Lattice-Based Cryptography in Homomorphic Encryption

Dr. Vincent Mooney, Leyan Pan, Holloway Noa

Outline

Overview of Secrecy in Computation

- Secure computation allows several parties to jointly compute a function over their inputs while keeping those inputs private.

- Homomorphic encryption ensures that the computation operates on encrypted data which cannot be decrypted by the entity performing the computation

- Secret sharing ensures that parts of the data are not known to each of two or more entities (i.e., parties) performing a computation such that possession of anything less than all of the data from all of the parties results in no information

Secure Computation

- Introduced in the 1980s by Andrew Yao[3]
- Several parties have data and what to learn the answer to a question using that data.
- Classic examples
  - Voting: Each party have their own private vote. The goal is to know who has the most vote without knowing anything about other people’s vote.
  - Bidding at an auction
  - The millionaire's problem[3]: two millionaires who want to know which party has more money but without revealing to each other the exact net worth

Secure Multiparty Computation

- ❖ Created by Andrew Yao in 1986
- ❖ An area of Cryptography where you are able to compute data on multiple sources whilst keeping the data private
- ❖ Keeps inputs private
- ❖ A subset of SMC is Secret Sharing: Divide information in to several pieces so that one must know all or some (more than 1) of the pieces to reconstruct the original information.
- ❖ Parties can enter data, split into several pieces and masked with random numbers.
- ❖ Encrypted data sent to multiple servers to ensure privacy.

Homomorphic Encryption

- Concept first introduced by Rivest, Adleman and Dertouzos in 1978

- Allows Computation on ciphertext without revealing the plaintext

- Users can upload data to untrusted server and allows them to perform computations without leaking any privacy

- Two main categories
  - Partially homomorphic encryption can only perform limited kinds of computations but are relatively faster.
  - Fully homomorphic encryption which can perform arbitrary computation but much slower (~1 million times slower than plaintext operations)

Secure Multiparty vs Homomorphic Encryption

- ❖ Secure Multiparty protocols may use Homomorphic Encryption in their underlying implementation

- ❖ Secure Multiparty protocols are already widely accepted for commercial use while Homomorphic Encryption is still only a research field due to its extremely slow speed.

- ❖ If Homomorphic Encryption were to become computationally acceptable in the future, machine learning models and data analysis may be trained over private data without leakage of information.

Terminology

- Encryption – The process of encoding a message or information in such a way that only authorized parties can access it.

- Cryptosystem/Encryption Scheme – A suite of cryptographic algorithms needed to implement a particular security service. Typically, it comprises at least 3 algorithms – one for key generation, one for encryption and one for decryption. (It can be composed of more.)

Terminology

❖ Symmetric Encryption – A kind of encryption that involves only one secret key to cipher and decipher information. The main disadvantage of the symmetric key encryption is that all parties involved have to exchange the key used to encrypt the data before they can decrypt it which can be obtained by an adversary.

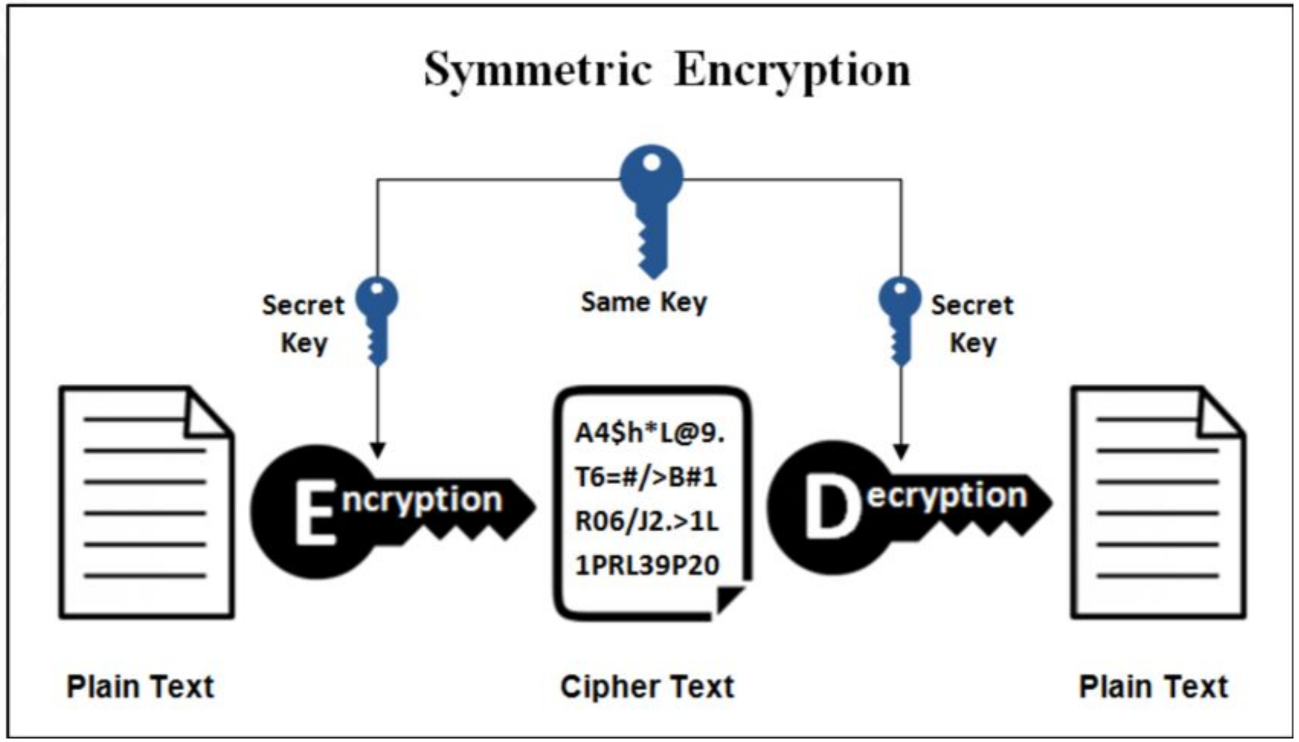


Figure 1. Symmetric Encryption Explained[1]

Terminology

❖ Asymmetric Encryption/Public Key Encryption – A kind of encryption that uses two keys – one to encrypt a plaintext (public key) and the other to decrypt a ciphertext (private key).

A message that is encrypted using a public key can only be decrypted using a private key, while also, a message encrypted using a private key can be decrypted using a public key.

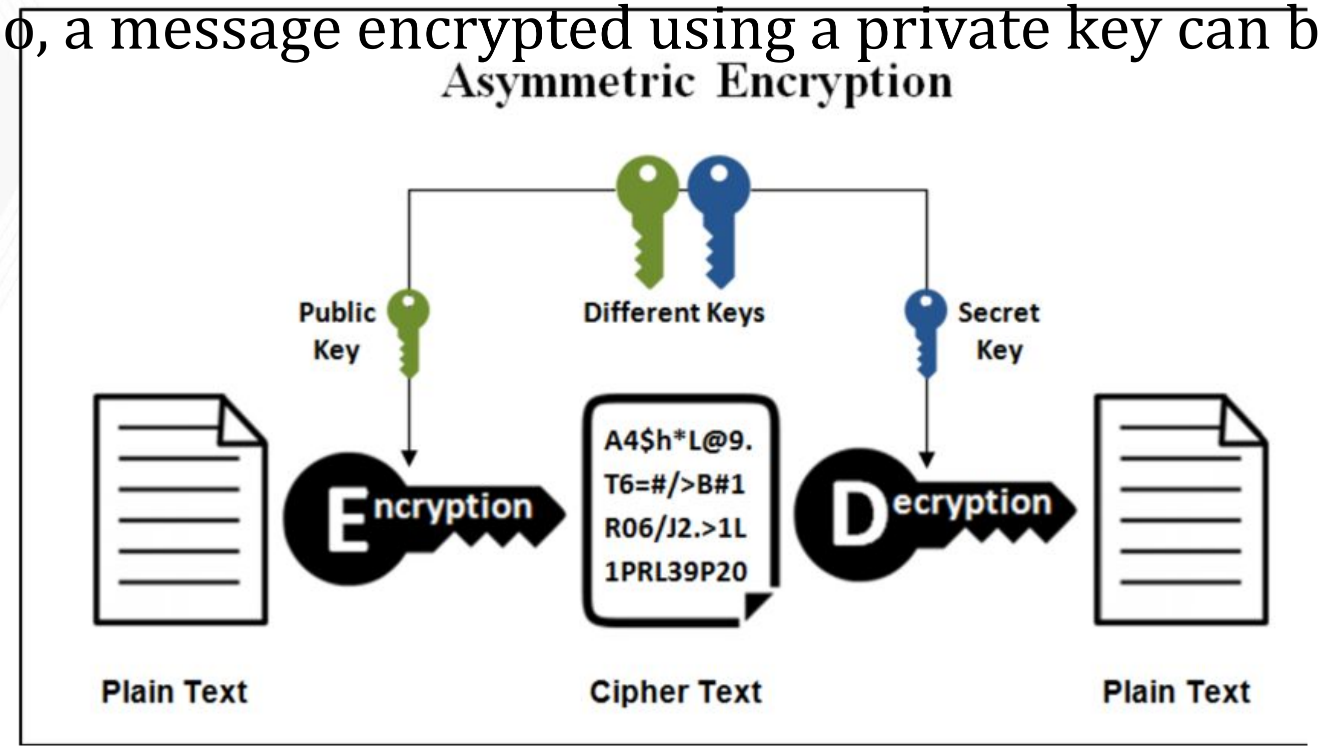


Figure 2. Asymmetric Encryption Explained[1]

Terminology

- ❖ Probabilistic Encryption - Probabilistic encryption is the use of randomness in an encryption algorithm, so that when encrypting the same message several times it will, in general, yield different ciphertexts.

- ❖ Homomorphic Encryption - An encryption scheme that allows computations to be performed on the ciphertexts without gaining any information about their plaintext

- ❖ Ciphertext - data that was the encrypted value of some encryption scheme

- ❖ Plain Text - opposing term which refers to data that was not encrypted

Georgia Tech
CREATING THE NEXT

Terminology

- Lattice - Mathematical structure that represents linear integer combinations of a vectors of an integer basis of n-dimensional space. More information in later sections.

- Bootstrapping - Homomorphically decrypting a double encrypted ciphertext in order to reduce the noise of the ciphertext. More information in later sections.

- KDM Security - A homomorphic encryption scheme is KDM secure if providing the encrypted information of the secret key does not comprehend security.

Georgia Tech
CREATING THE NEXT

Traditional Public Key Encryption

Traditional public key encryption scheme is composed of 3 algorithms:

KeyGen: generate public key *pk* and private key *sk* with randomness (The keys are different each time this algorithm is ran).
The private key is usually kept private by authorized individuals and the public key is known to everyone involved in the communication.

Encrypt(*pk, m*): Given public key *pk* and plaintext *m,* generate ciphertext ψ which represents *m*. In probabilistic encryption,ψ will be different each time with the same *pk* and *m*

Decrypt(*sk,* ψ): Given secret key *sk* and ciphertext ψ, generate plaintext *m*.

Homomorphic Encryption

In addition to the three conventional algorithms, a homomorphic encryption scheme have an algorithm *Evaluate*:

Evaluate$(pk, C, \psi_1, \psi_2, \ldots, \psi_n)$: C is a function with plaintext parameters, $\psi_1, \psi_2, \ldots, \psi_n$ are encrypted values of the arguments (parameters) of C.

Calculate the encrypted value of $C(m_1, m_2, \ldots, m_n)$ while only knowing the encrypted value of $m_1, m_2, \ldots, m_n$.

No information about the plaintext value of $m_1, m_2, \ldots, m_n$ *or* $C(m_1, m_2, \ldots, m_n)$ is revealed to the entity that is computing the *Evaluate* function.

Note that plaintext result can only be known after it's sent back to an entity with the private key.

Homomorphic Encryption

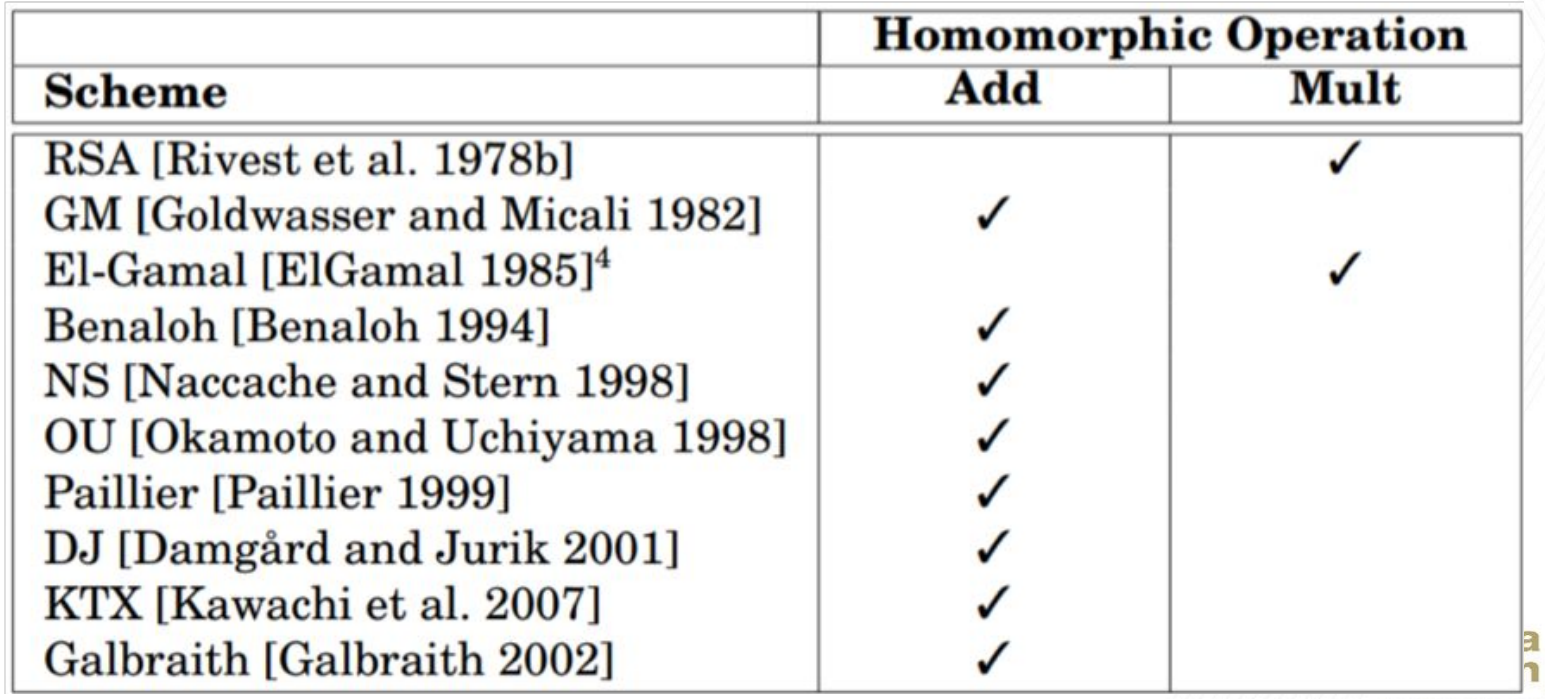
$$\begin{tikzcd}
\boxed{\psi} \arrow[r, "\text{Evaluate}(pk, C, \psi)"] & \boxed{\pi} \arrow[d, "\text{Decrypt}(sk, \pi)"] \\
\boxed{m} \arrow[u, "\text{Encrypt}(pk, m)"'] & \boxed{C(m)}
\end{tikzcd}$$

Types of Homomorphic Encryption

Several types of homomorphic encryption are differentiated based on what type of homomorphic operations are supported (C) for evaluation.

Partially Homomorphic Encryption - Allows unlimited evaluation of either addition or multiplication on ciphertexts

Note: In Homomorphic Encryption Addition and Multiplication are usually performed in modular sense. Under modular 2, Addition and Multiplication is equivalent to binary boolean operations XOR and AND.



	Homomorphic Operation	
Scheme	Add	Mult
RSA [Rivest et al. 1978b]		✓
GM [Goldwasser and Micali 1982]	✓	
El-Gamal [ElGamal 1985][4]		✓
Benaloh [Benaloh 1994]	✓	
NS [Naccache and Stern 1998]	✓	
OU [Okamoto and Uchiyama 1998]	✓	
Paillier [Paillier 1999]	✓	
DJ [Damgård and Jurik 2001]	✓	
KTX [Kawachi et al. 2007]	✓	
Galbraith [Galbraith 2002]	✓	

Table of PHEs[9]

Types of Homomorphic Encryption

Several types of homomorphic encryption are differentiated based on what type of homomorphic operations are supported (C) for evaluation.

Somewhat Fully Homomorphic Encryption - A somewhat homomorphic encryption (SHE) scheme can only perform a limited number of homomorphic computations before the noise parameter becomes large enough to render the scheme useless (i.e. it does no decrypt correctly). Abbreviated S/F Homomorphic Encryption in literature.

An example of an S/F Homomorphic Encryption scheme and noise parameter is given later.

Types of Homomorphic Encryption

Several types of homomorphic encryption are differentiated based on what type of homomorphic operations are supported (C) for evaluation.

Fully Homomorphic Encryption - allows the evaluation of arbitrary circuits of unbounded depth, and is the strongest notion of homomorphic encryption.

Craig Gentry, using lattice-based cryptography, described the first plausible construction for a fully homomorphic encryption scheme in 2009.

However, the Gentry-Halevi implementation of Gentry's original cryptosystem reported timing of about 30 minutes per basic (Homomorphic) bit operation.

Georgia Tech
CREATING THE NEXT

Noise Parameter

Most S/F Homomorphic Encryption schemes performs encryption on boolean values. i.e. the plaintext can only be 0 or 1.

This makes probabilistic encryption (same plaintext value corresponding to multiple ciphertexts) essential because otherwise an adversary can simply try encrypting both 0 and 1 using the public key and compare its value to the ciphertext to break the cryptosystem.

In order to incorporate randomness so that an adversary must make an infeasible amount of trials to break the ciphertext, S/F Homomorphic adds a random valuethat is bounded by some constraint, aka **noise parameter**, to the ciphertext during encryption.

Noise Parameter

Decryption only works when the noise parameter is in a certain constraint.

However, when producing a new ciphertext as the result of an homomorphic operation on other ciphertexts, the noise parameter changes and may exceed the constraint on the noise parameter.

When the noise parameter exceeded the constraint, the homomorphic scheme becomes useless because it no longer produces valid results.

Symmetric Example S/F Homomorphic Encryption Scheme[4]

Plaintext: $\boldsymbol{m} = \{0, 1\}$

KeyGen: Generate $\boldsymbol{k}$ = random large odd integer

Encrypt($\boldsymbol{k}, \boldsymbol{m}$): Generate random integer $-\boldsymbol{k}/4 < \boldsymbol{x} < \boldsymbol{k}/4$, random large integer $\boldsymbol{p}$

Ciphertext $\boldsymbol{c} = \boldsymbol{m} + 2\boldsymbol{x} + \boldsymbol{pk}$

Decrypt($\boldsymbol{k}, \boldsymbol{c}$): $\boldsymbol{m} = ((\boldsymbol{c} \bmod \boldsymbol{k}) \bmod 2)$

Where $\boldsymbol{a}$ mod $\boldsymbol{b}$ is the remainder when $\boldsymbol{a}$ is divided by $\boldsymbol{b}$

Symmetric Example S/F Homomorphic Encryption Scheme[4]

Encryption: $\boldsymbol{c} = \boldsymbol{m} + 2\boldsymbol{x} + \boldsymbol{pk}$

Decryption: $\boldsymbol{m} = ((\boldsymbol{c} \bmod \boldsymbol{k}) \bmod 2)$

The above algorithm works when "$\boldsymbol{c} \bmod \boldsymbol{k} = \boldsymbol{m} + 2\boldsymbol{x}$", which is only true when "$0 <= 2\boldsymbol{x} < \boldsymbol{k}$"

We define the noise parameter as $\boldsymbol{n} = 2\boldsymbol{x}$ and the constraint on the noise parameter would be $0 <= \boldsymbol{n} < \boldsymbol{k}$

We will now see what happens when we perform homomorphic XOR and AND (Addition and Multiplication in modulo 2) on two ciphertexts

Georgia Tech
CREATING THE NEXT

Symmetric Example S/F Homomorphic Encryption Scheme[4]

$$\begin{aligned}\text{Evaluate}(\boldsymbol{k}, \boldsymbol{xor}, \boldsymbol{c}_1, \boldsymbol{c}_2) &= \boldsymbol{c}_1 + \boldsymbol{c}_2 \\ &= \boldsymbol{m}_1 + 2\boldsymbol{x}_1 + \boldsymbol{p}_1\boldsymbol{k} + \boldsymbol{m}_2 + 2\boldsymbol{x}_2 + \boldsymbol{p}_2\boldsymbol{k} \\ &= (\boldsymbol{m}_1 + \boldsymbol{m}_2) + 2(\boldsymbol{x}_1 + \boldsymbol{x}_2) + (\boldsymbol{p}_1 + \boldsymbol{p}_2)\boldsymbol{k}\end{aligned}$$

New noise $\boldsymbol{n} = 2(\boldsymbol{x}_1 + \boldsymbol{x}_2)$ which may be larger than $\boldsymbol{k}$

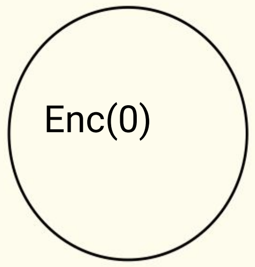
This Homomorphic XOR operation would only work if $\boldsymbol{n} = 2(\boldsymbol{x}_1 + \boldsymbol{x}_2) < \boldsymbol{k}$

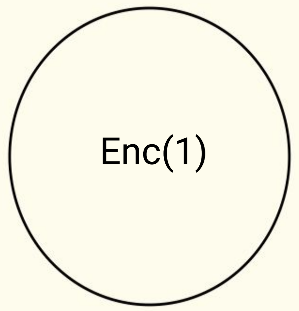
$$\begin{aligned}\text{Evaluate}(\boldsymbol{k}, \boldsymbol{and}, \boldsymbol{c}_1, \boldsymbol{c}_2) &= \boldsymbol{c}_1\boldsymbol{c}_2 \\ &= (\boldsymbol{m}_1 * \boldsymbol{m}_2)+2(\boldsymbol{m}_1\boldsymbol{x}_2+\boldsymbol{m}_2\boldsymbol{x}_1+2\boldsymbol{x}_1\boldsymbol{x}_2)+(\boldsymbol{p}_1\boldsymbol{p}_2\boldsymbol{k})\boldsymbol{k}\end{aligned}$$

New noise $\boldsymbol{n} = 2(\boldsymbol{m}_1\boldsymbol{x}_2+\boldsymbol{m}_2\boldsymbol{x}_1+2\boldsymbol{x}_1\boldsymbol{x}_2)$ which may also be larger than $\boldsymbol{k}$

Cipher Texts that Decrypts to 0

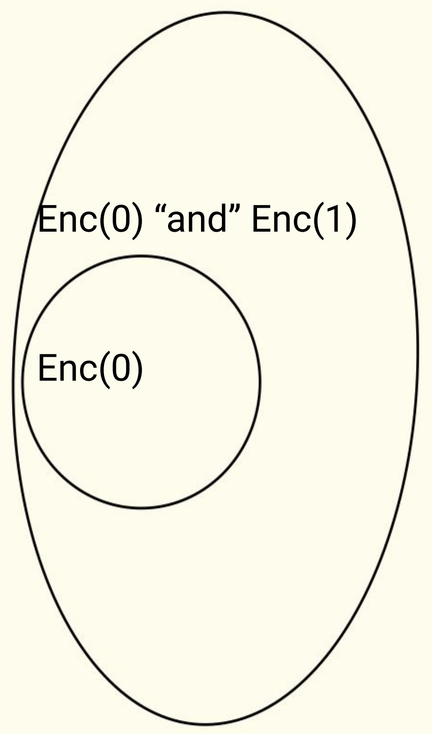
Cipher Texts that Decrypts to 1

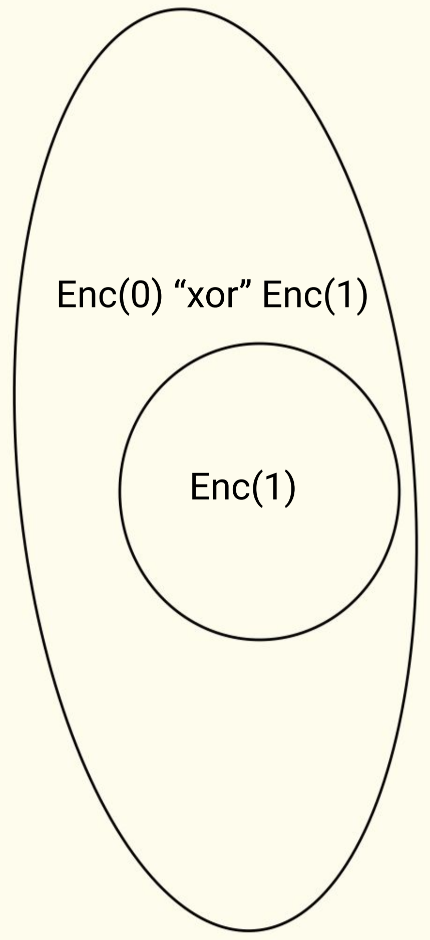


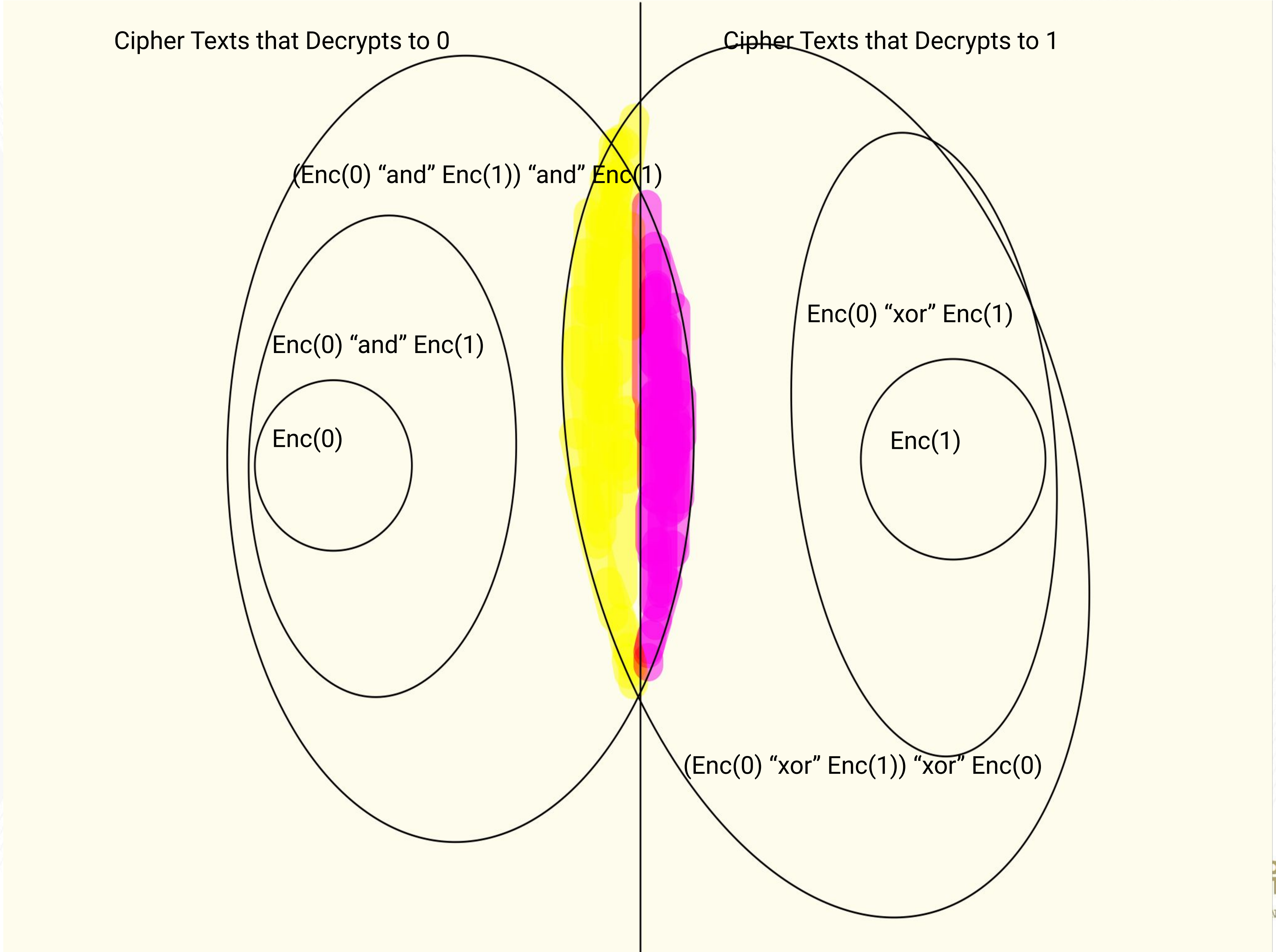


Cipher Texts that Decrypts to 0

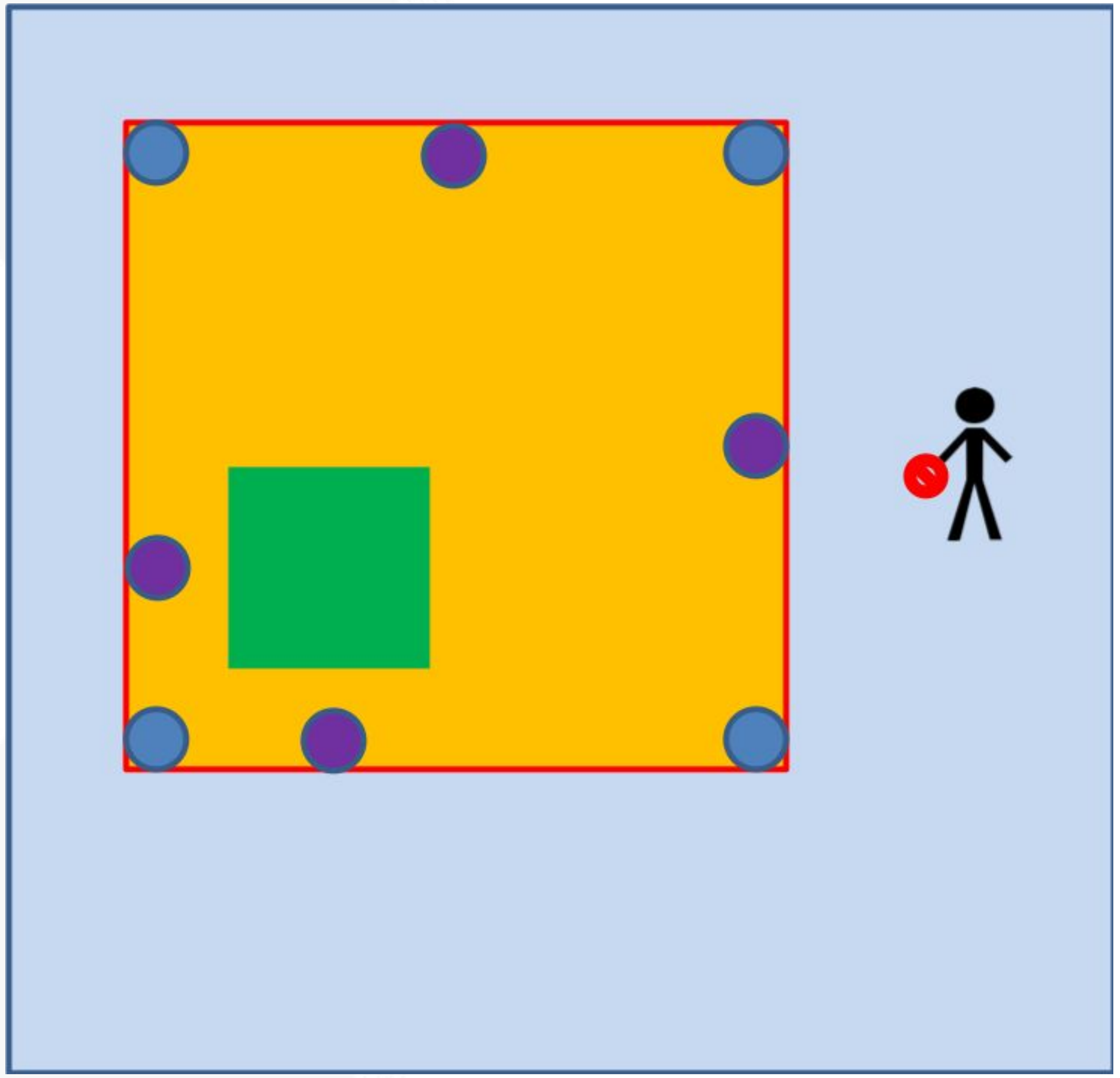
Cipher Texts that Decrypts to 1






Cipher Texts that Decrypts to 0
Cipher Texts that Decrypts to 1
(Enc(0) “and” Enc(1)) “and” Enc(1)
Enc(0) “and” Enc(1)
Enc(0)
Enc(0) “xor” Enc(1)
Enc(1)
(Enc(0) “xor” Enc(1)) “xor” Enc(0)

The Obtention Scenario[5]

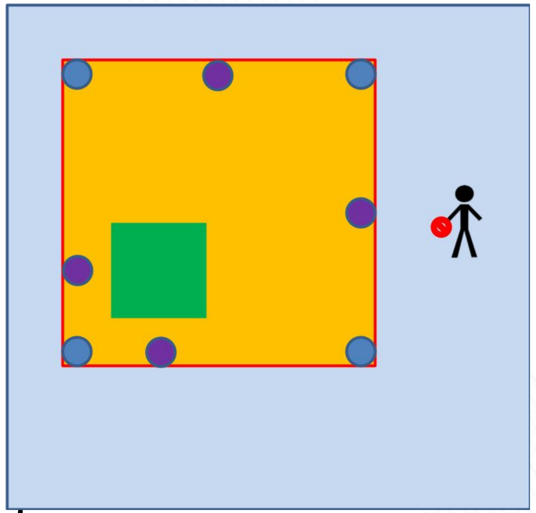


LEGEND:

🟩 →	Water tank
🧍 →	Adversary
● →	Alarm sensor
🟧 →	Area being monitored by the sensor(s)
▢ →	Floor
🚫 →	Contaminant
● →	Camera

Figure 3: Physical setting of the Obtention Scenario[5]

Description: Assumptions and Notes[5]



The adversary wishes to contaminate the water tank, without being caught.

Low noise in information obtained by sensors. [That is, assume noise to be negligible.]

The adversary make use of any opportunity presented to obtain a sensor belonging to the utilized security system, as long as he/she is sure of not being detected.

The presence of cameras is not known to adversary; only the alarm sensors are noticeable and known to the adversary.

File Size Comparison[5]

File Size comparison of periodically captured images: [6]

Commonly considered to be a quick-fix solution when it comes to quantifying the difference between images. Often utilized for distinguishing video thumbnails in YouTube, Bilibili, Nicovideo, etc.
Essentially involves JPEG compression of RAW images, followed by the comparison of their file size.

Often a sub-process in some steganographic analyses of images (especially where comprehension artefacts are noticeably different on compression)[6]

Final Mathematics: Given encrypted file sizes ***a***, ***b***, and encrypted threshold ***d***, determine the encrypted value of

$$|\boldsymbol{a} - \boldsymbol{b}| > \boldsymbol{d}$$

Description: Assumptions and Notes[5]

Processor/Computing Engine:

i.Converts captured raw image to JPEG, computes its file size (in bytes) and stores it in the storage unit.

ii.After doing the same with another image capture, it retrieves the 2 most recently stored file sizes and encrypts them using a homomorphic encryption scheme.

iii.Passes on the encrypted file sizes along with the threshold difference to the transmitter.

Deletes stored JPEG image sizes at set time intervals

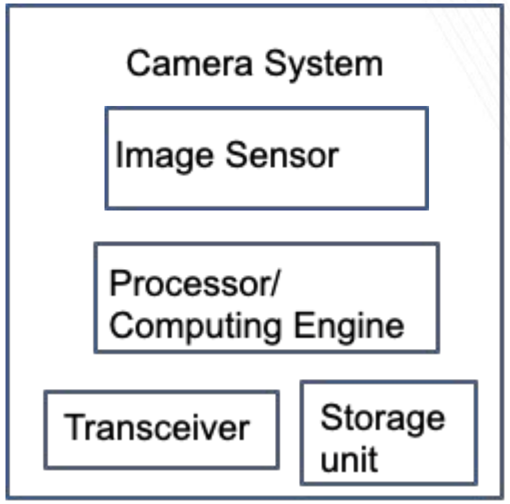
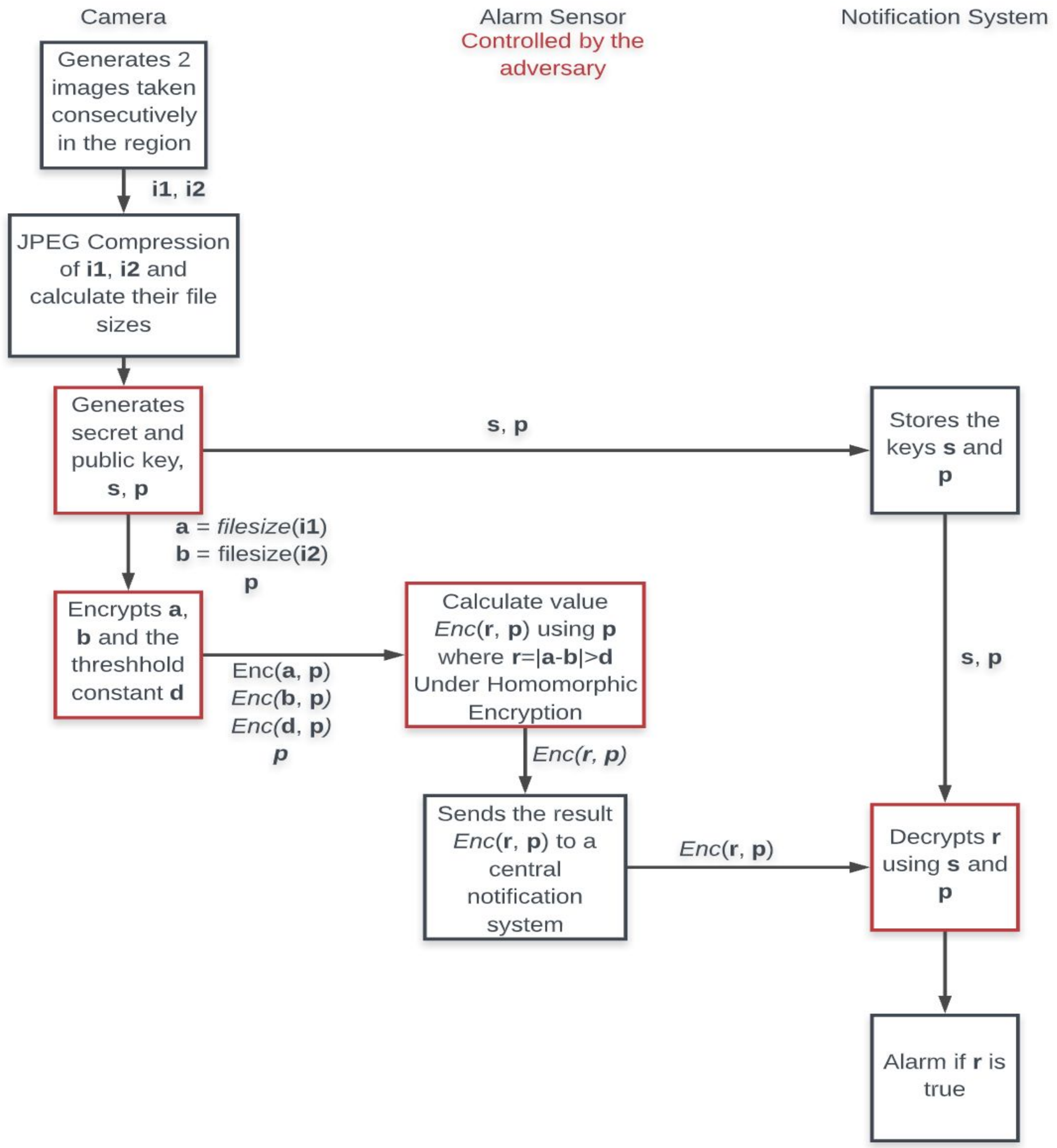


Figure 4. Illustration of the Camera System[5]

Our Process Flow

Our work focuses on the blocks marked red in the diagram, specifically, Key Generation, Encryption, Evaluation and Decryption



Camera
Alarm Sensor
Controlled by the adversary
Notification System
Generates 2 images taken consecutively in the region
i1, i2
JPEG Compression of i1, i2 and calculate their file sizes
Generates secret and public key, s, p
s, p
Stores the keys s and p
a = filesize(i1)
b = filesize(i2)
p
Encrypts a, b and the threshhold constant d
Enc(a, p)
Enc(b, p)
Enc(d, p)
p
Calculate value Enc(r, p) using p where r=|a-b|>d Under Homomorphic Encryption
Enc(r, p)
s, p
Sends the result Enc(r, p) to a central notification system
Enc(r, p)
Decrypts r using s and p
Alarm if r is true

Georgia Tech
CREATING THE NEXT

TFHE: Fast Fully Homomorphic Encryption over the Torus[2]

Fast Fully Homomorphic Encryption over the Torus (TFHE) is an online library that implements the scheme from the paper "Faster fully homomorphic encryption: Bootstrapping in less than 0.1 seconds"[7]. It is a C/C++ library that is intended to evaluate an arbitrary boolean circuit composed of binary gates, over encrypted data, without revealing any information on the data.

TFHE provides APIs for boolean encryption, decryption and logical operations on single bits using fully homomorphic encryption.

We need to transform our problem of threshold value comparison into a routine that only uses boolean operations on bits.

Transformation to Use Only Logic Circuits

$$|a - b| > d$$

$$(a - b > 0)\ ?\ (a - b) > d : (b - a) > d$$

$$(a > b)\ ?\ a > (b + d) : b > (a + d)$$

Operations such as MUX (***a*** ? ***b*** : ***c*** , operation which returns ***b*** if ***a*** is ***true*** and ***c*** if ***a*** is false) and other common boolean logic operations are already provided by the TFHE Library

Now we only need to implement 16-bit adders and comparators using logic circuits.

Performance

C Program wrote to implement Key Generation, Encryption, Evaluation and Decryption of the core computations in the Obtention Scenario[5], ran on Personal Computer

Configuration:
Integer Bit Size: 16 bits
Hardware: 2.3 GHz Intel Core i5

Performance

Key generation: 1.06 second

Encryption: 0.8ms

Encrypted Input Size: 98KB (The size of 3 encrypted 16-bit integers)

Homomorphic Computation: 3.6s

Encrypted Output Size: 2KB (The size of one encrypted bit, the result bit)

Decryption: 0.013 ms

Analysis

Despite significant improvement from Gentry-Halevi implementation of Gentry's Original Scheme[8], which costs 30 minute per basic bit operation, the TFHE library is still much slower than plaintext operations.

Note that hardware implementations of the library is still unavailable so TFHE still cannot be optimized for the alarm sensor in the scenario.

With the efficiency of the current implementation it is still not feasible for the alarm sensor because two consecutive images in video streams are less than one second apart.

Conclusion

Current homomorphic encryption schemes are rarely useful for commercial applications. If special hardware are created in the future for the purpose, however, Homomorphic Encryption might become more practical.

We do not understand the scheme underlying the TFHE library and we cannot guarantee the safety during the evaluation process in our implementation. Therefore, we decided to investigate further on the algorithms underlying the Homomorphic Encryption Schemes.

Georgia Tech
CREATING THE NEXT

Gentry’s Approach to Fully Homomorphic Encryption[4]

Craig Gentry, using lattice-based cryptography, described the first plausible construction for a fully homomorphic encryption scheme in 2009.

However, the Gentry-Halevi implementation of Gentry's original cryptosystem reported timing of about 30 minutes per basic (Homomorphic) bit operation.

The core idea behind Gentry’s construction of Fully Homomorphic Encryption is Bootstrapping, which allows the transformations of certain S/F Homomorphic Encryption schemes to be transformed into a Fully Homomorphic Scheme. Despite various improvements in FHE by multiple researchers after Gentry’s first proposal, bootstrapping is still the core technique behind almost all such schemes. In this report we would introduce the idea of bootstrapping.

All introductions in this report are informal for suitability for the length of this report.

Recrypt Algorithm[4]

How to prevent the "noise" in a S/F homomorphic Encryption Scheme from blowing up? (Note that this is during our Evaluate function)

A plaintext value does not have any "noise". If we can decrypt the plaintext value of a high noise ciphertext and encrypt the plaintext again, we would get a ciphertext with very low noise.

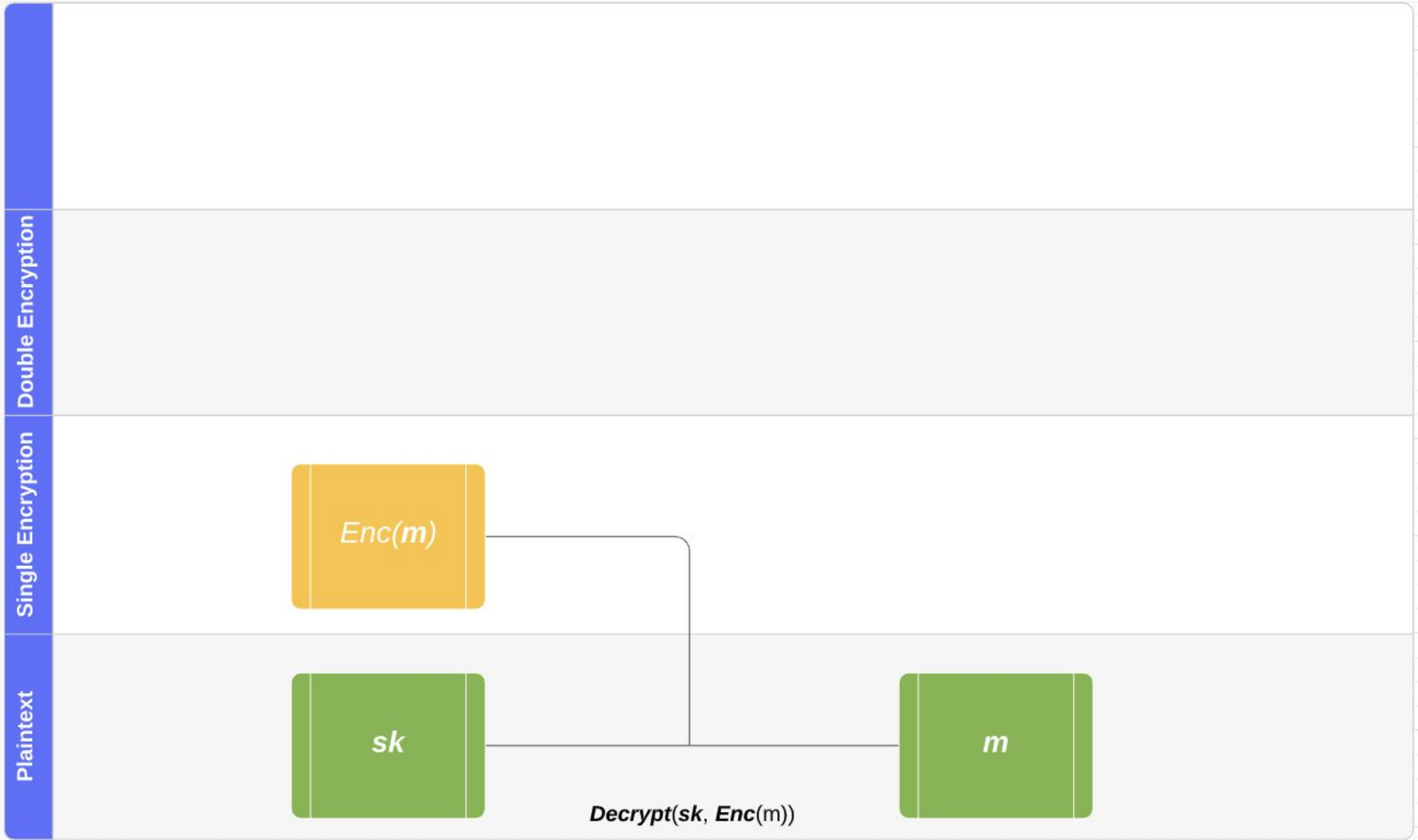
However, this is illegal because the secret key, ***sk***, must not be revealed to the entity performing the *Evaluate* function.

Recrypt Algorithm

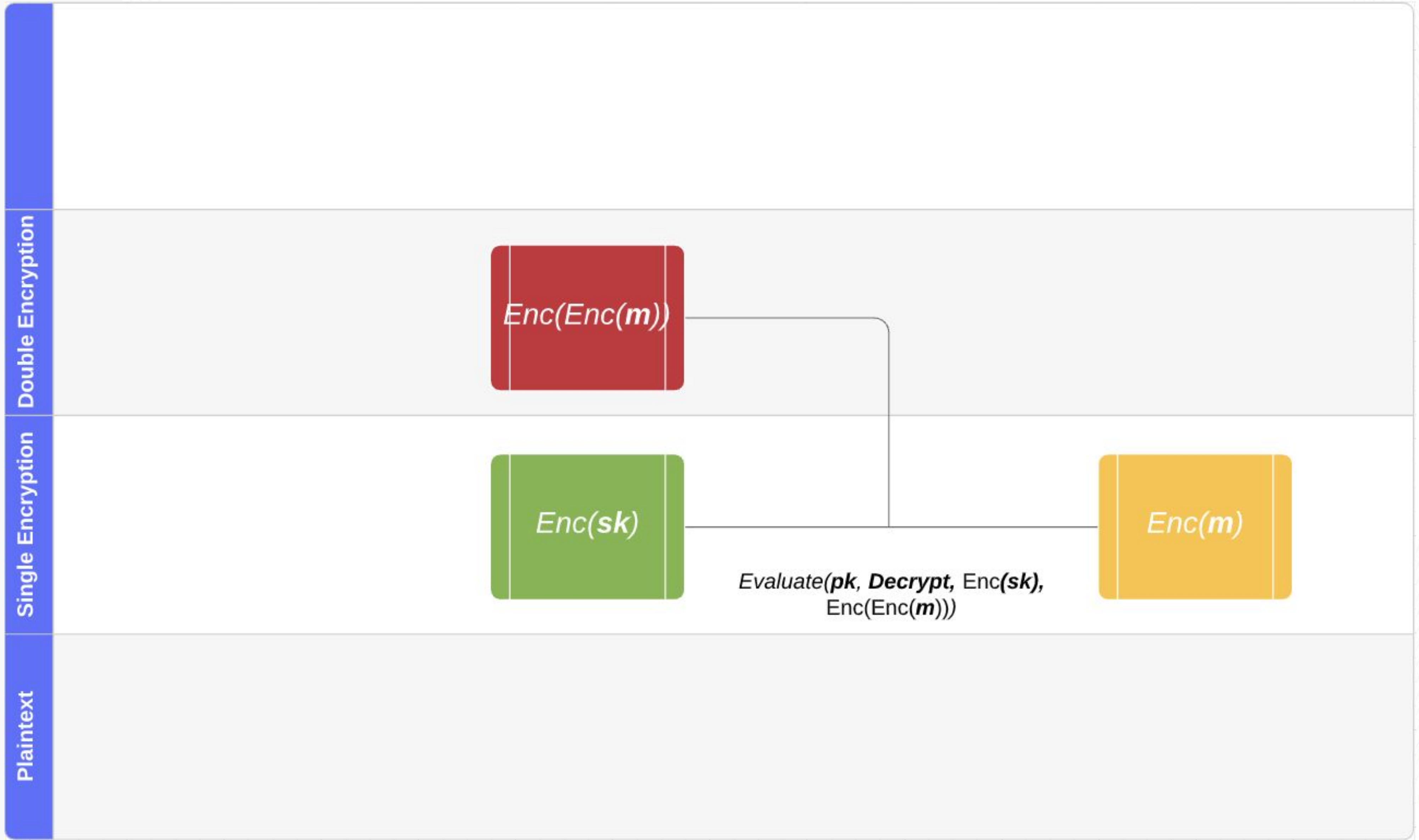
The Decryption algorithm of a encryption scheme, *Decrypt(**sk**, ψ),* is an function (algorithm) that conducts mathematical operations on the secret key ***sk*** and ciphertext value.

What if we can evaluate the *Decrypt* function homomorphically?

Decrypting


Double Encryption
Single Encryption
Plaintext
Enc(m)
sk
m
Decrypt(sk, Enc(m))

Decrypting Homomorphically



Recrypt Algorithm[4]

A encryption scheme is **bootstrappable** if it can homomorphically evaluate its own decryption circuit (and one extra homomorphic operation) without exceeding the noise parameter limit and is KDM secure.

If a scheme is bootstrappable, we define the following algorithm:

Recrypt Algorithm:

Input: High-noise ciphertext ψ, public key ***pk***, *Enc*(***sk***)

Let ψ’ = *Encrypt*(***pk***, ψ) (Doubly encrypted cipher text)

ψ_{new}= *Evaluate*(***pk***, *Decrypt*, Enc(***sk***), ψ’)

Output ψ_{new}

Recrypt Algorithm[4]

Input: High-noise ciphertext ψ, public key $\boldsymbol{pk}$, encrypted secret key $Enc(\boldsymbol{sk})$

Let $\psi' = Encrypt(\boldsymbol{pk}, \psi)$ (Doubly encrypted cipher text)

$\psi_{new} = Evaluate(\boldsymbol{pk}, Decrypt, \mathrm{Enc}(\boldsymbol{sk}), \psi')$

Output ψ_{new}

In this way, we get a ciphertext encrypting the same value but a different representation.

Note that if the *Decrypt* operation is not performed homomorphically it would result in plaintext and therefore eliminate all noise. Therefore, the homomorphic decryption would make the new ciphertext (ψ_{new}) noise independent of the original ciphertext noise and only incorporate noise during the evaluation of the decryption circuit. From the definition of bootstrappable, we know that this noise is small enough to allow us to perform another homomorphic operation before we need to *Recrypt* again.

Bootstrapping

Now we have an algorithm which reduces ciphertext noise, we can straightforwardly construct a method which takes a bootstrappable Somewhat Fully Homomorphic Encryption Scheme and construct a Fully Homomorphic Encryption Scheme.

1. start with original ciphertexts and encryption of the secretkey, $Enc(\boldsymbol{sk})$
2. perform homomorphic operations until it is impossible to perform further operations while guaranteeing correctness
3. perform *Recrypt* algorithm on the ciphertext with heavy noise to get a ciphertext with relatively low noise
4. perform further homomorphic operations until it is impossible to perform further operations while guaranteeing correctness
5. repeat step 3 and 4 until the required function is fully evaluated

Conclusion

For an S/F Homomorphic Encryption scheme to be bootstrappable, the scheme need to evaluate its own decryption circuit.

To create such a scheme, it is straightforward to divide the restriction into two parts:

1. The scheme should have short decryption circuit (Low number of calculations in the *Decrypt* function)
2. The scheme should have large evaluation strength (maximum number of homomorphic operations while guaranteeing correctness)

Many traditional public key encryption have long decryption circuits especially when they rely on exponential calculations. RSA, specifically, rely on modulus exponential calculations that requires too much calculation to make bootstrapping possible.

Conclusion

Gentry claims that many lattice-based encryption algorithms, specifically ideal lattices, have short decryption circuits that can be further reduced with minor modifications.

However, I am not familiar with any lattice-based encryption related materials and I am motivated to investigate into lattice theory and lattice-based encryption to get a better understanding of Gentry's constructed S/F Homomorphic Encryption Scheme.

Georgia Tech
CREATING THE NEXT

Motivation

Lattice-based Encryption is one of the most ideal candidates to post-quantum encryption standards.

Lattice-based Encryption Scheme security are based on the hardness of Lattice problems, which still have no known efficient solutions even using a Quantum Computer. Other encryption schemes such as RSA are already known to be insecure under quantum computer attack.

Many homomorphic encryption schemes have lattices-based encryption as their underlying schemes.

Lattice-based encryption schemes may have short decryption circuits that can be used for bootstrapping.

Georgia Tech
CREATING THE NEXT

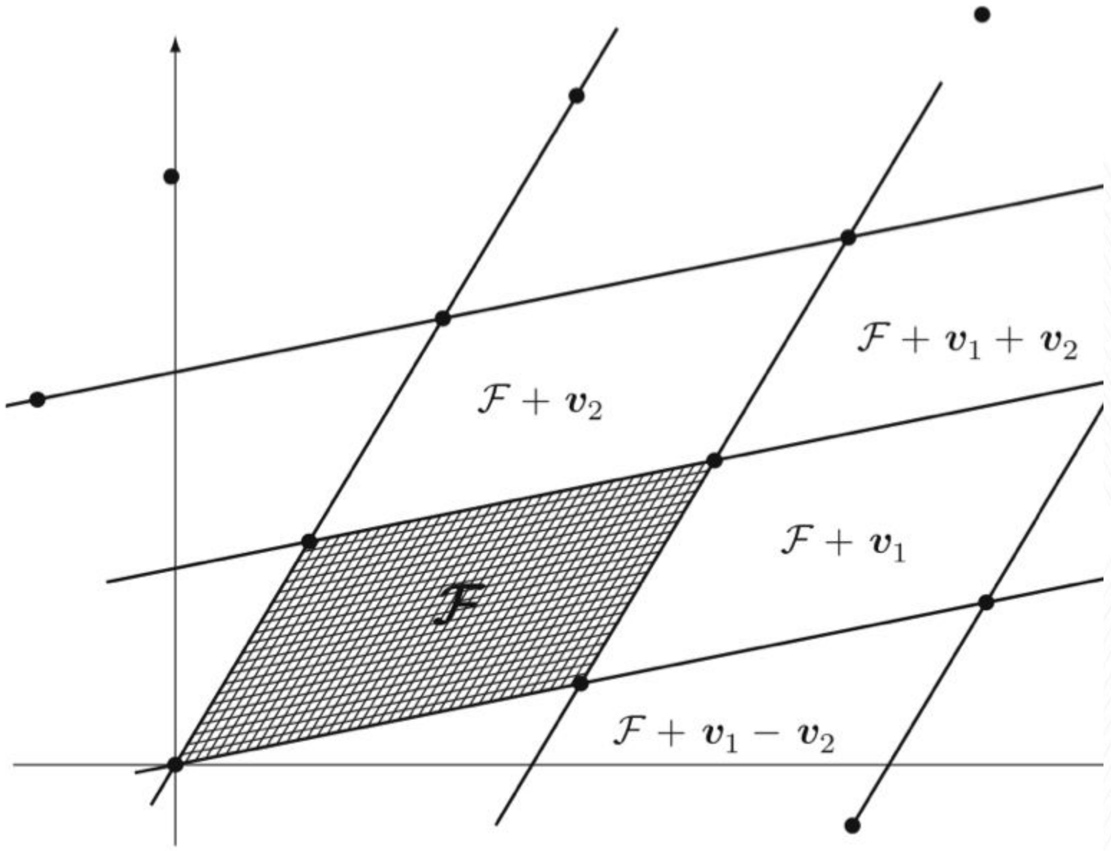
Lattice

Given two independent integer vectors in a two dimensional space, we define a (2D) lattice as all integer combinations of the two vectors.

The 2 selected vectors are called the ‘basis’ of a lattice

One lattice can have multiple basis

Can easily be extended to higher dimensions.



Structure of a Lattice [10]

Lattice Problems

Computationally hard problems involving lattices

This slide would focus on the Shortest Vector Problem: finding the shortest (or a relatively short) vector in a lattice.

For a "good" basis, this is easy

$$\begin{pmatrix} 6 & 0 & 1 \\ 1 & 3 & 1 \\ 7 & 3 & -5 \end{pmatrix}$$

Basis of the same lattice

For a bad one finding the SVP becomes terrible, in general the SVP problem is NP-hard

$$\begin{pmatrix} 494313 & 1267 & 2521 \\ 22814 & 586 & 1166 \\ 229598 & 870 & 37157 \end{pmatrix}$$

Lattice-Based Cryptography

Public key cryptography usually base its security on some know hard problems.
In algorithmic sense "hard" means a problem cannot be solved in a reasonable amount of time.

RSA, for example, based its security on the hardness of prime factorization.

Lattice-Based Cryptography bases its security on hard Lattice problem, and SVP is one of them.

Georgia Tech
CREATING THE NEXT

Knapsack Encryption: First Lattice Encryption Example[10]

A group of encryption schemes with relations to Lattice Problems.

Subset-sum problem:

Given $\boldsymbol{n}$ positive integers, $\boldsymbol{M} = \{\boldsymbol{M}_1, \boldsymbol{M}_2, \ldots, \boldsymbol{M}_n\}$ and positive integer $\boldsymbol{N}$, find a subset in the set $\boldsymbol{M}$ whose sum is $\boldsymbol{N}$

General problem known to be NP-Complete.

Potential application to Encryption

Knapsack Encryption[10]

Subset-sum problem:
Given ***n*** positive integers, ***r*** = {$\boldsymbol{r}_1$, $\boldsymbol{r}_2$, …, $\boldsymbol{r}_n$} and positive integer ***N***, find a subset in the set ***r*** whose sum is ***N***

Special case for subset-sum problem:
Assume that ***r*** is sorted in ascending order, and each number in number in ***r*** is more than two times the previous one (except for the first one), then the subset-sum problem can be easily and quickly solved.

Such sequence with each number at least two times as large as any smaller number is called an **superincreasing** sequence.
{3, 6, 13, 100} is **superincreasing** because 6 > 2 * 3, 13 > 2 * 6, 100 > 2 * 13

Symmetric Knapsack Encryption Example[10]

Key: Sorted superincreasing sequence of n distinct positive integers $\boldsymbol{k} = \boldsymbol{r} = \{\boldsymbol{r}_1, \dots , \boldsymbol{r}_n\}$

plaintext: $\boldsymbol{m}=\{0, 1\}^n$ (i.e. a bitstring with n bits), $\boldsymbol{m}_i$ represents the ith bit in $\boldsymbol{m}$

Encrypt(***k***, ***m***): the encrypted value is the sum of all values in ***r*** for which the corresponding $\boldsymbol{m}_i = 1$.

Decrypt(***k***, ψ): Compare $\psi_{remaining}$ (initialized to ψ) to elements in ***r*** in descending order. For each element $\boldsymbol{r}_i$, if its value if less than $\psi_{remaining}$ then set the corresponding bit $\boldsymbol{m}_i$ to 1 and subtract $\boldsymbol{r}_i$ from $\psi_{remaining}$. Otherwise set the corresponding $\boldsymbol{m}_i$ to 0. Output the final ***m***

Knapsack Encryption[10]

In the previous example if an entity have the key (all the numbers in the set), then they can both encrypt and decrypt the message.

Is it possible to create a set of numbers than can only be used for encryption but not for decryption? (i.e. creating a public key in an asymmetric encryption scheme?)

Knapsack Encryption[10]

Multiplying all members in $\boldsymbol{r}$ by $\boldsymbol{A}$ under modulo $\boldsymbol{B}$ can break the superincreasing property.

Using the “modified” $\boldsymbol{r}$ with the same plaintext bitstring $\boldsymbol{m}$ with the same encryption method generates a ciphertext that is equal to the original ciphertext times $\boldsymbol{A}$ under modulo $\boldsymbol{B}$

If $\boldsymbol{B}$ is large enough, we can guarantee the original (unmodified) ciphertext is smaller than $\boldsymbol{B}$ (i.e. $\boldsymbol{B}$ is larger than all elements in $\boldsymbol{r}$ combined)

Asymmetric Knapsack Encryption Example[10]

KeyGen: Generate random sorted superincreasing sequence of n distinct positive integers $\boldsymbol{r} = \{\boldsymbol{r}_1, \ldots, \boldsymbol{r}_n\}$, $\boldsymbol{B} > 2\boldsymbol{r}_n$, and $\boldsymbol{A}$ which must be relatively prime to $\boldsymbol{B}$. Calculate $\boldsymbol{r}' = \boldsymbol{Ar} \bmod \boldsymbol{B}$

$\boldsymbol{pk} = \boldsymbol{r}'$ $\boldsymbol{sk} = \{\boldsymbol{r}, \boldsymbol{A}, \boldsymbol{B}\}$

plaintext: $\boldsymbol{m}=\{0, 1\}^n$ (i.e. a bitstring with n bits), $\boldsymbol{m}_i$ represents the ith bit in $\boldsymbol{m}$

Encrypt(***pk***, ***m***): $\boldsymbol{pk} = \boldsymbol{r}'$, the encrypted value is the sum of all values in $\boldsymbol{r}'$ for which the corresponding $\boldsymbol{m}_i = 1$.

Asymmetric Knapsack Encryption Example[10]

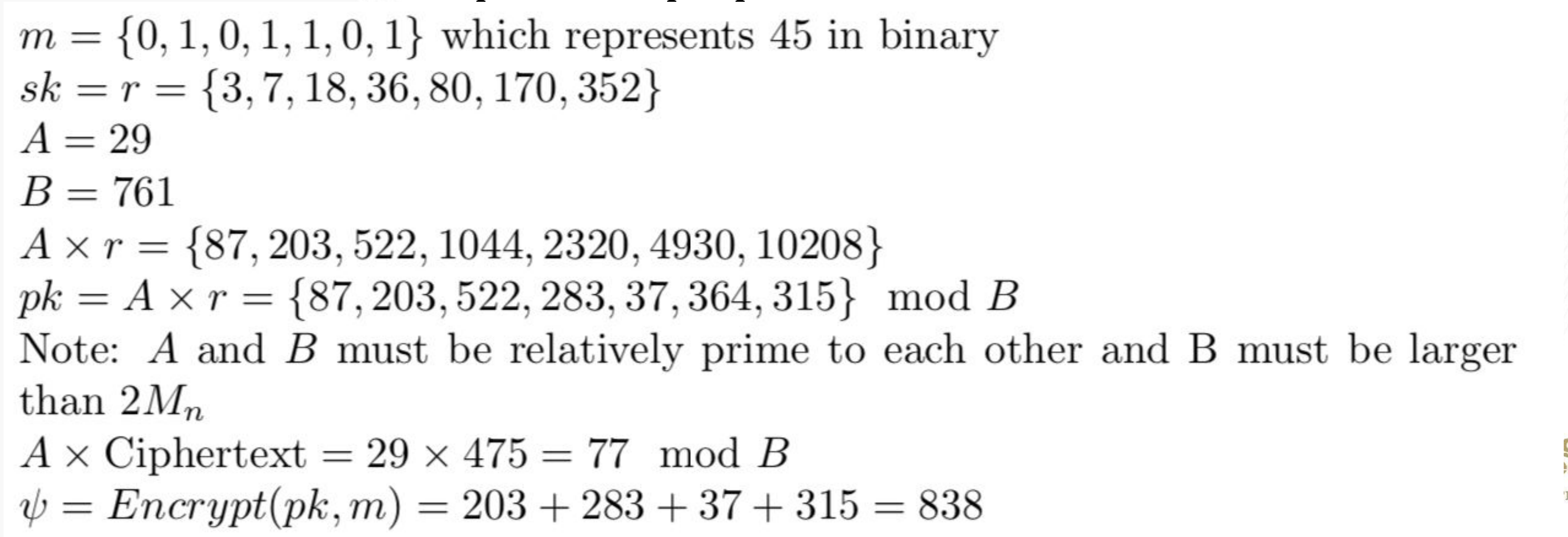
Decrypt(***sk***, ψ): Compute positive integer $\boldsymbol{F} < \boldsymbol{B}$ which satisfies $\boldsymbol{AF} = 1 \; mod \; \boldsymbol{B}$. Calculate ψ’ = ***F***ψ (mod B) and run same algorithm in the symmetric example on ψ’ with ***r***

Note that for the entity who knows only the public key, trying to decrypt a ciphertext requires solving an general subset-sum problem while for the entity holding the secret key decrypting a ciphertext only requires solving a superincreasing subset-sum problem.

Lattice in Knapsack Encryption

Suppose an adversary intends to break the security of the cryptosystem by cracking the non-superincreasing subset-sum problem, we're going to form the problem in terms of a lattice.

We would start with an example of the proposed scheme.



$m = \{0,1,0,1,1,0,1\}$ which represents 45 in binary

$sk = r = \{3,7,18,36,80,170,352\}$

$A = 29$

$B = 761$

$A \times r = \{87,203,522,1044,2320,4930,10208\}$

$pk = A \times r = \{87,203,522,283,37,364,315\} \mod B$

Note: A and B must be relatively prime to each other and B must be larger than $2M_n$

$A \times \text{Ciphertext} = 29 \times 475 = 77 \mod B$

$\psi = Encrypt(pk, m) = 203 + 283 + 37 + 315 = 838$

Lattice in Knapsack Encryption

$$\begin{pmatrix} 2 & 0 & 0 & 0 & 0 & 0 & 0 & 87 \\ 0 & 2 & 0 & 0 & 0 & 0 & 0 & 203 \\ 0 & 0 & 2 & 0 & 0 & 0 & 0 & 522 \\ 0 & 0 & 0 & 2 & 0 & 0 & 0 & 283 \\ 0 & 0 & 0 & 0 & 2 & 0 & 0 & 37 \\ 0 & 0 & 0 & 0 & 0 & 2 & 0 & 364 \\ 0 & 0 & 0 & 0 & 0 & 0 & 2 & 315 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 838 \end{pmatrix}$$

In the above (n+1) * (n + 1) matrix, the rightmost column is the numbers in the public key and the ciphertext to be broken. The left columns composes of an n * n matrix with diagonal values of 2 and all other values 0. The last row are all ones except for that last number.

Lattice in Knapsack Encryption

$$\begin{pmatrix} 2 & 0 & 0 & 0 & 0 & 0 & 0 & 87 \\ 0 & 2 & 0 & 0 & 0 & 0 & 0 & 203 \\ 0 & 0 & 2 & 0 & 0 & 0 & 0 & 522 \\ 0 & 0 & 0 & 2 & 0 & 0 & 0 & 283 \\ 0 & 0 & 0 & 0 & 2 & 0 & 0 & 37 \\ 0 & 0 & 0 & 0 & 0 & 2 & 0 & 364 \\ 0 & 0 & 0 & 0 & 0 & 0 & 2 & 315 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 838 \end{pmatrix}$$

We treat the rows in the above matrix as lattice basis. If we add up the rows whose corresponding term in ***m*** is 1 and subtract the last row, we would get the vector (-1, 1, -1, 1, 1, -1, 1, 0) in the lattice, which is of Euclidean distance √n .

Note that it is of very high probability there is no shorter non-zero vector in the above lattice, and we can easily obtain the plaintext from this short vector. Therefore, we can say that finding the shortest non-zero vector in this lattice is required to obtain the plaintext with only the public key.

Georgia Tech
CREATING THE NEXT

Future Work

1. To construct a encryption scheme that is capable of efficiently perform the operations in the Obtention Scenario, if possible.
2. In order to understand the mathematically behind fully homomorphic encryption, “Fully Homomorphic Encryption over the Integers” is another encryption scheme proposed a year later that implements FHE using bootstrapping without sophisticated mathematical knowledge about ideal lattices.
3. To investigate further in Lattice-based cryptography, it is necessary to learn more about lattice concepts in order to understand purely lattice-based encryption schemes.

References

[1] TBD

[2] Ilaria Chillotti, Nicolas Gama, Mariya Georgieva, and Malika Iz- abach`ene. TFHE: Fast fully homomorphic encryption library, August 2016. https://tfhe.github.io/tfhe/.

[3] A. C. Yao. Protocols for secure computations. In 23rd Annual Symposium on Foundations of Computer Science (sfcs 1982), pages 160–164, Nov 1982.

[4] Craig Gentry. A fully homomorphic encryption scheme. PhD thesis, Stanford University, 2009.

[5] Vignesh Raman. Security for Public Infrastructure using Homomorphic Encryption. Georgia Institute of Technology, 2019

[6] Abbas Cheddad, Joan Condell, Kevin Curran and Paul Mc Kevitt (Signal Processing, 2009). Digital image steganography: Survey and analysis of current methods.
[7] Ilaria Chillotti, Nicolas Gama, Mariya Georgieva, and Malika Iz- abach`ene. Faster fully homomorphic encryption: Bootstrapping in less than 0.1 seconds. Cryptology ePrint Archive, Report 2016/870, 2016. https://eprint.iacr.org/2016/870.
[8] Craig Gentry and Shai Halevi. Implementing gentry’s fully-homomorphic encryption scheme. Cryptology ePrint Archive, Report 2010/520, 2010. https://eprint.iacr.org/2010/520.

[9] Abbas Acar, Hidayet Aksu, A. Selcuk Uluagac, and Mauro Conti. A survey on homomorphic encryption schemes: Theory and implementation, 2017.

Georgia Tech
CREATING THE NEXT

References

[10] Jeffrey Hoffstein, Jill Pipher, and Joseph H. Silverman. An Introduction to Mathematical Cryptography. Springer Science+Business Media, 2014.