

# Homomorphic Encryption

*CS523 2025*

Sylvain Chatel | March 04, 2025 | v1.0.2

Some slides/ideas adapted from C. Mouchet, D. Fiore, JP Hubaux, C. Troncoso, and W. Lueks

# Introduction

## Homomorphic encryption

Lecture aim: study the cryptographic technique and related **toolbox for privacy engineering**



*tool*  
for building PETS



*cryptographic*  
primitive

**Application Layer**

**Network Layer**

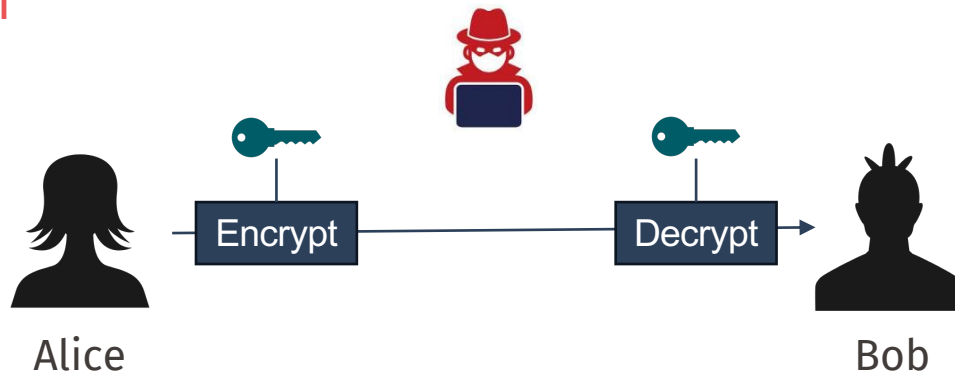
# Goals

## What should you learn today?

- Basic understanding of **homomorphic encryption**
- Understand **when to use** homomorphic encryption
- Understand **key properties**:
  - Communication and computation cost
  - Trust assumptions
  - Guarantees with respect to inputs
- Understand **practical issues** when using homomorphic encryption

# Introduction

## Encryption



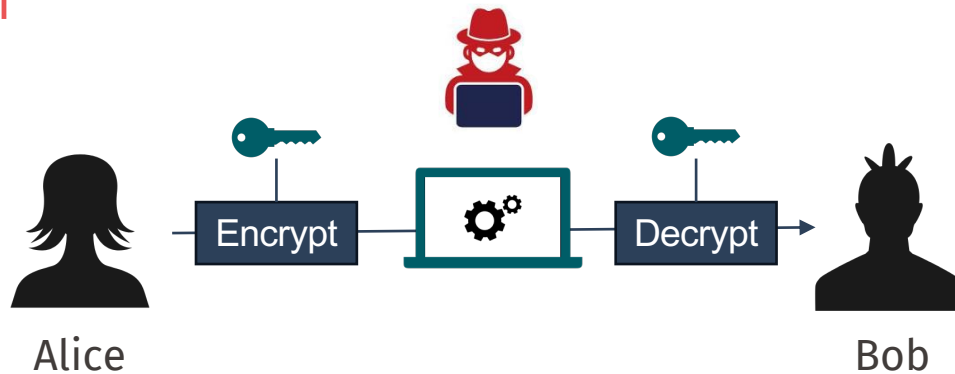
**Encryption** aims at data confidentiality

In transit

In storage

# Introduction

## Encryption



**Encryption** aims at data confidentiality

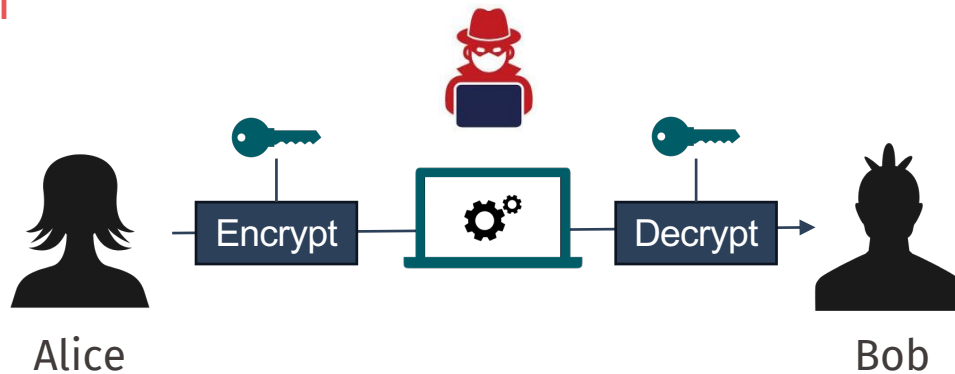
In transit

In storage

**Homomorphic Encryption** aims at also confidentiality in **computation**

# Introduction

## Encryption



**Encryption** aims at data confidentiality

In transit

In storage

**Homomorphic Encryption** aims at also confidentiality in **computation**

Health Data  
Analysis

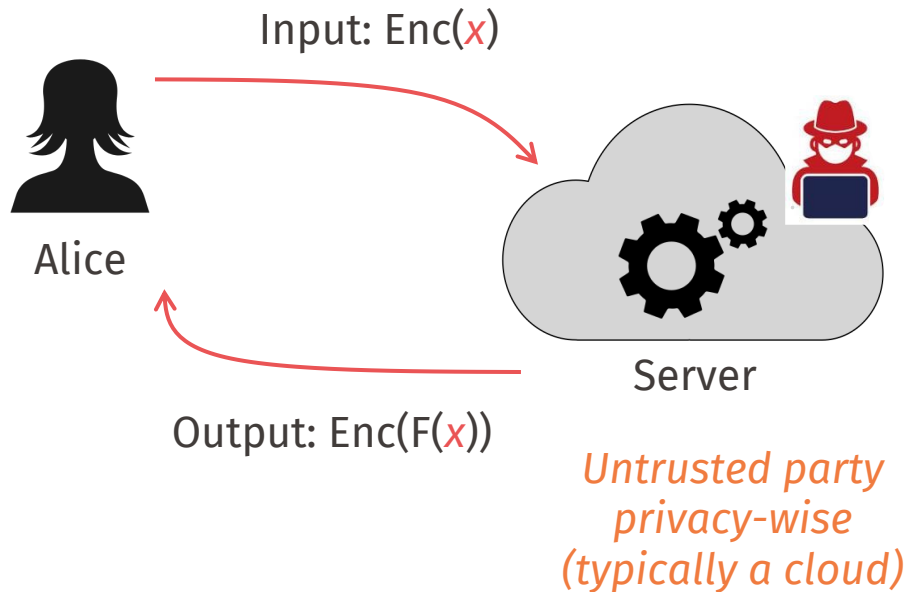
Financial Fraud  
Prevention

Govt Data: e.g.,  
demographics,  
vote, etc.

# Overview

## Homomorphic encryption

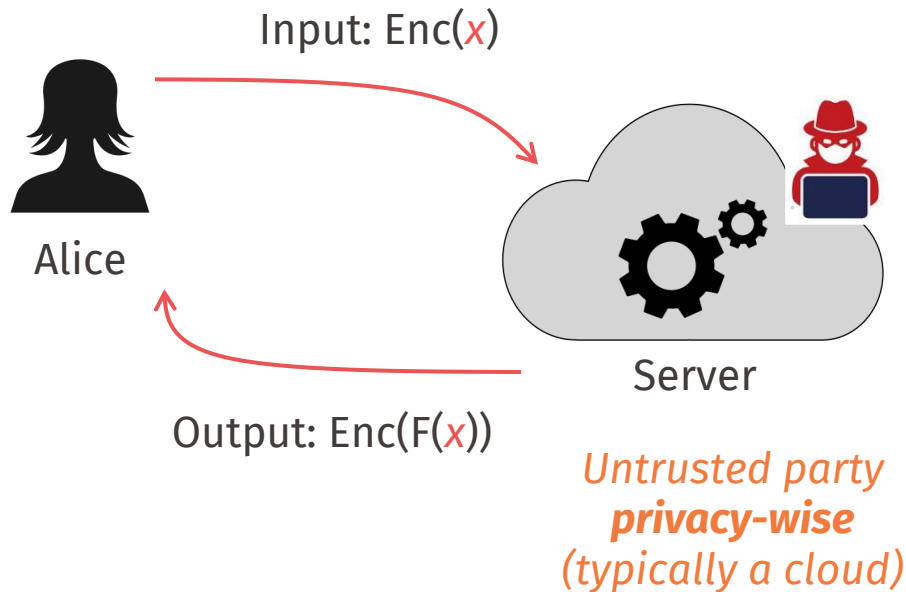
- A cryptographic primitive that enables the computation of functions in the encrypted domain



# Overview

## Homomorphic encryption

- A cryptographic primitive that enables the computation of functions in the encrypted domain



- **Security property:** the computing party cannot learn any information about the input or output
- **Correctness:** output is correct
- **Threat Model:** usually Honest-but-Curious

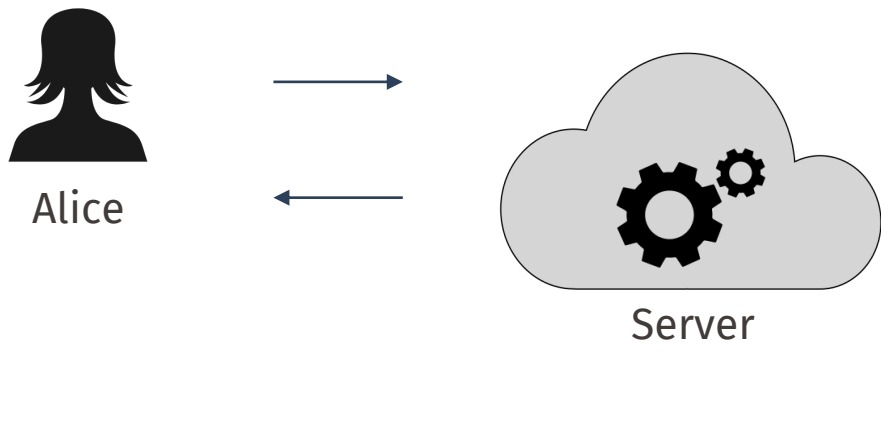


# Overview

## Homomorphic Encryption System Model

**Homomorphic encryption:** classically, has one computing party, and

- **one** party providing the input and reading the result, or

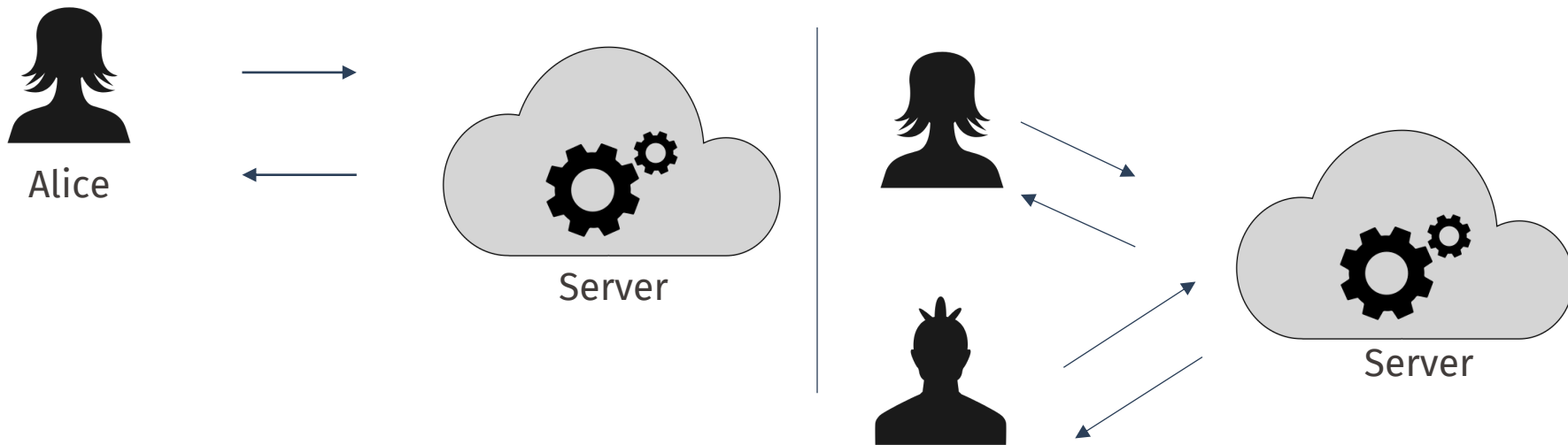


# Overview

## Homomorphic Encryption System Model

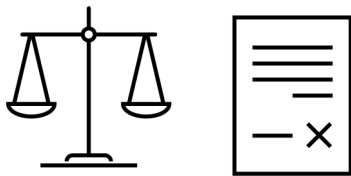
**Homomorphic encryption:** classically, has one computing party, and

- **one** party providing the input and reading the result, or
- **$n$  parties** providing the input and **another** reading the result, or
- **$n$  parties** providing data and learning the result (not in this class)



# What about Alternatives?

## NDAs, SMC, TEE

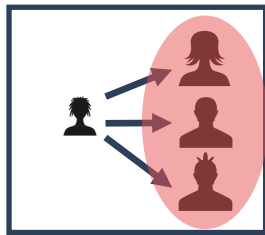


### Non-Disclosure Agreements

**Low tech**

**Lengthy Process**

**Limited protection**

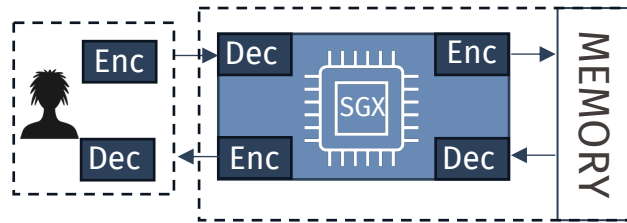


### Secret Sharing- SMC

**Non-collusion assumption**

If they do, they can  
join their shares and  
recover all the secrets!

**Hard to find in practice**



### Trusted Execution Env.

**Trust in manufacturer**

**Can be vulnerable to  
physical attacks**

# Homomorphic Encryption

## Objectives Consolidated



Enable **computation** on **encrypted data**

- Computation performed before decryption
- Decryption key known only to the receiver



**Outsourced computation setting:** party with data provides an encrypted version of the data to a *single* untrusted computation party (*a server*). No need for a non-collusion assumption.



Rely on **solely on the security of the cryptographic primitives**

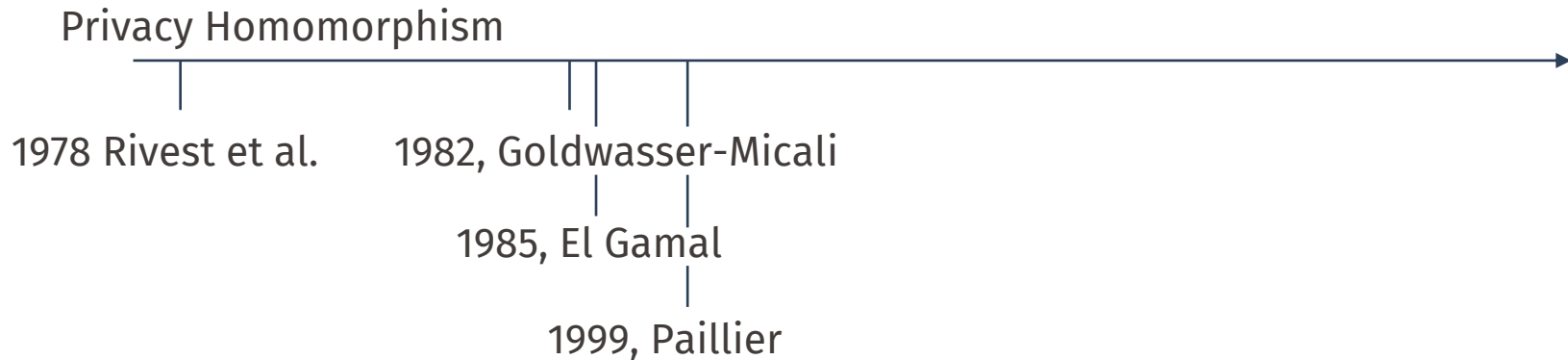
# Homomorphic Encryption History

Privacy Homomorphism

1978 Rivest et al.



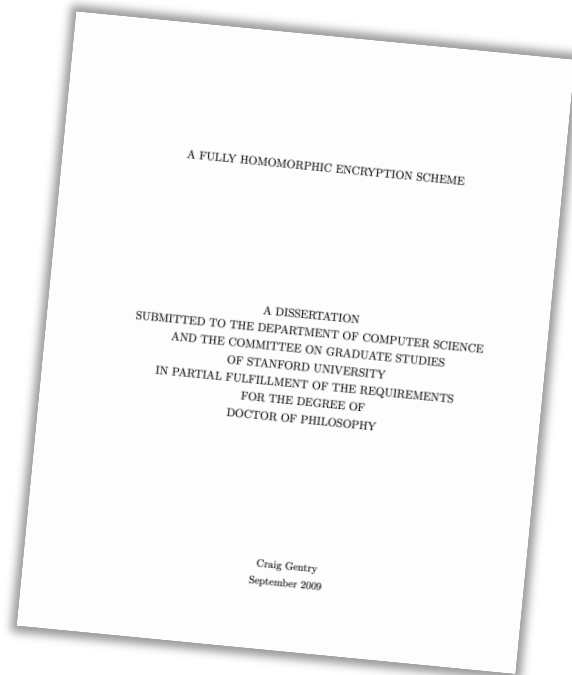
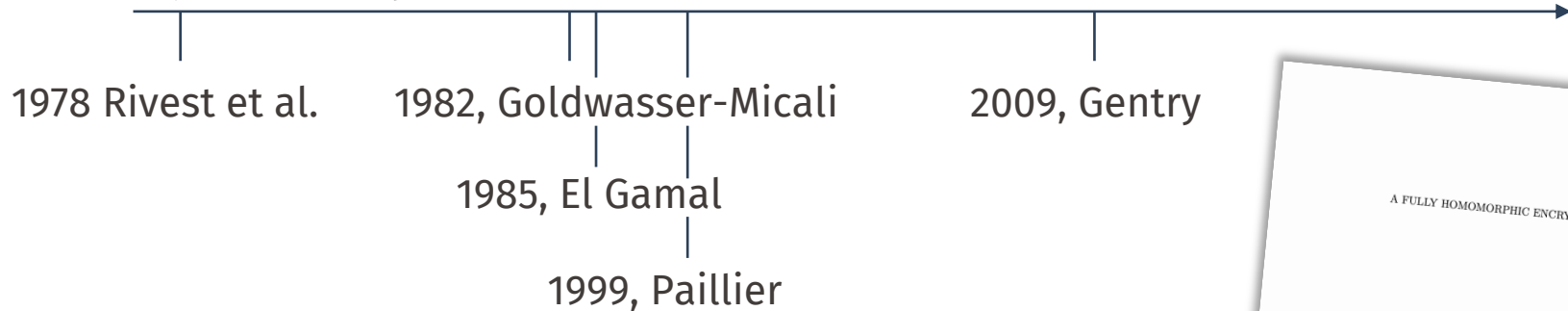
# Homomorphic Encryption History



- RSA 1977: modular multiplication
- Goldwasser-Micali: XOR
- El Gamal: modular multiplication
- Paillier: modular addition

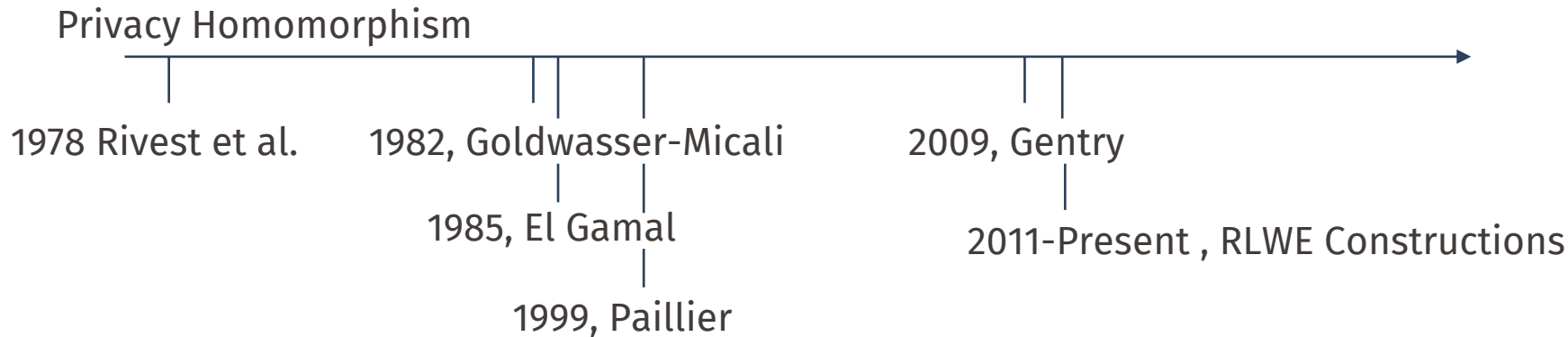
# Homomorphic Encryption History

## Privacy Homomorphism



# Homomorphic Encryption


## History



- Lattice-based schemes (e.g., BGV, BFV, CGGI, CKKS)
- Many libraries (e.g., Helib, SEAL, HEAAN, PALISADE, **OpenFHE**, **Lattigo**, **TFHE-rs**).
- Standardization 2017-soon





# Homomorphic Encryption Applications



**Swift**

## Announcing Swift Homomorphic Encryption


JULY 30, 2024

 Fabian Boerner  Karl Tarbe  Rehan Rishi

We're excited to announce a new open source Swift package for homomorphic encryption in Swift: [swift-homomorphic-encryption](#).

Homomorphic encryption (HE) is a cryptographic technique that enables computation on encrypted data *without* revealing the underlying unencrypted data to the operating process. It provides a means for clients to send encrypted data to a server, which operates on that encrypted data and returns a result that the client can decrypt. During the execution of the request, the server itself never decrypts the original data or even has access to the decryption key. Such an approach presents new opportunities for cloud services to operate while protecting the privacy and security of a user's data, which is obviously highly attractive for many scenarios.

Apple Live Caller ID - iOS18



**TUNE INSIGHT** Privacy-Preserving Federated Learning


## Secure Federated Learning with Tune Insight encrypted computing platform

 Tune Insight  
2,528 followers

November 21, 2023

Tune Insight Federated Learning

## ZAMA

 Zama's fhEVM Coprocessor is now available

### Zama's fhEVM

A simple and powerful solution for building decentralized apps with full privacy and confidentiality on Ethereum leveraging Fully Homomorphic Encryption (FHE).

[See the code](#)

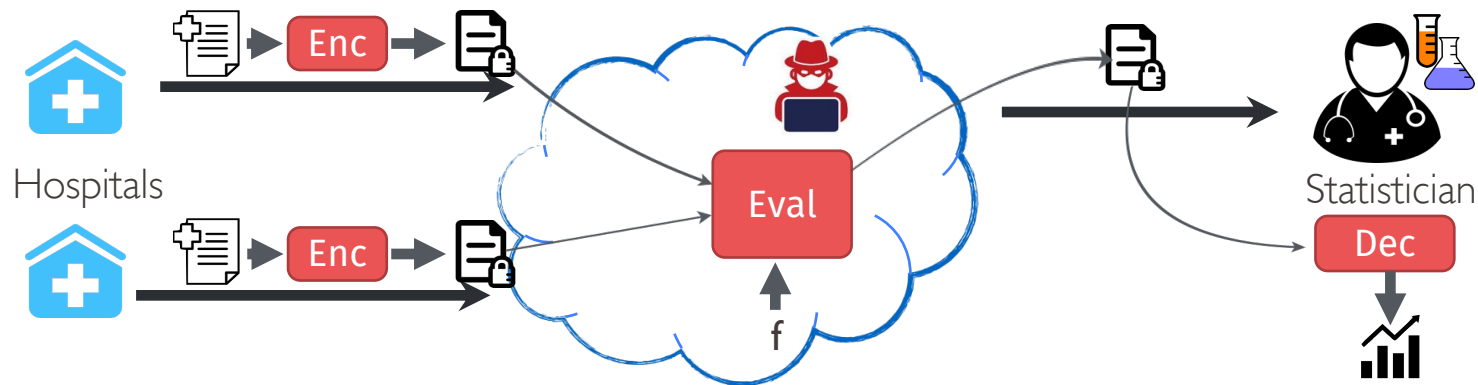
[Read the docs ↗](#)

Ready to use FHE for your business? Talk to our team.

Zama confidentiality for Ethereum

# Example application

## Privacy-preserving statistics on medical data

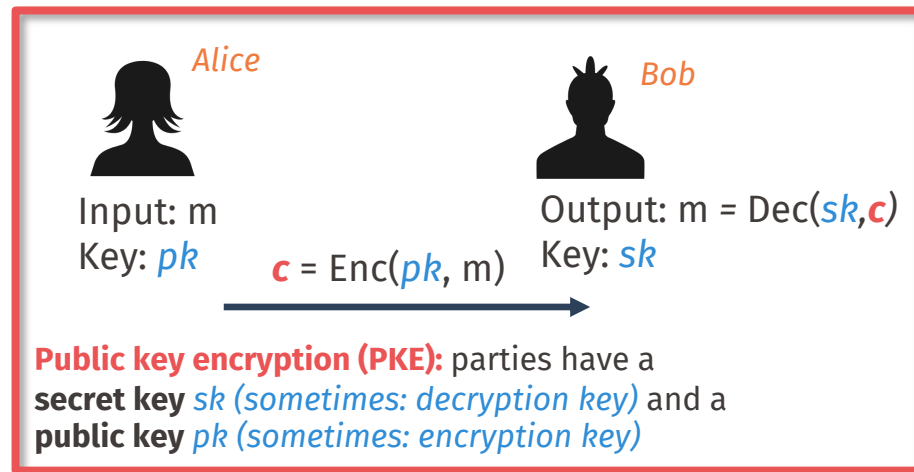
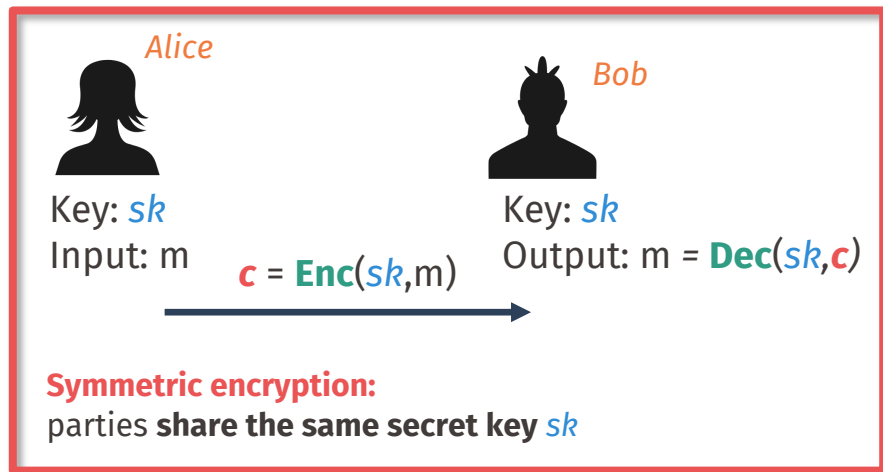


- **Goal:** a statistician wants to compute statistics on joint data from several hospitals
- **Privacy concern:** single data entries are privacy-sensitive patients' data
- **Using a cloud server and FHE**
  - Hospitals store encrypted patients' data on the cloud
  - Cloud compute statistics on joint (encrypted) datasets. Statistician decrypts results

# Definitions and Properties

# Formal definition

## Cryptographic Notation



**KeyGen** $(1^k) \rightarrow (sk, pk)$

**Enc** $(pk, m; r) \rightarrow c$

**Dec** $(sk, c) \rightarrow m$

*Generates a private/public key pair for a security parameter  $k$*   
*Encrypt the message  $m$  with randomness  $r$  to a ciphertext  $c$*   
*Decrypt a ciphertext  $m$  to obtain the message  $m$*

# Formal definition

## Homomorphic encryption

A **homomorphic encryption scheme** is given by the following four algorithms:

- **KeyGen**( $1^k$ )  $\rightarrow$  ( $sk$ ,  $pk$ )      *Generates a private/public key pair for a security parameter  $k$*
- **Enc**( $pk$ ,  $m$ ;  $r$ )  $\rightarrow$   $c$       *Encrypt the message  $m$  with randomness  $r$  to a ciphertext  $c$*
- **Dec**( $sk$ ,  $c$ )  $\rightarrow$   $m$       *Decrypt a ciphertext  $c$  to obtain the message  $m$*
- **Eval**( $pk$ ,  $f$ ,  $c_1, \dots, c_n$ )  $\rightarrow$   $c'$       *Evaluate function  $f$  on the encrypted input  $c_i$  to obtain a new ciphertext  $c'$*

**Eval()** is what makes HE different from standard PK encryption

(in some schemes **Eval** uses a special key  $evk$  instead of  $pk$ . If this is the case, then **KeyGen** generates a triplet)

(The **KeyGen** and **Enc** are randomized algorithms. The randomness parameter is often implicit)

# Properties

## Homomorphic encryption

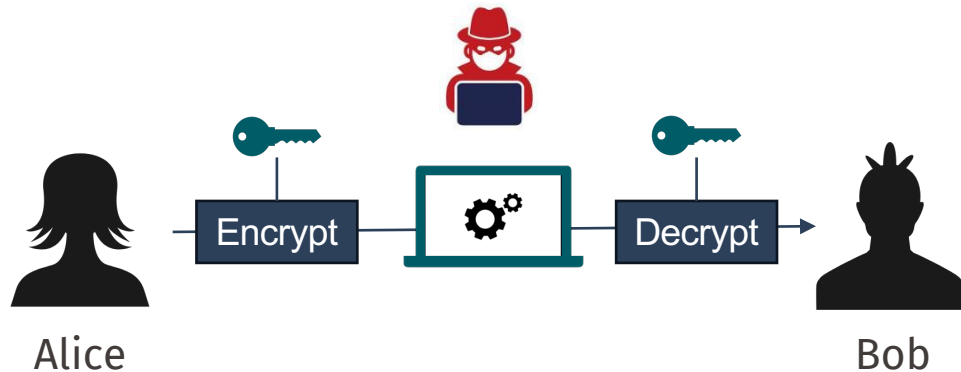
**Correctness**

**Security**

**Evaluation correctness**

**Composition**

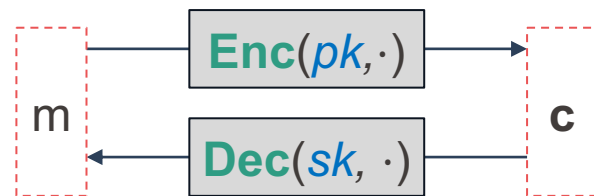
**Compactness**



# Homomorphic Encryption

## Encryption Correctness

**Correctness** – *intuition:*



# Homomorphic Encryption

## Encryption Correctness

**Correctness** – *intuition*: applying the deterministic **decryption** function to the randomized **encryption** always **returns the initial plaintext**.

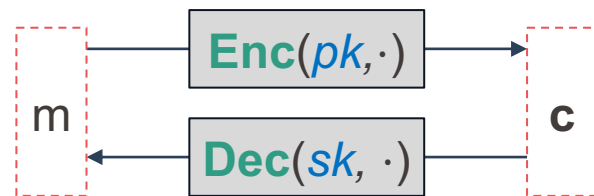
**Correctness.**

$\forall r \in \text{Rand}, \forall m \in \mathcal{P} \text{ and } (sk, pk) \leftarrow \text{KeyGen}(1^k):$

$$\text{Dec}(sk, \text{Enc}(pk, m; r)) = m$$

$\text{Dec}(sk, \cdot)$  is the deterministic inverse of the randomized  $\text{Enc}(pk, \cdot)$  function

$\mathcal{P}$  is the plaintext space and  $\text{Rand}$  a source of randomness





# Homomorphic Encryption

## Homomorphic Correctness

Objective: Evaluate the plaintext function  $f$  over the encryption in the ciphertext space

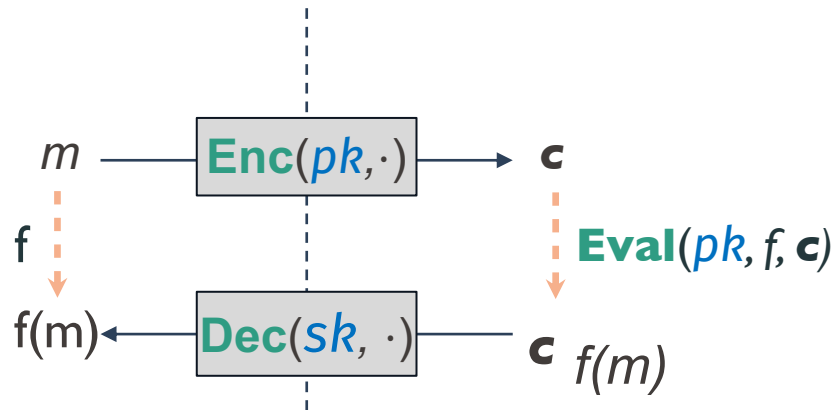
Recall an HE scheme:

**KeyGen** $(1^k) \rightarrow (sk, pk)$

**Enc** $(pk, m; r) \rightarrow \mathbf{c}$

**Dec** $(sk, \mathbf{c}) \rightarrow m$

**Eval** $(pk, f, \mathbf{c}) \rightarrow \mathbf{c}_{f()}$



# Homomorphic Encryption

## Wait... Homomorphic?

**Homomorphism:** mapping between two sets that preserves their algebraic structure

### Group homomorphism

Given two groups  $(G, +)$ ,  $(H, \boxtimes)$

$h: G \rightarrow H$  is a *group-homomorphism* if  $\forall x, y \in G$ :

$$h(x + y) = h(x) \boxtimes h(y)$$

For an **additive** HE scheme,  $\text{Dec}(sk, \cdot): \mathcal{C} \rightarrow \mathcal{P}$  is an homomorphism between  $(\mathcal{C}, \text{Eval.Add})$  and  $(\mathcal{P}, +)$

*A **group** consists of a **set of elements** and **one operation** that combines two elements of the set into another element of the set (operation must be associative, there is an identity element, and all elements have inverse)*

*A **ring** consists of a set of elements and **two operations** that combine two elements of the set into another element of the set (multiplication does not need to be commutative, and does not need inverse)*

# Homomorphic Encryption

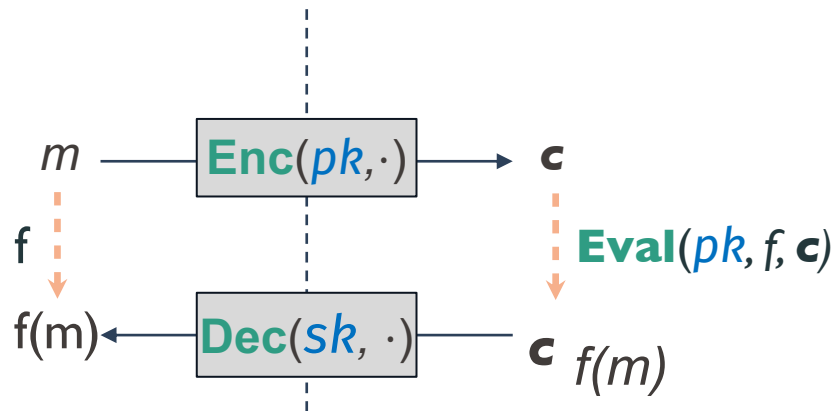
## Homomorphic Correctness

Objective: Evaluate the plaintext function  $f$  over the encryption in the ciphertext space

For a set  $\mathcal{F}$  of admissible functions

$\forall m \in \mathcal{P}, \forall f \in \mathcal{F}$ , and  
 $(sk, pk) \leftarrow \text{KeyGen}(1^k)$ :

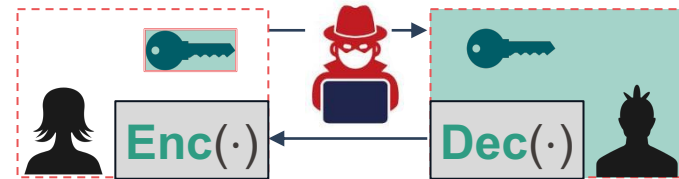
$\text{Dec}(sk, \text{Eval}(pk, f, \text{Enc}(pk, m))) = f(m)$



This definition is relaxed in practice

# Homomorphic Encryption Security

**Security** – *intuition:*

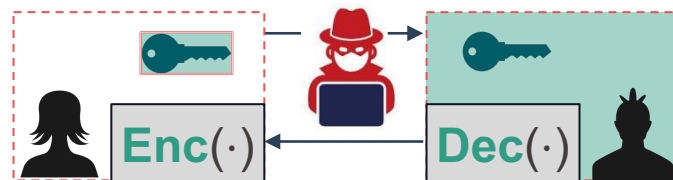


# Homomorphic Encryption

## Security

**Security** – *intuition*: The adversary learns nothing from the ciphertexts and public keys

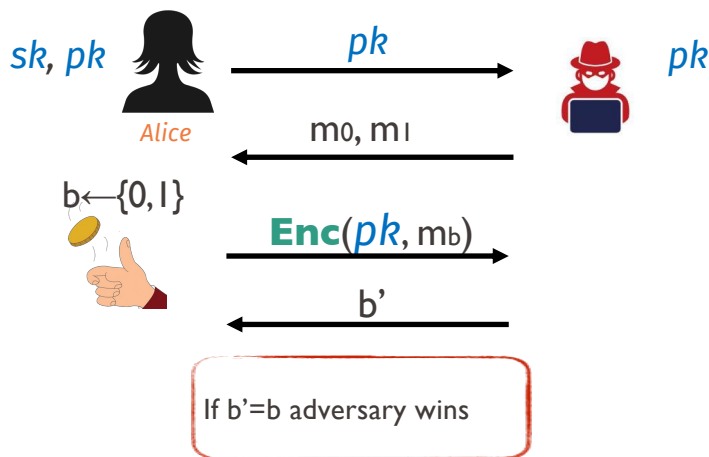
*How do we capture this?*



**Game-based security definition:** method to capture a security definition using a game between an adversary and a challenger. If the adversary has a non-negligible advantage to win the game, the adversary “breaks” the security property.

# Semantic security (IND-CPA) of Homomorphic encryption

*Game-based security*: is a method to capture a security definition using a game between an adversary and a challenger. If the adversary wins the game, the adversary “breaks” the security property.



$(sk, pk) \leftarrow \text{KeyGen}(1^k)$

$(m_0, m_1) \leftarrow \mathcal{A}(pk)$  // adversary chooses two messages

$b \leftarrow \{0, 1\};$  // random bit is chosen

$c \leftarrow \text{Enc}(pk, m_b)$  // message  $b$  is encrypted

$b' \leftarrow \mathcal{A}(c)$  // adversary must figure out  $b$

## Semantic security – intuition

seeing encryption of messages does not give the adversary information that helps them guessing the encrypted message better than random guess

**Semantic security.** HE is secure if  $\frac{1}{2} - \Pr[b' = b] = \text{negl}(k)$

# Homomorphic Encryption

## Is this an HE scheme?

Consider the following scheme built on top of a IND-CPA secure **PKE** scheme:

$$\mathbf{KeyGen}(1^k) = \mathbf{PKE.KeyGen}(1^k)$$

$$\mathbf{Enc}(pk, m) = \{c_{PKE} = \mathbf{PKE.Enc}(pk, m); \text{ return } \mathbf{c} = (c_{PKE}, \text{nil})\}$$

$$\mathbf{Eval}(pk, f, \mathbf{c}) = \{c_{PKE} = \mathbf{c}[0]; \text{ return } (c_{PKE}, f)\}$$

$$\mathbf{Dec}(sk, \mathbf{c}) = \{c_{PKE}, f \leftarrow \mathbf{c}; \text{ return } f(\mathbf{PKE.Dec}(sk, c_{PKE}))\}$$

1. Is it correct Encryption scheme?
2. Is it IND-CPA secure?
3. Is it useful?

# Homomorphic Encryption

## Is this an HE scheme?

Consider the following scheme built on top of a IND-CPA secure **PKE** scheme:

$$\text{KeyGen}(1^k) = \text{PKE.KeyGen}(1^k)$$

$$\text{Enc}(pk, m) = \{c_{\text{PKE}} = \text{PKE.Enc}(pk, m); \text{return } \mathbf{c} = (c_{\text{PKE}}, \text{nil})\}$$

$$\text{Eval}(pk, f, \mathbf{c}) = \{c_{\text{PKE}} = \mathbf{c}[0]; \text{return } (c_{\text{PKE}}, f)\}$$

$$\text{Dec}(sk, \mathbf{c}) = \{c_{\text{PKE}}, f \leftarrow \mathbf{c}; \text{return } f(\text{PKE.Dec}(sk, c_{\text{PKE}}))\}$$

1. Is it correct Encryption scheme? Yes, straightforward from the decryption
2. Is it IND-CPA secure? Yes, as it would break the IND-CPA from **PKE**
3. Is it useful? **No**, the evaluation of  $f$  is performed at decryption...



# Homomorphic Encryption

## Compactness

**Compactness** – *intuition*: the ciphertext size should not be growing through homomorphic operations

**Compactness.** HE compactly evaluates a family of functions  $\mathcal{F}$  if

$$\forall (sk, pk) \leftarrow \text{KeyGen}(1^k), \quad \forall f \in \mathcal{F}, \quad \forall m_i \in \mathcal{P}:$$

There exists a polynomial  $p()$  such that the size of

$$|\text{Eval}(pk, f, c_1, \dots, c_n)| < p(k), \text{ with } k \text{ the security parameter, } \underline{\text{independent of } f()};$$

i.e., the complexity of **Dec** is independent of  $f()$ .

This definition is  
relaxed in practice

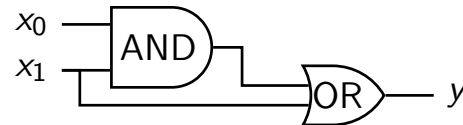
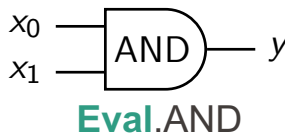
# Homomorphic Encryption

## Composition

**Composition** – *intuition*: Build HE computation from a set of simple operation (think Circuits and CPU)

Define **Eval** as circuit of simple “gates”. E.g., Boolean Algebra

$x_0$	$x_1$	$\text{AND}(x_0, x_1)$
0	0	0
0	1	0
1	0	0
1	1	1



$x_0$	$x_1$	$\text{OR}(x_0, x_1)$
0	0	0
0	1	1
1	0	1
1	1	1



$x_0$	$x_1$	$\text{OR}(\text{AND}(x_0, x_1), x_1)$
0	0	0
0	1	1
1	0	0
1	1	1

# Homomorphic Encryption

## Composition

**Composition** – *intuition*: Build HE computation from a set of simple operation (think Circuits and CPU)

$\forall f \in \mathcal{F}, f: \mathcal{P}^n \rightarrow \mathcal{P}, \forall (m_1, \dots, m_n) \in \mathcal{P}^n, (sk, pk) \leftarrow \text{KeyGen}(1^k), \text{ and } \forall i \in [n] \ c_i = \text{Enc}(pk, m_i)$

- **Correctness for n-ary functions:**

$$\text{Dec}(sk, \text{Eval}(pk, f, c_1, \dots, c_n)) = f(m_1, \dots, m_n)$$

- **Composability:**

$$\text{Dec}(sk, \text{Eval}(pk, f, c_1, \dots, c_n)) = f(\text{Dec}(sk, c_1), \dots, \text{Dec}(sk, c_n))$$

Informally extends correctness to any *valid* ciphertext (fresh or **Eval**)

# Homomorphic Encryption

## Summary of the properties

$\forall f \in \mathcal{F}, f: \mathcal{P}^n \rightarrow \mathcal{P}, \forall (m_1, \dots, m_n) \in \mathcal{P}^n,$   
 $(sk, pk) \leftarrow \text{KeyGen}(1^k), \text{ and } \forall i \in [n] \ c_i = \text{Enc}(pk, m_i)$

HE scheme:

$\text{KeyGen}(1^k) \rightarrow (sk, pk)$

$\text{Enc}(pk, m; r) \rightarrow c$

$\text{Dec}(sk, c) \rightarrow m$

$\text{Eval}(pk, f, c_1, \dots, c_n) \rightarrow c_{f()}$

### Correctness

$$\text{Dec}(sk, \text{Enc}(pk, m_i)) = f(m_i)$$

### IND-CPA Security

$$\text{Adv}_{\text{HE}, \mathcal{A}}^{\text{IND-CPA}} = \text{negl}(k)$$

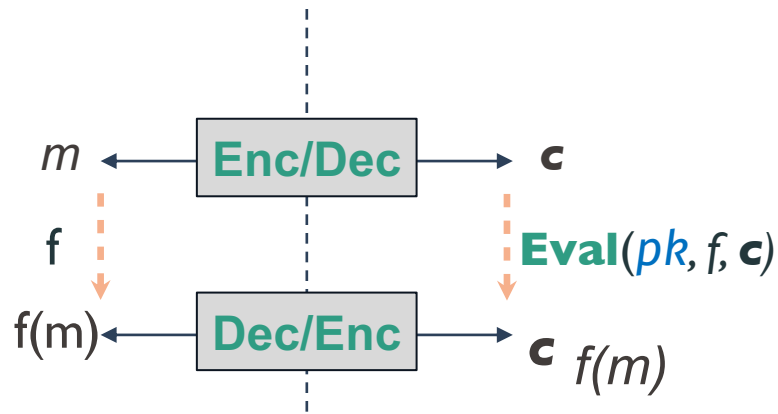
### Evaluation correctness and composition

$$\text{Dec}(sk, \text{Eval}(pk, f, c_1, \dots, c_n)) = f(m_1, \dots, m_n)$$

$$\text{Dec}(sk, \text{Eval}(pk, f, c_1, \dots, c_n)) = f(\text{Dec}(sk, c_1), \dots, \text{Dec}(sk, c_n))$$

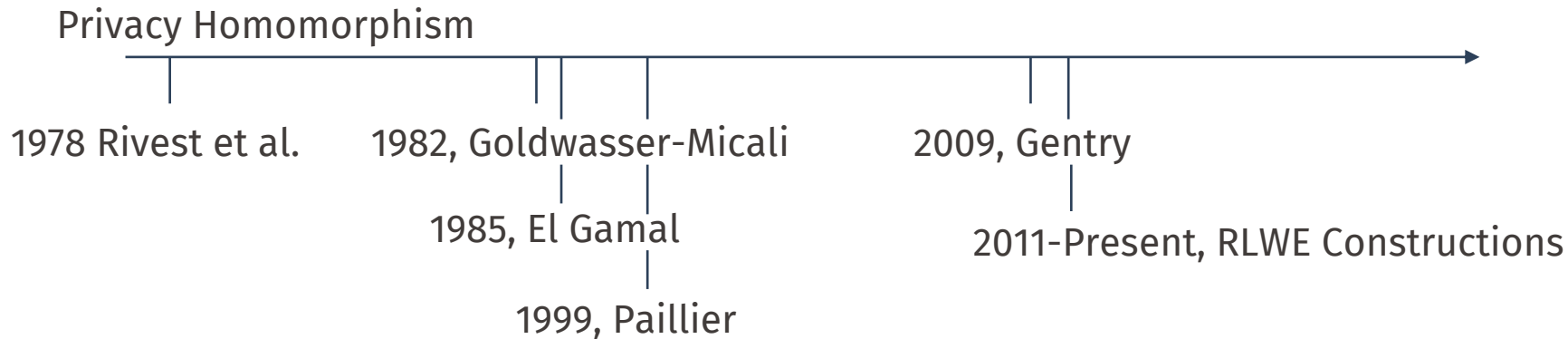
### Compactness

$$|\text{Eval}(pk, f, c_1, \dots, c_n)| < p(k) \text{ for } p \text{ polynomial Independent of } f$$



# Constructions

# Homomorphic Encryption History



We will cover

- ~~- RSA~~
- El Gamal
- ~~- Paillier~~
- RLWE

# El Gamal Encryption

## Properties

### ElGamal( $G, g$ ):

Let  $G$  be a cyclic group of order  $q=|G|$   
generated by  $g$

### KeyGen $\rightarrow (sk, pk)$ :

$$sk \leftarrow \{1, \dots, q-1\}, \quad pk = g^{sk}$$

### Enc( $pk, m \in G$ ) $\rightarrow c$

$$r \leftarrow \{1, \dots, q-1\}$$

$$c := (m \cdot pk^r, g^r)$$

### Dec( $sk, c$ ) $\rightarrow m$

$$m := c[0] / c[1]^{sk}$$

**Security** – Security from the Discrete Logarithm:

Given  $g$  and  $y = g^x \bmod p$ , find  $x$

**Correctness** –

**Evaluation** – Component-wise \_\_\_\_\_

**Compact** –

# El Gamal Encryption

## Properties

### ElGamal( $G, g$ ):

Let  $G$  be a cyclic group of order  $q=|G|$   
generated by  $g$

### KeyGen $\rightarrow (sk, pk)$ :

$$sk \leftarrow \mathbb{Z}\{1, \dots, q-1\}, \quad pk = g^{sk}$$

### Enc( $pk, m \in G$ ) $\rightarrow c$

$$r \leftarrow \mathbb{Z}\{1, \dots, q-1\}$$

$$c := (m \cdot pk^r, g^r)$$

### Dec( $sk, c$ ) $\rightarrow m$

$$m := c[0] / c[1]^{sk}$$

**Security** – Security from the Discrete Logarithm:

Given  $g$  and  $y = g^x \bmod p$ , find  $x$

**Correctness** – Follows by definition

$$\begin{aligned} \text{Dec}(sk, \text{Enc}(pk, m)) &= \text{Dec}(sk, (m \cdot pk^r, g^r)) \\ &= m \cdot pk^r \cdot (g^r)^{-sk} = m \cdot g^{r \cdot sk} \cdot (g^r)^{-sk} = m \end{aligned}$$

**Evaluation** – Component-wise \_\_\_\_\_

**Compact** –



# El Gamal Encryption

## Properties

### ElGamal( $G, g$ ):

Let  $G$  be a cyclic group of order  $q=|G|$  generated by  $g$

### KeyGen $\rightarrow (sk, pk)$ :

$$sk \leftarrow \mathcal{S}\{1, \dots, q-1\}, \quad pk = g^{sk}$$

### Enc( $pk, m \in G$ ) $\rightarrow c$

$$r \leftarrow \mathcal{S}\{1, \dots, q-1\}$$

$$c := (m \cdot pk^r, g^r)$$

### Dec( $sk, c$ ) $\rightarrow m$

$$m := c[0] / c[1]^{sk}$$

### Security – Security from the Discrete Logarithm:

Given  $g$  and  $y = g^x \bmod p$ , find  $x$

### Correctness – Follows by definition

$$\begin{aligned} \text{Dec}(sk, \text{Enc}(pk, m)) &= \text{Dec}(sk, (m \cdot pk^r, g^r)) \\ &= m \cdot pk^r \cdot (g^r)^{-sk} = m \cdot g^{r \cdot sk} \cdot (g^r)^{-sk} = m \end{aligned}$$

### Evaluation – Component-wise multiplication

$$\begin{aligned} c_1 &= \text{Enc}(pk, m_1) \text{ and } c_2 = \text{Enc}(pk, m_2) \\ \text{Mul}(c_1, c_2) &= (c_1[0] \cdot c_2[0], c_1[1] \cdot c_2[1]) \\ &= (m_1 \cdot pk^{r_1} \cdot m_2 \cdot pk^{r_2}, g^{r_1} \cdot g^{r_2}) \\ &= (m_1 \cdot m_2 \cdot pk^{r_1+r_2}, g^{r_1+r_2}) = c \\ \text{Dec}(sk, c) &= m_1 \cdot m_2 \end{aligned}$$

### Compact –

# El Gamal Encryption

## Properties

### ElGamal( $G, g$ ):

Let  $G$  be a cyclic group of order  $q=|G|$  generated by  $g$

### KeyGen $\rightarrow (sk, pk)$ :

$$sk \leftarrow \mathcal{S}\{1, \dots, q-1\}, \quad pk = g^{sk}$$

### Enc( $pk, m \in G$ ) $\rightarrow c$

$$r \leftarrow \mathcal{S}\{1, \dots, q-1\}$$

$$c := (m \cdot pk^r, g^r)$$

### Dec( $sk, c$ ) $\rightarrow m$

$$m := c[0] / c[1]^{sk}$$

### Security – Security from the Discrete Logarithm:

Given  $g$  and  $y = g^x \bmod p$ , find  $x$

### Correctness – Follows by definition

$$\begin{aligned} \text{Dec}(sk, \text{Enc}(pk, m)) &= \text{Dec}(sk, (m \cdot pk^r, g^r)) \\ &= m \cdot pk^r \cdot (g^r)^{-sk} = m \cdot g^{r \cdot sk} \cdot (g^r)^{-sk} = m \end{aligned}$$

### Evaluation – Component-wise multiplication

$$\begin{aligned} c_1 &= \text{Enc}(pk, m_1) \text{ and } c_2 = \text{Enc}(pk, m_2) \\ \text{Mul}(c_1, c_2) &= (c_1[0] \cdot c_2[0], c_1[1] \cdot c_2[1]) \\ &= (m_1 \cdot pk^{r_1} \cdot m_2 \cdot pk^{r_2}, g^{r_1} \cdot g^{r_2}) \\ &= (m_1 \cdot m_2 \cdot pk^{r_1+r_2}, g^{r_1+r_2}) = c \\ \text{Dec}(sk, c) &= m_1 \cdot m_2 \end{aligned}$$

### Compact – yes

# Additive El Gamal Encryption Properties

## AddElGamal(G, g):

Let G be a cyclic group of order  $q=|G|$  generated by  $g$

## KeyGen $\rightarrow (sk, pk)$ :

$$sk \leftarrow \mathbb{Z}_{q-1}, \quad pk = g^{sk}$$

## Enc(pk, $m \in G$ ) $\rightarrow c$

$$r \leftarrow \mathbb{Z}_{q-1}$$

$$c := (g^m \cdot pk^r, g^r)$$

## Dec(sk, c) $\rightarrow m$

$$m := \log_g (c[0] / c[1]^{sk})$$

Useful plaintext space:

$(\mathbb{Z}_n, +)$  closed additive group of integers modulo  $n$ .

Supports any linear combination:

$$f(x_1, \dots, x_\ell) = \sum_{k=1}^{\ell} \delta_i \cdot x_i \bmod n$$

We encode the plaintext in the exponent  $m \rightarrow g^m$

$$c_1 = \text{Enc}(pk, m_1) \text{ and } c_2 = \text{Enc}(pk, m_2)$$

$$\begin{aligned} \text{Mul}(c_1, c_2) &= (c_1[0] \cdot c_2[0], c_1[1] \cdot c_2[1]) \\ &= (g^{m_1} \cdot pk^{r_1} \cdot g^{m_2} \cdot pk^{r_2}, g^{r_1} \cdot g^{r_2}) \\ &= (g^{m_1+m_2} \cdot pk^{r_1+r_2}, g^{r_1+r_2}) = c \end{aligned}$$

$$\text{Dec}(sk, c) = m_1 + m_2$$

Not efficient for  
general case

# Paillier Properties

## Paillier(L):

**KeyGen**  $\rightarrow (sk, pk)$ :

$$p, q \leftarrow \mathbb{P}_L$$

$$n := pq$$

$$pk = (n, g = n + 1), sk = \phi(n)$$

**Enc**( $pk, m \in \mathbb{Z}_n$ )  $\rightarrow c$

$$r \leftarrow \mathbb{Z}_n^*$$

$$c := g^m \cdot r^n \bmod n^2$$

**Dec**( $sk, c$ )  $\rightarrow m = \frac{(c^{sk} \bmod n^2) - 1}{n} sk^{-1} \bmod n$

**Security** – Security from the Decisional Composite Residuosity assumption:

**Evaluation** –

**Correctness** – Uses two facts

1.  $\forall x \in \mathbb{Z}_{n^2}^*, (x^n)^{\phi(n)} = 1 \bmod n^2$
2. The base- $g$  DL in  $\mathbb{Z}_{n^2}^*$  is easy to compute

# Textbook RSA Encryption Properties

## RSA(L):

**KeyGen**  $\rightarrow (sk, pk)$ :

$p, q \leftarrow \mathbb{P}$  s.t.  $\log(pq) \geq L$

$n := pq, e \leftarrow \mathbb{Z}_{\phi(n)}^*$

$pk = (n, e), sk = e^{-1} \bmod \phi(n)$

**Enc**( $pk, m \in \mathbb{Z}_n^*$ )  $\rightarrow c = m^e \bmod n$

**Dec**( $sk, c$ )  $\rightarrow m = c^{sk} \bmod n$

**Security** – Security based on factoring hardness:

Given  $n = pq$  s.t.  $p, q \leftarrow \mathbb{P}$ , find  $p$  and  $q$

Equivalent to finding  $\phi(n)$

**Correctness** –

**Evaluation** –

**Compact** –

# Lattice-Based Constructions

# Fully Homomorphic Encryption

## A simple scheme for intuition

$$\mathcal{R}_q = \mathbb{Z}_q[X]/(X^N + 1)$$

### IdealHE:

**KeyGen**  $\rightarrow (sk, \cdot)$ :

$$sk := s \leftarrow \$ \mathcal{R}_q$$

**Enc**( $sk, m \in \mathcal{R}_q$ )  $\rightarrow \mathbf{c}$

$$a \leftarrow \$ \mathcal{R}_q$$

$$\mathbf{c} := (-a \cdot sk + m, a)$$

**Dec**( $sk, \mathbf{c}$ )  $\rightarrow m$

$$m = \mathbf{c}[0] + sk \cdot \mathbf{c}[1]$$

Consider the polynomial ring  $\mathcal{R}_q = \mathbb{Z}_q[X]/(X^N + 1)$   
i.e., degree  $N - 1$  polynomials with coefficients in  $\mathbb{Z}_q$

# Fully Homomorphic Encryption

## A simple scheme for intuition – Correctness

$$\mathcal{R}_q = \mathbb{Z}_q[X]/(X^N + 1)$$

IdealHE:

**KeyGen**  $\rightarrow (sk, \cdot)$ :

$$sk := s \leftarrow \$ \mathcal{R}_q$$

**Enc**( $sk, m \in \mathcal{R}_q$ )  $\rightarrow \mathbf{c}$

$$a \leftarrow \$ \mathcal{R}_q$$

$$\mathbf{c} := (-a \cdot sk + m, a)$$

**Dec**( $sk, \mathbf{c}$ )  $\rightarrow m$

$$m = \mathbf{c}[0] + sk \cdot \mathbf{c}[1]$$

Consider the polynomial ring  $\mathcal{R}_q = \mathbb{Z}_q[X]/(X^N + 1)$   
i.e., degree  $N - 1$  polynomials with coefficients in  $\mathbb{Z}_q$

**Correctness** –  $\forall sk \in \mathcal{R}_q$  and  $\forall m \in \mathcal{R}_q$

$$\begin{aligned} \text{Dec}(sk, \text{Enc}(sk, m)) &= (-a \cdot sk + m) + sk \cdot a \\ &= (-as + m) + s \cdot a = m \end{aligned}$$



# Fully Homomorphic Encryption

## A simple scheme for intuition – Addition

$$\mathcal{R}_q = \mathbb{Z}_q[X]/(X^N + 1)$$

IdealHE:

**KeyGen**  $\rightarrow (sk, \cdot)$ :

$$sk := s \leftarrow \$ \mathcal{R}_q$$

**Enc**( $sk, m \in \mathcal{R}_q$ )  $\rightarrow \mathbf{c}$

$$a \leftarrow \$ \mathcal{R}_q$$

$$\mathbf{c} := (-a \cdot sk + m, a)$$

**Dec**( $sk, \mathbf{c}$ )  $\rightarrow m$

$$m = \mathbf{c}[0] + sk \cdot \mathbf{c}[1]$$

**Evaluation** –  $\forall sk \in \mathcal{R}_q$  and  $\forall m_1, m_2 \in \mathcal{R}_q$

$$\mathbf{c}_1 = \text{Enc}(sk, m_1) \text{ and } \mathbf{c}_2 = \text{Enc}(sk, m_2)$$

# Fully Homomorphic Encryption

## A simple scheme for intuition – Addition

$$\mathcal{R}_q = \mathbb{Z}_q[X]/(X^N + 1)$$

IdealHE:

**KeyGen**  $\rightarrow (sk, \cdot)$ :

$$sk := s \leftarrow \$ \mathcal{R}_q$$

**Enc**( $sk, m \in \mathcal{R}_q$ )  $\rightarrow \mathbf{c}$

$$a \leftarrow \$ \mathcal{R}_q$$

$$\mathbf{c} := (-a \cdot sk + m, a)$$

**Dec**( $sk, \mathbf{c}$ )  $\rightarrow m$

$$m = \mathbf{c}[0] + sk \cdot \mathbf{c}[1]$$

**Evaluation** –  $\forall sk \in \mathcal{R}_q$  and  $\forall m_1, m_2 \in \mathcal{R}_q$

$$\mathbf{c}_1 = \text{Enc}(sk, m_1) \text{ and } \mathbf{c}_2 = \text{Enc}(sk, m_2)$$

**- Addition**

$$\begin{aligned} \mathbf{c}_1 + \mathbf{c}_2 &= (-a_1 \cdot sk + m_1, a_1) + (-a_2 \cdot sk + m_2, a_2) \\ &= (-a_1 \cdot sk + m_1 - a_2 \cdot sk + m_2, a_1 + a_2) \\ &= (-(a_1 + a_2) \cdot sk + m_1 + m_2, a_1 + a_2) \\ &= (-b \cdot sk + m_1 + m_2, b_{add}) \text{ with } b_{add} = a_1 + a_2 \end{aligned}$$

$$\mathbf{c}_1 + \mathbf{c}_2 = \text{Enc}(sk, m_1 + m_2)$$

$$\text{So Add}(\mathbf{c}_1, \mathbf{c}_2) = \mathbf{c}_1 + \mathbf{c}_2$$

# Fully Homomorphic Encryption

## A simple scheme for intuition – Multiplication

$$\mathcal{R}_q = \mathbb{Z}_q[X]/(X^N + 1)$$

IdealHE:

**KeyGen**  $\rightarrow (sk, \cdot)$ :

$$sk := s \leftarrow \$ \mathcal{R}_q$$

**Enc**( $sk, m \in \mathcal{R}_q$ )  $\rightarrow \mathbf{c}$

$$a \leftarrow \$ \mathcal{R}_q$$

$$\mathbf{c} := (-a \cdot sk + m, a)$$

**Dec**( $sk, \mathbf{c}$ )  $\rightarrow m$

$$m = \mathbf{c}[0] + sk \cdot \mathbf{c}[1]$$

**Evaluation** –  $\forall sk \in \mathcal{R}_q$  and  $\forall m_1, m_2 \in \mathcal{R}_q$

$$\mathbf{c}_1 = \text{Enc}(sk, m_1) \text{ and } \mathbf{c}_2 = \text{Enc}(sk, m_2)$$

- **Multiplication?**

$$\begin{aligned} \mathbf{c}_1 \cdot \mathbf{c}_2 &= (-a_1 \cdot sk + m_1, a_1) \cdot (-a_2 \cdot sk + m_2, a_2) \\ &= ((-a_1 \cdot sk + m_1) \cdot (-a_2 \cdot sk + m_2), a_1 \cdot a_2) \\ &= sk^2 a_1 a_2 - sk(a_1 m_2 + m_1 a_2) + m_1 m_2, a_1 \cdot a_2 \end{aligned}$$

Cross-terms that cannot be reconstructed at decryption!

# Fully Homomorphic Encryption

## A simple scheme for intuition – Multiplication

$$\mathcal{R}_q = \mathbb{Z}_q[X]/(X^N + 1)$$

IdealHE:

**KeyGen**  $\rightarrow (sk, \cdot)$ :

$$sk := s \leftarrow \$ \mathcal{R}_q$$

**Enc**( $sk, m \in \mathcal{R}_q$ )  $\rightarrow \mathbf{c}$

$$a \leftarrow \$ \mathcal{R}_q$$

$$\mathbf{c} := (-a \cdot sk + m, a)$$

**Dec**( $sk, \mathbf{c}$ )  $\rightarrow m$

$$m = \mathbf{c}[0] + sk \cdot \mathbf{c}[1]$$

**Evaluation** –  $\forall sk \in \mathcal{R}_q$  and  $\forall m_1, m_2 \in \mathcal{R}_q$

$$\mathbf{c}_1 = \text{Enc}(sk, m_1) \text{ and } \mathbf{c}_2 = \text{Enc}(sk, m_2)$$

- **Multiplication?**

$$\begin{aligned} \mathbf{c}_1 \cdot \mathbf{c}_2 &= (-a_1 \cdot sk + m_1, a_1) \cdot (-a_2 \cdot sk + m_2, a_2) \\ &= ((-a_1 \cdot sk + m_1) \cdot (-a_2 \cdot sk + m_2), a_1 \cdot a_2) \\ &= sk^2 a_1 a_2 - sk(a_1 m_2 + m_1 a_2) + m_1 m_2, a_1 \cdot a_2 \end{aligned}$$

Cross-terms that cannot be reconstructed at decryption!

# Fully Homomorphic Encryption

## A simple scheme for intuition – Multiplication

$$\mathcal{R}_q = \mathbb{Z}_q[X]/(X^N + 1)$$

IdealHE:

**KeyGen**  $\rightarrow (sk, \cdot)$ :

$$sk := s \leftarrow \$ \mathcal{R}_q$$

**Enc**( $sk, m \in \mathcal{R}_q$ )  $\rightarrow \mathbf{c}$

$$a \leftarrow \$ \mathcal{R}_q$$

$$\mathbf{c} := (-a \cdot sk + m, a)$$

**Dec**( $sk, \mathbf{c}$ )  $\rightarrow m$

$$m = \mathbf{c}[0] + sk \cdot \mathbf{c}[1]$$

**Evaluation** –  $\forall sk \in \mathcal{R}_q$  and  $\forall m_1, m_2 \in \mathcal{R}_q$

$$\mathbf{c}_1 = \text{Enc}(sk, m_1) \text{ and } \mathbf{c}_2 = \text{Enc}(sk, m_2)$$

- **Multiplication: Tensor product**

$$\mathbf{c} = \mathbf{c}_1 \times \mathbf{c}_2 = \begin{pmatrix} \mathbf{c}_1[0] \cdot \mathbf{c}_2[0] \\ \mathbf{c}_1[0] \cdot \mathbf{c}_2[1] + \mathbf{c}_1[1] \cdot \mathbf{c}_2[0] \\ \mathbf{c}_1[1] \cdot \mathbf{c}_2[1] \end{pmatrix}$$

$$\mathbf{c} = \mathbf{c}_1 \times \mathbf{c}_2 = \begin{pmatrix} sk^2 c_{mul} - sk b_{mul} + m_1 m_2 \\ -2 sk c_{mul} + b_{mul} \\ c_{mul} \end{pmatrix}$$

$$c_{mul} = a_1 \cdot a_2$$

$$b_{mul} = a_1 m_2 + m_1 a_2$$

# Fully Homomorphic Encryption

## A simple scheme for intuition – Multiplication

$$\mathcal{R}_q = \mathbb{Z}_q[X]/(X^N + 1)$$

IdealHE:

**KeyGen**  $\rightarrow (sk, \cdot)$ :

$$sk := s \leftarrow \$ \mathcal{R}_q$$

**Enc**( $sk, m \in \mathcal{R}_q$ )  $\rightarrow \mathbf{c}$

$$a \leftarrow \$ \mathcal{R}_q$$

$$\mathbf{c} := (-a \cdot sk + m, a)$$

**Dec**( $sk, \mathbf{c}$ )  $\rightarrow m$

$$m = \mathbf{c}[0] + sk \cdot \mathbf{c}[1]$$

**Evaluation** –  $\forall sk \in \mathcal{R}_q$  and  $\forall m_1, m_2 \in \mathcal{R}_q$

$$\mathbf{c}_1 = \text{Enc}(sk, m_1) \text{ and } \mathbf{c}_2 = \text{Enc}(sk, m_2)$$

- **Multiplication: Tensor product + Decryption**

$$\mathbf{c}_1 \times \mathbf{c}_2 = \begin{pmatrix} sk^2 c_{mul} - sk b_{mul} + m_1 m_2 \\ -2 sk c_{mul} + b_{mul} \\ c_{mul} \end{pmatrix}$$

Define **Dec'\_2**( $sk, \mathbf{c}$ )  $\rightarrow m = \mathbf{c}[0] + sk \cdot \mathbf{c}[1] + sk^2 \cdot \mathbf{c}[2]$

This extend to larger depth.



Relinearization: converts  
to degree 1-ciphertext

Degree-2  
ciphertext

# Fully Homomorphic Encryption

## A simple scheme for intuition – Multiplication

$$\mathcal{R}_q = \mathbb{Z}_q[X]/(X^N + 1)$$

IdealHE:

**KeyGen**  $\rightarrow (sk, \cdot)$ :

$$sk := s \leftarrow \$ \mathcal{R}_q$$

**Enc**( $sk, m \in \mathcal{R}_q$ )  $\rightarrow \mathbf{c}$

$$a \leftarrow \$ \mathcal{R}_q$$

$$\mathbf{c} := (-a \cdot sk + m, a)$$

**Dec**( $sk, \mathbf{c}$ )  $\rightarrow m$

$$m = \mathbf{c}[0] + sk \cdot \mathbf{c}[1]$$

**Compactness** – Need for **relinearization**

We have seen that homomorphic multiplication **increases the size of the ciphertexts**.

**Relinearization:** converts a degree-2 ciphertext  $\mathbf{c}$  into a degree-1 ciphertext  $\mathbf{c}'$ .

**Requires:** a relinearization key  $rlk = \text{Enc}(sk, sk^2)$

$$\mathbf{c}' = \text{Relin}(rlk, \mathbf{c}) = \begin{pmatrix} \mathbf{c}[0] + \mathbf{c}[2] \cdot rlk[0] \\ \mathbf{c}[1] + \mathbf{c}[2] \cdot rlk[1] \end{pmatrix}$$

# Fully Homomorphic Encryption

## A simple scheme for intuition – Summary

$$\mathcal{R}_q = \mathbb{Z}_q[X]/(X^N + 1)$$

### IdealHE:

**KeyGen**  $\rightarrow (sk, rlk)$ :

$$rlk = \text{Enc}(sk, sk^2)$$

$$sk := s \leftarrow \$ \mathcal{R}_q$$

**Enc**( $sk, m \in \mathcal{R}_q$ )  $\rightarrow \mathbf{c}$

$$a \leftarrow \$ \mathcal{R}_q$$

$$\mathbf{c} := (-a \cdot sk + m, a)$$

$$\text{Relin}(rlk, \mathbf{c}) = (c[0], c[1]) + c[2] \cdot rlk$$

$$\text{Dec}(sk, \mathbf{c}) \rightarrow m = c[0] + sk \cdot c[1]$$

$$\text{Add}(\mathbf{c}_1, \mathbf{c}_2) = \mathbf{c}_1 + \mathbf{c}_2$$

$$\text{Mul}(\mathbf{c}_1, \mathbf{c}_2) = \text{Relin}(rlk, \mathbf{c}_1 \times \mathbf{c}_2)$$

**Correctness** –

**Evaluation** –

**Compactness** –

**Public Key** –

**Security** –



# Fully Homomorphic Encryption

## A simple scheme for intuition – Summary

$$\mathcal{R}_q = \mathbb{Z}_q[X]/(X^N + 1)$$

### IdealHE:

**KeyGen**  $\rightarrow (sk, rlk)$ :

$$rlk = \text{Enc}(sk, sk^2)$$

$$sk := s \leftarrow \$ \mathcal{R}_q$$

**Enc**( $sk, m \in \mathcal{R}_q$ )  $\rightarrow \mathbf{c}$

$$a \leftarrow \$ \mathcal{R}_q$$

$$\mathbf{c} := (-a \cdot sk + m, a)$$

$$\text{Relin}(rlk, \mathbf{c}) = (\mathbf{c}[0], \mathbf{c}[1]) + \mathbf{c}[2] \cdot rlk$$

$$\text{Dec}(sk, \mathbf{c}) \rightarrow m = \mathbf{c}[0] + sk \cdot \mathbf{c}[1]$$

$$\text{Add}(\mathbf{c}_1, \mathbf{c}_2) = \mathbf{c}_1 + \mathbf{c}_2$$

$$\text{Mul}(\mathbf{c}_1, \mathbf{c}_2) = \text{Relin}(rlk, \mathbf{c}_1 \times \mathbf{c}_2)$$

**Correctness** – Yes

**Evaluation** – Additions and Multiplications

**Compactness** – with **relinearization**

**Public Key** – Yes,  $pk = \text{Enc}(sk, 0)$  and  $u \cdot pk$  to **Enc** for  $u \leftarrow \mathcal{R}_q$

**Security** –

# Fully Homomorphic Encryption

## A simple scheme for intuition – Summary

$$\mathcal{R}_q = \mathbb{Z}_q[X]/(X^N + 1)$$

### IdealHE:

**KeyGen**  $\rightarrow (sk, rlk)$ :

$$rlk = \text{Enc}(sk, sk^2)$$

$$sk := s \leftarrow \$ \mathcal{R}_q$$

**Enc**( $sk, m \in \mathcal{R}_q$ )  $\rightarrow \mathbf{c}$

$$a \leftarrow \$ \mathcal{R}_q$$

$$\mathbf{c} := (-a \cdot sk + m, a)$$

$$\text{Relin}(rlk, \mathbf{c}) = (\mathbf{c}[0], \mathbf{c}[1]) + \mathbf{c}[2] \cdot rlk$$

$$\text{Dec}(sk, \mathbf{c}) \rightarrow m = \mathbf{c}[0] + sk \cdot \mathbf{c}[1]$$

$$\text{Add}(\mathbf{c}_1, \mathbf{c}_2) = \mathbf{c}_1 + \mathbf{c}_2$$

$$\text{Mul}(\mathbf{c}_1, \mathbf{c}_2) = \text{Relin}(rlk, \mathbf{c}_1 \times \mathbf{c}_2)$$

**Correctness** – Yes

**Evaluation** – Additions and Multiplications

**Compactness** – with **relinearization**

**Public Key** – Yes,  $pk = \text{Enc}(sk, 0)$  and  $u \cdot pk$  to **Enc** for  $u \leftarrow \mathcal{R}_q$

**Security** – **No!** The problem is not hard. We need the error!

# Lattice-Based Homomorphic Encryption

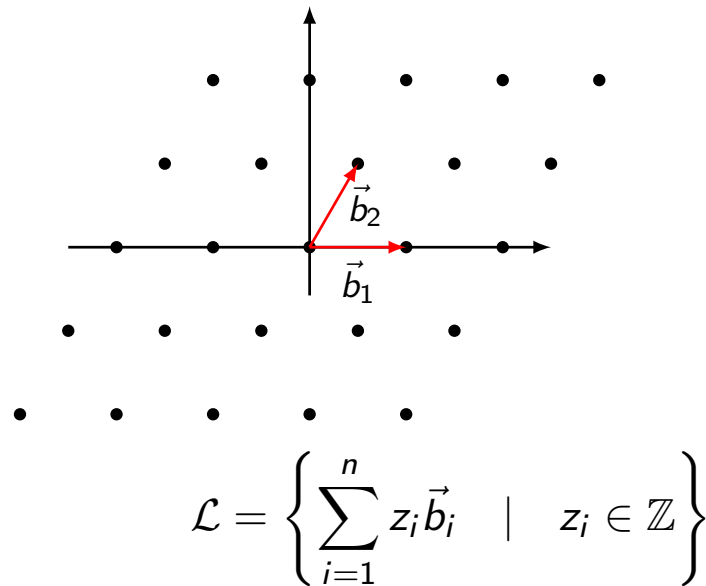
## Background on lattices

A lattice is a mathematical object: a subset of  $\mathbb{R}^n$  that is an additive subgroup and discrete

There exists many problems deemed hard related to lattices

e.g., Shortest Vector Problem: find non-zero vector of low norm

- Large mathematical objects
- Several lattice-based hard problems
- Assumed quantum resistant for appropriate parametrization



# Ring learning with error

## Intuition

Define  $\mathcal{R}_q = \mathbb{Z}_q[X]/(X^N + 1)$  Degree  $N - 1$  polynomials with coefficients in  $\mathbb{Z}_q$

Given a secret vector  $s \in \mathcal{R}_q$ , the  $\text{RLWE}_{\mathcal{R}_q, \chi}^s$  distribution outputs:

$$(a \ b) = (a, as + e) \in \mathcal{R}_q^2$$

where  $\vec{a} \leftarrow \mathcal{R}_q$  and  $e \leftarrow \chi$   $m = \text{poly}(n)$ , appropriate  $q$ , and  $\chi$  of error rate  $\alpha < 1$

Cannot distinguish between  $\text{RLWE}_{\mathcal{R}_q, \chi}^s$  and a uniform distribution

# Ring learning with error-based FHE scheme

## Issue with correctness

$$\mathcal{R}_q = \mathbb{Z}_q[X]/(X^N + 1)$$

**FHE:**

**KeyGen**  $\rightarrow (sk, \cdot)$ :  $a, b \leftarrow \mathcal{R}_q$   
 $sk := s \leftarrow \$ \mathcal{R}_q$

**Enc**( $sk, m \in \mathcal{R}_q$ )  $\rightarrow \mathbf{c}$   
 $a \leftarrow \$ \mathcal{R}_q$   
 $\mathbf{c} := (-a \cdot sk + m, a)$

**Dec**( $sk, \mathbf{c}$ )  $\rightarrow m = \mathbf{c}[0] + sk \cdot \mathbf{c}[1]$

**Idea: add noise everywhere!**

$(a, b) = (a, as + e) \in \mathcal{R}_q^2$  with  $\vec{a} \leftarrow \mathcal{R}_q$  and  $e \leftarrow \chi$

**Informal:** RLWE assumption says this is indistinguishable from  $(a, b \leftarrow \$ \mathcal{R}_q)$ .

# Ring learning with error-based FHE scheme

## Issue with correctness

$$\mathcal{R}_q = \mathbb{Z}_q[X]/(X^N + 1)$$

**FHE:**

**KeyGen**  $\rightarrow (sk, \cdot)$ :  $a, b \leftarrow \mathcal{R}_q$   
 $sk := s \leftarrow \$ \mathcal{R}_q$

**Enc**( $sk, m \in \mathcal{R}_q$ )  $\rightarrow \mathbf{c}$   
 $a \leftarrow \$ \mathcal{R}_q$   $e_{enc} \leftarrow \chi$   
 $\mathbf{c} := (-a \cdot sk + e_{enc} + m, a)$

**Dec**( $sk, \mathbf{c}$ )  $\rightarrow m = \mathbf{c}[0] + sk \cdot \mathbf{c}[1]$

**Idea: add noise everywhere!**

$(a, b) = (a, as + e) \in \mathcal{R}_q^2$  with  $\vec{a} \leftarrow \mathcal{R}_q$  and  $e \leftarrow \chi$

**Correctness**  $- \forall sk \in \mathcal{R}_q$  and  $\forall m \in \mathcal{R}_q$

$$\begin{aligned} \text{Dec}(sk, \text{Enc}(sk, m)) &= (-a \cdot sk + e_{enc} + m) + sk \cdot a \\ &= (-as + e_{enc} + m) + s \cdot a \\ &= e_{enc} + m \neq m \end{aligned}$$

**We need to perform error correction:**

1. Message scaling
2. Noise scaling

# Ring learning with error-based FHE scheme

## Adding error correction – message scaling

$$\mathcal{R}_q = \mathbb{Z}_q[X]/(X^N + 1)$$

**FHE:**

**KeyGen**  $\rightarrow (sk, \cdot)$ :  $a, b \leftarrow \mathcal{R}_q$   
 $sk := s \leftarrow \$ \mathcal{R}_q$

**Enc**( $sk, m \in \mathcal{R}_q$ )  $\rightarrow \mathbf{c}$   
 $a \leftarrow \$ \mathcal{R}_q$   $e_{enc} \leftarrow \chi$   
 $\mathbf{c} := (-a \cdot sk + e_{enc} + \Delta m, a)$

**Dec**( $sk, \mathbf{c}$ )  $\rightarrow m$   
 $M := \mathbf{c}[0] + sk \cdot \mathbf{c}[1]$   
 $m := \lfloor M/\Delta \rfloor$

Let  $\Delta \in \mathbb{Z}_q$  be a positive factor less than  $q$ .

If  $\Delta > 2e$  and  $\Delta m < q$ , then the division and rounding remove the error.

We now have correctness again!

# Ring learning with error-based FHE scheme

## Adding error correction – message scaling

$$\mathcal{R}_q = \mathbb{Z}_q[X]/(X^N + 1)$$

**FHE:**

**KeyGen**  $\rightarrow (sk, \cdot)$ :  $a, b \leftarrow \mathcal{R}_q$   
 $sk := s \leftarrow \$ \mathcal{R}_q$

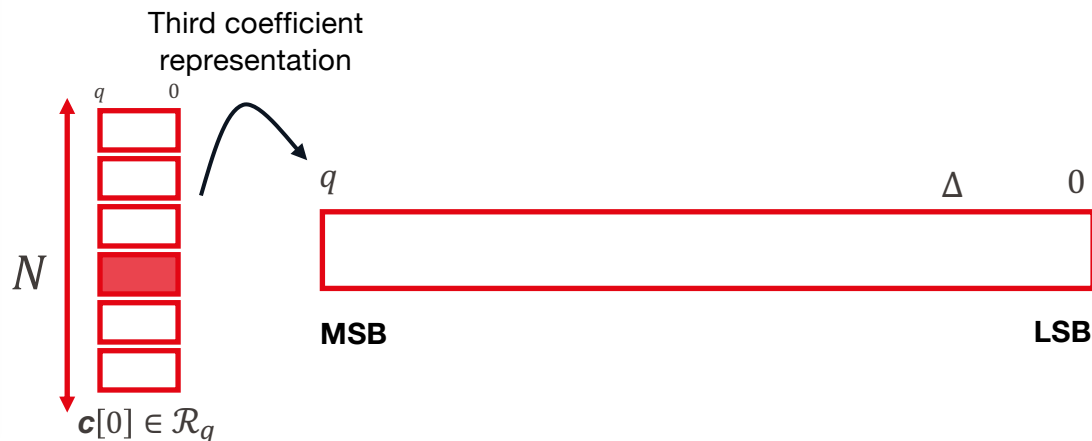
**Enc**( $sk, m \in \mathcal{R}_q$ )  $\rightarrow \mathbf{c}$   
 $a \leftarrow \$ \mathcal{R}_q$   $e_{enc} \leftarrow \chi$   
 $\mathbf{c} := (-a \cdot sk + e_{enc} + \Delta m, a)$

**Dec**( $sk, \mathbf{c}$ )  $\rightarrow m$   
 $M := \mathbf{c}[0] + sk \cdot \mathbf{c}[1]$   
 $m := \lfloor M/\Delta \rfloor$

For simplicity, the relinearization and PKE have been omitted. In practice,  
 Disclaimer: it is non-trivial to enable relinearization: use gadgets products

Let us look at the **impact** of the scale  $\Delta$

Consider the following representation of the polynomial coefficients





# Ring learning with error-based FHE scheme

## Adding error correction – message scaling

$$\mathcal{R}_q = \mathbb{Z}_q[X]/(X^N + 1)$$

**FHE:**

**KeyGen**  $\rightarrow (sk, \cdot)$ :  $a, b \leftarrow \mathcal{R}_q$

$sk := s \leftarrow \$ \mathcal{R}_q$

**Enc**( $sk, m \in \mathcal{R}_q$ )  $\rightarrow \mathbf{c}$

$a \leftarrow \$ \mathcal{R}_q$   $e_{enc} \leftarrow \chi$

$\mathbf{c} := (-a \cdot sk + e_{enc} + \Delta m, a)$

**Dec**( $sk, \mathbf{c}$ )  $\rightarrow m$

$M := \mathbf{c}[0] + sk \cdot \mathbf{c}[1]$

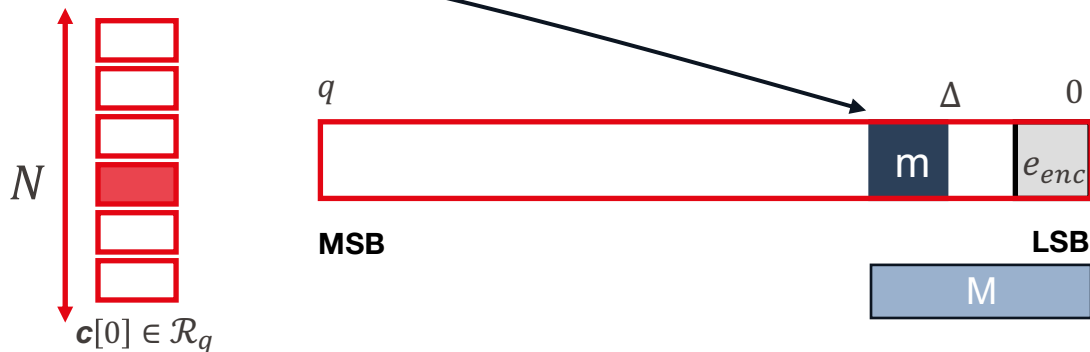
$m := \lfloor M/\Delta \rfloor$

For simplicity, the relinearization and PKE have been omitted. In practice, Disclaimer: it is non-trivial to enable relinearization: use gadgets products

Let us look at the **impact** of the scale

We display the scaled message  $m$  and the encryption noise  $e_{enc}$

Third coefficient representation



# Ring learning with error-based FHE scheme

## Adding error correction – message scaling

$$\mathcal{R}_q = \mathbb{Z}_q[X]/(X^N + 1)$$

**FHE:**

**KeyGen**  $\rightarrow (sk, \cdot)$ :  $a, b \leftarrow \mathcal{R}_q$

$sk := s \leftarrow \$ \mathcal{R}_q$

**Enc**( $sk, m \in \mathcal{R}_q$ )  $\rightarrow \mathbf{c}$

$a \leftarrow \$ \mathcal{R}_q$   $e_{enc} \leftarrow \chi$

$\mathbf{c} := (-a \cdot sk + e_{enc} + \Delta m, a)$

**Dec**( $sk, \mathbf{c}$ )  $\rightarrow m$

$M := \mathbf{c}[0] + sk \cdot \mathbf{c}[1]$

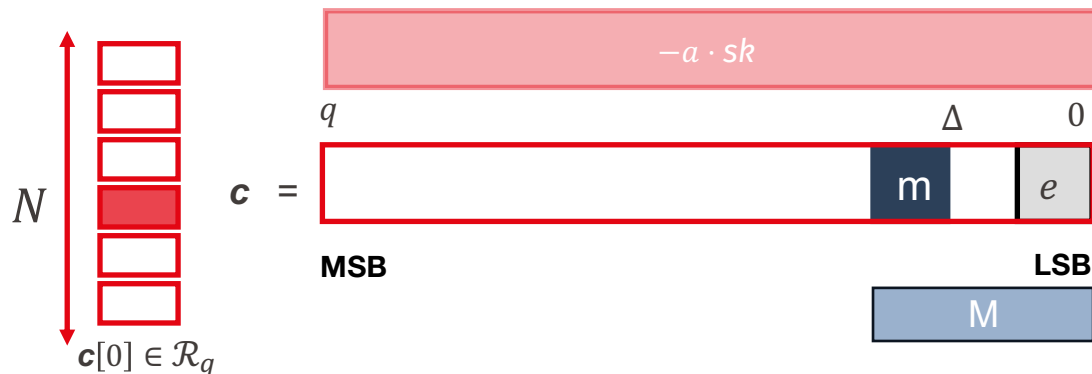
$m := \lfloor M/\Delta \rfloor$

For simplicity, the relinearization and PKE have been omitted. In practice, Disclaimer: it is non-trivial to enable relinearization: use gadgets products

Let us look at the **impact** of the scale

We now add the masking

Third coefficient  
representation



# Ring learning with error-based FHE scheme

## Adding error correction – message scaling and Ops

### FHE:

$$\mathcal{R}_q = \mathbb{Z}_q[X]/(X^N + 1)$$

**KeyGen**  $\rightarrow (sk, \cdot)$ :  $a, b \leftarrow \mathcal{R}_q$

$sk := s \leftarrow \$ \mathcal{R}_q$

**Enc**( $sk, m \in \mathcal{R}_q$ )  $\rightarrow \mathbf{c}$

$a \leftarrow \$ \mathcal{R}_q$   $e_{enc} \leftarrow \chi$

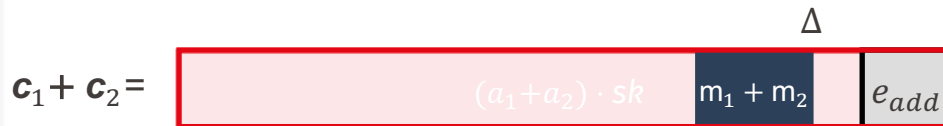
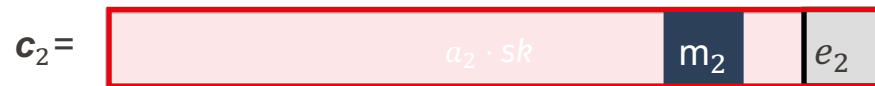
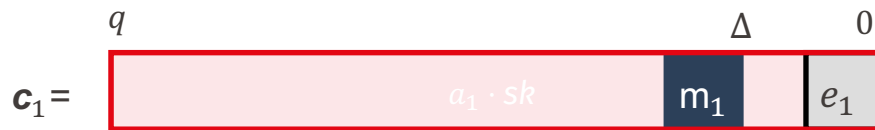
$\mathbf{c} := (-a \cdot sk + e_{enc} + \Delta m, a)$

**Dec**( $sk, \mathbf{c}$ )  $\rightarrow m$

$M := \mathbf{c}[0] + sk \cdot \mathbf{c}[1]$

$m := \lfloor M/\Delta \rfloor$

For simplicity, the relinearization and PKE have been omitted. In practice,  
Disclaimer: it is non-trivial to enable relinearization: use gadgets products



The noise growth is linear in the #add in the worst case.



The noise growth is quadratically in the #mult in the worst case.  
Output message scaled by  $\Delta^2$

# Ring learning with error-based FHE scheme

## Adding error correction – message scaling and Ops

### FHE:

$$\mathcal{R}_q = \mathbb{Z}_q[X]/(X^N + 1)$$

**KeyGen**  $\rightarrow (sk, \cdot)$ :  $a, b \leftarrow \mathcal{R}_q$

$$sk := s \leftarrow \$ \mathcal{R}_q$$

**Enc**( $sk, m \in \mathcal{R}$ )  $\rightarrow \mathbf{c}$

$$a \leftarrow \$ \mathcal{R}_q \quad e_{enc} \leftarrow \chi$$

$$\mathbf{c} := (-a \cdot sk + e_{enc} + \Delta m, a)$$

**Dec**( $sk, \mathbf{c}$ )  $\rightarrow m$

$$M := \mathbf{c}[0] + sk \cdot \mathbf{c}[1]$$

$$m := \lfloor M/\Delta \rfloor$$



What can we do? – Noise management



# Ring learning with error-based FHE scheme

## Adding error correction – message scaling and Ops

### FHE:

$$\mathcal{R}_q = \mathbb{Z}_q[X]/(X^N + 1)$$

**KeyGen**  $\rightarrow (sk, \cdot)$ :  $a, b \leftarrow \mathcal{R}_q$

$sk := s \leftarrow \$ \mathcal{R}_q$

**Enc**( $sk, m \in \mathcal{R}_t$ )  $\rightarrow \mathbf{c}$

$a \leftarrow \$ \mathcal{R}_q$   $e_{enc} \leftarrow \chi$

$\mathbf{c} := (-a \cdot sk + e_{enc} + \Delta m, a)$

**Dec**( $sk, \mathbf{c}$ )  $\rightarrow m$

$M := \mathbf{c}[0] + sk \cdot \mathbf{c}[1]$

$m := \lfloor M/\Delta \rfloor$

For simplicity, the relinearization and PKE have been omitted. In practice,  
Disclaimer: it is non-trivial to enable relinearization: use gadgets products



What can we do? – Noise management

### 1. Select $\Delta$ wisely:

Encrypt messages in  $\mathcal{R}_t$ ,  $t \ll q$ . Set  $\Delta = \lfloor \frac{q}{t} \rfloor$



2. **Rescale** from  $\Delta^2$  needed (multiply by  $\frac{t}{q}$  in  $\mathcal{R}$ )

3. **Perform noise refresh:** Bootstrapping



# Ring learning with error-based FHE scheme

## Summary

**Correctness** – Yes

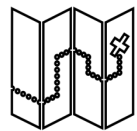
**Evaluation** – Additions and Multiplications

**Compactness** – with **relinearization**

**Security** – RLWE assumption

### Note:

- Modulus  $q$  is large: **decompose** it in smaller primes (Chinese remainder Theorem)
- Plaintext space is  $\mathcal{R}_t$ : **batch**  $N$  values in  $\mathbb{Z}_t$
- By selecting appropriate parameters, enable **fast multiplication** and **SIMD**



Amortize operations!  
Pack as much as possible

### FHE:

$$\mathcal{R}_q = \mathbb{Z}_q[X]/(X^N + 1)$$

**KeyGen**  $\rightarrow (sk, pk, rlk)$ :

$$sk := s \leftarrow \$ \mathcal{R}_q$$

$$rlk = \text{GenRLK}(sk) \quad rlk = \text{GenPK}(sk)$$

**Enc**( $sk, m \in \mathcal{R}_t$ )  $\rightarrow c$

$$a \leftarrow \$ \mathcal{R}_q \quad e_{enc} \leftarrow \chi, \Delta = \lfloor \frac{q}{t} \rfloor$$

$$c := (-a \cdot sk + e_{enc} + \Delta m, a)$$

**Relin**( $rlk, c$ )  $\rightarrow c'$

**Dec**( $sk, c$ )  $\rightarrow m$

$$\text{Add}(c_1, c_2) = c_1 + c_2$$

$$\text{Mul}(c_1, c_2) = \text{Relin}(rlk, \mathcal{R}(c_1 \times c_2) / \Delta)$$

SIMD: single input multiple data

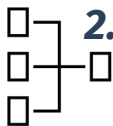
# FHE in Practice

## Challenges

### 1. **Selecting cryptographic parameters**



Interdependencies between  $N$ ,  $q$ ,  $t$ , and  $\chi$   
Relies on estimators to assess the hardness



### 2. **Circuit definition**

How to represent functions into circuits  
Minimize the multiplicative depth



Optimize the costly operations (bootstrap, rescale, etc.)



### 3. **Bearing with the costs**

FHE induces both computation and communication overhead  
Not all plaintexts space can be easily handled

#### Naïve operation

$$(((m_1 \cdot m_2) \cdot m_3) \cdot m_4)$$

Diagram illustrating the Naïve operation: A sequence of three nested multiplications. The first multiplication  $m_1 \cdot m_2$  is bracketed and labeled  $+e$  below it. This result is then multiplied by  $m_3$ , with a bracket labeled  $+e^2$  below it. Finally, this result is multiplied by  $m_4$ , with a bracket labeled  $+e^3$  below it.

#### Optimization

$$((m_1 \cdot m_2) \cdot (m_3 \cdot m_4))$$

Diagram illustrating the Optimization: Two parallel multiplications,  $m_1 \cdot m_2$  and  $m_3 \cdot m_4$ , are each bracketed and labeled  $+e$  below them. These two results are then multiplied together, with a bracket labeled  $+e^2$  below it.

# FHE in Practice

## Parameterization

- **Dimension  $N$ :** between  $2^{11}$  and  $2^{16}$
- **Ciphertext space:**  $\mathcal{R}_q$  with  $q$  100s of bits
- **Message space:**  $t$  application dependent

polynomial size (keys, ciphertexts)

coefficient size of ciphertexts

can be as small as 20 bits

**Example:** compute Squared Distance between two 2D

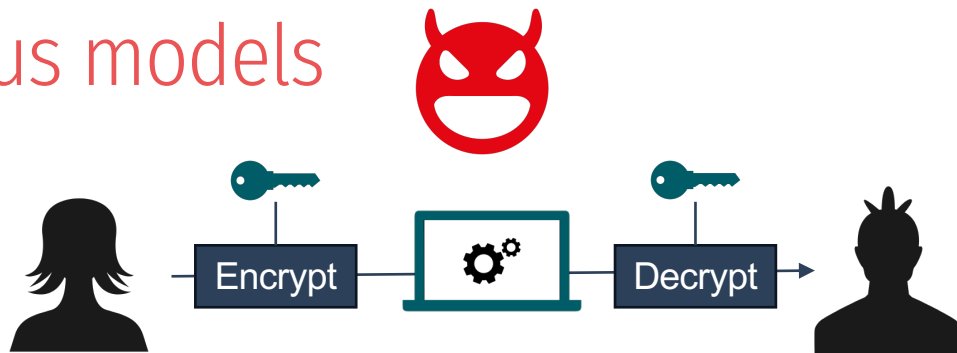
- $\log N = 13, \log q = 218, \log t = 16$
- One ciphertext: 450kB
- **Setup:** 7.4 ms
- **Encode + Enc 1 vector:** 2.7 ms
- **Eval:** 7 ms
- **Dec + Decode:** 2 ms

*But costs can quickly go up!  
For example for  $\log N = 15$   
and depth 16, you might  
need >100ms for one  
multiplication*



# FHE in Practice

## Extension to malicious models



### *Malicious threat model?*

So far, the evaluator can only **evaluate** the function on the input it receives.

In reality, evaluator could be:

- **Honest but Curious:** Parties will follow the HE protocol honestly, but try to learn as much as possible from the messages they receive
- **Malicious:** Parties can arbitrarily deviate from the HE protocol to learn as much as possible

⇒ **Requires heavy machinery e.g., Zero-Knowledge Proofs**

Chiara Marcolla, Victor Sucasas, *Member, IEEE*, Marc Manzano, Riccardo Bassoli, *Member, IEEE*,  
Frank H.P. Fitzek, *Senior Member, IEEE* and Najwa Aaraj

**Abstract**—Data privacy concerns are increasing exponentially in the context of Internet of Things (IoT), cloud services, edge computing, artificial intelligence applications, and other applications involving data generation and processing. Homomorphic Encryption (HE) is a cryptographic primitive that can help address data privacy challenges by enabling multiple operations to be performed on encrypted messages without decryption. This paper comprehensively addresses homomorphic encryption from both theoretical and practical perspectives. The paper delivers insights into the mathematical foundations required to understand fully homomorphic encryption (FHE). It consequently covers design fundamentals and security properties of FHE, and describes the main FHE schemes based on various mathematical problems. On a more practical level, the paper provides a view on privacy-preserving Machine Learning (ML) using homomorphic encryption. It surveys FHE at length from an engineering angle, covering the potential application of FHE in fog computing, and cloud computing services. It also provides a comprehensive analysis of existing state-of-the-art FHE libraries and compares them against existing state-of-the-art secure computation, secure multi-party computation, and secure data aggregation protocols.

**Index Terms**—Fully Homomorphic Encryption, Homomorphic Encryption, Lattices, Neural Networks, Fog Computing, Cloud Computing, IoT.

## 1. INTRODUCTION

The notion of fully homomorphic encryption, originally called privacy homomorphism, was introduced by Rivest, Adleman and Dertouzos [1] in 1978. For more than 30 years, this concept was considered to be the holy grail of cryptography, until 2009, when Gentry proposed the first fully homomorphic encryption scheme in his PhD thesis [2]. Homomorphic encryption enables operations on plaintexts without decryption. Namely, a set of operations can be performed over ciphertexts such that these operations are reflected as additions and multiplications on the corresponding plaintexts. Thus, homomorphic encryption allows data manipulation in the encrypted domain. This has a tremendous application potential since it allows privacy-preserving data processing, and can be adopted in new emerging fields such as machine

C. Marcella, V. Sucasas and N. Aamj are with the Technology Innovation Institute, Masdar City Abu Dhabi, United Arab Emirates. (e-mail: {chiara.marcella,victor.sucasas,noua.aamj}@tii.ae).

M. Manzano is with SandboxesAQ, Palo Alto, CA, U.S. (e-mail: marc@sandboxesaq.com). M. Manzano is also with the Electronics and Computing Department, Faculty of Engineering, Mondragón Universities, Mondragón, Spain.

F.H.P. Fitzek is also with Centre for Tactile Internet with Human in-the-Loop (CeTI), Cluster of Excellence, Dresden, Germany. (e-mail: [fitzek@tu-dresden.de](mailto:fitzek@tu-dresden.de)).

learning, cloud computing, or in the different data processing layers of new generation networks.

Homomorphic encryption schemes that allow one type of operation, or a limited number of operations, have existed for a long time. Some examples are the RSA cryptosystem by Rivest, Shamir and Adleman [3] (1978), encryption schemes of Goldwasser and Micali [4] (1982), ElGamal [5] (1985), Benaloh [6] (1994), Paillier and Naccache and Stern [7] (1998), Paillier [8] (1999) and Boneh, Goh and Nissim [9] (2005). In particular in [9], the authors proposed the first scheme capable of performing basic operations: an arbitrary number of additions and just one multiplication, then again an arbitrary number of additions. Later, Aguilar Melchor, Gaborit and Herranz [10] (2008) proposed a theoretical approach that permits chaining several homomorphic schemes in order to have a fixed amount of multiplications, i.e. more than one, for a given public key.

However, it was not until 2009, when Gentry [21] proposed the first fully homomorphic encryption scheme, that the use of homomorphic encryption in arbitrary circuits. In his work, Gentry not only proposed an FHE scheme, but also provided a method for constructing a general homomorphic evaluation algorithm for arbitrary circuits. The homomorphic evaluation algorithm is based on the idea of homomorphic encryption of a scalar product. Since then, homomorphic encryption has triggered a series of researches, and many new schemes have been proposed with significant interest, and better constructions [22–24]. For example, the GGHV [25] and BGV [26] schemes are based on the hardness of the LWE problem, and the CKKS [27] is the most representative.

The majority of research efforts for FHE schemes are focused on public key encryption. However, some researchers have focused on symmetric key encryption among the scientific community, due to their more limited applicability to cloud computing. Also, some proposed symmetric key encryption schemes are based on the idea of homomorphic encryption [28–30]. Nevertheless, there are some papers that proposed symmetric key FHE schemes, which can be divided into two categories: (1) [17, 19]–[21], in which symmetric key FHE schemes are constructed by using both symmetric and asymmetric key FHE schemes. It is also worth commenting that, in 2011, Rothblum [31] proposed an encryption method to construct a symmetric key homomorphic encryption capacity, into a symmetric one. This survey only covers public key encryption schemes, but more information on symmetric key encryption can

This survey provides a comprehensive vision on homomorphic encryption since its genesis. We extend previous surveys on the topic [16], [23]–[26], to cover the most relevant advances on FHE and its applications. Specifically, the survey is structured as follows: i) Section II provides the preliminaries

Alexander Viand  
ETH Zurich  
alexander.viand@inf.ethz.ch

Patrick Jattke  
ETH Zurich  
pjattke@ethz.ch

**Abstract**—Fully Homomorphic Encryption (FHE) allows a third party to perform arbitrary computations on encrypted data, learning neither the inputs nor the computation results. Hence, it provides resilience in situations where computations are carried out by an untrusted or potentially compromised party. This powerful concept was first conceived by Rivest et al. in the 1970s. However, it remained unrealized until Craig Gentry

The advent of the massive collection of sensitive data in cloud services, coupled with a plague of data breaches, moved highly regulated businesses to increasingly demand confidential and secure computing solutions. This demand, in turn, has led to a recent surge in the development of FHE tools. To understand the landscape of recent FHE tool developments, we conducted an extensive survey and experimental evaluation to explore the current state of the art and identify areas for future development.

In this paper, we survey, evaluate, and systematize FHE tools and compilers. We perform experiments to evaluate these tools' performance and usability aspects on a variety of applications. We conclude with recommendations for developers intending to develop FHE-based applications and a discussion on future directions for FHE tools development.

## 1. INTRODUCTION

Recent years have seen unprecedented growth in the adoption of cloud computing services. More and more highly regulated businesses and organizations (e.g., banks, governments, insurance, health), where data security is paramount, move their data and services to the cloud. This trend has led to a surge in demand for secure and confidential computing solutions that protect data confidentiality while in transit, rest, and in-use. This is an amply justified and expected demand, particularly in the light of the numerous reports of data breaches [1, 2]. Fully Homomorphic Encryption (FHE) is a key technological enabler for secure computation and has recently matured to be practical for real-world use [3–9].

FHE allows arbitrary computations to be performed over encrypted data, eliminating the need to decrypt the data and expose it to potential risk while in use. While first proposed in the 1970s [10], FHE was long considered impossible in the 1990s [11]. However, thanks to advances in the underlying

practical. However, thanks to advances in hardware and more efficient theory, general hardware improvements, and more efficient implementations, it has become increasingly practical. In 2009, breakthrough work from Craig Gentry proposed the first feasible FHE scheme [11]. In the last decade, FHE has gone from a theoretical concept to reality, with performance improving by up to five orders of magnitude. For example, times for a multiplication between ciphertexts dropped from 30 minutes to less than 20 milliseconds. While this is still around

Patrick Jattke  
ETH Zurich  
pjattke@ethz.ch

Anwar Hithnawi  
ETH Zurich  
anwar.hithnawi@inf.ethz.ch

seven orders of magnitude slower than an IMUL instruction on a modern CPU, it is sufficient to make many applications practical. Additionally, modern schemes introduced SIMD-style parallelism, encoding thousands of plaintext values into

single enterprise. These have enabled a wide range of applications. These applications have been enabled by a wide range of domains. These include mobile computing, mobile devices, mobile networks, mobile applications, and mobile services. These applications have been enabled by a wide range of domains. These include mobile computing, mobile devices, mobile networks, mobile applications, and mobile services. These applications have been enabled by a wide range of domains. These include mobile computing, mobile devices, mobile networks, mobile applications, and mobile services.

encryptions. Despite these recent breakthroughs, building secure and efficient FHE-based applications remains a challenging task. This is largely attributed to the differences between traditional programming paradigms and FHE's computation model, which poses unique challenges. For example, virtually all stateful programming paradigms rely on data-dependent branching, e.g., if-else statements and loops. On the other hand, FHE computations are, by definition, data-independent, or they would violate the privacy guarantees. Working with FHE also introduces significant engineering challenges in practice. Different schemes offer varying performance tradeoffs, and optimal choices are heavily application-dependent. To address some of the engineering challenges in this space, we have seen a surge of work on tools that aim to improve accessibility and ease of use, to enter in this field.

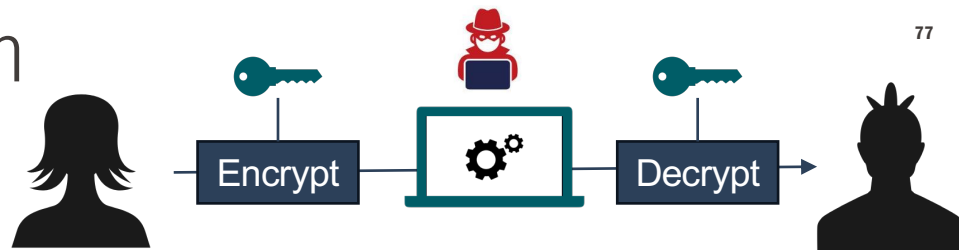
Without tool support, realizing FHE-based computations by implementing the required mathematical operations directly or using an arbitrary-precision arithmetic library is complex, requiring considerable expertise in both cryptography and high-performance numerical computation. Therefore, FHE libraries like the Simple Encrypted Arithmetic Library (SEAL) [20] or the Fast Fully Homomorphic Encryption Library over the

C. Marcolla et al. "Survey on fully homomorphic encryption, theory, and applications." *Proc. of the IEEE* 2022

A. Viand et al. "SoK: Fully homomorphic encryption compilers." *2021 IEEE SP*

# Homomorphic Encryption

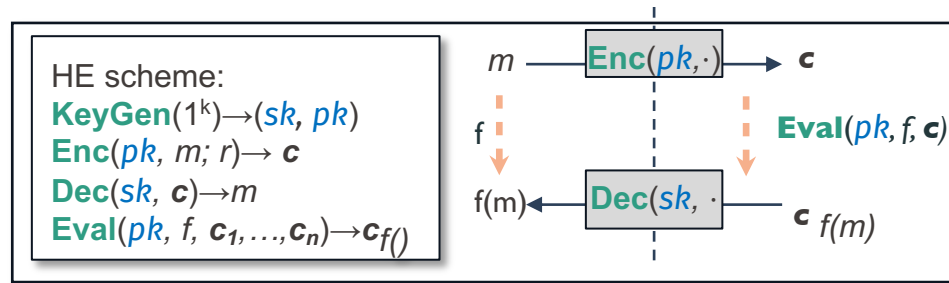
## Conclusion



Enables **computation over encryption**

Has become increasingly **practical**

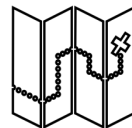
Modern lattice-based schemes plausibly **quantum resistant**



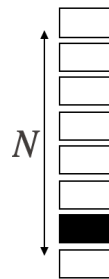
### Challenges:



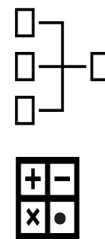
Parameters  
selection



Plaintext  
encoding



Amortization



Circuit  
design



Noise  
management

# References

## Software Libraries

- [HElib] <https://github.com/shaih/HElib> Halevi and Shoup
- [SEAL] <https://www.microsoft.com/en-us/research/project/simple-encrypted-arithmetic-library/> SEAL
- [NFLlib] <https://github.com/quarkslab/NFLlib> French consortium
- [HEAAN] <https://github.com/kimandrik/HEAAN> Korean researchers
- [TFHE] <https://tfhe.github.io/tfhe/> inpher + French researchers
- [PALISADE] <https://git.njit.edu/palisade/PALISADE> New Jersey Institute of Technology
- [cuHE] <https://github.com/vernamlab/cuHE>
- [Lattigo] <https://github.com/tuneinsight/lattigo>
- [OpenFHE] <https://openfhe.org/>
- [THFE-rs] <https://docs.zama.ai/tfhe-rs>