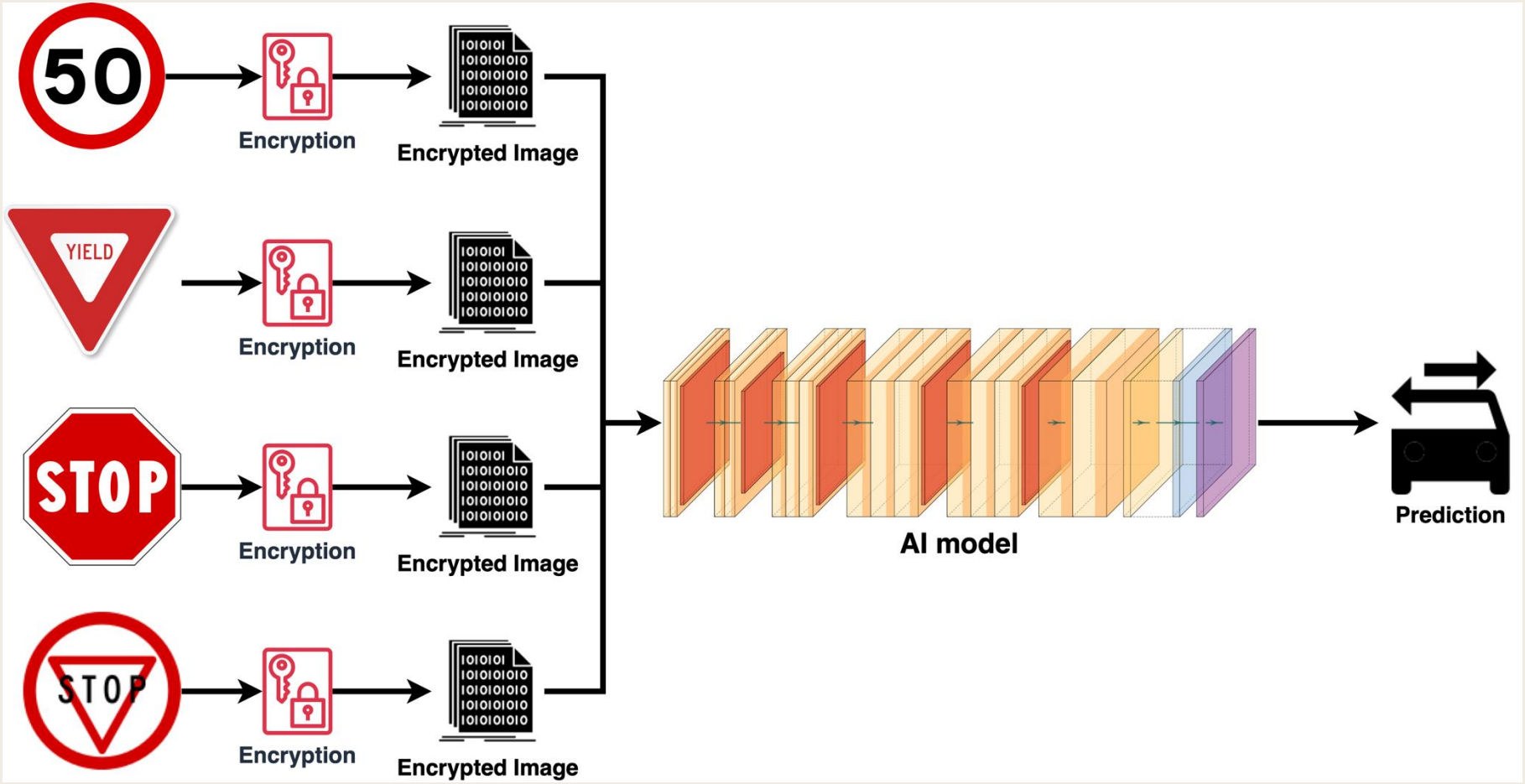
F. Ozgur Catak – f.ozgur.Catak@uis.no

University of Stavanger

DAT945: Cryptography

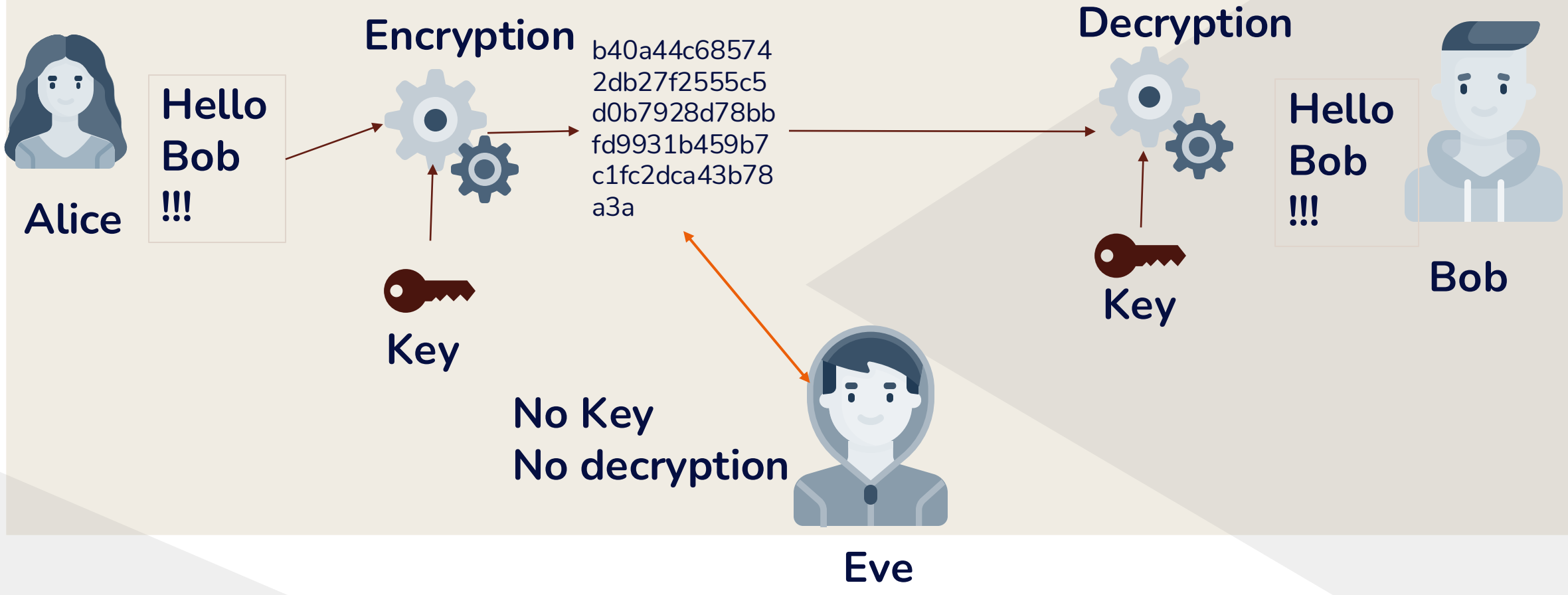


Cryptography in AI



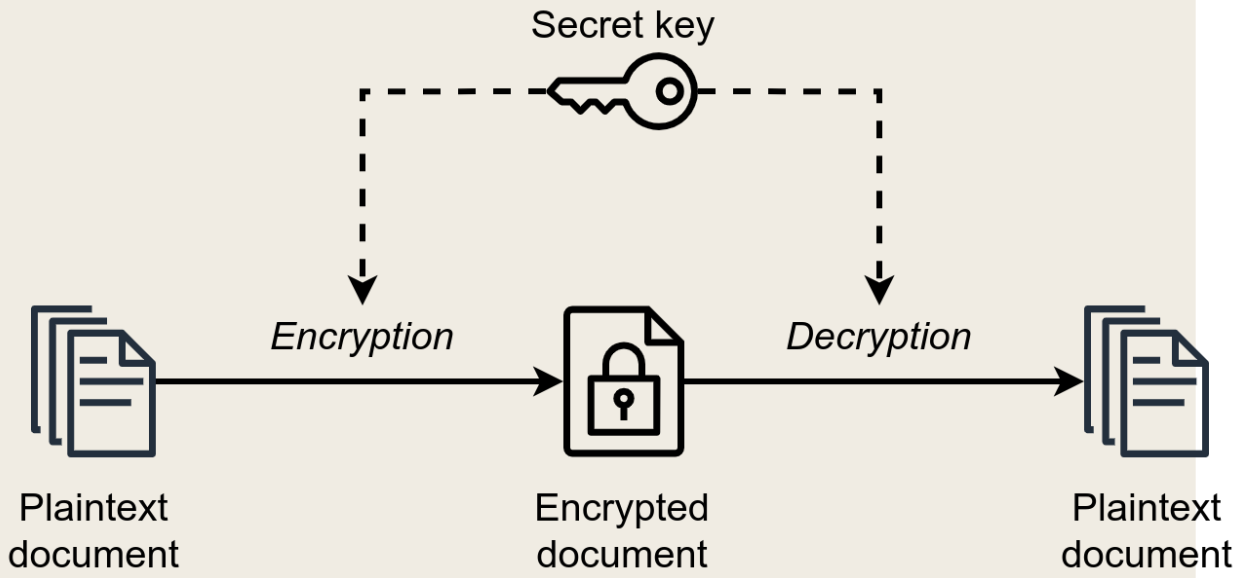
Encryption, Decryption, Crypto Key(s)

- What are they?
- We want to send a senstive message over unsecure channel (like internet)
- We want to store the sensitive data in a server or a harddisk



Basics of Encryption and Decryption

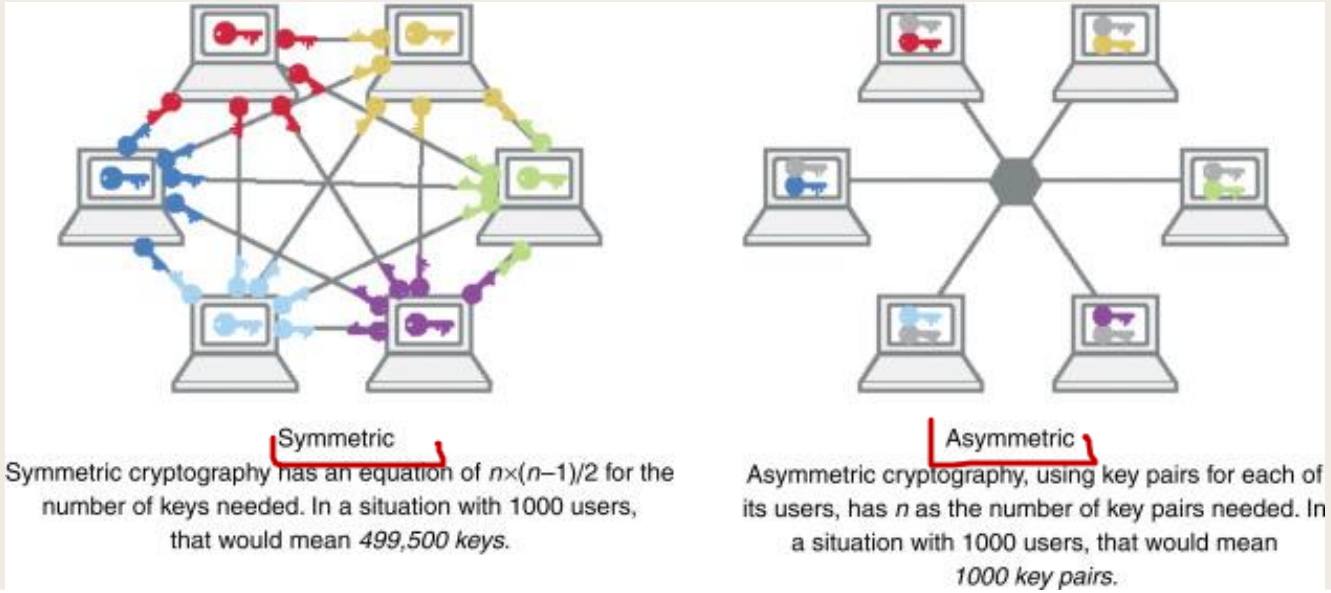
- a single secret key is used for both encryption and decryption of messages.
- This key must be shared between the sender and the recipient to enable secure communication.
- The encryption process can be represented by the mathematical expression $c=Ek(p)$, where k represents the secret key and p is the plaintext message to be encrypted.
- The encryption function Ek is typically a permutation or substitution cipher, which operates on blocks of fixed length.
- The decryption function, on the other hand, is expressed as $p=Dk(c)$, where k is the same secret key used for encryption and c is the ciphertext obtained after encryption.



The security of symmetric encryption algorithms depends on
- the key length
- the quality of the key generation process
- the mathematical operations used for encryption and decryption.

Public-key cryptography

- Everyone generates both a public and private key
- Can be used to distribute keys for symmetric encryption/decryption
- Can also be used to create a digital signature
- Assures confidentiality, authenticity, and non-repudiability



Symmetric

Symmetric cryptography has an equation of $n \times (n-1)/2$ for the number of keys needed. In a situation with 1000 users, that would mean *499,500 keys*.

Asymmetric

Asymmetric cryptography, using key pairs for each of its users, has n as the number of key pairs needed. In a situation with 1000 users, that would mean *1000 key pairs*.

Public Key Cryptography

In a publiy key (asymmetric) cryptosystem all the users have two keys: **public key** and a **private key**

→ these keys are not independent of each other

→ the **private key** can decrypt a message that has been encrypted with the **public key** and vica versa

Everyone can send a decrypted message to a given user using his/her public key and only the given user can decrypt that message using the private key

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

Homomorphism in Cryptography

Let's consider a simple encryption function Enc and a corresponding decryption function Dec. The homomorphic property can be expressed mathematically as:

$$\mathrm{Dec}(\mathrm{Enc}(a) \circ \mathrm{Enc}(b)) = a \circ b$$

Here:

- a and b are plaintext values.
- Enc is the encryption function.
- Dec is the decryption function.
- $\circ$ represents an arbitrary arithmetic operation (e.g., addition or multiplication) in the encrypted domain.

In simpler terms, applying an operation on the encrypted values is equivalent to applying the same operation on the corresponding plaintext values after decryption.

Now, let's break down the concept with an example using addition as the arithmetic operation:

$$\mathrm{Dec}(\mathrm{Enc}(a) \oplus \mathrm{Enc}(b)) = a + b$$

This means that adding the encrypted values of a and b, and then decrypting the result, is the same as adding the plaintext values of a and b.

Similarly, for multiplication:

$$\mathrm{Dec}(\mathrm{Enc}(a) \otimes \mathrm{Enc}(b)) = a \times b$$

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**

Fully Homomorphic Encryption

When you can perform mathematical operations on the ciphertext

BGV, BFV, CKKS

(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

Comparison and Examples

Type of Homomorphic Encryption	Supported Operations	Examples	Use Cases
Partially Homomorphic (PHE)	Addition or Multiplication	Paillier (addition), RSA (multiplication)	Secure vote tallying, sensor data aggregation
Somewhat Homomorphic (SHE)	Limited Additions and Multiplications	BGV (Brakerski-Gentry-Vaikuntanathan)	Simple statistical analysis on encrypted data
Fully Homomorphic (FHE)	Unlimited Additions and Multiplications	Gentry's FHE, BGV with bootstrapping	Complex computations like encrypted ML, secure multi-party computation

Paillier Partially Homomorphic Encryption

```
val1            : 0.6938852677239302
val2            : 0.45216488110666964
**************************************************
Plain domain addition operation
val1 + val2 : 1.1460501488305999
Plain domain multiplication operation
val1 * val2 : 0.31375054958206056
**************************************************
Encryption
Encrypted val1-1            : len( 1850 ) 1173054473 ... 2158203387
Encrypted val1-1            : len( 1849 ) 8738752806 ... 2573986158
--------------------------------------------------
Encrypted val2-1            : len( 1849 ) 1064589880 ... 4500701565
Encrypted val2-2            : len( 1849 ) 7418872070 ... 8048295908
**************************************************
Encrypted sum1_1_enc        : len( 1850 ) 1155738019 ... 6974619344
Encrypted sum1_2_enc        : len( 1849 ) 5468292477 ... 2140675507
--------------------------------------------------
Encrypted sum2_1_enc        : len( 1850 ) 1147157501 ... 3975436517
Encrypted sum2_2_enc        : len( 1849 ) 5322453946 ... 5897549084
--------------------------------------------------
Encrypted sum2_enc          : len( 1848 ) 2209235844 ... 2115361092
Encrypted sum2_enc          : len( 1849 ) 6916582660 ... 0259314695
**************************************************
Decrypted sum1_1_dec        : 1.1460501488305999
Decrypted sum1_2_dec        : 1.1460501488305999
--------------------------------------------------
Decrypted sum2_1_dec        : 1.1460501488305999
Decrypted sum2_2_dec        : 1.1460501488305999
--------------------------------------------------
Decrypted sum1_1_dec        : 1.1460501488305999
Decrypted sum3              : 1.1460501488305999
**************************************************
Decrypted mult1_1_enc       : 0.31375054958206056
```

Jupyter Notebook

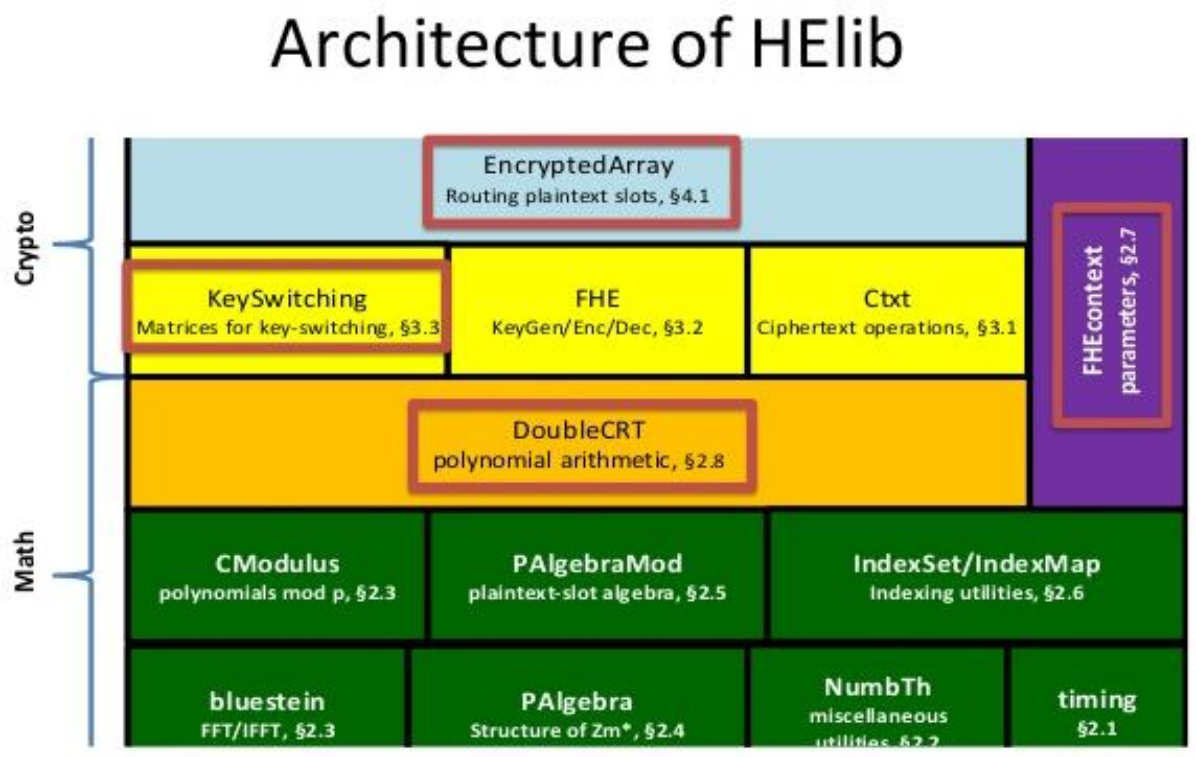
Open Source Libraries for HE

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
```

Exact vs Approximate - Integers vs Floats

Exact Homomorphism

```
resSum = HE.decryptInt(ctxtSum)

byte_data1 = cx1.__bytes__()
hex_string1 = ''.join([hex(byte)[2:].zfill(2) for byte in byte_data1])

byte_data2 = cx2.__bytes__()
hex_string2 = ''.join([hex(byte)[2:].zfill(2) for byte in byte_data2])

byte_data_sum = ctxtSum.__bytes__()
hex_string_sum = ''.join([hex(byte)[2:].zfill(2) for byte in byte_data_sum])

print('-'*50,'\nEncrypted domain operations')
print('Enc(x1) \t:',hex_string1[:20],'...',hex_string1[-20:])
print(' '*40,'+')
print('Enc(x2) \t:',hex_string2[:20],'...',hex_string2[-20:])
print(' '*40,'=')
print('Enc(x1+x2)\t:',hex_string_sum[:20],'...',hex_string_sum[-20:])
print('Dec(Enc(x1+x2))\t:', resSum[0])
```

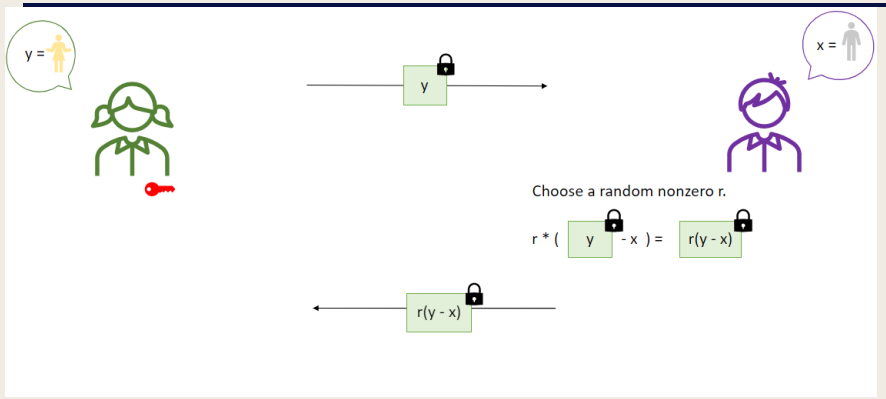
```
--------------------------------------------------
Plain domain operations
Plain operation: [127] + [-2] = [125]
--------------------------------------------------
Encrypted domain operations
Enc(x1)         : 5ea11004000000007100 ... 0000eaf97fc7291d0000
                                           +
Enc(x2)         : 5ea11004000000007100 ... 010038a10aa42fcc0100
                                           =
Enc(x1+x2)      : 5ea11004000000007100 ... 0000229b8a6b59e90100
Dec(Enc(x1+x2)) : 125
```

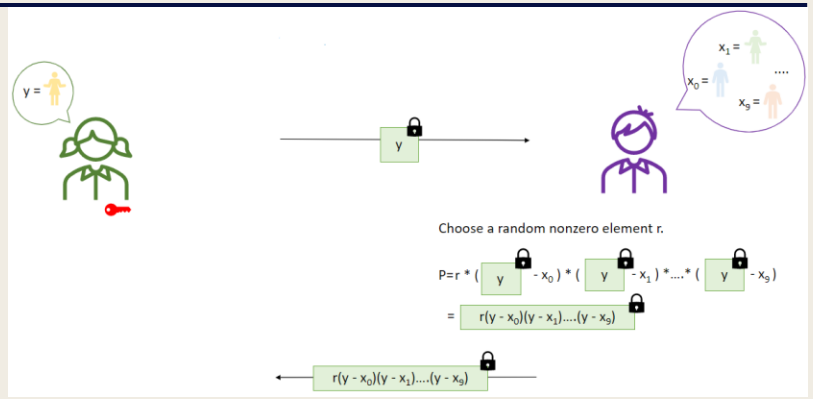
Approximate Homomorphism

```
resSum = HE.decryptFrac(ctxtSum)

byte_data1 = cx1_float.__bytes__()
hex_string1 = ''.join([hex(byte)[2:].zfill(2) for byte in byte_data1])

byte_data2 = cx1_float.__bytes__()
hex_string2 = ''.join([hex(byte)[2:].zfill(2) for byte in byte_data2])

byte_data_sum = ctxtSum.__bytes__()
hex_string_sum = ''.join([hex(byte)[2:].zfill(2) for byte in byte_data_sum])

print('-'*50,'\nEncrypted domain operations')
print('Enc(x1) \t:',hex_string1[:20],'...',hex_string1[-20:])
print(' '*40,'+')
print('Enc(x2) \t:',hex_string2[:20],'...',hex_string2[-20:])
print(' '*40,'=')
print('Enc(x1+x2)\t:',hex_string_sum[:20],'...',hex_string_sum[-20:])
print('Dec(Enc(x1+x2))\t:', resSum[0])
print('-'*50,'\nPlain domain operations')
print("Plain operation:", arr_x[0], '+', arr_y[0], '=', str(arr_x + arr_y))
```

```
--------------------------------------------------
Encrypted domain operations
Enc(x1)         : 5ea11004000000007100 ... 0000bccbe73700000000
                                           +
Enc(x2)         : 5ea11004000000007100 ... 0000bccbe73700000000
                                           =
Enc(x1+x2)      : 5ea11004000000007100 ... 00005a55263700000000
Dec(Enc(x1+x2)) : 5.8598724419438675
--------------------------------------------------
Plain domain operations
Plain operation: 3.141592653589793 + 2.718281828459045 = [5.85987448]
```

Example: Private Set Intersection





 https://bit-ml.github.io/blog/post/private-set-intersection-an-implementation-in-python/

Neural Cryptography: Similar Discussion in Cryptography

- A branch of cryptography dedicated to analyzing the application of stochastic algorithms, especially **ANN algorithms, for use in encryption and cryptanalysis.**

LEARNING TO PROTECT COMMUNICATIONS WITH ADVERSARIAL NEURAL CRYPTOGRAPHY

Martín Abadi and David G. Andersen *
Google Brain

ABSTRACT

We ask whether neural networks can learn to use secret keys to protect information from other neural networks. Specifically, we focus on ensuring confidentiality properties in a multiagent system, and we specify those properties in terms of an adversary. Thus, a system may consist of neural networks named Alice and Bob, and we aim to limit what a third neural network named Eve learns from eavesdropping on the communication between Alice and Bob. We do not prescribe specific cryptographic algorithms to these neural networks; instead, we train end-to-end, adversarially. We demonstrate that the neural networks can learn how to perform forms of encryption and decryption, and also how to apply these operations selectively in order to meet confidentiality goals.

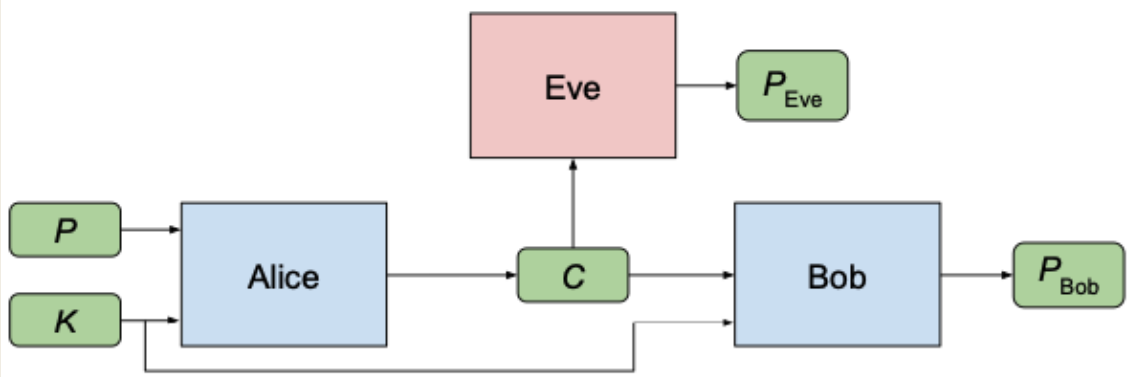


Figure 1: Alice, Bob, and Eve, with a symmetric cryptosystem.

https://arxiv.org/pdf/1610.06918.pdf

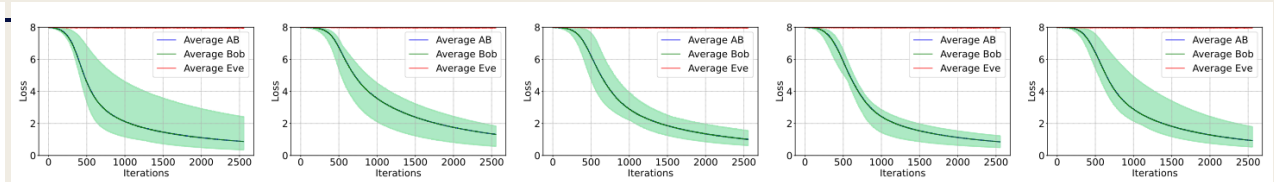
Neural Networks Meet Elliptic Curve Cryptography: A Novel Approach to Secure Communication

Neural Networks Meet Elliptic Curve Cryptography: A Novel Approach to Secure Communication

Mina Cecilie Wøien*, Ferhat Ozgur Catak*
*Electrical Eng. and Computer Science
University of Stavanger
Rogaland, Norway
mc.woien@stud.uis.no, f.ozgur.catak@uis.no

Murat Kuzlu
Batten College of Eng. and Tech.
Old Dominion University
Norfolk, VA, USA
mkuzlu@odu.edu

Umit Cali
School of Physics, Eng. and Technology
University of York
York, UK
umit.cali@york.ac.uk



(a) Curve: secp224r1 (b) Curve: secp256k1 (c) Curve: secp256r1 (d) Curve: secp384r1 (e) Curve: secp521r1

Fig. 5. Loss functions for the ABE model, Bob and Eve, with different curves.

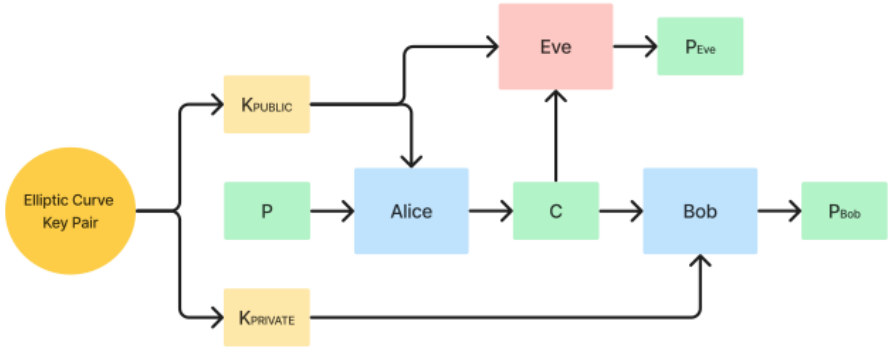


Fig. 3. Overview of the system, the interaction between Alice, Bob, and Eve.

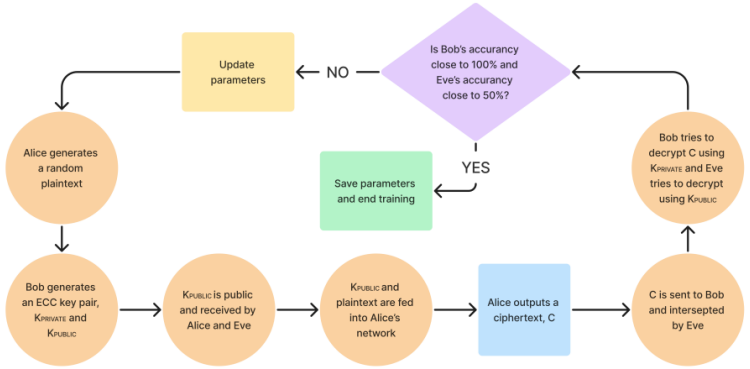
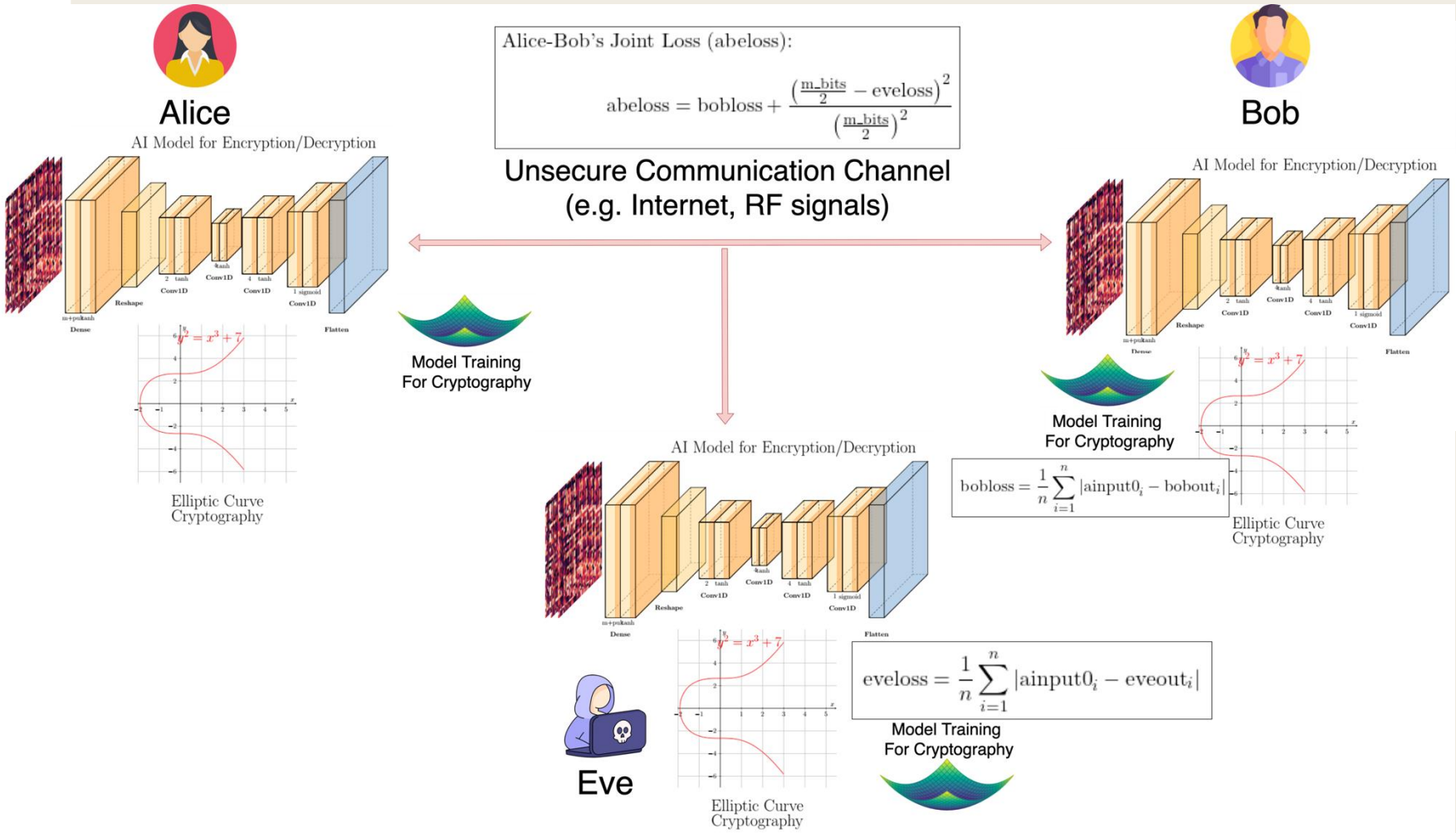
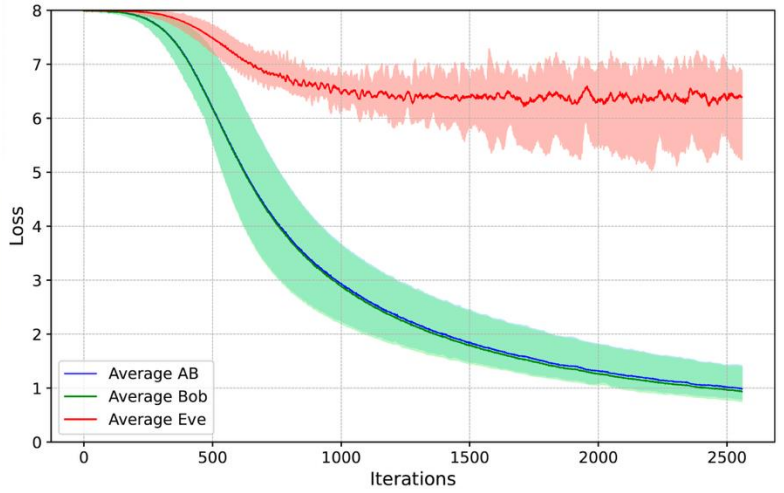


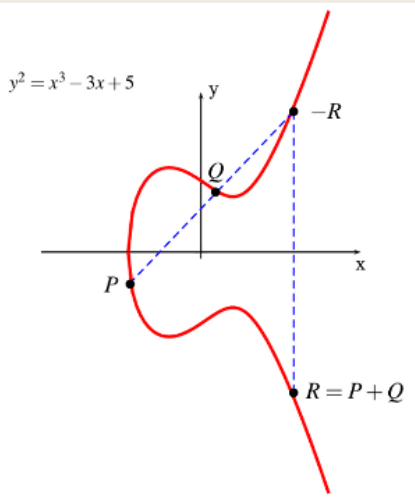
Fig. 4. Training flow of the neural networks.

Wøien, M. C., Catak, F. O., Kuzlu, M., & Cali, U. (2024, December). Neural Networks Meet Elliptic Curve Cryptography: A Novel Approach to Secure Communication. In *2024 IEEE Virtual Conference on Communications (VCC)* (pp. 1-6). IEEE.

Our Contribution: Asymmetric Cryptography







https://github.com/ocatak/Neural-Cryptography

Neural Cryptography: Arithmetic Operations

- Complex Models: GPT, DALL-E, Whisper

Input

Andrew is free from 11 am to 3 pm, Joanne is free from noon to 2 pm and then 3:30 pm to 5 pm. Hannah is available at noon for half an hour, and then 4 pm to 6 pm. What are some options for start times for a 30 minute meeting for Andrew, Hannah, and Joanne?

Output

Andrew: 11 am - 3 pm
Joanne: 12 pm - 2 pm, 3:30 pm - 5 pm
Hannah: 12 pm - 12:30 pm, 4 pm - 6 pm

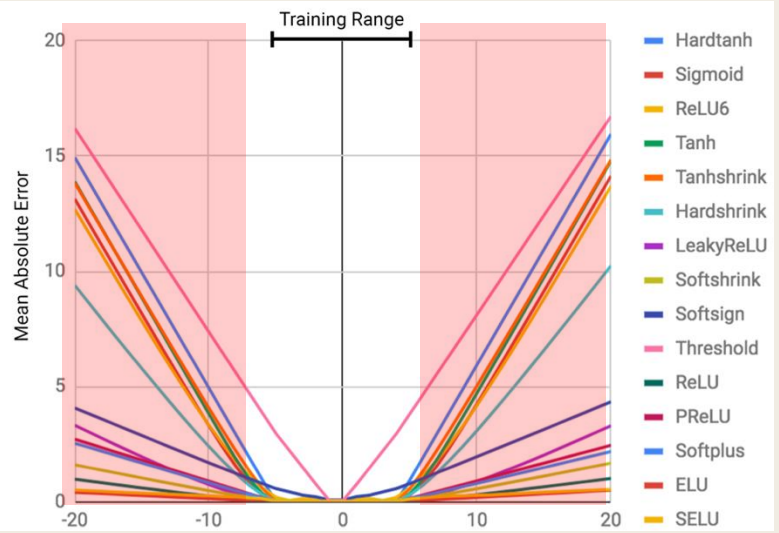
Common availability for a 30-minute meeting: 12 pm - 12:30 pm

We need simple operations also: The number sense and Arithmetic Operations (+, - , / , *)

Numerical reasoning appears widespread across species, indicating a fundamental aspect of intelligence.

Numerical Extrapolation Failures in Neural Networks

- Example: **Identity function**
  - *f(X) = X*
  - MLPs learn the identity within their training range, with error sharply rising beyond it.
  - The model is trained between -5 and 5, and average their ability to encode the numbers between -20 and 20



https://arxiv.org/pdf/1808.00508.pdf

Neural Arithmetic Logical Units (NALU)

Interpolation

```
for _ in range(10):
    x1 = np.random.rand()*10
    x2 = np.random.rand()*10
    # Test the model for addition
    x_test_add = tf.constant([[x1, x2]], dtype=tf.float32)
    result_add = model.predict(x_test_add, verbose=0)

    print("-"*30)
    print("Original :", x1, "+", x2, "\t=", (x1+x2))
    print("Predicted:", x1, "+", x2, " \t=", result_add[0][0])
```

```
------------------------------
Original : 0.03223643801577247 + 0.905602319794282      = 0.9378387578100544
Predicted: 0.03223643801577247 + 0.905602319794282      = 0.9378373
------------------------------
Original : 6.699403185449919 + 9.181966281418722        = 15.881369466868641
Predicted: 6.699403185449919 + 9.181966281418722        = 15.881344
------------------------------
Original : 5.570541726870961 + 9.196359999324198        = 14.766901726195158
Predicted: 5.570541726870961 + 9.196359999324198        = 14.766877
------------------------------
Original : 0.15014735627107978 + 3.5919890355623507     = 3.7421363918334305
Predicted: 0.15014735627107978 + 3.5919890355623507     = 3.7421305
```

Extrapolation

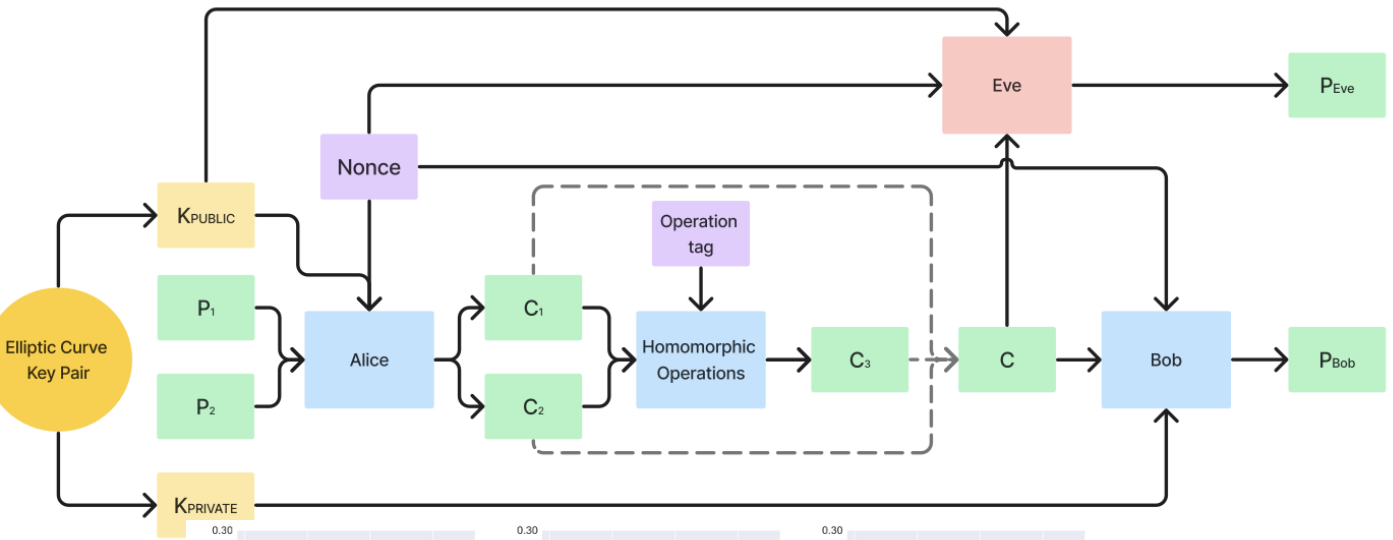
```
for _ in range(10):
    x1 = np.random.rand()*1000
    x2 = np.random.rand()*1000
    # Test the model for addition
    x_test_add = tf.constant([[x1, x2]], dtype=tf.float32)
    result_add = model.predict(x_test_add, verbose=0)

    print("Predicted result for addition:", x1, "+", x2, " \t=", result_add[0][0])
    print("Original result for addition:", x1, "+", x2, "\t=", (x1+x2))
    print("-"*30)
```

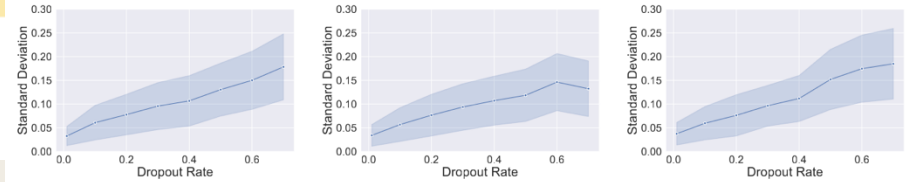
```
Predicted result for addition: 174.00640392449972 + 896.3278818666447    = 1070.3325
Original result for addition: 174.00640392449972 + 896.3278818666447     = 1070.3342857911443
------------------------------
Predicted result for addition: 921.806429604309 + 478.7109349687702      = 1400.5151
Original result for addition: 921.806429604309 + 478.7109349687702       = 1400.5173645730792
------------------------------
Predicted result for addition: 828.8336957801187 + 961.1299482327117     = 1789.9607
Original result for addition: 828.8336957801187 + 961.1299482327117      = 1789.9636440128304
------------------------------
Predicted result for addition: 955.3939866435168 + 114.14540287514829    = 1069.5377
Original result for addition: 955.3939866435168 + 114.14540287514829     = 1069.539389518665
```

We will combine Neural Cryptography with NALU to create Homomorphic Neural Cryptography Model

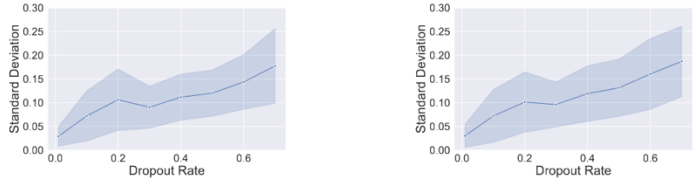
University of Stavanger

Neural Cryptography with Homomorphism





(a) Curve: secp224r1 (b) Curve: secp256k1 (c) Curve: secp256r1



(d) Curve: secp384r1 (e) Curve: secp521r1

Figure 4.2: Mean (solid line) and standard deviation (shaded area) of the standard deviation of five ciphertexts created with the same key and plaintext for different elliptic curves.

Table 4.1: Bob's decryption accuracy with different curves and dropout rates.

Plaintext	Curve	Rate							
		0.01	0.1	0.2	0.3	0.4	0.5	0.6	0.7
P1	secp224r1	100	100	100	100	100	100	99.96	99.99
	secp256k1	100	100	99.99	100	99.93	99.87	99.75	100
	secp256r1	100	100	100	100	100	100	100	100
	secp384r1	100	99.85	99.78	99.94	100	100	99.99	100
	secp521r1	100	99.8	99.73	100	100	100	100	99.99
P2	secp224r1	100	100	100	100	100	100	99.9	100
	secp256k1	100	100	100	99.96	99.96	99.9	99.86	100
	secp256r1	100	100	100	100	100	100	100	100
	secp384r1	100	99.82	99.87	99.99	100	100	100	100
	secp521r1	100	99.85	99.79	100	100	100	100	100
P1+P2	secp224r1	99.99	99.93	99.9	99.92	99.82	99.82	99.5	99.58
	secp256k1	99.93	99.92	99.83	99.72	99.58	99.27	99.11	74.73
	secp256r1	99.92	99.96	99.94	99.89	99.92	99.9	99.89	99.73
	secp384r1	99.99	98.9	99.43	99.86	99.86	99.85	99.86	99.85
	secp521r1	100	98.97	99.93	99.87	99.89	99.94	99.94	99.82
P1×P2	secp224r1	96.79	93.22	97.8	98.09	97.96	98.35	98.19	97.18
	secp256k1	91.34	97.07	98.06	97.64	98.16	93.25	97.98	79.83
	secp256r1	97.04	93.6	98.3	98.05	98.24	95.28	95.31	98.02
	secp384r1	97.04	93.05	97.41	98.27	98.3	98.27	98.05	91.74
	secp521r1	96.71	95.2	97.74	98.33	98.31	96.68	95.84	97.43

https://github.com/ocatak/mina-he-ai

R1: Channel Estimation in 5G Networks with **Federated Learning** & **Homomorphic Encryption**

- Accepted in IEEE COMSOC VCC Conference
- MATLAB – LTE and 5G toolboxes. The the physical downlink shared channel (PDSCH) carries the user data and is transmitted from the base station (eNodeB or gNB) to the user equipment (UE).

DM-RS Configuration:
- Base station configures DM-RS positions within PDSCH.
- Known DM-RS pattern and structure.

PDSCH and DM-RS Transmission:
- Base station transmits PDSCH with embedded DM-RS symbols.

Channel Estimation Model:
- Receiver captures the received signal (PDSCH + DM-RS).
- Constructs a channel estimation model using the known DM-RS symbols.

DM-RS Detection:
- Receiver performs DM-RS detection.
- Extracts received DM-RS symbols using known pattern and structure.

Channel Estimation:
- Receiver applies signal processing techniques for estimation.
- Interpolation, equalization, and specific algorithms based on channel model.

Channel Compensation:
- Estimated channel info used for compensating impairments.
- Enhances data detection, demodulation, and decoding.

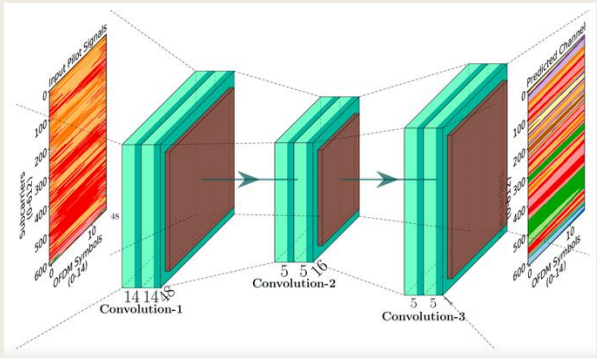
TABLE I
EACH OF THE CHANNEL CHARACTERISTIC PARAMETERS AND VALUES

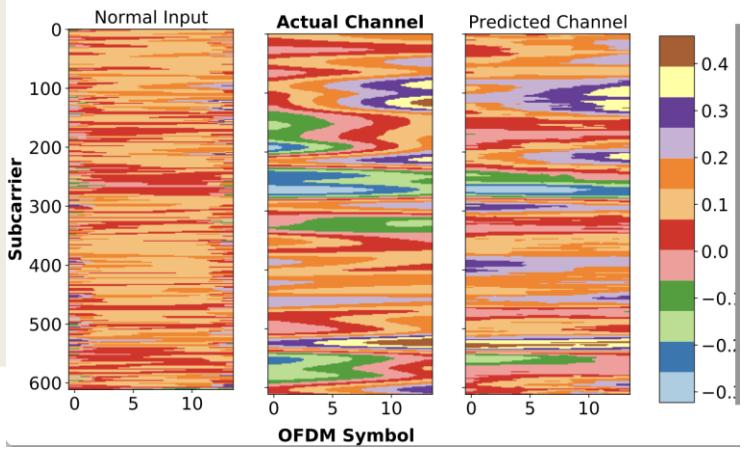
Channel Parameter	Value
Delay Profile	TDL-A, TDL-B, TDL-C, TDL-D, TDL-E
Delay Spread	1-300 ns
Maximum Doppler Shift	5-400 Hz
NFFT	1024
Sample Rate	30720000
Symbols Per Slot	14
Windowing	36
Slots Per Subframe	2
Slots Per Frame	20
Polarization	Co-Polar
Transmission Direction	Downlink
Num. Transmit Antennas	1
Num. Receive Antennas	1
Fading Distribution	Rayleigh
Modulation	16QAM

F1	F2	F3	F4
0.15+0.90j	0.26+0.90j	0.32+0.90j	0.41+0.88j
-0.39-0.84j	-0.46-0.83j	-0.55-0.79j	-0.61-0.72j
-0.26-0.89j	-0.38-0.87j	-0.44-0.84j	-0.50-0.80j
-0.56+0.78j	-0.45+0.82j	-0.37+0.89j	-0.28+0.89j
⋮	⋮	⋮	⋮
-0.86-0.43j	-0.88-0.35j	-0.87-0.23j	-0.89-0.12j

⇓

F1-1	F1-2	F2-1	F2-2	F3-1	F3-2	F4-1	F4-2
0.15	0.90	0.26	0.90	0.32	0.90	0.41	0.88
-0.39	0.84	-0.46	0.83	-0.55	0.79	-0.61	0.72
-0.26	0.89	-0.38	0.87	-0.44	0.84	-0.50	0.80
-0.56	0.78	-0.45	0.82	-0.37	0.89	-0.28	0.89
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
-0.86	0.43	-0.88	0.35	-0.87	0.23	-0.89	0.12





System Model

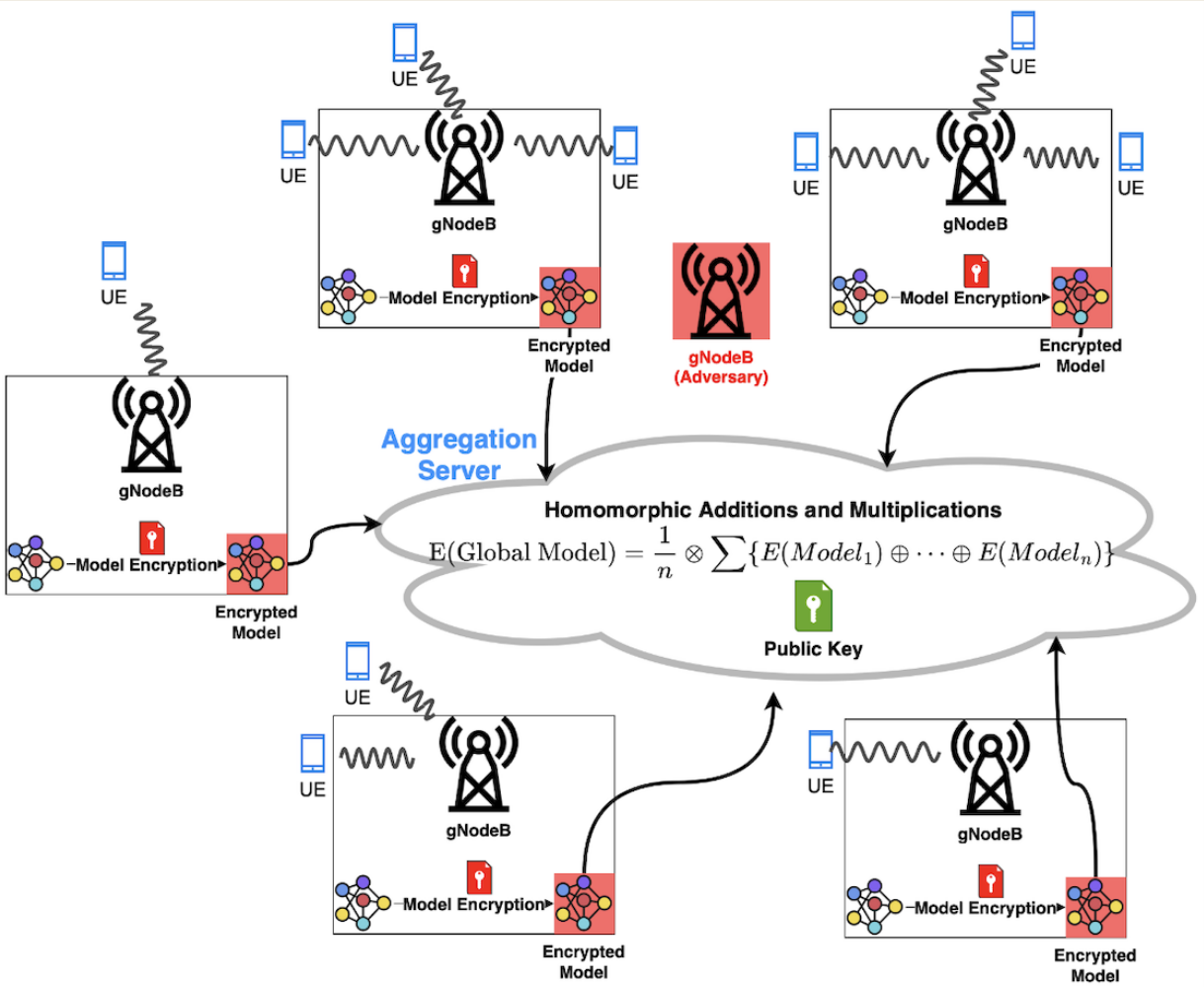
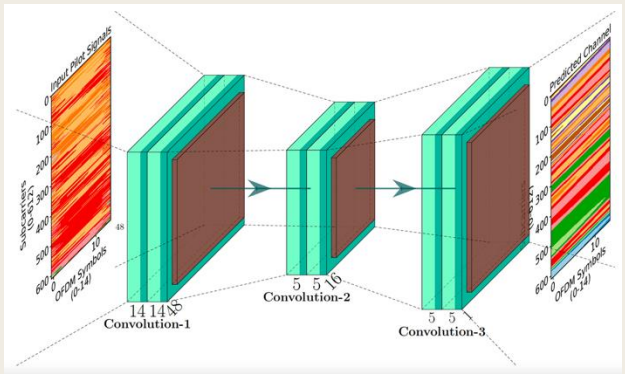


Fig. 1. System overview of privacy-Preserving FL Channel Estimation Model.



- Each gNodeB has same model architecture
- There is no data sharing between nodes
- Only encrypted model parameters will be shared to the aggregation server
- Aggregation server is responsible to perform homomorphic addition and multiplication.
- **The only model weights as ciphertext are shared.**

Cryptographic FL Algorithms Client/Server Side

Algorithm 1 Training Phase

Require: (F: Global model, R: learning rate, T: the number of FL iterations, N: the number of clients)

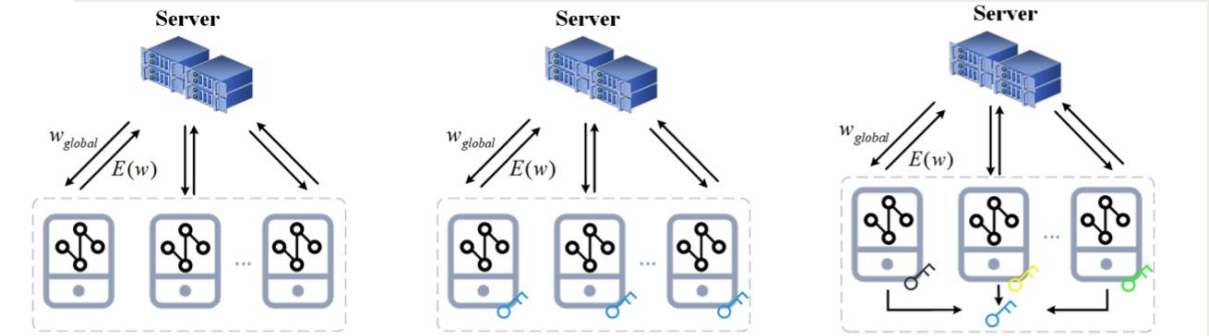
0: Download the initial model, F, from the server
0: **for all** $t \in T$ **do**
0: **for all** $i \in N$ **do**
0: $F^{(i)}, W^{(i)} \longleftarrow SGD(R, \mathcal{D}^{(i)})$ Train the model with local dataset $\mathcal{D}^{(i)}$
0: $W^{(i)} \leftarrow \mathcal{E}(W^{(i)}, key_{pub})$ Encrypt the local weights
0: Upload encrypted local model weight vector $W^{(i)}$ to the server
0: **end for**
0: **end for**
0: **return** local model $F^{(i)}$ =0

Algorithm 2 Encrypted FL in the server

Require: $(W^{(1)}, \ldots, W^{(|N|)}, \eta, m)$

0: Initialize global model parameters
0: **for** i = 1, 2, …, N **do**
0: Download all $|N|$ encrypted local model weights
0: $W \longleftarrow \underbrace{\mathcal{E}(\frac{1}{|N|}, key_{pub})}_{\text{Encrypt the number of clients}} \otimes \underbrace{\sum_{i=1}^{|N|} W^{(i)}}_{\text{Homomorphic addition}}$
0: Broadcast the encrypted global model weight vector W to all gNodeBs
0: **end for**
0: **return** encrypted global model weight vector W =0

R3: Multi-Key Homomorphism with FL



(a) Federated learning without encryption

(b) Homomorphic encryption (HE)-based federated learning using the same keys for all parameters

(c) Multi-key HE-based federated learning using distinct encryption keys. Decryption is possible only combining all keys

Figure (1) Different federated learning schemes with various types of encryption. We propose multi-key homomorphic encryption in which parameters are not able to decrypt other parameters encrypted weights.

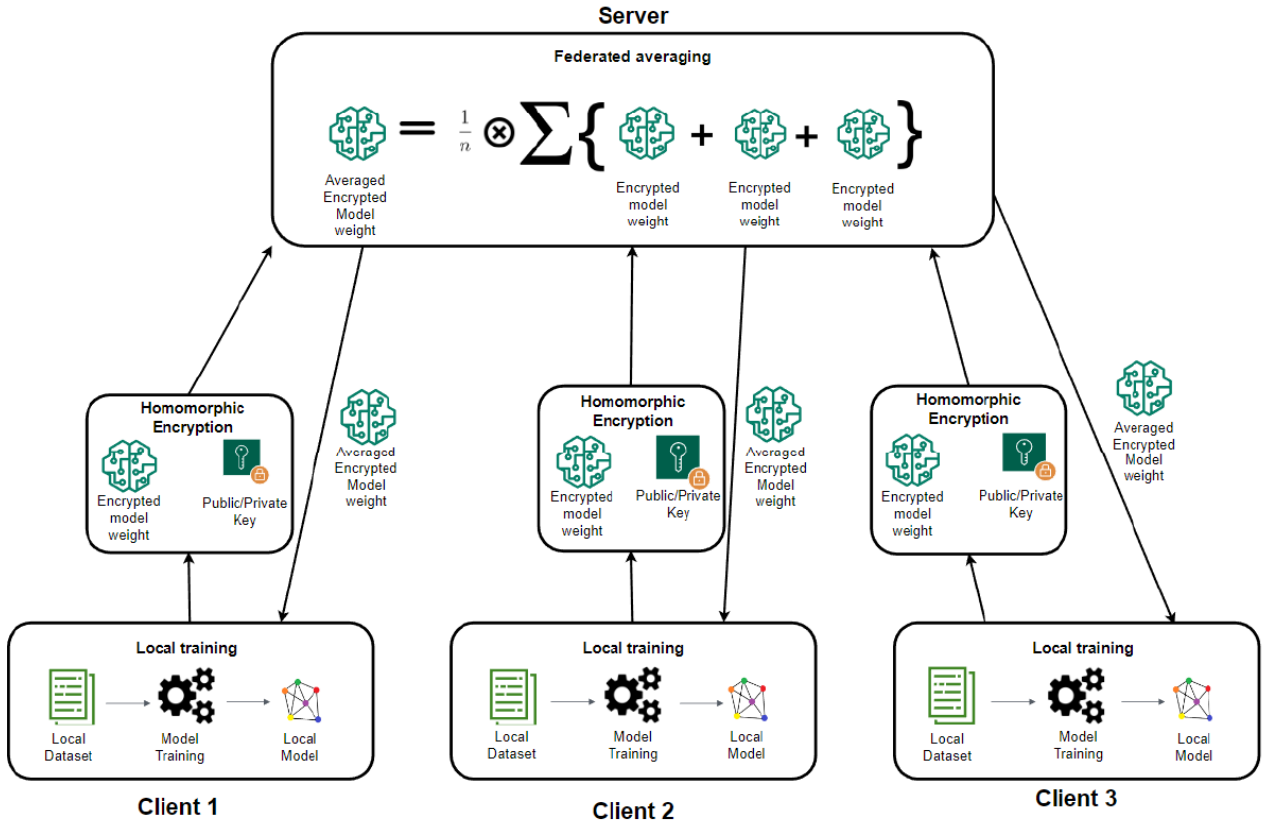


Figure 1. Centralized FL with privacy preserving encryption.

Walskaar, I.; Tran, M.C.; Catak, F.O. A Practical Implementation of Medical Privacy-Preserving Federated Learning Using Multi-Key Homomorphic Encryption and Flower Framework. *Cryptography* **2023**, *7*, 48. https://doi.org/10.3390/cryptography7040048

Implementation

Flower A Friendly Federated Learning Framework

A unified approach to federated learning, analytics, and evaluation. Federate any workload, any ML framework, and any programming language.

Take the tutorial to learn federated learning

GitHub

https://github.com/ocatak/MSc-thesis-xmkckks

A Practical Implementation of Medical Privacy-Preserving Federated Learning Using Multi-Key Homomorphic Encryption and Flower Framework

Ivar Walskaar †, Minh Christian Tran † and Ferhat Ozgur Catak *,†

README.md

Prototype of Federated Learning with Multi-Key Homomorphic Encryption

This repository contains an implementation that integrates the xMK-CKKS homomorphic encryption scheme into a federated learning architecture to provide a demonstration of how to incorporate Multi-Key Homomorphic Encryption (MKHE) into a federated learning system.

Cite The Research
If you find those results useful please cite the research :

```
@Article{cryptography7040048,
  AUTHOR = {Walskaar, Ivar and Tran, Minh Christian and Catak, Ferhat Ozgur},
  TITLE = {A Practical Implementation of Medical Privacy-Preserving Federated Learning Using Mu
  JOURNAL = {Cryptography},
  VOLUME = {7},
  YEAR = {2023},
  NUMBER = {4},
  ARTICLE-NUMBER = {48},
  ISSN = {2410-387X},
  DOI = {10.3390/cryptography7040048}
}
```

Dependencies

This project is built upon two external repositories:

- `rlwe-xmkckks` (Modified RLWE for xMK-CKKS)
- `flower-xmkckks` (Modified Flower library to incorporate MKHE using `rlwe-xmkckks`)

To install these packages locally, clone the repositories, navigate into their respective directories and execute `pip install .`

In addition to the above packages, this project requires certain Python packages, which can be installed using the `requirements.txt` file. Run the following command to install these dependencies:

```
pip install -r requirements.txt
```

Results

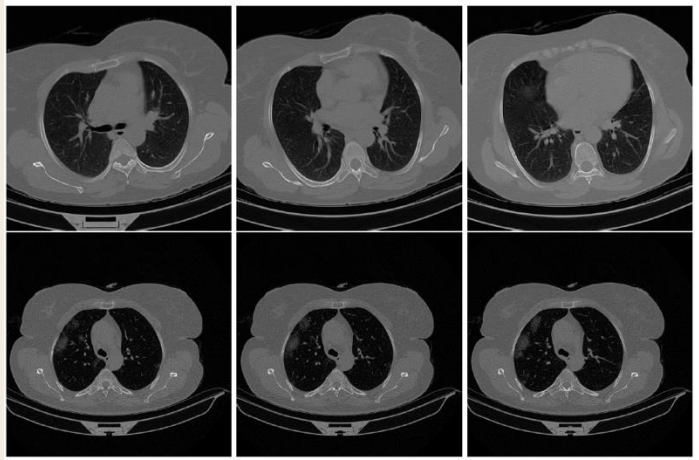


Figure 5. An example of an X-ray lung scan image taken from the dataset. The first row is COVID-19 negative, the second row is COVID-19 positive.

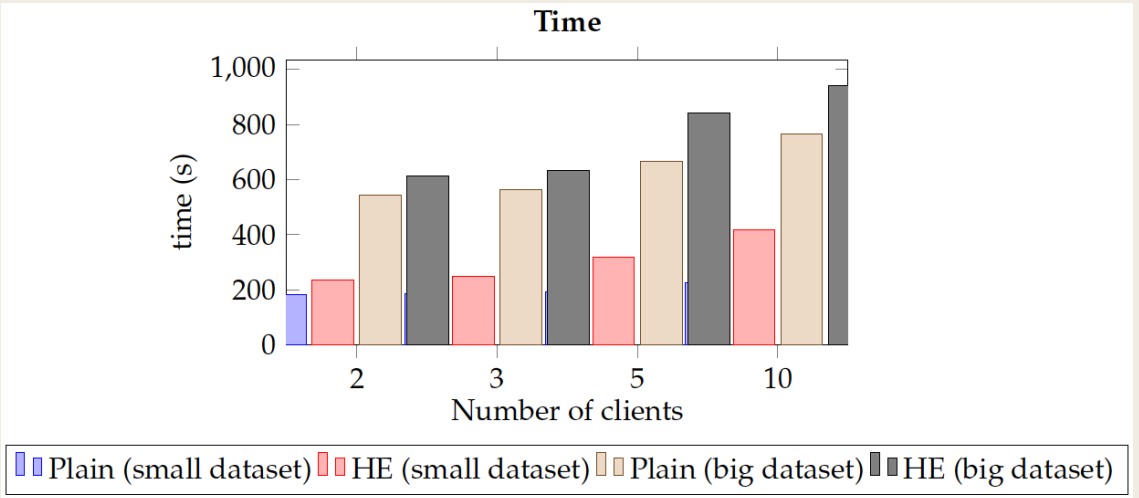


Figure 12. Time taken for different numbers of clients.

Client	Accuracy		Precision		Recall		F1-Score		Time		Server Memory		Client Memory	
	Plain	HE	Plain	HE	Plain	HE	Plain	HE	Plain	HE	Plain	HE	Plain	HE
2	0.973	0.967	0.98	0.98	0.97	0.97	0.97	0.97	542 s	612 s	1398 MB	1601 MB	1504 MB	1906 MB
3	0.968	0.964	0.98	0.98	0.97	0.97	0.97	0.96	564 s	633 s	1596 MB	1710 MB	1322 MB	1730 MB
5	0.962	0.933	0.96	0.98	0.97	0.89	0.97	0.93	667 s	841 s	1965 MB	1977 MB	1001 MB	1175 MB
10	0.938	0.927	0.94	0.98	0.94	0.89	0.94	0.92	765 s	940 s	2528 MB	2659 MB	901 MB	1088 MB

Experimental Results

Table 4: MSE results with different numbers of clients

	# of Clients	MSE
Init.	Without Encryption	0.029410
	Encrypted Without FL	0.032801
Encrypted FL	3	0.030709
	4	0.030549
	5	0.029964
	6	0.032894
	7	0.032433
	8	0.033806
	9	0.032733

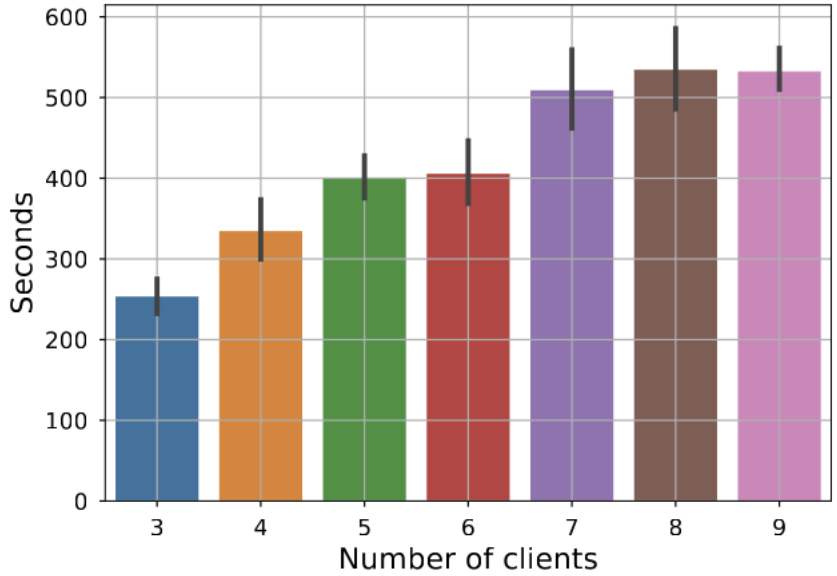


Fig. 2: Execution time with different numbers of clients