

Machine learning over encrypted data

Rener Oliveira

Rodrigo Targino

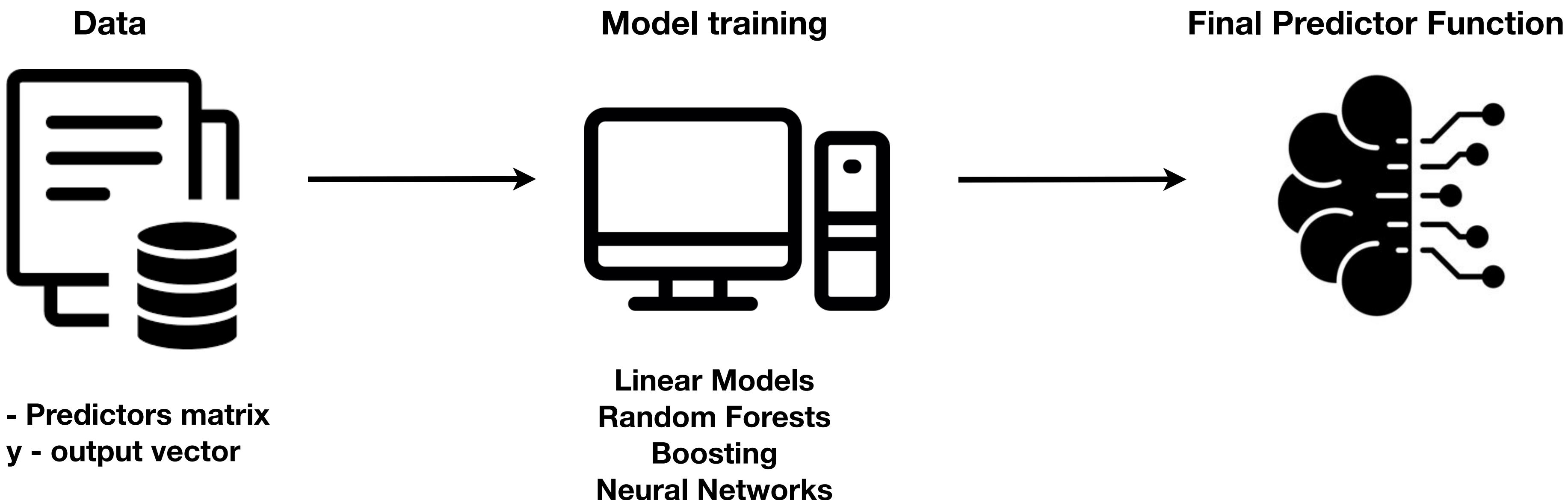
June 27th, 2022

Table of Contents

- **Introduction**
- **Fully Homomorphic Encryption**
- **Somewhat Homomorphic Encryption**
- **Bootstrapping**
- **Literature overview**
- **Calendar**
- **References**

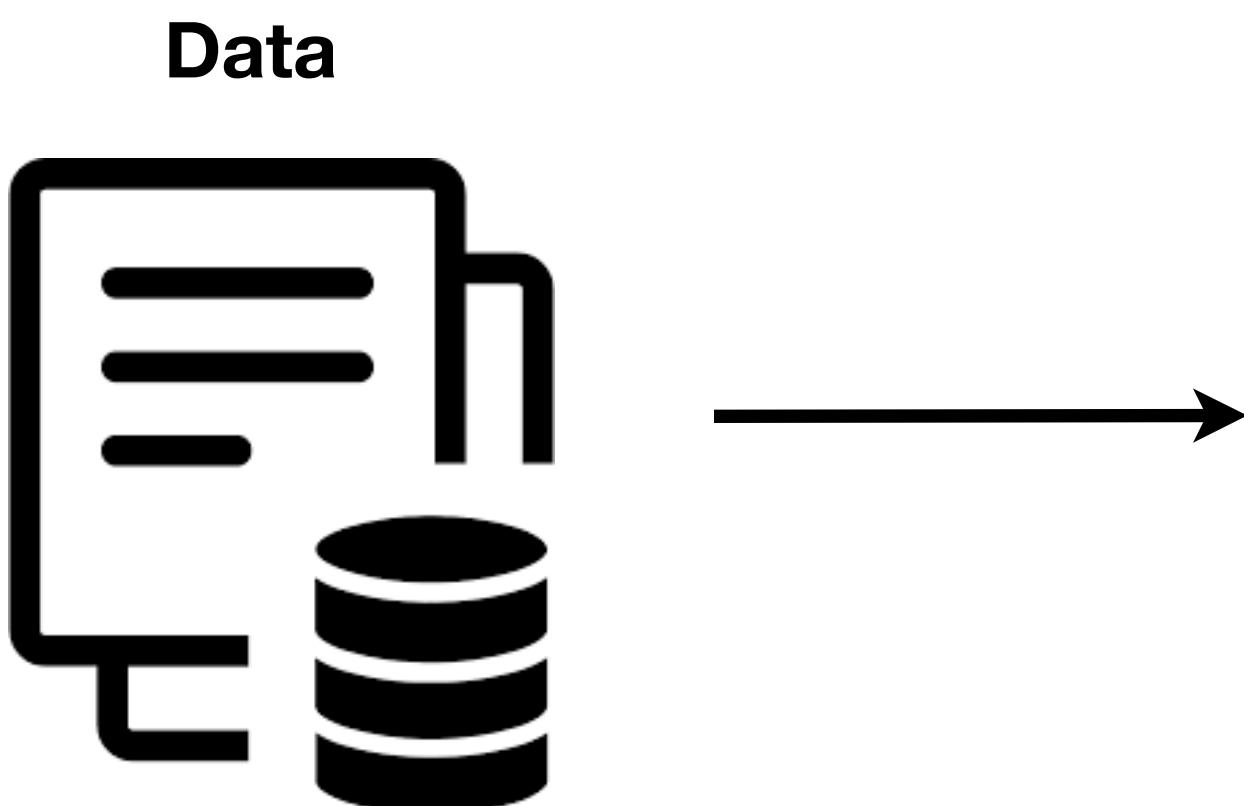
Introduction

Supervised machine learning setup:



Introduction

Supervised machine learning setup:



X - Predictors matrix
y - output vector

What if Data is sensitive?

- Medical records
- Financial data
- Genetic data

Introduction

Supervised machine learning setup:

Encrypted Data



What if Data is sensitive?

- Medical records
- Financial data
- Genetic data

Just encrypt it ;-)

%\$*#!7E% - Encrypted predictors
\$&h%\$8 - Encrypted target vector

Introduction

Supervised machine learning setup:

Encrypted Data



What if Data is sensitive?

- Medical records
- Financial data
- Genetic data

Just encrypt it ;-)

But, can we still train the models over encrypted data?

%\$*#!7E% - Encrypted predictors
\$&h%\$8 - Encrypted target vector

Introduction

Supervised machine learning setup:

Encrypted Data



What if Data is sensitive?

- Medical records
- Financial data
- Genetic data

Just encrypt it ;-)

But, can we still train the models
over encrypted data?
Maybe :)

%\$*#!7E% - Encrypted predictors
\$&h%\$8 - Encrypted target vector

Privacy Homomorphisms, 1978

ON DATA BANKS AND PRIVACY HOMOMORPHISMS

Ronald L. Rivest

Len Adleman

Michael L. Dertouzos

Massachusetts Institute of Technology
Cambridge, Massachusetts

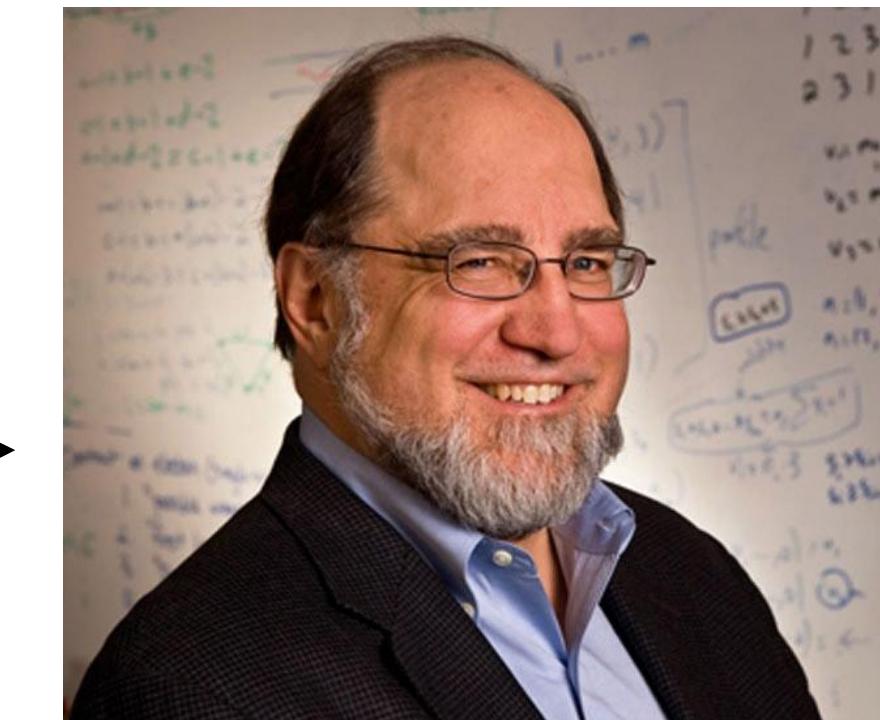
Privacy Homomorphisms, 1978

ON DATA BANKS AND PRIVACY HOMOMORPHISMS

Ronald L. Rivest

Len Adleman

Michael L. Dertouzos



Massachusetts Institute of Technology
Cambridge, Massachusetts

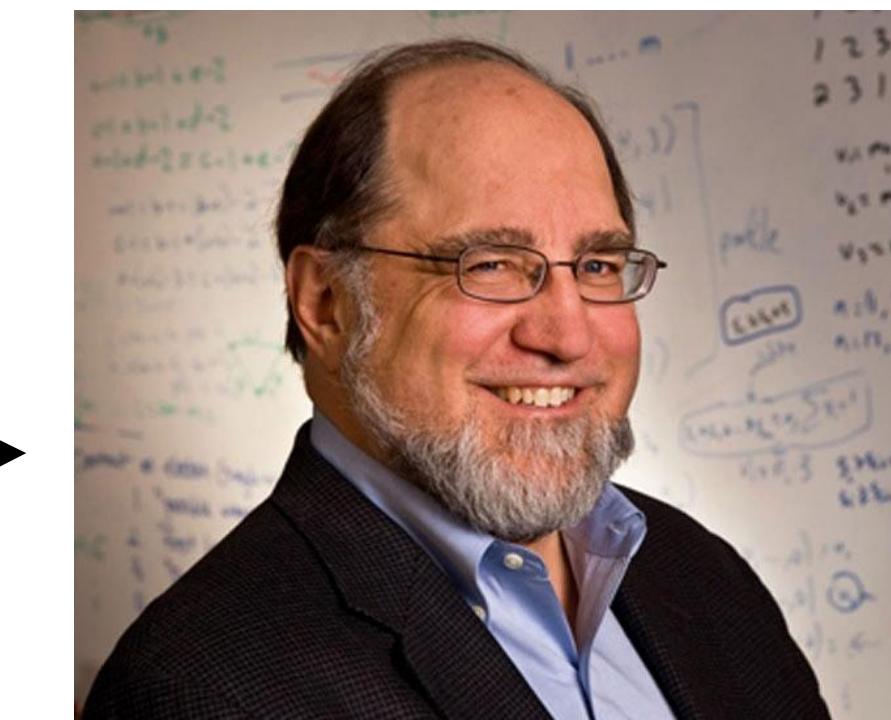
Privacy Homomorphisms, 1978

ON DATA BANKS AND PRIVACY HOMOMORPHISMS

Ronald L. Rivest

Len Adleman

Michael L. Dertouzos



Massachusetts Institute of Technology
Cambridge, Massachusetts

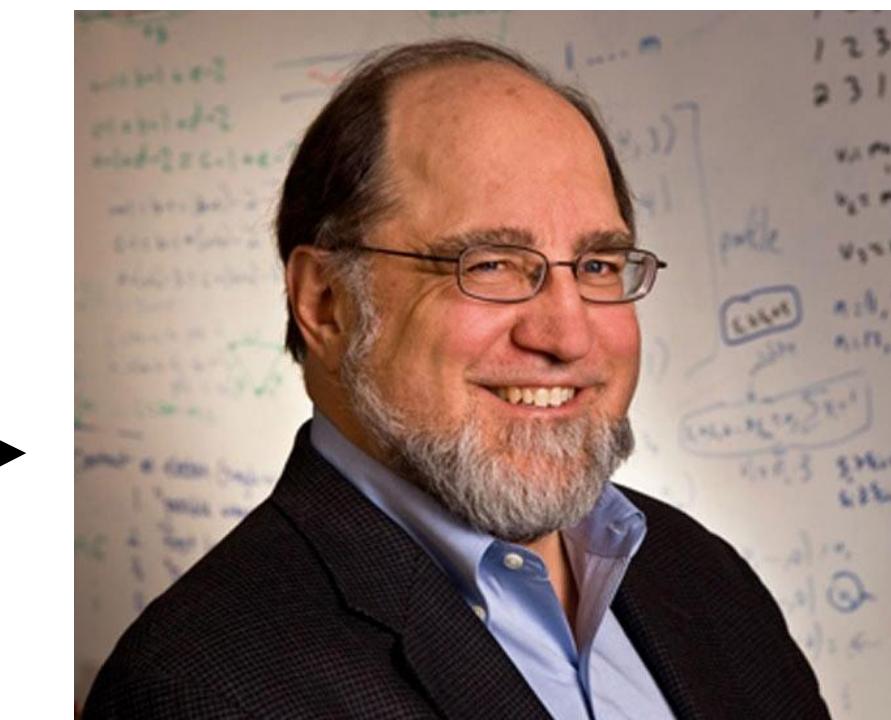
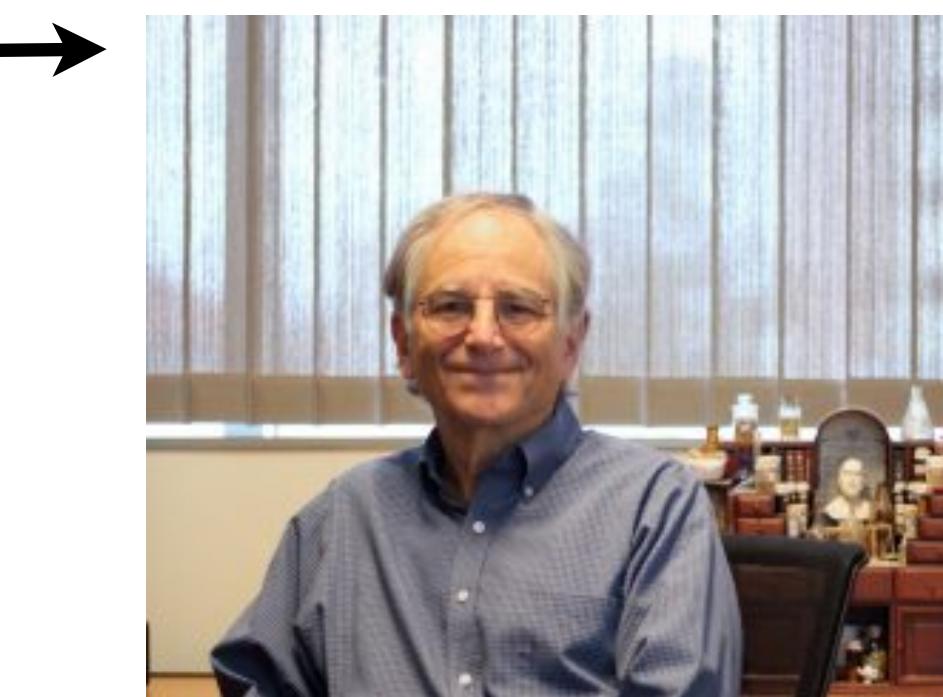
Privacy Homomorphisms, 1978

ON DATA BANKS AND PRIVACY HOMOMORPHISMS

Ronald L. Rivest

Len Adleman

Michael L. Dertouzos



Massachusetts Institute of Technology
Cambridge, Massachusetts

limitations on what can be accomplished, we shall see that it appears likely that there exist encryption functions which permit encrypted data to be operated on without preliminary decryption of the operands, for many sets of interesting operations. These special encryption functions we call "privacy homomorphisms"; they form an interesting subset of arbitrary encryption schemes

Privacy Homomorphisms, 1978

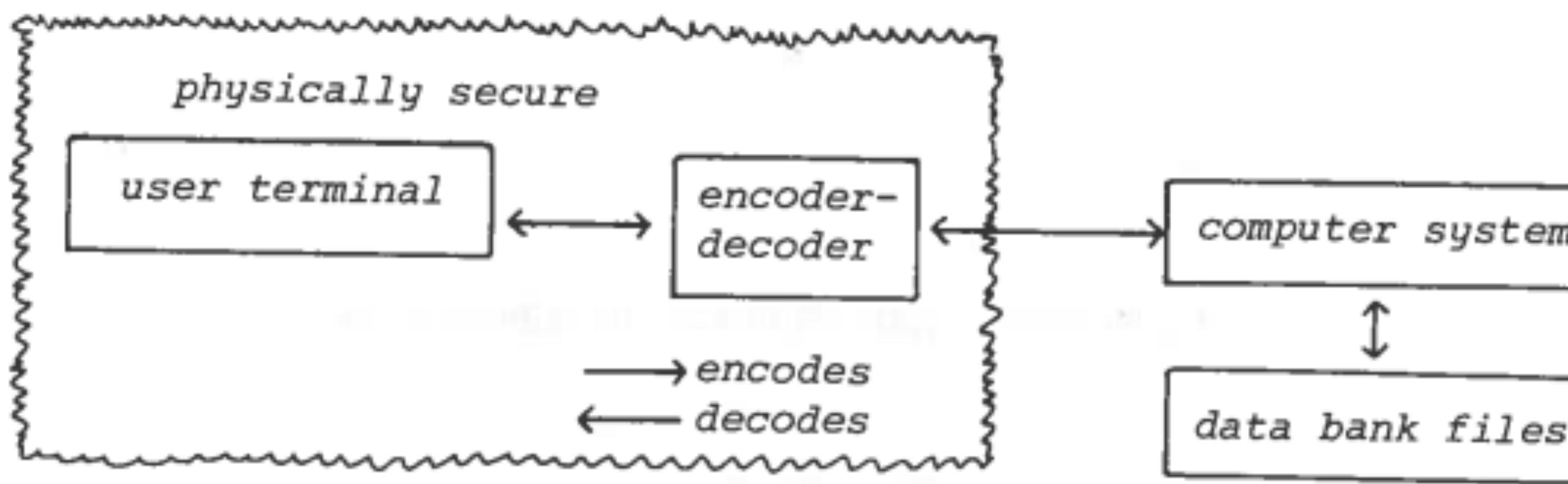


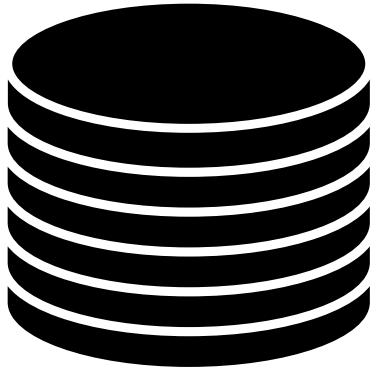
Figure 1

Fully Homomorphic Encryption

Fully Homomorphic Encryption

How to securely compute $f(m_1, m_2, \dots)$?

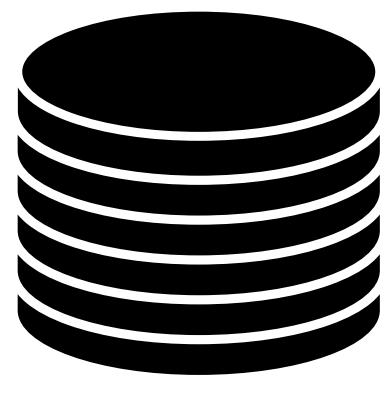
Message space:
 (m_1, m_2, \dots)



Fully Homomorphic Encryption

How to securely compute $f(m_1, m_2, \dots)$?

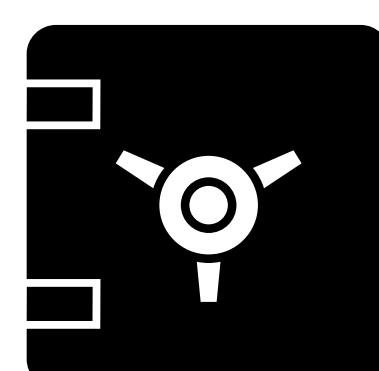
Message space:
 (m_1, m_2, \dots)



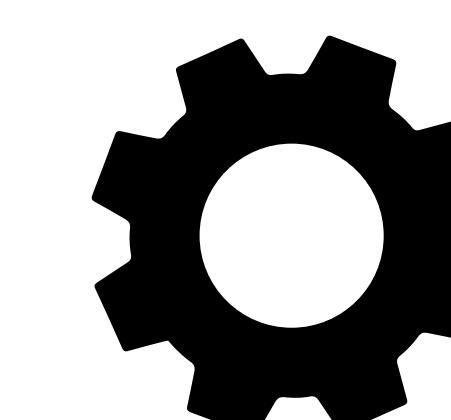
$c_1 = \text{Encrypt}(pk, m_1)$



Ciphertext space:
 (c_1, c_2, \dots)



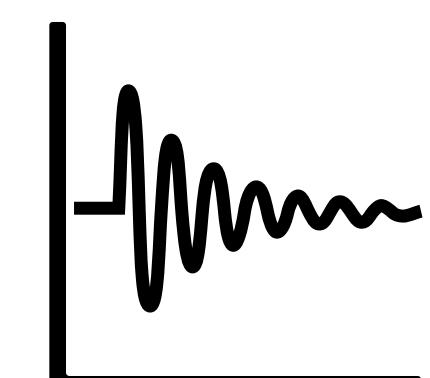
Encrypted evaluation
 $F = f'(c_1, c_2, \dots)$



$\text{Decrypt}(sk, F)$



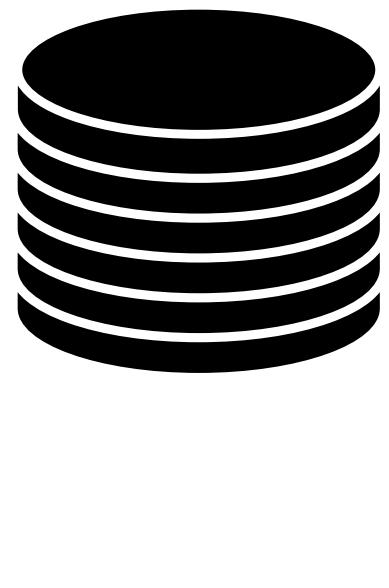
$f(m_1, m_2, \dots)$



Fully Homomorphic Encryption

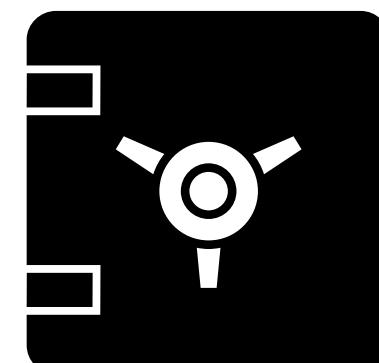
How to securely compute $f(m_1, m_2, \dots)$?

Message space:
 (m_1, m_2, \dots)

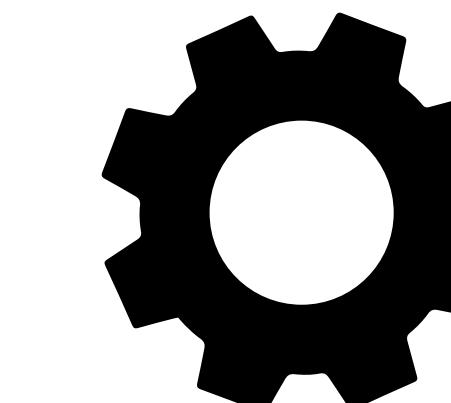


$c_1 = \text{Encrypt}(pk, m_1)$
 $pk = \text{public key}$

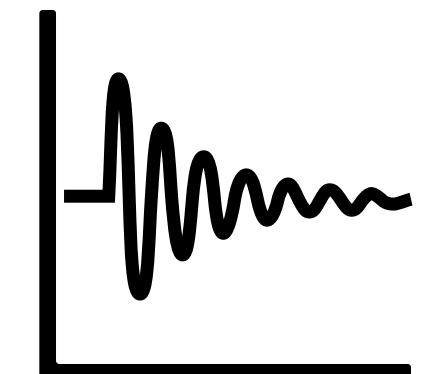
Ciphertext space:
 (c_1, c_2, \dots)



Encrypted evaluation
 $F = f'(c_1, c_2, \dots)$



$f(m_1, m_2, \dots)$
 $sk = \text{secret key}$



An encryption scheme is called Fully Homomorphic,
if it can handle any desired function f !

Fully Homomorphic Encryption

**Craig Gentry built the first FHE scheme,
on his Ph.D. Thesis in 2009**

Message space: {0,1}
Permitted functions:
additions and multiplications



Semantically secure encryption

- Encrypt the same message 2+ times must generate different ciphertexts
- Encryption must be random

Semantically secure encryption

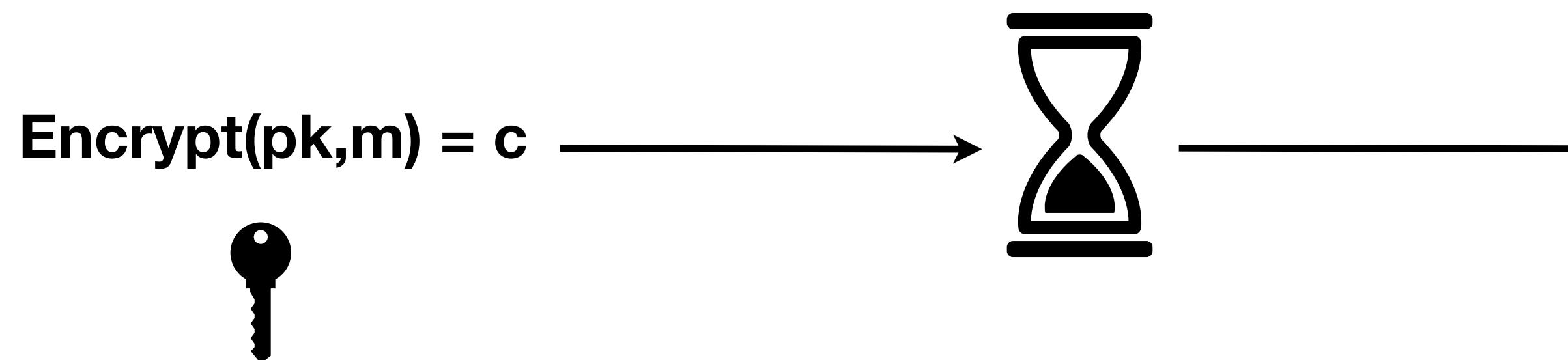
- Encrypt the same message 2+ times must generate different ciphertexts
- Encryption must be random

Encrypt(pk,m) = c



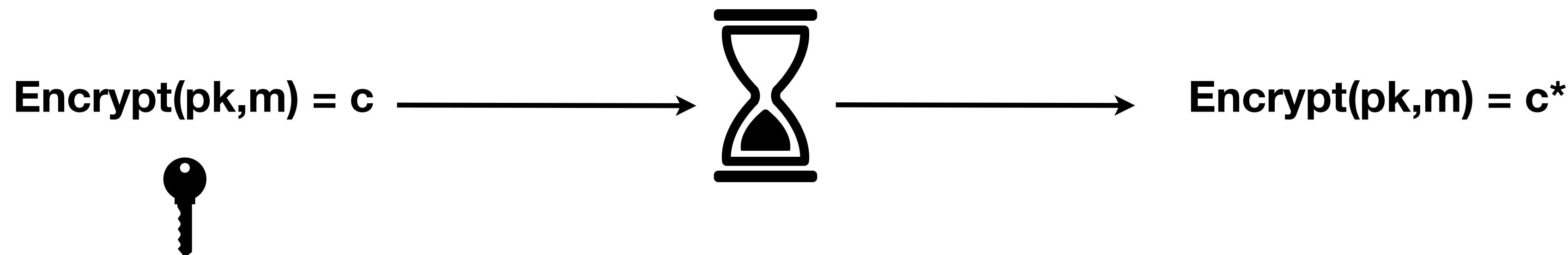
Semantically secure encryption

- Encrypt the same message 2+ times must generate different ciphertexts
- Encryption must be random



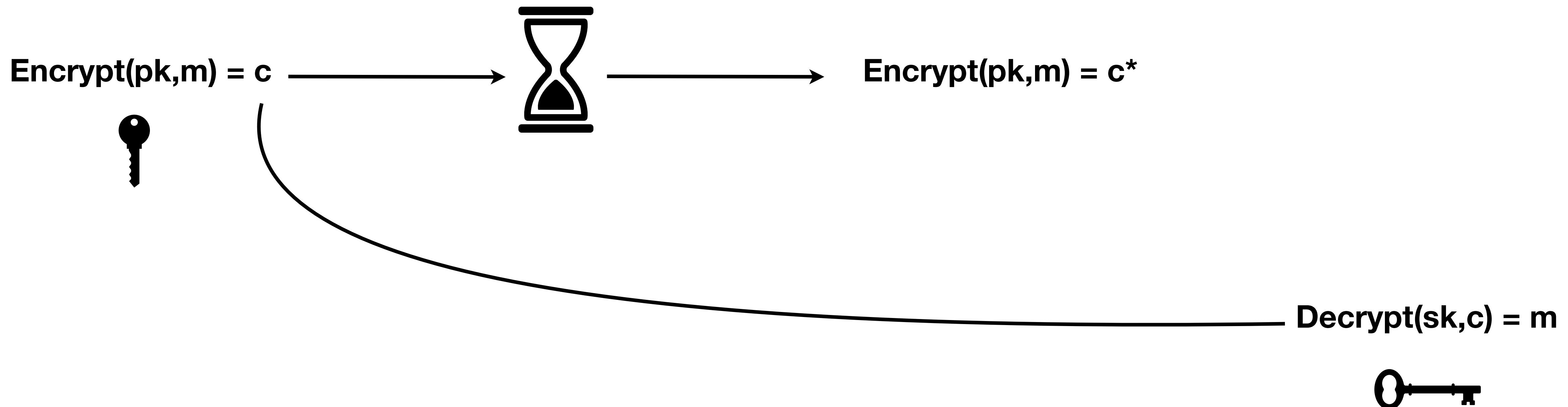
Semantically secure encryption

- Encrypt the same message 2+ times must generate different ciphertexts
- Encryption must be random



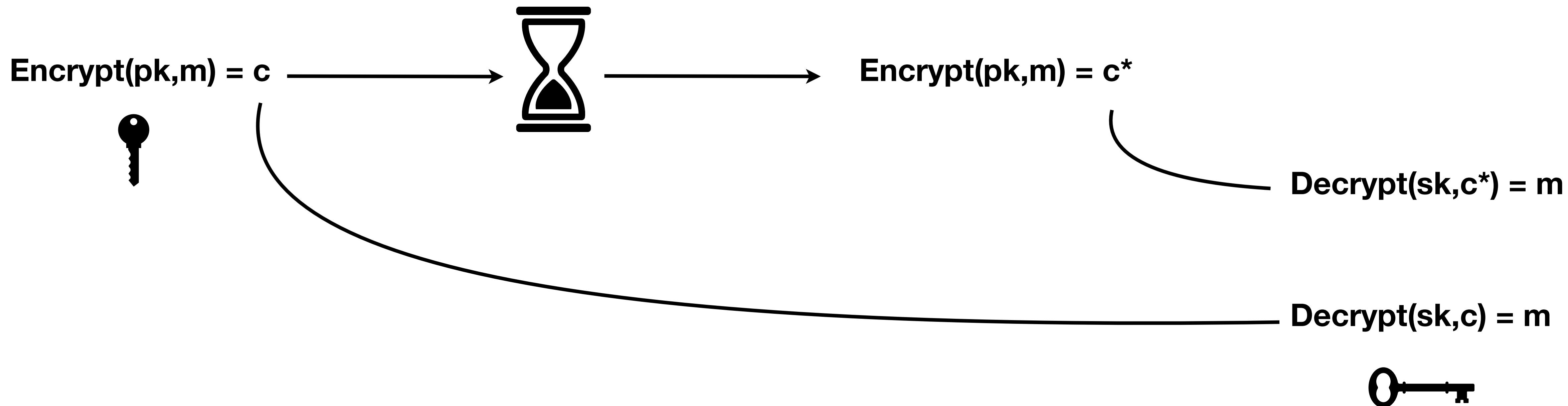
Semantically secure encryption

- Encrypt the same message 2+ times must generate different ciphertexts
- Encryption must be random



Semantically secure encryption

- Encrypt the same message 2+ times must generate different ciphertexts
- Encryption must be random



Somewhat Homomorphic Encryption (SHE)

Somewhat Homomorphic Encryption (SHE)

A SHE scheme can homomorphically evaluate functions f with a limited depth

Somewhat Homomorphic Encryption (SHE)

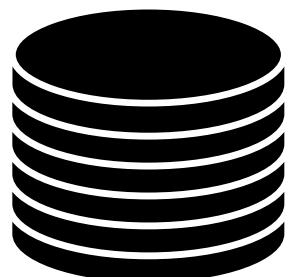
A SHE scheme can homomorphically evaluate functions f with a limited depth

A simple SHE:

Somewhat Homomorphic Encryption (SHE)

A SHE scheme can homomorphically evaluate functions f with a limited depth

A simple SHE:

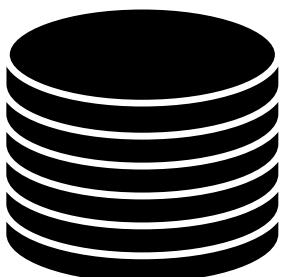


Message space:
 $\{0,1\}$

Somewhat Homomorphic Encryption (SHE)

A SHE scheme can homomorphically evaluate functions f with a limited depth

A simple SHE:



Message space:
 $\{0,1\}$

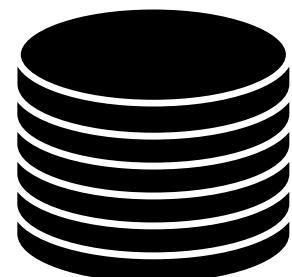


Key:
 p , odd integer

Somewhat Homomorphic Encryption (SHE)

A SHE scheme can homomorphically evaluate functions f with a limited depth

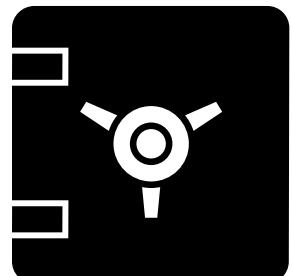
A simple SHE:



Message space:
 $\{0,1\}$



Key:
 p , odd integer

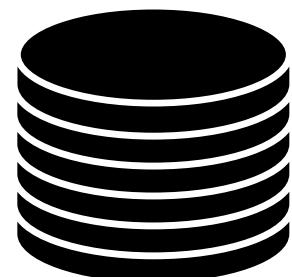


Ciphertext space:
integers

Somewhat Homomorphic Encryption (SHE)

A SHE scheme can homomorphically evaluate functions f with a limited depth

A simple SHE:



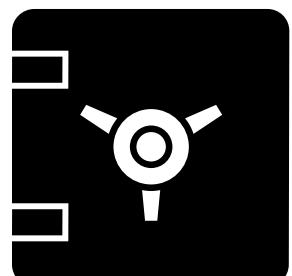
Message space:
 $\{0,1\}$



Encryption function:
 $c = pq + 2r + m$



Key:
 p , odd integer

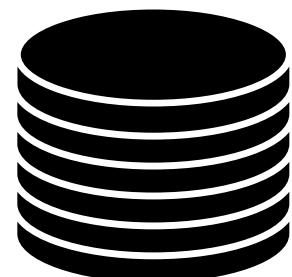


Ciphertext space:
integers

Somewhat Homomorphic Encryption (SHE)

A SHE scheme can homomorphically evaluate functions f with a limited depth

A simple SHE:



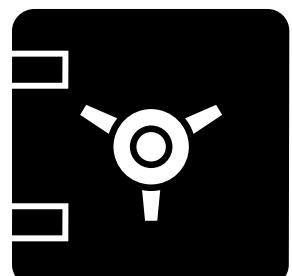
Message space:
 $\{0,1\}$



Encryption function:
 $c = pq + 2r + m$
(q,r random)



Key:
 p , odd integer

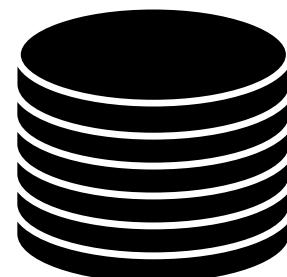


Ciphertext space:
integers

Somewhat Homomorphic Encryption (SHE)

A SHE scheme can homomorphically evaluate functions f with a limited depth

A simple SHE:



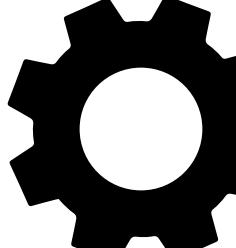
Message space:
 $\{0,1\}$



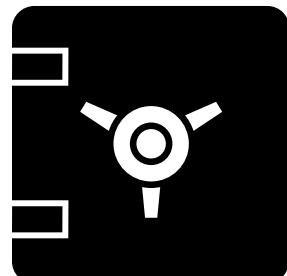
Encryption function:
 $c = pq + 2r + m$
(q,r random)



Key:
 p , odd integer



Decryption function:
 $(c \bmod p) \bmod 2$

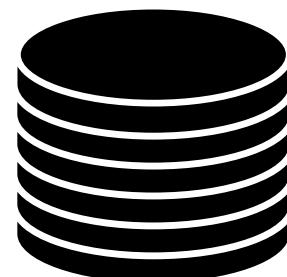


Ciphertext space:
integers

Somewhat Homomorphic Encryption (SHE)

A SHE scheme can homomorphically evaluate functions f with a limited depth

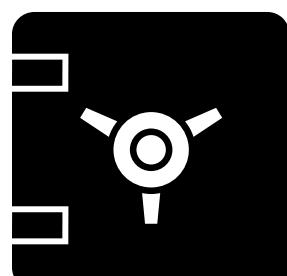
A simple SHE:



Message space:
 $\{0,1\}$



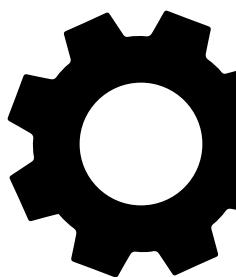
Key:
 p , odd integer



Ciphertext space:
integers



Encryption function:
 $c = pq + 2r + m$
(q,r random)



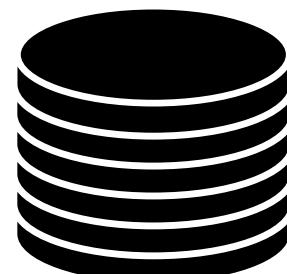
Decryption function:
 $(c \bmod p) \bmod 2$

If $2r < p$, the decryption is correct:

Somewhat Homomorphic Encryption (SHE)

A SHE scheme can homomorphically evaluate functions f with a limited depth

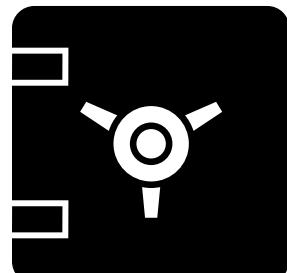
A simple SHE:



Message space:
 $\{0,1\}$



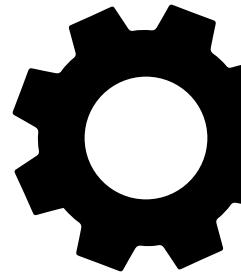
Key:
 p , odd integer



Ciphertext space:
integers



Encryption function:
 $c = pq + 2r + m$
(q,r random)



Decryption function:
 $(c \bmod p) \bmod 2$

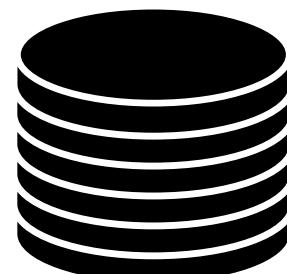
If $2r < p$, the decryption is correct:

$$\begin{aligned} & (c \bmod p) \bmod 2 \\ &= ((pq + 2r + m) \bmod p) \bmod 2 \\ &= ((2r + m) \bmod p) \bmod 2 \\ &= (2r + m) \bmod 2 \\ &= m \end{aligned}$$

Somewhat Homomorphic Encryption (SHE)

A SHE scheme can homomorphically evaluate functions f with a limited depth

A simple SHE:



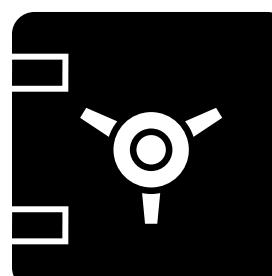
Message space:
 $\{0,1\}$



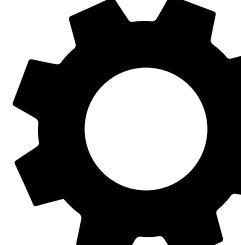
Encryption function:
 $c = pq + 2r + m$
(q, r random)



Key:
 p , odd integer



Ciphertext space:
integers



Decryption function:
 $(c \bmod p) \bmod 2$

If $2r < p$, the decryption is correct:

$$\begin{aligned} & (c \bmod p) \bmod 2 \\ &= ((pq + 2r + m) \bmod p) \bmod 2 \\ &= ((2r + m) \bmod p) \bmod 2 \\ &= (2r + m) \bmod 2 \\ &= m \end{aligned}$$

If we add/multiply two ciphertexts c_1 and c_2 , the noise will be added/multiplied too

Bootstrapping

**Let's suppose we have a ciphertext c
of a message m , and want to reduce its noise**

c

Bootstrapping

Let's suppose we have a ciphertext c of a message m , and want to reduce its noise

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

Bootstrapping

Let's suppose we have a ciphertext c of a message m , and want to reduce its noise

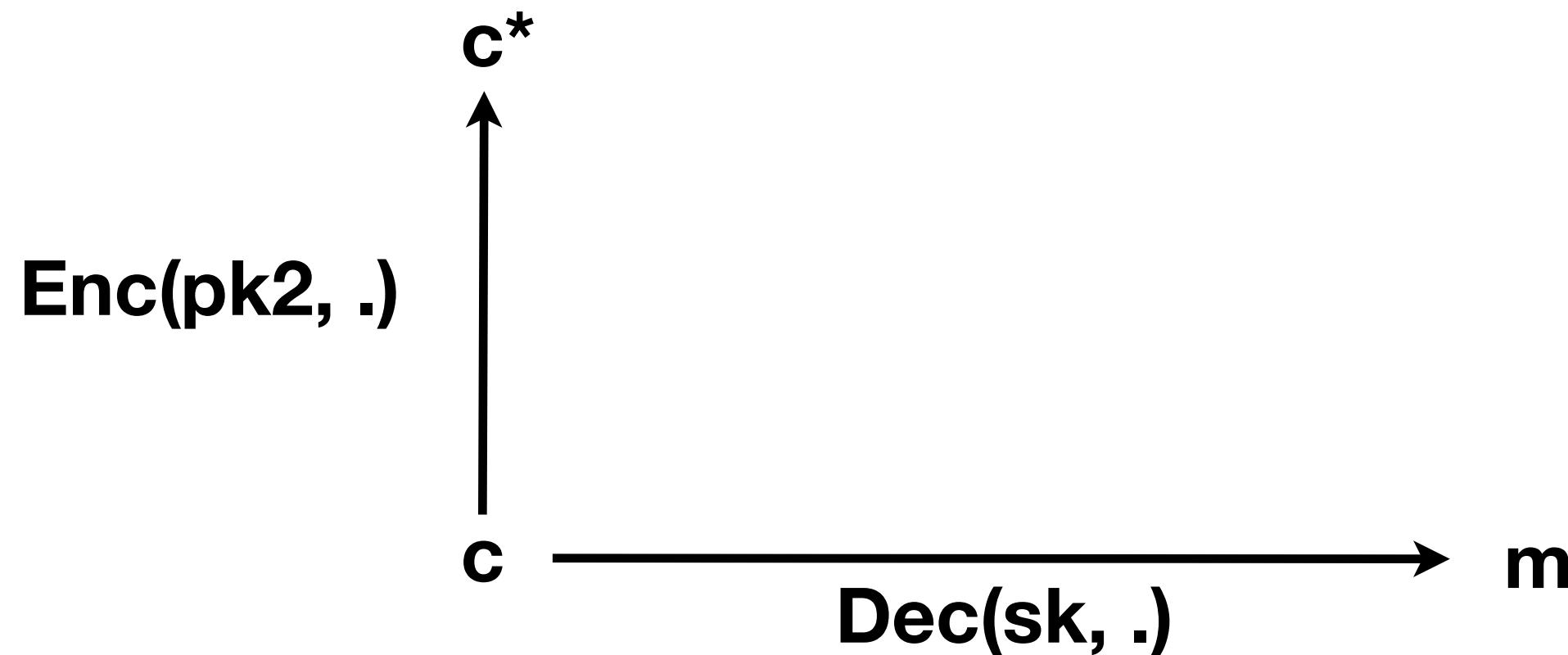
**The Decryption function completely removes any noise !!
But reveals the message and uses the private key :(**

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

Bootstrapping

Let's suppose we have a ciphertext c of a message m , and want to reduce its noise

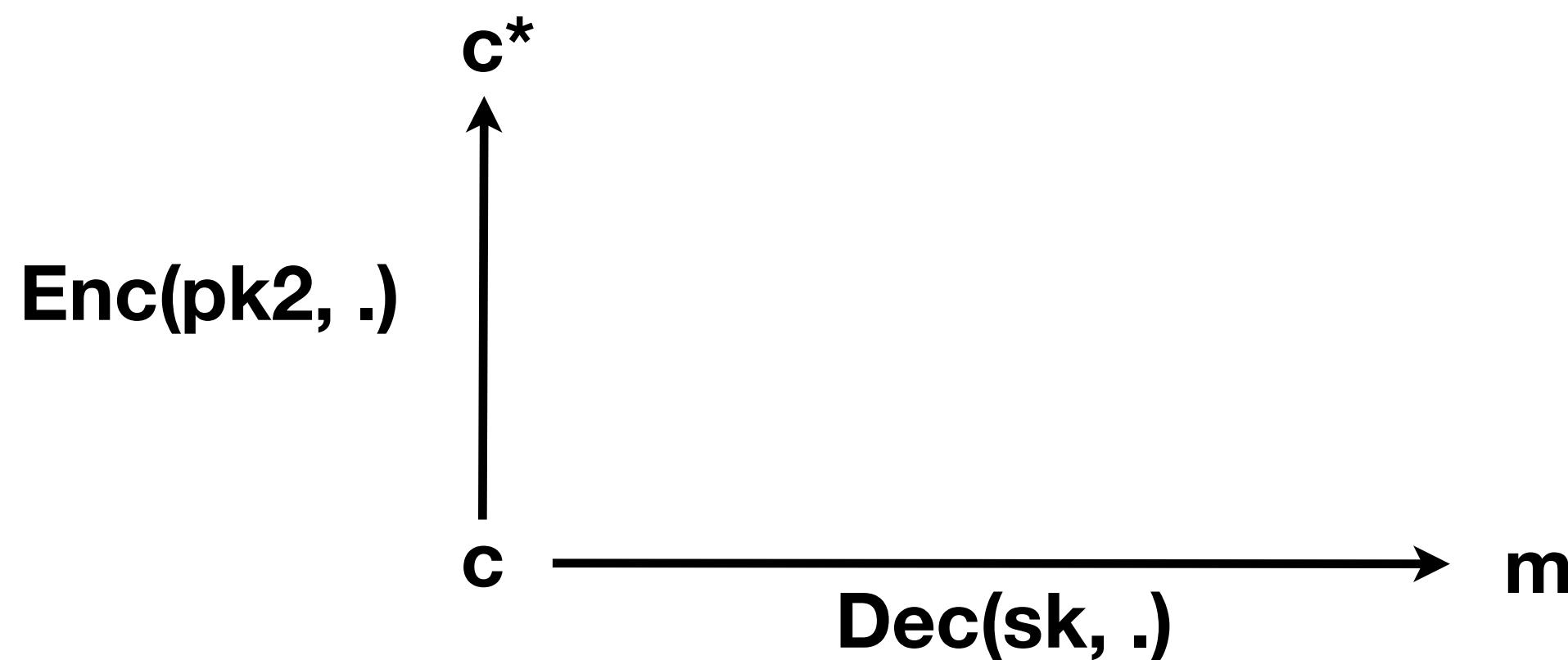
The Decryption function completely removes any noise !!
But reveals the message and uses the private key :(



c^* is a double encryption of m

Bootstrapping

Let's suppose we have a ciphertext c of a message m , and want to reduce its noise



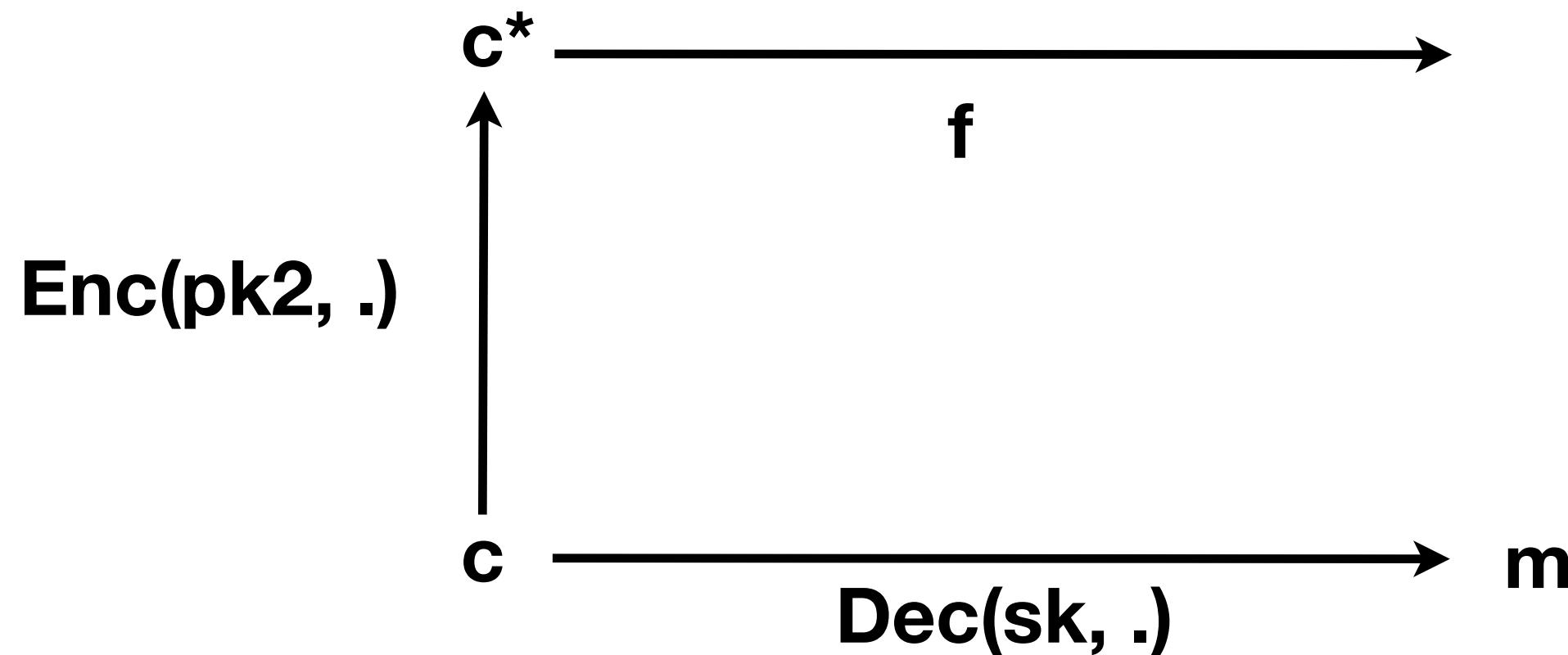
The Decryption function completely removes any noise !!
But reveals the message and uses the private key :(

c^* is a double encryption of m

We can homomorphically apply permitted functions f to c^*

Bootstrapping

Let's suppose we have a ciphertext c of a message m , and want to reduce its noise



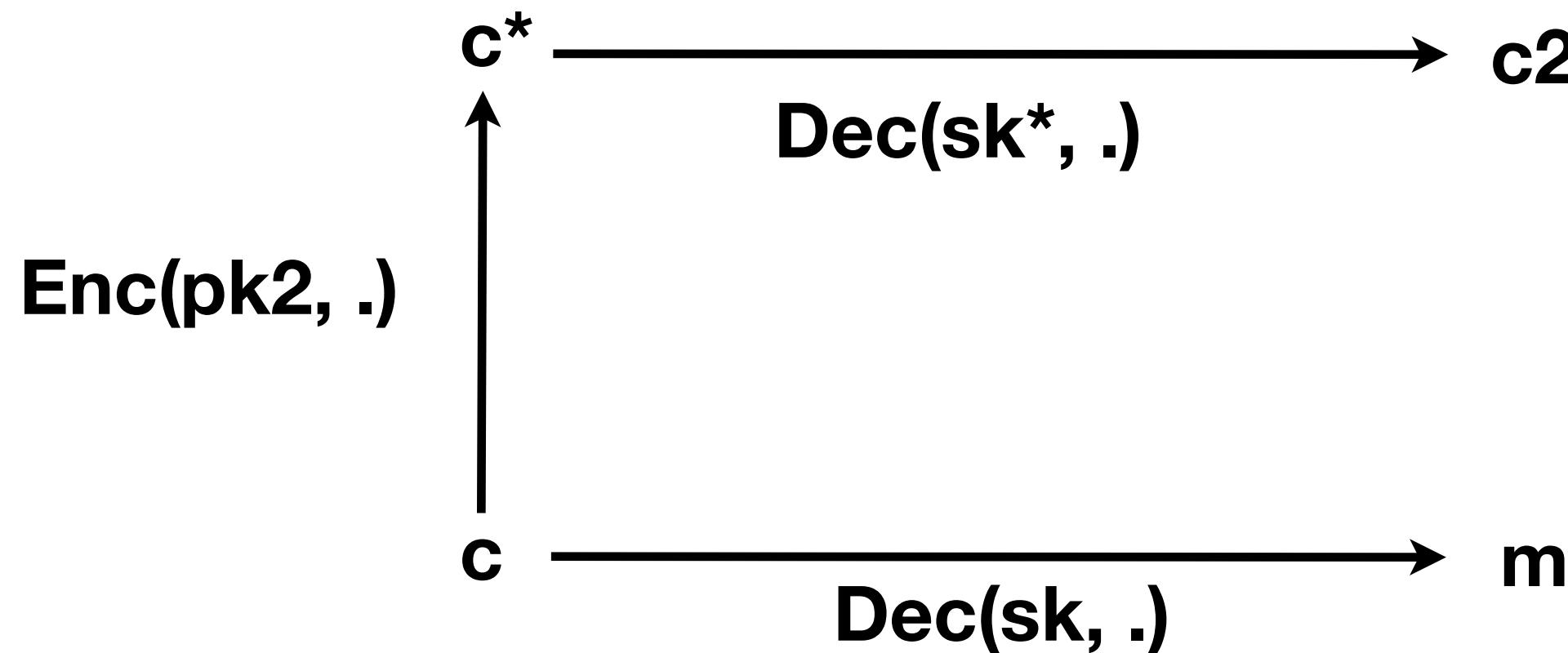
The Decryption function completely removes any noise !!
But reveals the message and uses the private key :(

c^* is a double encryption of m

We can homomorphically apply permitted functions f to c^*

Bootstrapping

Let's suppose we have a ciphertext c of a message m , and want to reduce its noise



The Decryption function completely removes any noise !!
But reveals the message and uses the private key :(

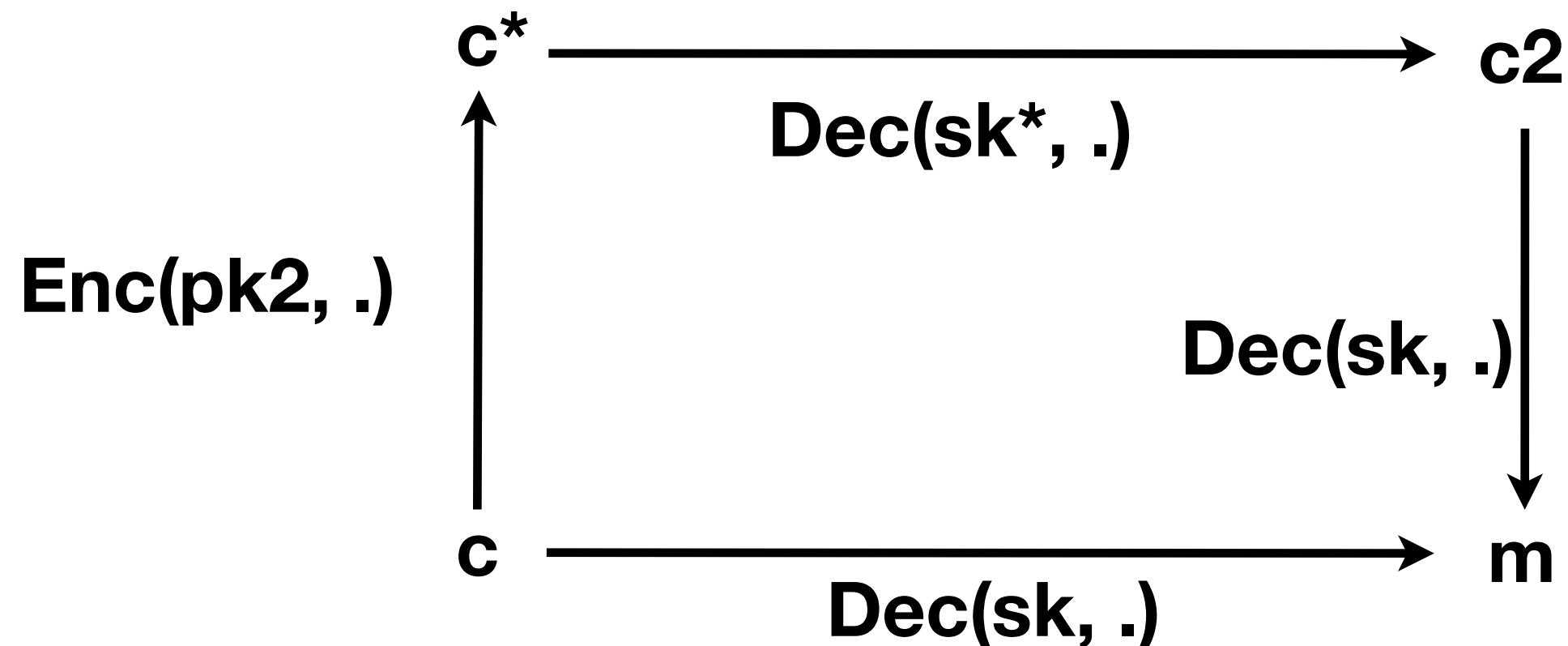
c^* is a double encryption of m

We can homomorphically apply permitted functions f to c^*

So, apply $f(.) = \text{Dec}(\text{sk}^*, .)$

Bootstrapping

Let's suppose we have a ciphertext c of a message m , and want to reduce its noise



The Decryption function completely removes any noise !!
But reveals the message and uses the private key :(

c^* is a double encryption of m

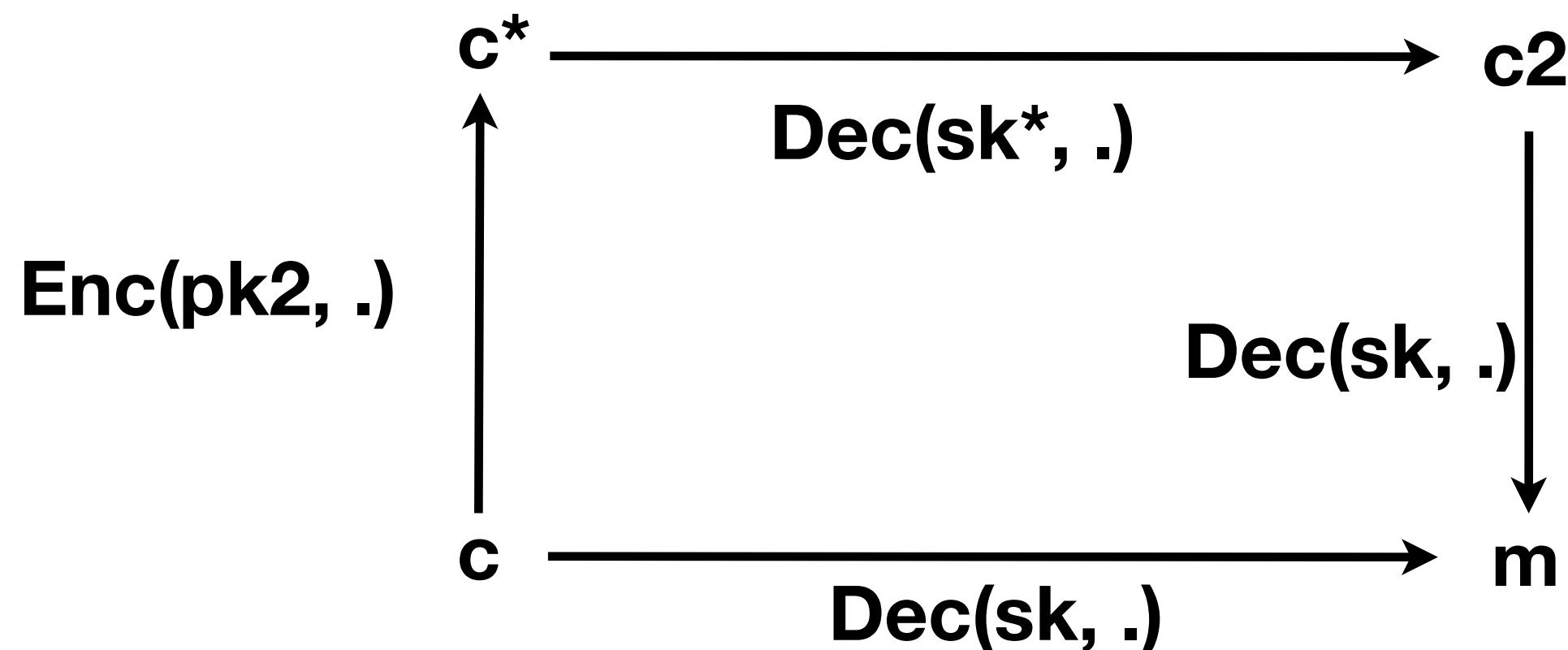
We can homomorphically apply permitted functions f to c^*

So, apply $f(.) = \text{Dec}(\text{sk}^*, .)$

c_2 is a new ciphertext that encrypts m , with less noise!

Bootstrapping

Let's suppose we have a ciphertext c of a message m , and want to reduce its noise



pk_2 = second public key
 $\text{sk}^* = \text{Enc}(\text{pk}_2, \text{sk})$

The Decryption function completely removes any noise !!
But reveals the message and uses the private key :(

c^* is a double encryption of m

We can homomorphically apply permitted functions f to c^*

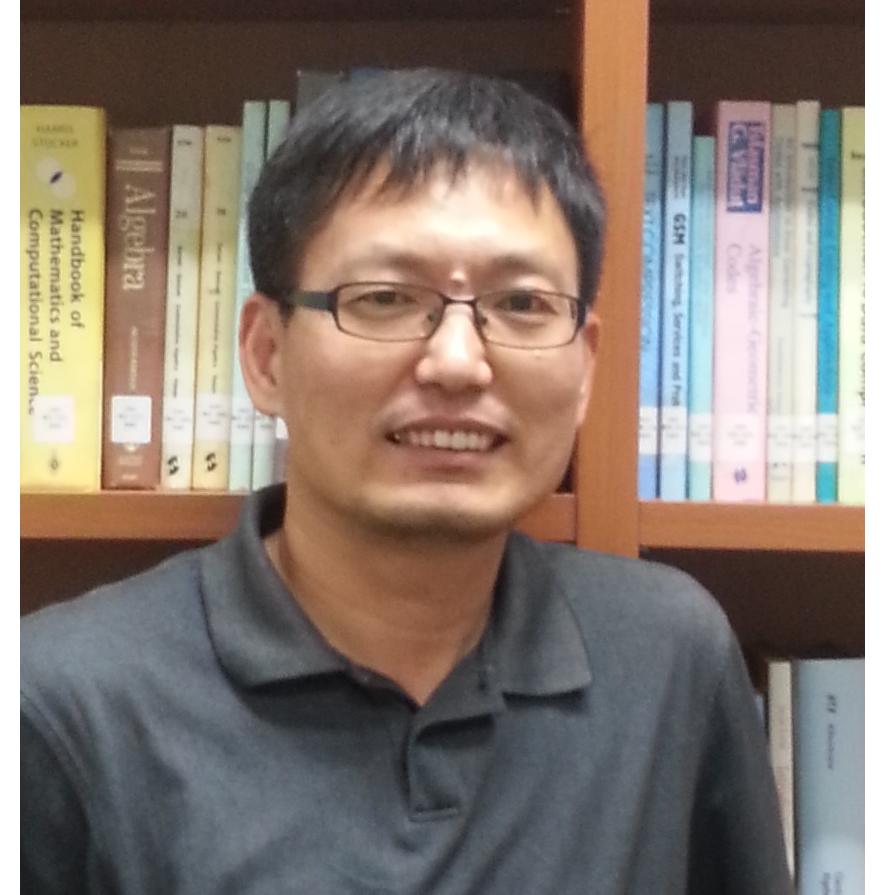
So, apply $f(\cdot) = \text{Dec}(\text{sk}^*, \cdot)$

c_2 is a new ciphertext that encrypts m , with less noise!

A SHE scheme is called bootstrappable,
if the Decryption function is a permitted function

CKKS scheme

Cheon, Kim, Kim and Song (CKKS, 2017) - Homomorphic Encryption for arithmetic of approximate numbers



Jun Hee Cheon
Seoul National University



Andrey Kim
Samsung Advanced
Institute of Technology



Miran Kim
Hanyang University



Yongsoo Song
Seoul National University

CKKS scheme

Plaintext Space: Polynomials

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

Ciphertext Space: Polynomials tuples

$$\mathcal{R}_q \times \mathcal{R}_q.$$

CKKS scheme

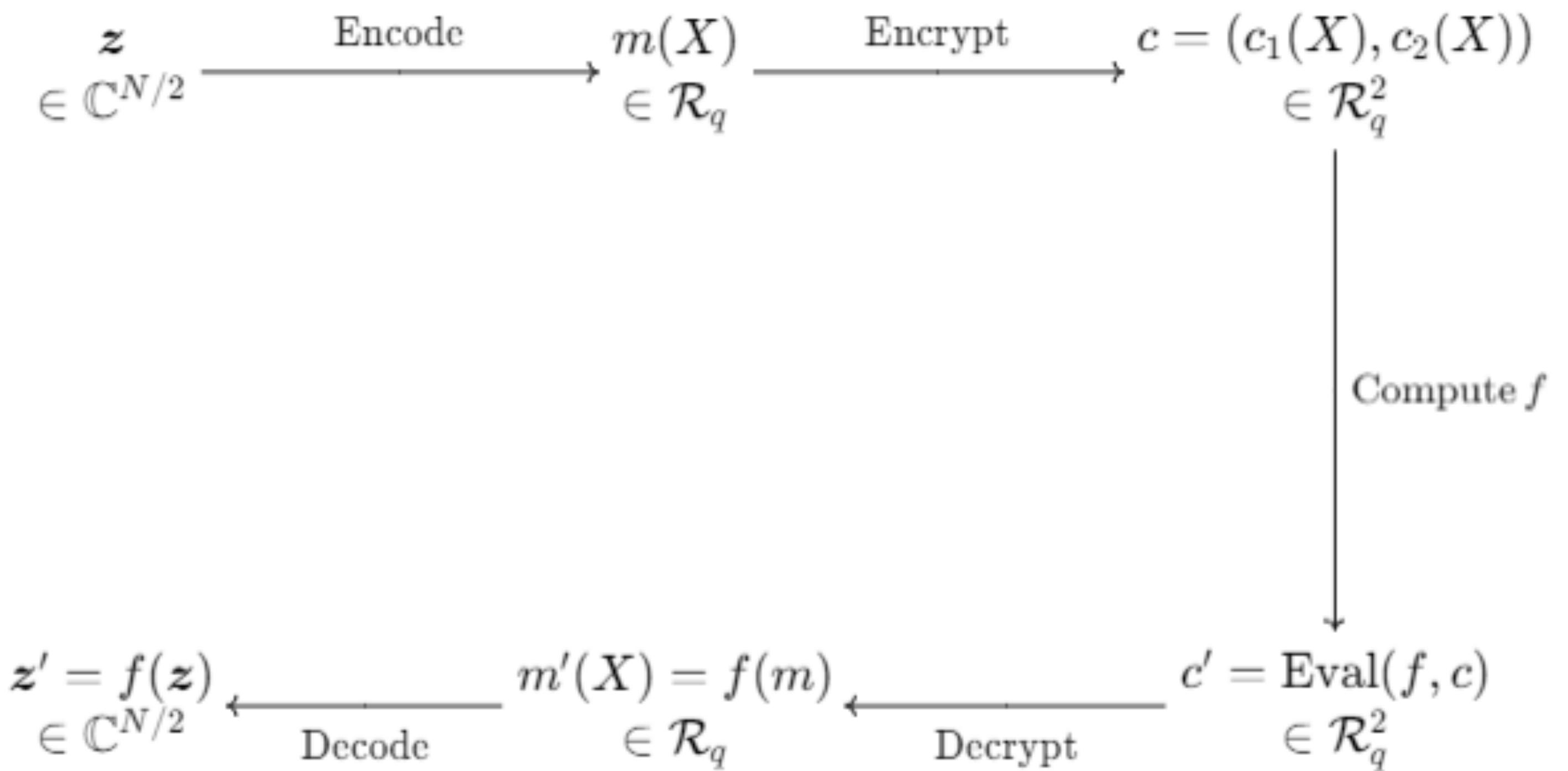
Complex Numbers are supported through encoding.

Plaintext Space: Polynomials

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

Ciphertext Space: Polynomials tuples

$$\mathcal{R}_q \times \mathcal{R}_q.$$



CKKS scheme

Plaintext Space: Polynomials

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

Ciphertext Space: Polynomials tuples

$$\mathcal{R}_q \times \mathcal{R}_q.$$

Secret key: (1,s)

s - random polynomial from \mathcal{R}_q

Public Key:

| $\text{pk} = (b, a)$, with $b = [-a \cdot s + e]_{q_L}$

$$a \xleftarrow{\mathcal{U}} \mathcal{R}_{q_L} \text{ and } e \xleftarrow{\mathcal{D}G(\sigma^2)} \mathcal{R}.$$

Encryption:

$$\text{Enc}(\text{pk}, m) = [v \cdot \text{pk} + (m + e_0, e_1)]_{q_L}$$

Decryption:

$$\text{Dec}(\text{sk}, c) = [\langle c, \text{sk} \rangle]_{q_\ell} = [c_1 + c_2 \cdot s]_{q_\ell}, \text{ for } c = (c_1, c_2)$$

Allowed operations:

Additions, Multiplications, Rotations

Security - Learning with Errors (LWE)

$$\mathbf{a}_i \xleftarrow{\mathcal{U}} \mathbb{Z}_q^n$$

$$b_i = \langle \mathbf{a}_i, \mathbf{s} \rangle + e_i \pmod{q}$$

$$e_i \xleftarrow{\chi} \mathbb{Z}_q$$

LWE Problem: Guess \mathbf{s} , given (\mathbf{a}_i, b_i)

If $e_i = 0$, we could easily find \mathbf{s} , solving $A\mathbf{s} = \mathbf{b}$

With $e_i > 0$, this is a NP-hard problem

Ring Learning with Errors (RLWE)

$$a_i \xleftarrow{\mathcal{U}} \mathcal{R}_q$$

$$b_i = a_i \cdot s + e_i \pmod{q}$$

$$e_i \xleftarrow{\chi} \mathcal{R}$$

RLWE Problem: Guess s , given (a_i, b_i)

CKKS public key security is based on RLWE:

| $\text{pk} = (b, a)$, with $b = [-a \cdot s + e]_{q_L}$

Logistic Regression

$$\mathbf{X} \in \mathbb{R}^{n \times d}$$

$$\mathbf{y} \in \{0, 1\}^n$$

Logistic Regression

$$\mathbf{X} \in \mathbb{R}^{n \times d}$$

$$\mathbf{y} \in \{0, 1\}^n$$

$$\ln \left(\frac{p(\mathbf{x})}{1 - p(\mathbf{x})} \right) = \mathbf{w}^T \mathbf{x}'$$

$$\mathbf{w} = (w_0, w_1, \dots, w_d)^T \in \mathbb{R}^{d+1}$$

$$\mathbf{x}' = (1; \mathbf{x})$$

Logistic Regression

$$\mathbf{X} \in \mathbb{R}^{n \times d}$$

$$\mathbf{y} \in \{0, 1\}^n$$

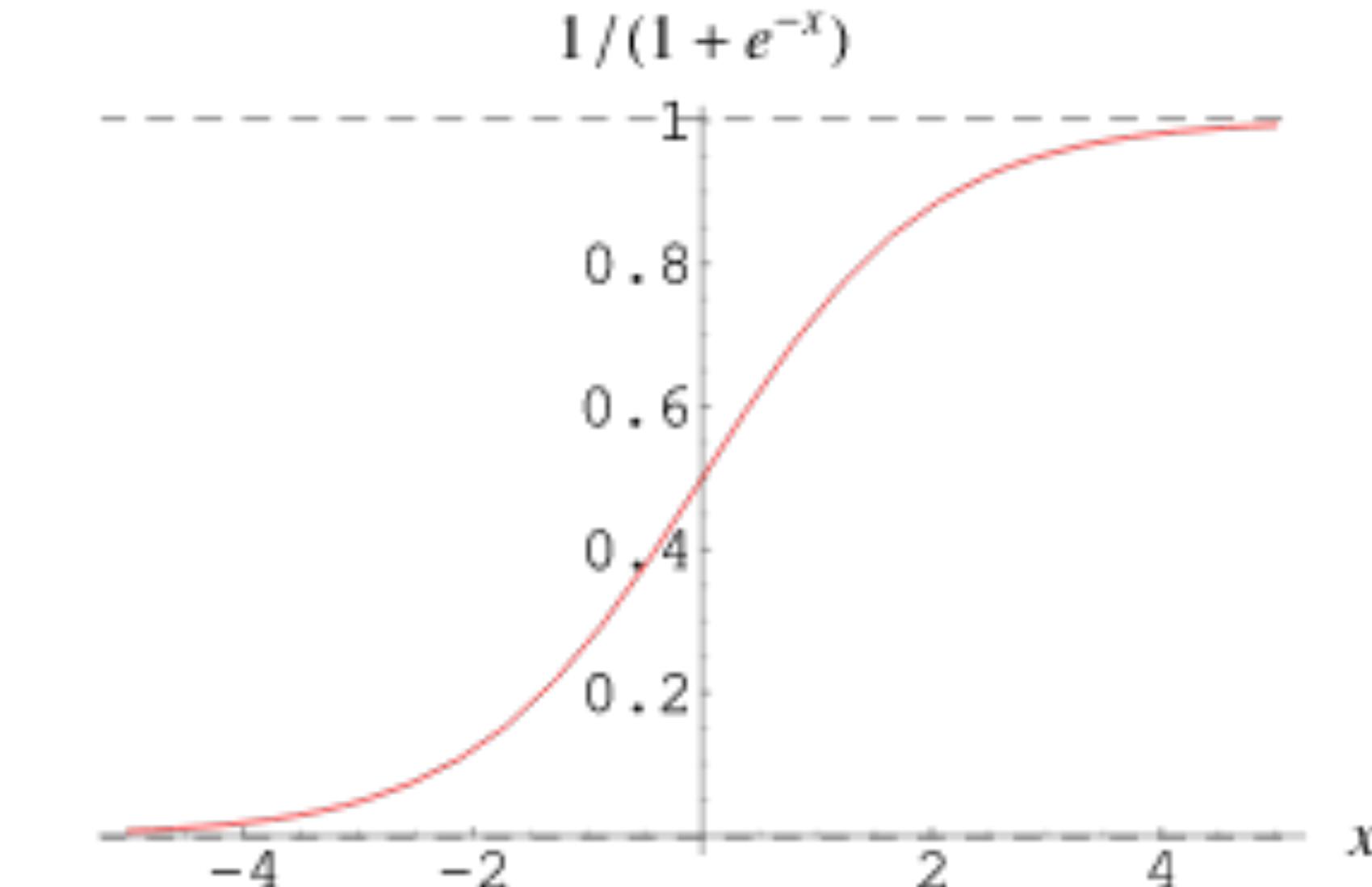
$$\ln \left(\frac{p(\mathbf{x})}{1 - p(\mathbf{x})} \right) = \mathbf{w}^T \mathbf{x}'$$

$$\mathbf{w} = (w_0, w_1, \dots, w_d)^T \in \mathbb{R}^{d+1}$$

$$\mathbf{x}' = (1; \mathbf{x})$$

$$p(\mathbf{x}) = \sigma(\mathbf{w}^T \mathbf{x}')$$

$$\sigma(x) = 1/(1 + e^{-x})$$



Logistic Regression - Training

Training algorithm: Find w that maximizes the likelihood ...

$$\mathcal{L}(w) = \prod_{i=1}^n \sigma(w^T x'_i)^{y_i} (1 - \sigma(w^T x'_i))^{1-y_i}$$

Logistic Regression - Training

Training algorithm: Find w that maximizes the likelihood ...

$$\mathcal{L}(w) = \prod_{i=1}^n \sigma(w^T x'_i)^{y_i} (1 - \sigma(w^T x'_i))^{1-y_i}$$

... or minimizes the negative log-likelihood

$$\ell(w) = -\frac{1}{n} \ln(\mathcal{L}(w))$$

Logistic Regression - Training

Training algorithm: Find w that maximizes the likelihood ...

$$\mathcal{L}(w) = \prod_{i=1}^n \sigma(w^T x'_i)^{y_i} (1 - \sigma(w^T x'_i))^{1-y_i}$$

... or minimizes the negative log-likelihood

$$\ell(w) = -\frac{1}{n} \ln(\mathcal{L}(w))$$

Gradient Descent Method:

$$w_{i+1} = w_i - \alpha \nabla \ell(w_i),$$

$$\text{where } \nabla \ell(w_i) = -\frac{1}{n} \sum_{j=1}^n \sigma(-w_i^T z_j) z_j$$

**Two inner products involved!
exponential function in sigmoid!**

Logistic Regression - Private Training

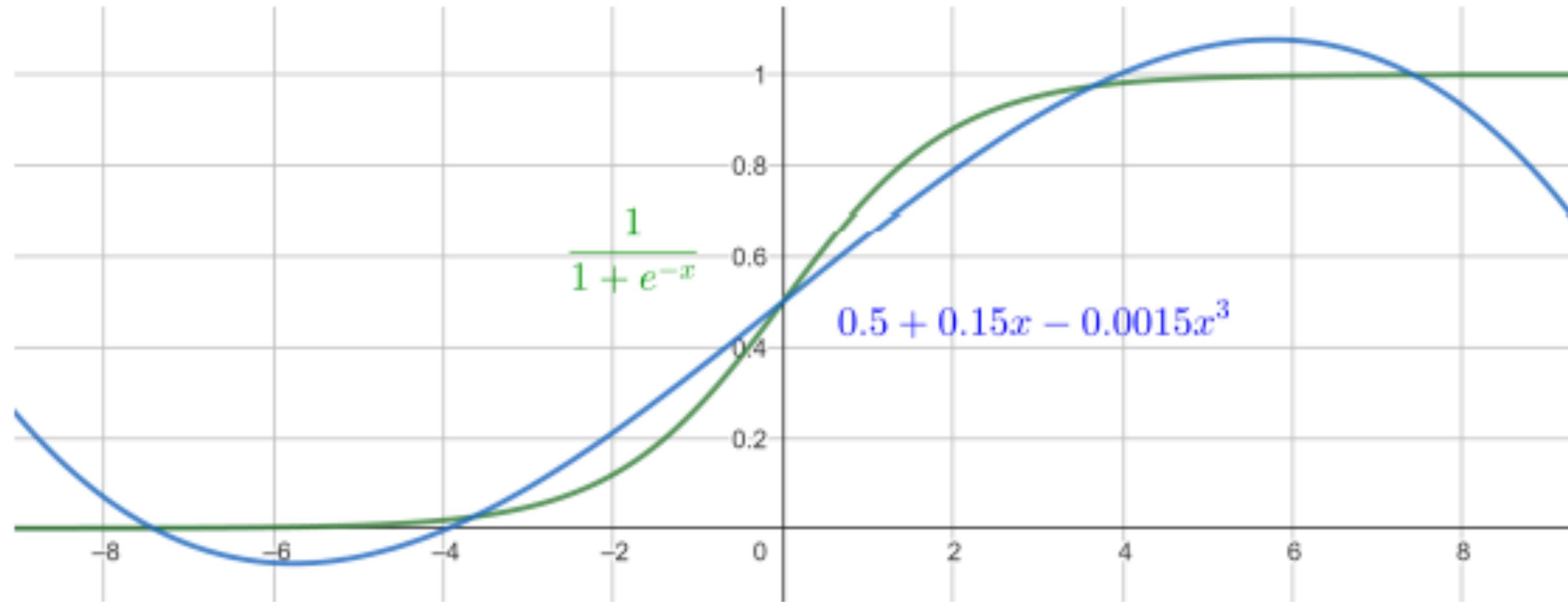


Figure 6 – Sigmoid polynomial approximation

Logistic Regression - Private Training

Calculating Inner products

Naive approach: encrypt each entry as a ciphertext

$$\vec{a} = \begin{array}{|c|c|c|c|} \hline 1 & 2 & 3 & 4 \\ \hline \end{array}$$

$$\vec{b} = \begin{array}{|c|c|c|c|} \hline 5 & 6 & 7 & 8 \\ \hline \end{array}$$

$$a_1.b_1 + a_2.b_2 + a_3.b_3 + a_4.b_4 = 70$$

4 multiplications, 3 additions required

Logistic Regression - Private Training

Calculating Inner products

SIMD approach: encrypt each vector as a ciphertext

$$\vec{a} = \begin{array}{|c|c|c|c|} \hline 1 & 2 & 3 & 4 \\ \hline \end{array}$$

$$\vec{b} = \begin{array}{|c|c|c|c|} \hline 5 & 6 & 7 & 8 \\ \hline \end{array}$$

Logistic Regression - Private Training

1st Step: Do 1 SIMD multiplication

$$\vec{a} = \begin{bmatrix} 1 & 2 & 3 & 4 \end{bmatrix}$$

$$\vec{b} = \begin{bmatrix} 5 & 6 & 7 & 8 \end{bmatrix}$$

$$c = a.b$$

$$\vec{c} = \begin{bmatrix} 5 & 12 & 21 & 32 \end{bmatrix}$$

Logistic Regression - Private Training

2nd Step: $c = c + \text{left_rotate}(c, 1)$

5	12	21	32
---	----	----	----

+

12	21	32	5
----	----	----	---

=

17	33	53	37
----	----	----	----

Logistic Regression - Private Training

3nd Step: $c = c + \text{left_rotate}(c, 2)$

$$\begin{array}{cccc} 17 & 33 & 53 & 37 \\ + \\ 53 & 37 & 17 & 33 \\ = \\ 70 & 70 & 70 & 70 \end{array}$$

Logistic Regression - Private Training

3nd Step: $c = c + \text{left_rotate}(c, 2)$

$$\begin{array}{cccc} 17 & 33 & 53 & 37 \\ + \\ 53 & 37 & 17 & 33 \\ = \\ 70 & 70 & 70 & 70 \end{array}$$

Total cost: 1 multiplications + 2 additions

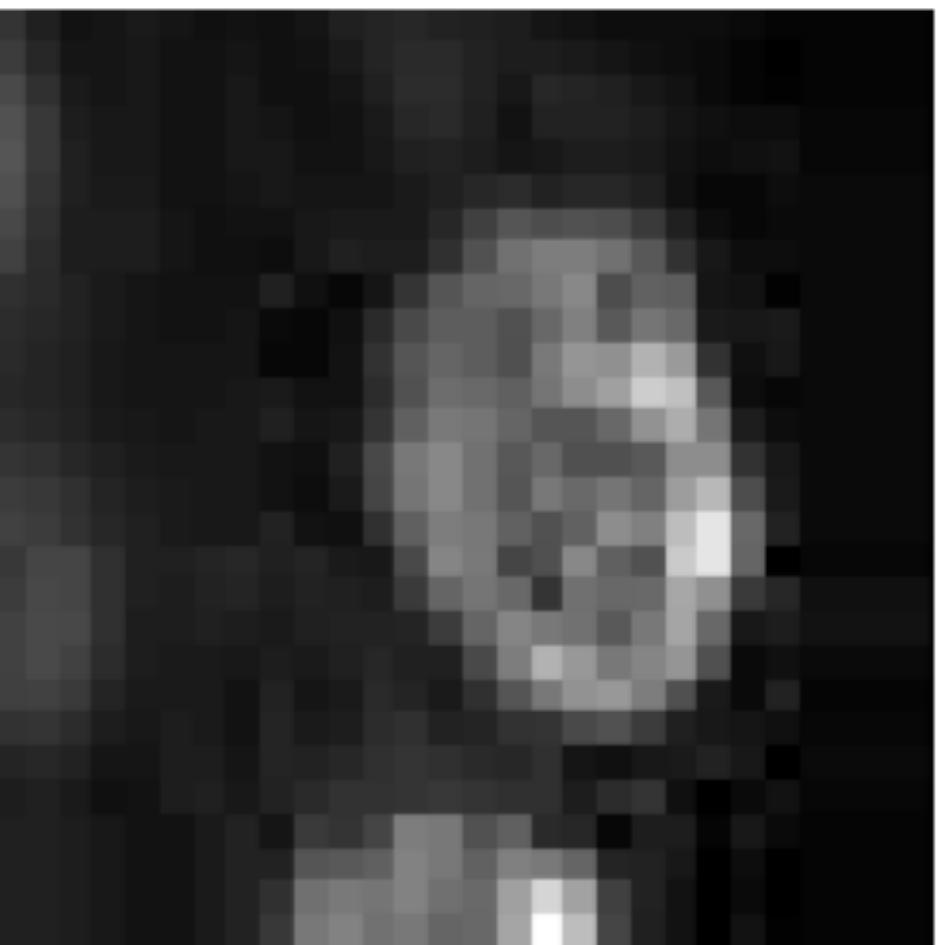
Logistic Regression - Private Training

In general a d-dimensional inner product requires:

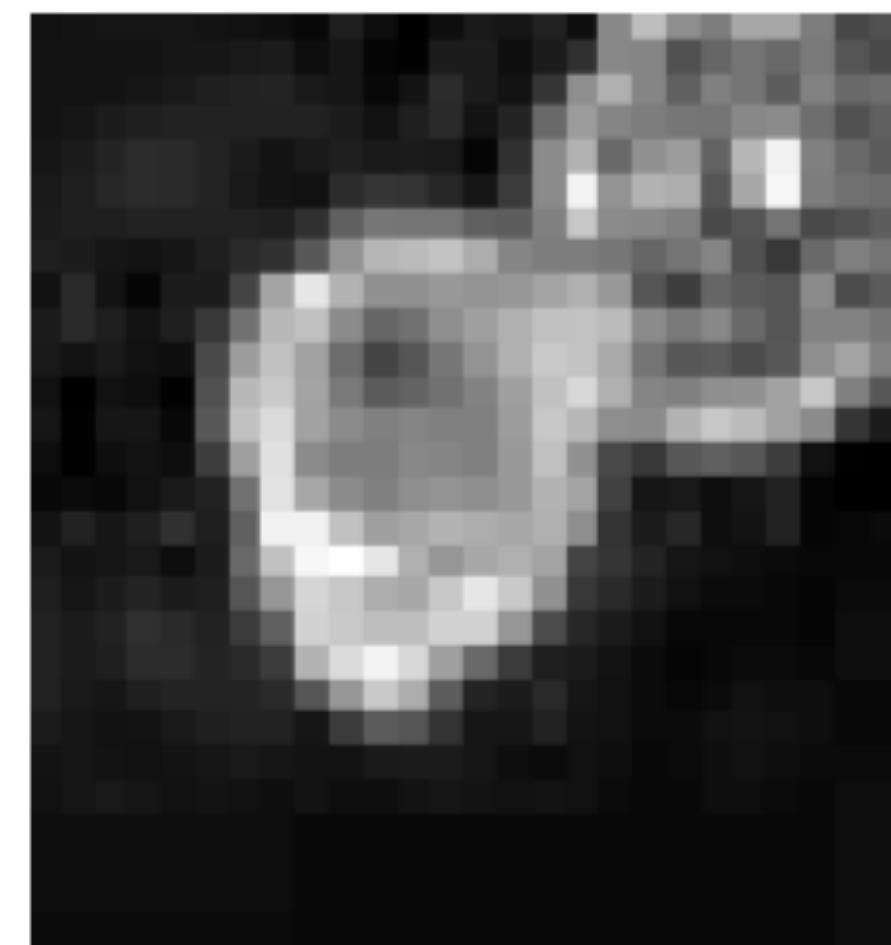
- Naive approach: d multiplications, $d-1$ additions
- SIMD approach: 1 multiplication, $\log(d)$ additions

Simulation Results

- HEAAN + HELR C++ packages;
- Dataset: Subset of the **TissueMNIST** dataset; 92672 rows and 196 columns;
- Machine: 64-bit quad-core Intel Core i5-6200U 2.3GHz CPU, 16GB of RAM;



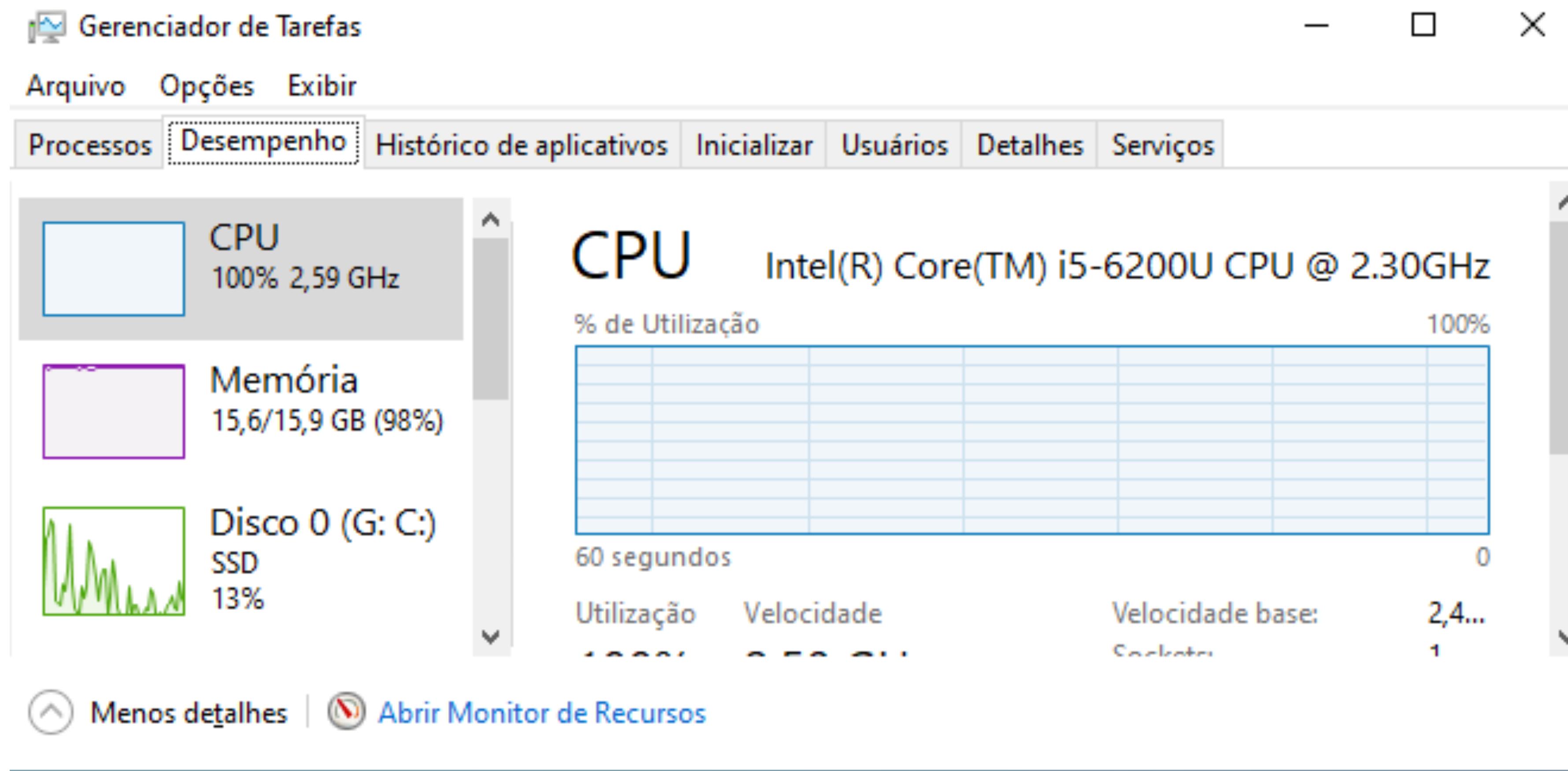
(a) Collecting Duct



(b) Proximal Tubule Segments

Figure 7 – TissueMNIST

Simulation Results



Simulation Results

KeyGen time	8.52 min
Encrypt time	19.17 min
Training time	6.19 hours
Public key size	2.62 GB

	Encrypted	Unencrypted
Accuracy	64.6211%	64.6363%
AUC	81.7039%	81.6996%

References

Rivest, 1978 - ON DATA BANKS AND PRIVACY HOMOMORPHISMS;

Gentry, 2009 - Fully Homomorphic Encryption using Ideal Lattices;

Gentry, 2010 - Fully Homomorphic Encryption over the Integers;

CHEON, Jung Hee; KIM, Andrey, et al., 2017 - Homomorphic Encryption for Arithmetic of Approximate Numbers;

HAN, Kyoohyung et al., 2018 Efficient Logistic Regression on Large Encrypted Data.