이학석사 학위논문

Application of Homomorphic Encryption in Black-Scholes Equation

(블랙숄츠 방정식에서 동형암호이론의 적용)

2014년 7월

서울대학교 대학원
수 리 과 학 부
권 민 우

Application of Homomorphic Encryption in Black-Scholes Equation

(블랙숄츠 방정식에서 동형암호이론의 적용)

지도교수 이 기 암

이 논문을 이학석사 학위논문으로 제출함

2014년 7월

서울대학교 대학원

수 리 과 학 부

권 민 우

권민우의 이학석사 학위논문을 인준함

2014년 7월

위 원 장	천 정 희	인
부 위 원 장	이 기 암	인
위 원	김 판 기	인

Application of Homomorphic Encryption in Black-Scholes Equation

by

Min-Woo Kwon

A DISSERTATION

Submitted to the faculty of the Graduate School
in partial fulfillment of the requirements
for the degree Master of Science
in the Department of Mathematics
Seoul National University
August, 2014

Abstract

This paper propose the method that is the calculation for price of option to apply a fully homomorphic encryption.

In chapter 1,we provide a brief introduction about how to apply that information. In chapter 2, we describe a typical option pricing model which is the Black-Scholes equation and derive its solution. In chapter 3, we introduce CRT-based FHE [8] published at Seoul National University, and BGV algorithm [7] that used in the design of HElib.

Finally, In chapter 4, we show that the results of calculated by modifying c++ code of [8] implementd NTL written by Dr. Lee Hyung-Tae (Nanyang Technological University). And we discuss to improve ways.

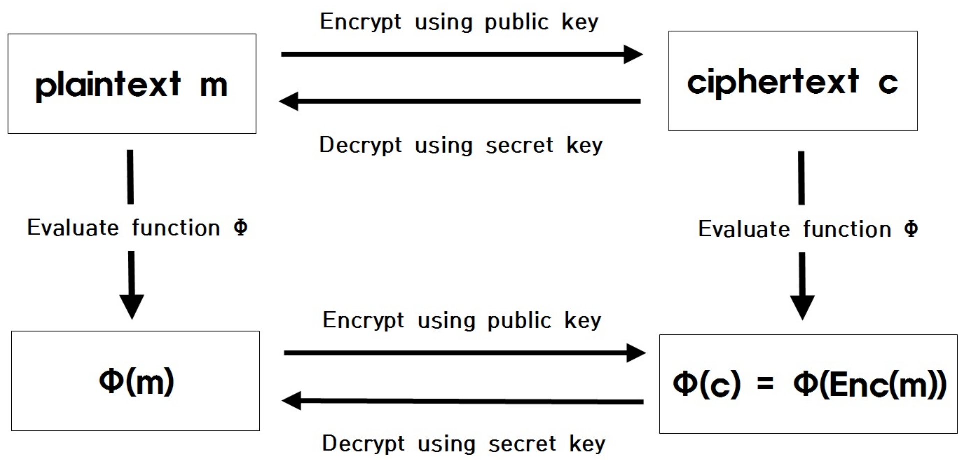
Key words : Black-Scholes Equation, option price, option greeks, fully homomorphic encryption
Student number : 2011-23199

Contents

1 Introduction

1.1 Homomorphic encryption

A **homomorphic encryption** is a form of encryption which allows specific types of computations to be carried out on ciphertext without decryption. It generate an encrypted result which, when decrypted, matches the result of operations performed on the plaintext.



There are some types of homomorphic encryption. Among them, typically, SWHE and FHE is.

- Somewhat Homomorphic Encryption(SWHE)
  : somewhat means it works for some functions
- Fully Homomorphic Encryption(FHE)
  : fully means it works for all functions

1.2 Option of Stock

An option is a contract which gives the buyer (the owner) the right, but not the obligation, to buy or sell an underlying asset at a specified strike price on or before a specified date.

Call option : the right to buy / Put option : the right to sell

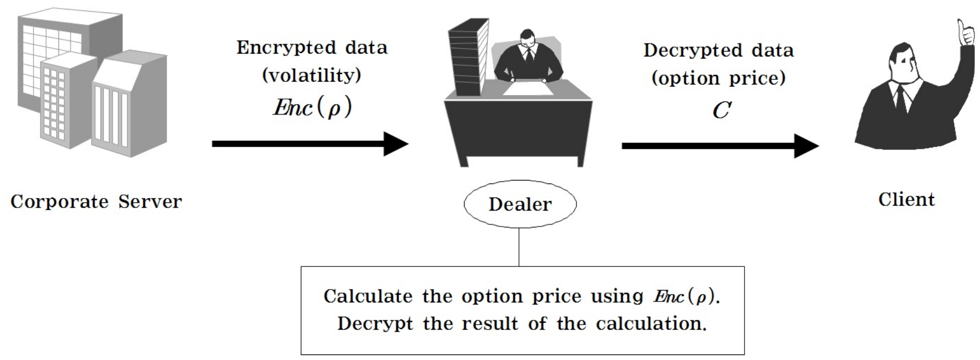
Arbitrage is defined as any trading strategy requiring no cash input (zero investment) that has some probability of making profits, without any risk of a loss. We assume that assets with the same payoffs must have the same prices. It called as no-arbitrage principle. In short, no free lunch.

1.3 Our Work

Option price is determinds by stock price S, exercise price K, interest rate r, expiration date T and volatility σ. That is, the call option price $C = C(S, K, T, \sigma, r)$ and the put option price $P = P(S, K, T, \sigma, r)$. We assume that the volatility is a trade secret. How can we calculate opion price without revealing to a secret information? The answer is a homomorphic encryption.

Company gives the encrypted data to the dealer. Let $Enc(\sigma)$ be an encypted volatility. Of course, the dealer does not know the volatility. He knows only an encrypted data. Then he calculates price of option by using $Enc(\sigma)$ for
By the idea of homomorphic encryption, the decryptions of $C(S, K, T, Enc(\sigma), r)$ and $P(S, K, T, Enc(\sigma), r)$ are $C(S, K, T, \sigma, r)$ and $P(S, K, T, \sigma, r)$, respectively.

The dealer gives an option price to the client. The dealer and the client still does not know the original volatility. They know only the result of computation for the option price.



2 Option pricing

2.1 Continuous model

Suppose that and amount A is invested for n years at an interest rate of R per annum. If the rate is compounded once per annum, the terminal value of the invest rate is

$$A(1+R)^n$$

If the rate is compounnded m times per annum, the terminal value of the investment is

$$A\left(1+\frac{R}{m}\right)^{mn}$$

The limit as the compounding frequency, m, tends to infinity is known as *continuous compounding*. With continuous compounding, it can be shouwn that an amount A invested for n years at rate R grows to

$$Ae^{Rn}$$

The *arbitrage* involves locking in a riskless profit by simultaneously entering into transactions in two or more markets. Actually, any available arbitrage opportunities disappear very quickly. So we will assume that there are no arbitrage opportunities.

We now derive an important relationship between European call option and European put option. We will use the following notation:

- K : Strike price of option
- T : Time to expiration of option
- X_t : stock price of time $t \leq T$
- C_t : the call option price of time $t \leq T$
- P_t : the put option price of time $t \leq T$
- r : Continuously compounded risk-free rate of interest for an investment maturing in time T

Now, consider the following two portfolio :

$PortfolioA$: one European call option + an amount of cash equal to Ke^{-rT}
$PortfolioB$: one European put option + one share

Both are worth $\max(X_T,\ K)$ at expiration of the options. Because the options are European, they cannot be exercised prior to the expiration date. The portfolios must therefore have identical values time t.
This means that

$$C_t + Ke^{-rT} = P_t + X_0 \tag{1}$$

This relationship is known as $put-call\ parity$. It shows that the value of a European call with a certain exercise price and exercise date can be deduced from the value of a European put with the same exercise price and exercise date, and vice versa. If equation (1) does not hold, there are arbitrage opportunities.

We consider a riskless asset (a money market account or bank account), X_t, started at time 0 that grows with the constant continuously compounded risk-free rate of return r. The value of our money market account at time $\triangle t$ is

$$X_{t+\triangle t} = X_t e^{r\triangle t}$$

For sufficiently small $\triangle t$,

$$\frac{\triangle X_t}{X_t} = \frac{X_{t+\triangle t} - X_t}{X_t} \approx r\triangle t$$

(Note that $e^x \approx 1 + x$ for sufficiently small x)

So, rate r can be seen as a kind of return rate on the more general concept. However, the return on risk assets such as stocks has a significant random fluctuation. There were many factors to the cause of the fluctuation. The impact of the sudden news, changes in investor sentiment, etc. Hence, let μ be mean return rate. Then

$$\frac{\triangle X_t}{X_t} \approx (\mu + noise)\triangle t$$

We model its time evolution by some diffusion process with Brownian motion B_t. Let σ be deviation of returns, say *volatility*. We define

$$noise\triangle t := \sigma \triangle B_t$$

Then

$$\frac{\triangle X_t}{X_t} \approx \mu \triangle t + \sigma \triangle B_t$$

Finally, we have the diffrential form

$$\frac{dX_t}{X_t} = \mu dt + \sigma dB_t \tag{2}$$

Or

$$dX_t = \mu X_t dt + \sigma X_t dB_t \tag{3}$$

2.2 Ito integral formula

By integration of (2) both sides,

$$X_t = X_0 + \int_0^t \mu \mathrm{X_s}\ ds + \int_0^t \sigma \mathrm{X_s}\ dB_s$$

First integration term $\int_0^t \mu \mathrm{X_s}\ ds$ is defined by the Reimann integral. But, Secon intergration term $\int_0^t \sigma \mathrm{X_s}\ dB_s$ is not a function of bounded varaiation, since B_s is a brownian motion. Furthemore, we can not define a Riemann-Stieltjes integral for B_s.

We consider stochastic differential equation which is generalized form of (3)

$$dX_t = b(t, X_t)dt + \sigma(t, X_t)dB_t$$

And we know that for any two variable function $f(t, x)$,

$$df(t, x) = f_t\ dt + f_x\ dx$$

Theorem 2.1. ***(Ito formula)***
Let $f(t, x)$ be a C^2 function and an upper bounded.
And let X_t be a solution of ().
Then

$$df(t, X_t) = (f_t + bf_x + \frac{1}{2}\sigma^2 f_{xx})\ dt + \sigma f_x\ dB_t$$

Theorem 2.2. ***(Integration by parts formula)***
Let $b_X = b(t, X_t),\ b_Y = b(t, Y_t),\ \sigma_X = \sigma(t, X_t),\ \sigma_Y = \sigma(t, Y_t).$
And let X_t, Y_t *be solutions of*

$$dX_t = b_X\ dt + \sigma_X\ dB_t,\ dY_t = b_Y\ dt + \sigma_Y dB_t,\ \textit{respectively}$$

Then

$$d(X_tY_t) = X_t\ dY_t + Y_t\ dX_t + \sigma_X\sigma_Y\ dt$$

2.3 Black-scholes equation

Let a_t, b_t be amount of stocks and bonds, repectively
We assume that

$$E[\int_0^T |\mathrm{a_t}|^2\,\mathrm{d}t] < \infty, \quad E[\int_0^T |\mathrm{b_t}|^2\,\mathrm{d}t] < \infty$$

And suppose that a portfolio of the trading strategy (a_t, b_t) is self-financing. i.e changes in value of portfolio is only due to price fluctuations of stocks and bonds. The self-financing implies that

$$d(a_tX_t + b_te^{rt}) = a_t\ dX_t + b_tre^{rt}\ dt$$

Using integration by parts formula,

$$d(a_tX_t + b_te^{rt}) = a_t\ dX_t + X_t\ da_t + \sigma_a\sigma_X\ dt + e^{rt}\ db_t + b_tre^{rt}\ dt$$

So, we have

$$X_t\ da_t + e^{rt}\ db_t + \sigma_a\sigma_X\ dt = 0$$

Let $P(t, X_t)$ be the price of option.
By no-arbitrage principle,

$$a_tX_t + b_te^{rt} = P(t, X_t)$$

Differentiate both sides, from () and (),

$$d(a_tX_t + b_te^{rt}) = (a_t\mu X_t + b_tre^{rt})\ dt + a_t\sigma X_t\ dB_t \tag{4}$$

From Ito Fomula,

$$dP(t, X_t) = \left(\frac{\partial P}{\partial t} + \mu X_t \frac{\partial P}{\partial x} + \frac{1}{2}\sigma^2 {X_t}^2 \frac{\partial^2 P}{\partial x^2} \right) \, dt + \sigma X_t \frac{\partial P}{\partial x} \, dB_t \tag{5}$$

Since (4)=(5),

$$a_t = \frac{\partial P}{\partial x}, \quad b_t = e^{-rt} \left(P - X_t \frac{\partial P}{\partial x} \right)$$

Therefore,

$$\frac{\partial P}{\partial t} + \frac{1}{2}\sigma^2 {X_t}^2 \frac{\partial^2 P}{\partial x^2} + r \left(X_t \frac{\partial P}{\partial x} - P \right) = 0$$

Theorem 2.3. ***(The Black-Scholes Equation)***
Let $X_t = x$ be price of risk asset at time t. the no-arbitrage price of option $P(t, x)$ statisfies

$$\begin{cases} \frac{\partial P}{\partial t} + \frac{1}{2}\sigma^2 {X_t}^2 \frac{\partial^2 P}{\partial x^2} + r \left(X_t \frac{\partial P}{\partial x} - P \right) = 0 \\ P(T, x) = h(x) \qquad \textit{where} \quad 0 < t < T, \quad x > 0 \end{cases} \tag{6}$$

Remark. *The assumptions of the $Black - Scholes$ equation are as follows :*

- *The stock price follows the process geometric Brownian motion with μ and σ constant*
- *The short selling of securities with full use of proceeds is permitted.*
- *There are no transactions costs or taxes. All securities are perfectly divisible.*
- *There are no dividends during the life of the derivative.*
- *There are no riskless arbitrage opportunities.*
- *Security trading is continuous.*
- *The risk-free rate of interest, r, is constant and the same for all maturities.*

Theorem 2.4. *The function*

$$u(t,x) = \int_{-\infty}^{\infty} \frac{1}{\sqrt{4\pi tc^2}} e^{-\frac{(y-x)^2}{4tc^2}} f(y)dy$$

solves the heat equation

$$\begin{aligned} &u_t = c^2 u_{xx}, \quad -\infty < x < \infty, \quad t > 0 \\ &u(0,x) = f(x) \end{aligned}$$

Lemma 2.5. *Let* $\tau = T - t, \quad y = ln(\frac{x}{K}), \quad \nu = \frac{P}{K}$.
By the change variables, (6) as

$$\begin{cases} \frac{\partial \tau}{\partial \nu} = \frac{1}{2}\sigma^2 \frac{\partial^2 \nu}{\partial y^2} + \left(r - \frac{1}{2}\sigma^2\right) \frac{\partial \nu}{\partial y} - r\nu \\ \nu(0,y) = K^{-1}h(Ke^y) \qquad \textit{where} \quad 0 < \tau < T, \quad y > 0 \end{cases} \tag{7}$$

Lemma 2.6. *Let* $\omega = e^{-(\alpha y + \beta \tau)}\nu$.
And let $\alpha = -\frac{1}{2}(k-1), \quad \beta = -\frac{1}{8}\sigma^2(k+1)^2, \quad k = \frac{2r}{\sigma^2}$.
By the change variables, (7) as

$$\begin{cases} \frac{\partial \omega}{\partial \tau} = \frac{1}{2}\sigma^2 \frac{\partial^2 \omega}{\partial y^2} \\ \omega(0,y) = e^{-\alpha y}K^{-1}h(Ke^y) \qquad \textit{where} \quad 0 < \tau < T, \quad \omega > 0 \end{cases} \tag{8}$$

Lemma 2.7. *Let* $\tau = T - t, \quad y = ln(\frac{x}{K}), \quad \nu = \frac{P}{K}$.
By the change variables, (8) as

$$\begin{cases} \frac{\partial \tau}{\partial \nu} = \frac{1}{2}\sigma^2 \frac{\partial^2 \nu}{\partial y^2} + \left(r - \frac{1}{2}\sigma^2\right) \frac{\partial \nu}{\partial y} - r\nu \\ \nu(0,y) = K^{-1}h(Ke^y) \qquad \textit{where} \quad 0 < \tau < T, \quad y > 0 \end{cases} \tag{9}$$

Theorem 2.8. ***(Black-Scholes formula)***
Given pay-off function $h(x) = (x-K)^+$ *and* $(x-K)^-$, *respectivly.*
The prices of European call option C *and European put option* P *are*

$$C(t,x) = xN(d_1) - Ke^{-r(T-t)}N(d_2) \tag{10}$$

$$P(t,x) = -xN(-d_1) + Ke^{-r(T-t)}N(-d_2) \tag{11}$$

where

$$d_1 = \frac{ln(x/K) + (r + \frac{1}{2}\sigma^2)(T-t)}{\sigma\sqrt{T-t}}, \tag{12}$$

$$d_2 = \frac{ln(x/K) + (r - \frac{1}{2}\sigma^2)(T-t)}{\sigma\sqrt{T-t}}, \tag{13}$$

$$N(z) = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{z} e^{-\frac{y^2}{2}} dy \tag{14}$$

Remark. *Polynimial approximation*
We introduce to a polynomial approximation that gives six-decimal-place accuracy for $N(x)$.

$$N(x) = \begin{cases} 1 - N'(x)(a_1 k + a_2 k^2 a_3 k^3 + a_4 k^4 + a_5 k^5) & \textit{if } x \geq 0 \\ 1 - N(-x) & \textit{if } x < 0 \end{cases}$$

where

$$N'(x) = \frac{1}{\sqrt{2\pi}} e^{-\frac{x^2}{2}}, \quad k = \frac{1}{1+\gamma x}, \quad \gamma = 0.2316419$$

$$a_1 = 0.319381530,\ a_2 = -0.356563782,\ a_3 = 1.781477937,$$
$$a_4 = -1.821255978,\ a_5 = 1.330274429$$

Example. *The stock price six months from the expiration of an option is* 42 *dollar, the exercise price of the option is* 40 *dollar, the risk-free interest rate is* 0.1, *and the volatility is* 0.2.

$$d_1 = \frac{ln(42/40) + (0.1 + \frac{1}{2}0.2^2)\frac{6}{12}}{0.2\sqrt{0.5}} = 0.7693$$
$$d_2 = \frac{ln(42/40) + (0.1 - \frac{1}{2}0.2^2)\frac{6}{12}}{0.2\sqrt{0.5}} = 0.6278$$

$$Ke^{-rT} = 40e^{-0.05} = 38.049$$
$$N(0.7693) = 0.7791, \quad N(0.6278) = 0.7349$$
$$C(t,x) = 42N(0.7693) - 38.049N(0.6278) = 4.76$$

2.4 Option Greeks

The call option price $C = C(t, x)$ is actualy fuction of four variables for S, T, σ, r. i.e, $C = C(S, T, \sigma, r)$ where $x = S$. So, change of the variables effects on the option price.
From $\triangle C$ (rate of change of C) during time h, using the taylor serise we have

$$\begin{aligned} dC &= \frac{\partial C}{\partial S} \cdot dS + \frac{\partial C}{\partial T} \cdot dT + \frac{\partial C}{\partial \sigma} \cdot d\sigma + \frac{\partial C}{\partial r} \cdot dr + \frac{1}{2} \frac{\partial^2 C}{\partial S^2} \cdot (dS)^2 \\ &= \triangle \cdot dS - \theta \cdot dT + \nu \cdot d\sigma + \rho \cdot dr + \frac{1}{2}\, \Gamma \cdot (dS)^2 \end{aligned}$$

2.4.1 Delta

The delta ($\triangle$) of an option is defined as the rate of change of the option price with respect to the price of the underlying asset.

For a call option price C, a put option price P,

$$\begin{aligned} \triangle_C &= \frac{\partial C}{\partial S} = N(d_1) \\ \triangle_P &= \frac{\partial P}{\partial S} = -N(-d_1) = N(d_1) - 1 \end{aligned}$$

If a stock price S is increasing, then C is incrresing. So, $\triangle_C > 0$. But, P is decreasing. Hence, $\triangle_P < 0$

2.4.2 Gamma

The gamma (Γ) of an option on an underlying asset is the rate of change of the option's delta with respect to the price of the underlying asset. It is the second partial derivative of the option with respect to asset price.

$$\Gamma = \frac{\partial^2 C}{\partial S^2} = \frac{\partial^2 P}{\partial S^2} = \frac{N'(d_1)}{\sigma S \sqrt{T}} = \frac{e^{-\frac{d_1^2}{2}}}{\sigma S \sqrt{2\pi T}}$$

$\Gamma = \frac{\partial \triangle}{\partial S}$ >0, since $\triangle$ is increasing as S increasing. If $\triangle$ is constant, then $\Gamma = 0$.

2.4.3 Theta

The theta (θ) of an option is the rate of change of the value of the option price with respect to the passage of time. θ is sometimes referred to as the *timedecay* of the option. Usually, time is measured in days so that θ is the change in the option value when one day passes with all else remaining the same.

$$\theta_C = -\frac{\partial C}{\partial T} = -\frac{\sigma S \cdot e^{-\frac{d_1^2}{2}}}{2\sqrt{2\pi T}} - rKe^{-rT} \cdot N(d_2)$$
$$\theta_P = -\frac{\partial P}{\partial T} = -\frac{\sigma S \cdot e^{-\frac{d_1^2}{2}}}{2\sqrt{2\pi T}} + rKe^{-rT} \cdot N(-d_2)$$

As the time to maturity decreases with all else remaining the same, the option tends to become less valuable. So, $\theta_C < 0$ usually.

2.4.4 Vega

Up to now, our assumption is the volatility of an option is constant. In practice, volatility change over time. This means that the value of an option is liable to change because of movements in volatility. The vega(ν) of an option is the rate of change of the value of the option with respect to the volatility of the underlying asset.

$$\nu = \frac{\partial C}{\partial \sigma} = \frac{\partial P}{\partial \sigma} = S\sqrt{T} \cdot N'(d_1) = \frac{S\sqrt{T} \cdot e^{-\frac{d_1^2}{2}}}{\sqrt{2\pi}}$$

By put-call parity, the stock price doesn't affect the volatility. So, $\frac{\partial C}{\partial \sigma} = \frac{\partial P}{\partial \sigma}$. And if the volatility is increasing, then a price of call(or put) option is inceasing. Hence, $\nu > 0$.

2.4.5 Rho

The rho(ρ) of an option is the rate of change of the value of the option with respect to the interest rate. It measures the sensitivity of the value of an option to interest rates.

$$\rho_C = \frac{\partial C}{\partial r} = TKe^{-rT} \cdot N(d_2)$$
$$\rho_P = \frac{\partial P}{\partial r} = -TKe^{-rT} \cdot N(-d_2)$$

If the interest rate is increasing, then a price of call option is incresing and a price of put option is decreasing. So, $\rho_C > 0$ and $\rho_P < 0$.

Example. *We use conditions of the previous example. Then*

$$\triangle_C = N(0.7693) = 0.7791$$
$$\triangle_P = N(0.7693) - 1 = -0.2208$$
$$\Gamma = \frac{e^{-\frac{(0.7693)^2}{2}}}{0.2 \cdot 42\sqrt{2\pi(0.5)}} = 0.0499$$
$$\theta_C = -\frac{(0.2) \cdot 42 \cdot e^{-\frac{(0.7693)^2}{2}}}{2\sqrt{2\pi(0.5)}} - (0.1) \cdot 40 \cdot e^{-0.05} \cdot N(0.6278) = -4.5590$$
$$\theta_P = -\frac{(0.2) \cdot 42 \cdot e^{-\frac{(0.7693)^2}{2}}}{2\sqrt{2\pi(0.5)}} + (0.1) \cdot 40 \cdot e^{-0.05} \cdot N(-0.6278) = -0.7541$$
$$\nu = \frac{42\sqrt{0.5} \cdot e^{-\frac{(0.7693)^2}{2}}}{\sqrt{2\pi}} = 8.8134$$
$$\rho_C = (0.5) \cdot 40e^{-0.05} \cdot N(0.6278) = 13.982$$
$$\rho_P = -(0.5) \cdot 40e^{-0.05} \cdot N(-0.6278) = -5.0425$$

3 Fully homomorphic encryption

Given ciphertexts that encrypt $\pi_1, \ldots, \pi_t$, fully homomorphic encrytion should allow anyone to ouput a ciphertext that encrypts $f(\pi_1, \ldots, \pi_t)$ for any efficiently computable function f. And we have no information about $\pi_1, \ldots, \pi_t$ or $f(\pi_1, \ldots, \pi_t)$ or any related plaintext. That is, the input and ouput, intermediate values are always encrypted. We know nothing about the original data.

3.1 Basic Definitions

In cryptography, encryption is the process of encoding messages or information in such a way that only authorized parties can read it. In an encryption scheme, the message or information (plaintext) is encrypted using an encryption algorithm, turning it into an unreadable ciphertext.

Let $\mathcal{P}$ be plaintext space and $\mathcal{C}$ be ciphertexts space. we consider a mapping

$$Enc : \mathcal{P} \to \mathcal{C}$$

Enc*(Encrypt)* is a rule of trasforming a plaintext into a ciphertext using some key. This is a procedure that takes in inputs and returns a value. Furthemore, it is randomized fuction. The randomize input determines which of the many possible ciphertexts a plaintext may be mapped to. And ***Dec****(Decrypt)* is rule of decryption

$$Dec = Enc^{-1} : \mathcal{C} \to \mathcal{P} \quad \textit{such that} \quad Dec(Enc(m)) = m \quad \forall m \in \mathcal{P}$$

The term homomorphic from the algebraic term homomorphism. We say an encryption shceme is ***homomorphic*** with respect to an operation $\diamond$ on $\mathcal{P}$ and some operation $*$ on $\mathcal{C}$, if

$$\textit{Dec(Enc(}m_1\textit{)}*\textit{Enc(}m_2\textit{))=Dec(Enc(}m_1 \diamond m_2\textit{))=}m_1 \diamond m_2 \quad \forall m_1, m_2 \in \mathcal{P}$$

There are a few types of homomorphic encryption shceme. We introduce two types of schme. ***Somewhat homomorphic encryption scheme*** **(SWHE)** is that perform only a limited number of operations on encrypted data. Another one is ***Fully homomorphic encryption shceme*** **(FHE)** that can perform an unlimited number of both types of operations on encrypted data.

Definition 3.1. (Homomorphic encrypyion scheme)
A homomorphic encryption schme $\mathcal{E}$ consists of four algorithmes :

1. ***(pk, sk)*** $\leftarrow$ ***KeyGen***$_{\mathcal{E}}$***(***λ***)*** : the key generation is a randomized algorithm that takes the security parameter λ as input, and outputs a pair of key *(pk,sk)*. a public key *pk* defines a plaintext space $\mathcal{P}$ and a secret key *sk* defines a ciphertext space $\mathcal{C}$.

2. $\psi \leftarrow$ ***Enc***$_{\mathcal{E}}$***(pk,*** π***)*** : randomized algrithm takes a public key *pk* and plaintext $\pi \in \mathcal{P}$, ouputs a ciphertext $\psi \in \mathcal{C}$.

3. ***Dec***$_{\mathcal{E}}$***(sk,*** ψ***)*** $\rightarrow \pi$: algorithm takes a secret key *sk* and $\psi$and ouputs the plaintext π.

4. $\Psi \leftarrow$ ***Evaluate***$_{\mathcal{E}}$***(pk,*** C***,*** Ψ***)*** : (possible randomized) efficieent algorithm which takes as input the public key *pk*, a circuit C from a permitted set $C_{\mathcal{E}}$ of circuits, and a tuple of ciphertexts $\Psi \ = <\psi_1, \dots, \psi_t>$ for the input wires of C. It outputs a ciphertext ψ.
   That is, if $\psi_i \leftarrow$ *Enc*$_{\mathcal{E}}$*(pk,* π_i*)*, then $\Psi \leftarrow$ *Evaluate*$_{\mathcal{E}}$*(pk,* C*,* Ψ*)* means that encyrpt $C(\pi_1, \dots, \pi_t)$ under *pk* where $C(\pi_1, \dots, \pi_t)$ is the output of C on inputs $\pi_1, \dots, \pi_t$.

Note that *correctness* of encryption scheme is defined by if *(pk, sk)* $\leftarrow$ *KeyGen*$_{\mathcal{E}}$*(*λ*)* and $\psi \leftarrow$ *Enc*$_{\mathcal{E}}$*(pk,* π*)*, then *Dec*$_{\mathcal{E}}$*(sk,* ψ*)* $\rightarrow \pi$. But, we need *correctness* of homomorphic encryption.

Definition 3.2. (Correctness of homomorphic encrypyion)
For any key-pair *(pk, sk)* output by *KeyGen*$_{\mathcal{E}}$*(*λ*)* and any circuit $C \in C_{\mathcal{E}}$, $\pi_1, \dots \pi_t \in \mathcal{P}$ and $\Psi =<\psi_1, \dots, \psi_t>\in \mathcal{C}$ with $\psi_i \leftarrow$ *Enc*$_{\mathcal{E}}$*(pk,* π_i*)* satisfies that

$$if \quad \psi \leftarrow Evaluate_{\mathcal{E}}(pk,\ C,\ \Psi), \quad then \quad Dec_{\mathcal{E}}(sk,\ \psi) \rightarrow C(\pi_1, \dots \pi_t).$$

except with negligible probability over the radom coins in *Evaluate*$_{\mathcal{E}}$.
Then we say that a homomorphic encryption schme $\mathcal{E}$ is **correct** for circuits in $C_{\mathcal{E}}$.

Definition 3.3. (Compact homomorphic encrypyion)
A homomorphic encryption scheme $\mathcal{E}$ is **compact**, if there is a polynomial f such that for every value of the security parameter λ, $\mathcal{E}$'s decryption algorithm can be expressed as a circuit $D_{\mathcal{E}}$ of size at most $f(\lambda)$.
If $\mathcal{E}$ is compact and also correct for circuits in $\mathcal{C}_{\mathcal{E}}$, then we say that $\mathcal{E}$ **compactly evaluates** circuits in $\mathcal{C}_{\mathcal{E}}$.

Note than compactness of homomorphic encrytion is an upper bound of the length of ciphertexts output by $Evaluate_{\mathcal{E}}$. Furthemore, it is an upper bound on the size of the decrytion $D_{\mathcal{E}}$ for the scheme $\mathcal{E}$ thas depends only in the security parameter.

Definition 3.4. (Fully homomorphic encrypyion)
A homomorphic encryption scheme $\mathcal{E}$ is **fully homomorphic**, if it is compact evaluates all circuits.

Definition 3.5. (Leveled Fully homomorphic encrypyion)
A family of homomorphic encryption scheme $\{\mathcal{E}^{(d)} : d \in \mathbb{Z}^+\}$ is **leveled fully homomorphic**, if they all use the same decryption circuit, $\mathcal{E}^{(d)}$ compactly evaluates all circuits of depth at most d (that use some specified set of gates) and the computational complexity of $\mathcal{E}^{(d)}$'s algorithms is polynomial in λ, d and (in the case of $Evaluate_{\mathcal{E}}$) the size of the circuit $\mathcal{C}$.

Now, we introduce two encryption schems that are SWHE and (Leveled) FHE. We will show only the construction algorithm. For further details or proof will not be covered here. Refer to [7], [8] for full contents of schems.

3.2 CRT-based fully homomorphic encryption over the integers

This scheme is the content of papers published from Seoul National University in 2013[8]. An encryption of a message and a decrption are used to the chinese remainder theorem. And it is a symmetric key encryption shceme that allows only bounded number of modular addition and multiplications. Hence, is a somewhat homomorphic encryption scheme. But, it can be extended to a fully homomorphic encryption through bootstrapping.

3.2.1 Notations

Denoted by $a \leftarrow A$ as choose an element a from a set A randomly.
$\mathbb{Z}_p := \mathbb{Z} \bigcap (\frac{-p}{2}, \frac{p}{2}]$ and $[x]_p = x \bmod p := x \; modulo \; p$ denotes a number in $\mathbb{Z}_p$.

For relatively prime integer p_0, p_1, that is $gcd(p_1, p_2) = 1$, we define

$$CRT_{(p_1,p_2)}(x_1, x_2) := \sum_{i=1}^{2} x_i \hat{p}_i (\hat{p}_i^{\,-1} \bmod p_i) \bmod N$$
$$\text{where } N = p_1 p_2 \text{ and } \hat{p}_i = \frac{N}{p_i}$$

For η-bit prime p and l_Q-bit integers Q, define distributions

$$\mathcal{D}_{\gamma,\rho}(p) = \{choose\ q \leftarrow \mathbb{Z} \bigcap [0, \frac{2^\gamma}{p})\ and \quad e \leftarrow \mathbb{Z} \bigcap (-2^\rho, 2^\rho\ )\}$$
output : $pq + e$

$$\mathcal{D}_\rho(p; q) = \{choose\ e_0 \leftarrow \mathbb{Z} \bigcap [0, q_0)\ and \quad e_1 \leftarrow \mathbb{Z} \bigcap (-2^\rho, 2^\rho\ )\}$$
output : $CRT_{q,p}(e_0, e_1)b$

$$\mathcal{D}_\rho(p; Q; q) = \{choose\ e_0 \leftarrow \mathbb{Z} \bigcap [0, q_0)\ and \quad e_1 \leftarrow \mathbb{Z} \bigcap (-2^\rho, 2^\rho\ )\}$$
output : $CRT_{q,p}(e_0, e_1 Q_1)$

3.2.2 The construction

$\mathbb{Z}_Q$: the message space
λ : the security parameter
ρ : the bit length of the error
η : the bit length of the secret primes
γ : the bit length of a ciphertext
τ : the number of encryptions of zero in public key
l_Q : the bit length of Q

KeyGen$(\lambda, \rho, \eta, \gamma, \tau, l_Q)$: Choose η-bit prime p and $q \leftarrow \mathbb{Z} \bigcap [0, \frac{2^\gamma}{p})$ and set $N = pq$. Choose l_Q-bit integers Q with $\gcd(Q,\ N) = 1$.
Output the public-key pk.

$$pk \;=\; (N,\ Q,\ X = \{x_j = CRT_{(q,p)}(e_{j0}, e_{j1}Q)\},\ y = CRT_{(q,p)}(e'_0, e'_1Q + 1))$$

where $e_{j0}, e'_0 \leftarrow \mathbb{Z} \bigcap [0, q_0)$ and $e_{j1}, e'_1 \leftarrow \mathbb{Z} \bigcap (-2^\rho, 2^\rho)$ for $j \in [1, \tau]$.
Output the secret-key $sk = p$.

Enc$(pk,\ m)$: For any $m \in \mathbb{Z}_Q$, Output $c = my + \sum_{j \in S} x_j \bmod N$ where S is a random subset of $\{1, \dots, \tau\}$. A ciphertext c can be written of the form

$$\begin{aligned} c &= my + \sum_{j \in S} x_j \bmod N \\ &= CRT_{(q,p)}(e'_0 m, e'_1 Qm) + CRT_{(q,p)}(\sum_{j \in S} e_{j0}, \sum_{j \in S} e_{j1}Q) \\ &= CRT_{(q,p)}(e_0, e_1Q + m) \quad \text{for some} e_0 \in \mathbb{Z} \bigcap [0, q), e_1 \in \mathbb{Z} \bigcap (-2^{\rho'}, 2^{\rho'}) \\ &\qquad\qquad\qquad\qquad\qquad\quad where\ \rho' = max\{\rho + l_Q, 2\rho + log\tau\} \end{aligned}$$

Dec$(sk,\ c)$: Output $m = (c \bmod p) \bmod Q$

Eval$(pk, \mathcal{C}, c = (c_1, \dots, c_t))$: Permitted circuit $\mathcal{C}$ with t inputs defined below and a t-tuple of ciphertextes c. Output $C(c_1, \dots, c_t)$ using Add and Mul.

Add$(pk,\ c_1, c_2)$: Output $c_1 + c_2 \bmod N$
Mul$(pk,\ c_1, c_2)$: Output $c_1 \times c_2 \bmod N$

Note that to decrypt a ciphertext correctly after operations of ciphertext, the size of e_0, e_1 and Q must be sufficiently smaller than p.

3.2.3 Corretness and the multiplicative depth

Let $\mathcal{C}$ be an integer circuit(Add, Mul) with t inputs. We define that $\mathcal{C}$ is a **permitted circuit**, if an output of $\mathcal{C}$ has absolute value at most $2^{\alpha(\eta-4)}$ whenever the absolute value of each t input is smaller than $2^{\alpha(\rho'+l_Q)}$ for any $\alpha \geq 1$.

Suppoes $c \leftarrow \mathrm{Enc}(pk, m)$ for $m \in \mathbb{Z}_{\mathbb{Q}}$. Then

$$\begin{aligned} c &= CRT_{(q,p)}(e_0, e_1Q + m) \\ &= pa + e_1Q + m \text{ for some } a \text{ and } |e_1Q + m| < 2^{\rho'+l_Q}. \end{aligned}$$

Let $\mathcal{C}_{\mathcal{E}}$ be a permitted circuit, $\mathcal{C} \in \mathcal{C}_{\mathcal{E}}$ and $c_j \leftarrow \mathrm{Enc}(pk, m_j)$ for $j = 1, \ldots, t$.
Let $m' =\leftarrow C(m_1, \ldots, m_t)$ and $c' \leftarrow Eval(pk, \mathcal{C}, c_1, \ldots, c_t)$.
If f is the polynomial compuetd by $\mathcal{C}$. Then

$$\begin{aligned} c' \bmod p &= f(c_1, \ldots, c_t) \bmod p \\ &= f(c_1 \bmod p, \ldots, c_t \bmod p) \bmod P. \end{aligned}$$

Since $\mathcal{C} \in \mathcal{C}_{\mathcal{E}}$ and $|c_j \bmod P| < 2^{\rho'+l_Q}$,

$$|f(c_1 \bmod p, \ldots, c_t \bmod p)| < 2^{\eta-4} < \frac{p}{8} \qquad \text{by ().}$$

So, $c' \bmod p = f(c_1 \bmod p, \ldots, c_t \bmod p)$.
Hence

$$\begin{aligned} (c' \bmod p) \bmod Q &= f(c_1 \bmod p, \ldots, c_t \bmod p) \bmod Q \\ &= f((c_1 \bmod p) \bmod Q, \ldots, (c_t \bmod p) \bmod Q) \bmod Q \\ &= f((m_1, \ldots, m_t) \bmod Q \\ &= m' \bmod Q \end{aligned}$$

It follow that the shceme given 3.2.2 is **correct** for a permitted circuit $\mathcal{C}_{\mathcal{E}}$.

Now, we consider a noise of a result by the opreartions. Actually, a noise of Add(pk, c_1, c_2) will incease at mose 1-bit. But, the bit length of a noise of Mul(pk, c_1, c_2) may grow larger than $2(\rho' + l_Q)$. So, we will keep an eye on the multiplicative depth of permiited circuit rather than the additive depth.

3.3 Fully homomorphic encryption without Bootstrapping (BGV)

Zvika Brakerski, Craig Gentry and Vinod Vaikuntanathan suggest the way of constructing leveled fully homomorphic encryption schemes[7]. It's based on the general learing with error(GLWE) problems.

Definition 3.6. (GLWE)
For security parameter λ, let $n = n(\lambda)$ be an integer dimension, let $f(x) = x^d + 1$ *where* $d = d(\lambda)$ is a power of 2, let $q = q(\lambda) \geq 2$ be a prime integer, let $R = \mathbb{Z}[x]/(f(x))$ and $R_q = R/qR$, and let $\chi = \chi(\lambda)$ be a distribution over R.
The $\mathrm{GLWE}_{n,f,g,\chi}$ problem is to distinguish the following two distributions :
In the first distribution, one samples (a_i, b_i) uniformly from R_q^{n+1}.
In the second distribution, one first draws $s \leftarrow R_q^n$ uniformly and then samples $(a_i, b_i) \in R_q^{n+1}$ by sampling $a_i \leftarrow R_q^n$ uniformly, $e_i \leftarrow \chi$, and setting $b_i = \langle a_i, s\rangle + e_i$
The $\mathrm{GLWE}_{n,f,g,\chi}$ assumption is that the $\mathrm{GLWE}_{n,f,g,\chi}$ problem is infeasible.

Remark. *The GLWE assumption implies that the distribution* $\{(a_i, \langle a_i, s\rangle + te_i)\}$ *is computational indistinguishable from uniform for any* t *relatiely prime to* q. *This fact will be convenint for encryption.*

3.3.1 Notations

We use a ring R, here either $R = \mathbb{Z}$ or $R = \mathbb{Z}[x]/(x^d + 1)$.
For $v \in R^n$, $v[i]$ refers to the i-th coefficient of v. And $\langle u, v\rangle = \sum_{i=1}^{n} u[i] \cdot v[i]$ for $u, v \in R^n$.
If R is a polynomial ring, then $\|r\|$ for $r \in R$ is the Euclidean norm of r's coefficeint vector. $\gamma(R) = \{\|a \cdot b\|/\|a\|\|b\| : a, b \in R\}$.

For an integer q, $R_q = R/qR$. And $[a]_q = a \bmod q$ into range $(-q/2, q/2)$.
For a real number z, $\lceil z \rceil$ the rounding of z up, that is the unique integers in the $[z, z+1)$. $\lfloor z \rfloor$ the rounding of z down, the unique integer in $(z-1, z]$.
Note that $\lceil z \rceil = 1 + \lfloor z \rfloor$.

3.3.2 Construction (with no homomorphic opertaions)

Let λ be the security parameter.
And let $q = q(\lambda)$ be an odd modulus, $\chi = \chi(\lambda)$ a noise distribution. $R = R(\lambda)$. Assume that the plaintext space is $R_2 = R/2R$, though larger plaintext space are certainly possible.

E.Setup$(1^\lambda, 1^\mu, b)$

Use the bit $b \in \{0, 1\}$ to determine whether we are setting parameters. Choose a μ-bit modulus q and choose the other parameters $d = d(\lambda, \mu, b),\ n = n(\lambda, \mu, b),\ N = \lceil (2n+1)\ log\ q \rceil, \chi = \chi(\lambda, \mu, b)$ aprropriately to ensure that the scheme is based on a GLWE instance that acheieves 2^λ security against known attacks.

Let $R = \mathbb{Z}[x]/(x^d + 1)$ and let $params = (q, d, n, N, \chi)$.

E.SecretKeyGen$(params)$

Draw $\hat{s} \leftarrow \chi^n$. Set $sk = \mathbf{s}\ \leftarrow (1,\ \hat{s}[1],\ \ldots,\ \hat{s}[\mathrm{n}]) \in {R_q}^{n+1}$.

E.PublicKeyGen$(params,\ sk)$

Takes as input a secret key $sk = \mathbf{s} = (1,\ \hat{s})$ with $\hat{s}[0]$=1 and $\hat{s} \in {\mathrm{R}_q}^{n+1}$ and the *params*. Generate matrix $\hat{\mathbf{A}} \leftarrow {\mathrm{R}_q}^{N \times n}$ uniformly and a vector $\mathbf{e} \leftarrow \chi^N$ and set $\mathbf{b} \leftarrow \hat{\mathbf{A}}\hat{\mathbf{s}} + 2\mathbf{e}$. Set $\mathbf{A}$ to be the $(n+1)$-column matrix consisting of $\mathbf{b}$ followed by the n columns of $-\hat{\mathbf{A}}$. (Observe : $\mathbf{A}{\cdot}\mathbf{s} = 2\mathbf{e}$).

Set the public key $pk = \mathbf{A}$.

E.Enc$(params,\ pk,\ m)$

To encrypt a message $\mathrm{m} \in R_2$, set $\mathbf{m} \leftarrow (\mathrm{m},0,\ldots,0) \in {R_q}^{n+1}$.

Sample $\mathbf{r} \leftarrow {R_2}^N$ and output the ciphertext $\mathbf{c} \leftarrow \mathbf{m} + \mathbf{A}^{\mathrm{T}}\mathbf{r} \in {R_q}^{n+1}$.

E.Dec$(params,\ sk,\ \mathbf{c})$ <u>Output</u> : $m \leftarrow [\langle \mathbf{c},\ \mathbf{s} \rangle]_q$ mod 2.

Exampe for algorithm

Let $n = 1$, $q = 3$ and $N = \lceil 3 \cdot \log 3 \rceil = 4$.

$\hat{s} \leftarrow \chi^1$, the secret key $sk = \mathbf{s} = \begin{pmatrix} 1 \\ \hat{s} \end{pmatrix}$

Generate a matrix $\widehat{\mathbf{A}} = (a_i) \leftarrow R_5^{4\times 1}$ uniformly.
a vector $\mathbf{e} = (e_i) \leftarrow \chi^4 \quad$ for $i = 1, 2, 3, 4$

Set $\mathbf{b} = (b_i) = (a_i \cdot s + 2e_i) \;\leftarrow\; \widehat{\mathbf{A}} \cdot \hat{s} + 2\mathbf{e}$
Set a martix $\mathbf{A} = (\mathbf{b} \mid -\widehat{\mathbf{A}}) = (\bar{a}_{ij}) \quad$ where $\bar{a}_{i1} = b_i$ and $\bar{a}_{i2} = -a_i$
The public key pk $= \mathbf{A}$.

Encrypt :

$m \in R_2$, set $\mathbf{m} = (m, 0) \in R_3^2$ and sample $\mathbf{r} \leftarrow R_2^4$.

Ouput ciphertext $\mathbf{c} \leftarrow \mathbf{m} + \mathbf{A}^{\mathbf{T}}\mathbf{r} \in R_3^2$

$$\mathbf{c} = \begin{pmatrix} m + \sum_{i=1}^{4}(a_i\hat{s} + 2e_i) \cdot r_i \\ -\sum_{i=1}^{4} a_i r_i \end{pmatrix}$$

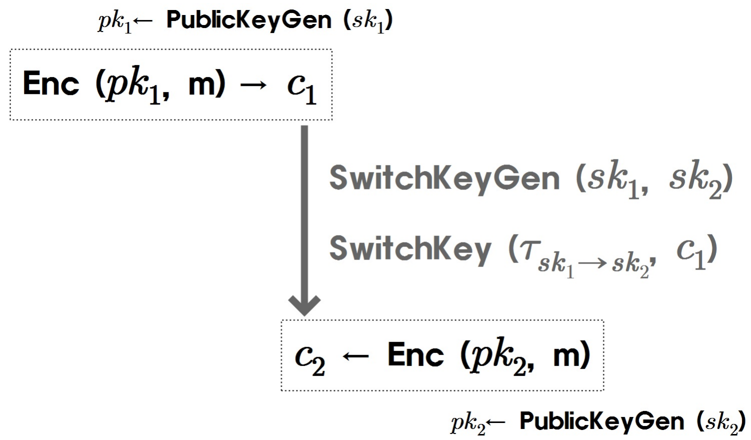
Decrypt :

$\langle \mathbf{c},\ \mathbf{s} \rangle = m + 2\sum_{i=1}^{4} e_i r_i$

Output $m \leftarrow [\langle \mathbf{c},\ \mathbf{s} \rangle]_5 \bmod 2$

3.3.3 Key Switching

Brakerski and Vaikuntanathan's Key switching produres may used to reduce the dimension of the ciphertext that transform a ciphertext c_1(decryptable under secret key s_1) to a different ciphertext c_2(encrypts the same message of c_1, decryptable under secret key s_2).



BitDecomp$(x \in R_q^n, q)$: decomposes **x**into its bit representaion.

$$x = \sum_{j=0}^{\lfloor \log\ q \rfloor} 2^j \cdot u_j \quad \forall u_j \in R_2^n$$

$$\underline{\text{Output}} \quad (u_0,\ u_1,\ \ldots,\ u_{\lfloor \log\ q \rfloor}) \in R_2^{n \cdot \lceil \log\ q \rceil}$$

Powerof2$(x \in R_q^n, q)$: $\underline{\text{Output}}$ $(x,\ 2 \cdot x,\ \ldots,\ 2^{\lfloor \log\ q \rfloor} \cdot x) \in R_q^{n \cdot \lceil \log\ q \rceil}$

So, if c and s are vectors of equal length,
then we have $\langle c, s \rangle$ mod $q = \langle$BitDecomp(c, q), Powerof2(s, q)$\rangle$.

Key switching consist of two procedures :

Step1. **SwitchKeyGen**$(s_1 \in R_q^{n_1},\ s_2 \in R_q^{n_2})$

1. Run $A \leftarrow$ E.PublicKeyGen$(s_2,\ N)$ for $N = n_1 \cdot \lceil \log\ q \rceil$

2. Set $B \leftarrow A + \text{Powerof2}(s_1)$ (Add $\text{Powersof2}(s_1) \in R_q^N$ to A's first column Ouput $\tau_{s_1 \to s_2} = B$

Step2. $\textbf{SwitchKey}(\tau_{s_1 \to s_2},\, c_1)$: Output $c_2 = \text{BitDecomp}(c_1)^T \cdot B \in R_q^{n_2}$

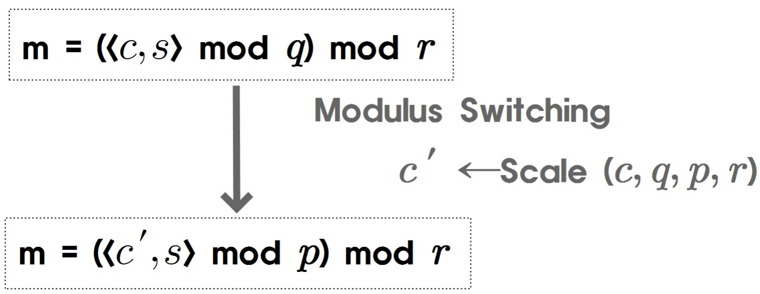
Note that the matrxi A consists of encryption of 0 unsder the key s_2.
And the marix B consists of encryptions of pieces of s_1 under the key s_2.

By Key Switching procedure, we have

$$\langle c_2, s_2 \rangle = 2\langle \text{BitDecomp}(c_1),\, e_2 \rangle + \langle c_1, s_1 \rangle \mod q.$$

3.3.4 Modulus Switching

We will call $[\langle c, s \rangle]_q$ the noise associated to ciphertext c under key s.
The modulus switching technique can to manage the noise in FHE.
The evaluator who does not know the secret key, can reduce the magnitude of the noise without knowing the secret key. In brief, it can transform a ciphertext c modulo q into a a diffrent ciphertext modulo p while preserving correctness. Furthemore, if $p << q$, then $\|[\langle c, s \rangle]_p\| < \|[\langle c, s \rangle]_q\|$.



Modulus Switching is the follwing steps :

Let L be a depth of a circiut for evaluate.

1. Start a lage modulus q_L and the noise of size $\eta << q_L$.

2. After first multplication, the noise grows to size η^2.

3. Modulus switching to $q_{L-q} \approx q_L/\eta$. the noise reduced to $\eta^2/\eta \approx \eta$.
4. After next multiplication, noise again grows to η^2.
5. Switch to $q_{L-2} \approx q_{L-1}/\eta$ to reduce the noise to η.
6. Setting $q_{i-1} \approx q_i/\eta$.
7. Untill the last modulus just barely satisfies $q_0 > \eta$.

Definition 3.7. (Scale)
For integer vector x and integer $m < p < q$,
$\widehat{\mathbf{x}} \leftarrow \mathbf{Scale}(\mathbf{x},\ \mathbf{q},\ \mathbf{p},\ \mathbf{r})$ is defined as the R-vector closest to $(p/q) \cdot x$ such that $\hat{x} = x \bmod r$.

Definition 3.8. (l_1^R-norm)
The l_1^R-norm is defined as $\boldsymbol{l_1^R}(\mathbf{s}) := \sum_i \|s[i]\|$ for $s \in R^n$

Lemma 3.1. *Let d be the degree of the ring.*
And let $r < p < q$ be positive integer satisfying $q = p = 1 \bmod r$.
Suppose that $c \in R^n$, $\widehat{\mathbf{c}} \leftarrow \boldsymbol{Scale}(\mathbf{c},\ \mathbf{q},\ \mathbf{p},\ \mathbf{r})$ and
$\|[\langle c, s\rangle]_q\| < q/2 - (q/p) \cdot (r/2) \cdot \sqrt{d} \cdot \gamma(R) \cdot l_1^R(s)$ for any $s \in R^n$.

Then we have

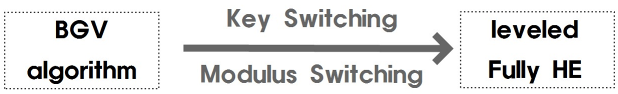
$$[\langle \hat{c}, s\rangle]_p = [\langle c, s\rangle]_q \bmod r \quad \textit{and}$$
$$\|[\langle \hat{c}, s\rangle]_p\| < (p/q) \cdot \|[\langle c, s\rangle]_q\| + (r/2) \cdot \sqrt{d} \cdot \gamma(R) \cdot l_1^R(s)$$

Corollary 3.2. *Let p and q be tow odd moduli. Suppose* $\mathbf{c}$ *is an ecryption of bit m under key* $\mathbf{s}$ *for modulus q. i.e, $m = [\langle \mathbf{c},\ \mathbf{s}\rangle] \bmod r$.*
Suppose that $\mathbf{s}$ *is a farily short key and the noise of $[\langle \mathbf{c},\ \mathbf{s}\rangle]$ has small magnitude - precisely, assume that $\|[\langle c, s\rangle]_q\| < q/2 - (q/p) \cdot (r/2) \cdot \sqrt{d} \cdot \gamma(R) \cdot l_1^R(s)$.*
Then $\widehat{\mathbf{c}} \leftarrow \boldsymbol{Scale}(\mathbf{c},\ \mathbf{q},\ \mathbf{p},\ \mathbf{r})$ is an encryption of bit m under key $\mathbf{s}$ *for modulus p. i.e, $m = [\langle \hat{c}, s\rangle]_p \bmod r$*

$$\|[\langle \hat{c}, s\rangle]_p\| < (p/q) \cdot \|[\langle c, s\rangle]_q\| + (r/2) \cdot \sqrt{d} \cdot \gamma(R) \cdot l_1^R(s)$$

3.3.5 (Leveled) FHE base on GLWE without Bootstrapping

We will use a parameter L indicating the number of levels of arithmetic circuit that we want our scheme to be capable of evaluating. And a parameter d indicating the degree of the polynomials to be evaluated.



FHE.Setup$(1^\lambda, 1^\mu, b)$

Take as input the security parameter, a number of level L, and a bit b$\in \{0,1\}$. Let $\mu = \mu(\lambda, L, b) = \theta(\log\lambda + \log L)$. For $j = L$ (input level of circuit) to be 0 (output level), run $params_j \leftarrow$ **E.Setup**$(1^\lambda, 1^{(j+1)\mu}, b)$ to obtain a ladder of decresing moduli from $q_L((L+1)\cdot\mu$ bit) down to q_0 (μ bits). For $j = L-1$ to 0, replace the value of d_j in $params_j$ with $d = d_L$ and the distribution χ_j with $\chi = \chi_L$. (That is, the ring demension and noise distribution do not depend on the circuit level, but the vector dimension n_j still might.)

FHE.KeyGen$(\{parmas_j\})$

For $j = L$ down to 0, do the following :

1. Run $s_j \leftarrow$ E.SecretKeyGen$(params_j)$
   and $A_j \leftarrow$ E.PublicKeyGen$(params_j,\ s_j)$.
2. Set $\hat{s}_j \leftarrow s_j \otimes s_j \in R_{q_j}^{\binom{n_j+1}{2}}$. That is $\hat{s}_j$ is a tensoring of s_j with itself whose coefficients are each the product fo two cefficients fo s_j in R_{q_j}.
3. Set $\bar{s}_j \leftarrow$ BitDecomp$(\hat{s}_j,\ q_j)$.
4. Run $\tau_{\bar{s}_{j+1}\rightarrow s_j} \leftarrow$ SwitchKeyGen$(\bar{s}_j,\ s_{j-1})$. Omit this step when $j = L$.

FHE.Enc$(params,\ pk,\ m)$: Take a message in R_2. Run **E.Enc**$(A_L,\ m)$.

FHE.Dec(*params*, *sk*, *c*)

Suppose the ciphertext in under key s_j. Run **E.Dec**(s_j, c). The ciphertext could be augmented with an index indicationg which level it belong to.

FHE.Add(*params*, c_1, c_2)

Take two ciphertexts encrypted under the same s_j. If they are not initially, use FHE.Refresh (blow) to make it so. Set $\mathbf{c_3} \leftarrow \mathbf{c_1} + \mathbf{c_2} \bmod \mathbf{q_j}$. Interpret c_3 as a ciphertext under $\hat{s}_j$ ($\hat{s}_j$'s coefficients include all of s_j's since $\hat{s}_j = s_j \otimes s_j$ and s_j's first coefficient is 1)

<u>output</u> : $c_4 \leftarrow$ FHE.Refresh(c_3, $\tau_{\bar{s}_j \to s_{j-1}}$, q_j, q_{j-1})

FHE.Mult(*params*, c_1, c_2)

Take two ciphertext encrypted under the same s_j. If they are not inially, same as FHE.Add. First, multiply : the new ciphertext, under the secret key $\hat{s}_j = s_j \otimes s_j$, is the coefficient vector c_3 of the linear equation $L^{long}_{c_1,c_2}(x \otimes x)$.

<u>output</u> : $c_4 \leftarrow$ FHE.Refresh(c_3, $\tau_{\bar{s}_j \to s_{j-1}}$, q_j, q_{j-1})

FHE.Refresh(c, $\tau_{\bar{s}_j \to s_{j-1}}$, q_j, q_{j-1})

Take a ciphertext encrypted under $\hat{s}_j$, the auxiliary information $\tau_{\bar{s}_j \to s_{j-1}}$ to facilitate key switching, and the current and next moduli q_j and q_{j-1}. Do the following :

1. Expand : Set $c_1 \leftarrow$ Powersof2(c, q_j).
2. Switch Moduli : Set $c_2 \leftarrow$ Scale(c_1, q_j, q_{j-1}, 2), a ciphertext under the key $\bar{s}_j$ for modulus q_{j-1}.
3. Switch Keys : Output $c_3 \leftarrow$ SwitchKey($\tau_{\bar{s}_j \to s_{j-1}}$, c_2, q_{j-1}), a ciphertext under the key s_{j-1} for modulus q_{j-1}.

4 Computations using NTL

NTL is a C++ library for doing number theory. NTL supports arbitrary length integer and arbitrary precision floating point arithmetic, finite fields, vectors, matrices, polynomials, lattice basis reduction and basic linear algebra. It is written and maintained by Victor Shoup[10].

4.1 Computation method

We use the algorithm which is CRT-based homomorphic encryption over the intergers. From Theorem 2.8, $C(t,x)$ is function over $\mathbb{R}$. For using the algorithm, we consider *Integeration* for float type number. If $x \in \mathbb{R}$ is not integer, then $10^k x$ is integer for some k.

Integeration :
(i) $x \in \mathbb{Q}$ is not integer $\Longrightarrow$ $x = (x_0,\ x_1) = (10^k x_0,\ 10^k x_1)$
(ii) $x \in \mathbb{Q}$ is integer $\Longrightarrow$ $x = (x_0,\ x_1) = (x,\ 1)$

We define operations : For $x = (x_0,\ x_1)$, $y = (y_0,\ y_1)$,

$$\begin{aligned} x + y \bmod N &= (x_0 y_1 + x_1 y_0 \bmod N,\ x_1 y_1 \bmod N) \\ x \cdot y \bmod N &= (x_0 y_0 \bmod N,\ x_1 y_1 \bmod N) \\ x/y \bmod N &= (x_0 y_1 \bmod N,\ x_1 y_0 \bmod N) \\ k \cdot x \bmod N &= (k_0 x_0 \bmod N,\ k_1 x_1 \bmod N) \quad \text{for } k = \frac{k_0}{k_1} \in \mathbb{Q} \end{aligned}$$

In this sense, we consider that $c_i \leftarrow \mathrm{Enc}(pk, m_i) = \mathrm{Enc}(m_i)$ for $i = 0, 1$ where $m = (m_0,\ m_1) \in \mathbb{Z} \times \mathbb{Z}$ and $c = (c_0,\ c_1)$. Then we have

$$\begin{aligned} Enc(m) + y \bmod N &= \big(y_1 c_0 + y_0 x_1 \bmod N, \quad y_1 c_1 \bmod N\big) \\ Enc(m) \cdot y \bmod N &= \big(c_0 y_0 \bmod N, \quad c_1 y_1 \bmod N\big) \end{aligned}$$

At (12) and (13), d_1 and d_2 are major parst of our computation. Suppose that the volutality σ is a secret data. That is, we will use $Enc(pk, \sigma)$ instead of σ.

Then

$$
\begin{aligned}
d_1 =& d_1(x, K, r, Enc(\sigma), T, \sqrt{T}) \\
=& [ln(\frac{x}{K} + (r + \frac{1}{2}Enc(\sigma) \cdot Enc(\sigma)) \cdot T]_N \ / \ [Enc(\sigma) \cdot \sqrt{T}]_N \bmod N
\end{aligned}
$$

$$
\begin{aligned}
d_2 =& d_2(x, K, r, Enc(\sigma), T, \sqrt{T}) \\
=& [ln(\frac{x}{K} + (r - \frac{1}{2}Enc(\sigma) \cdot Enc(\sigma)) \cdot T]_N \ / \ [Enc(\sigma) \cdot \sqrt{T}]_N \bmod N
\end{aligned}
$$

Now, Let's think about the function $N(z)$. The polynomial approximation in remark has number of large bit. Then the decrypt may not work, because of our scheme is still somewhat homomorphic encrytion sheme. So, we suggest adopting the Taylor series for e^x.

Assume $z > 0$.

$$
\begin{aligned}
N(z) &= \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{z} e^{-\frac{y^2}{2}} dy \\
&= \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{0} e^{-\frac{y^2}{2}} dy + \frac{1}{\sqrt{2\pi}} \int_{0}^{z} e^{-\frac{y^2}{2}} dy \\
&= 0.5 \ + \ \frac{1}{\sqrt{2\pi}} \int_{0}^{z} \sum_{k=0}^{\infty} \frac{(-1)^k}{2^k k!} y^{2k} dy \\
&= 0.5 \ + \ \frac{1}{\sqrt{2\pi}} (z - \frac{z^3}{6} + \frac{z^5}{40} - \frac{z^7}{336} + \frac{z^9}{3456} - \cdots)
\end{aligned}
$$

Note that for $0 < x < 1$, we measure the error of

$$
f(x) = e^{-\frac{x^2}{2}} = \sum_{k=0}^{\infty} \frac{(-1)^k}{2^k k!} x^{2k} = 1 - \frac{x^2}{2} + \frac{x^4}{8} - \frac{x^6}{48} + \frac{x^8}{384} - \cdots
$$

By the taylor theorem,

$$
\begin{aligned}
&|e^{-\frac{x^2}{2}} - 1| \le \max\{|f'(t)| : t \in [0, x]\} \cdot |x| \le \frac{1}{\sqrt{e}} \cdot |x| \approx 0.6065 \cdot |x| \\
&|e^{-\frac{x^2}{2}} - (1 - \frac{1}{2}x^2)| \le \max\{|f''(t)| : t \in [0, x]\} \cdot \frac{|x|^2}{2!} \le \frac{|x|^2}{2}
\end{aligned}
$$

4.2 Performance

We compute example in chapter 2.

Companents of call option (no encrypted original data)				
d_1	d_2	$N(d_1)$	$N(d_2)$	call price
0.7693	0.6278	0.7791	0.7349	4.7599

Greeks of call option (no encrypted original data)				
$\triangle_C$	Γ	θ_C	ν	ρ_C
0.7791	0.0499	-4.5590	8.8134	13.9820

For $z > 0$, we use $N(z) \approx \frac{1}{2} + \frac{1}{\sqrt{2\pi}} \int_0^z \sum_{k=0}^{\infty} \frac{(-1)^k}{2^k k!} y^{2k} dy$

$N_m(z) := \frac{1}{2} + \frac{1}{\sqrt{2\pi}} \sum_{k=0}^{m} \frac{(-1)^k y^{2k+1}}{2^k k!(2k+1)}$ and use $N_0(d_1)$, $N_0(d_2)$.

Results of computations using encrypted volatility					
decimal point	d_1	d_2	$N(d_1)$	$N(d_2)$	call price
10^{-2}	0.7200	0.5800	0.7808	0.7262	5.1689
10^{-3}	0.7606	0.6194	0.8027	0.7465	5.3098
10^{-4}	0.7687	0.6273	0.8066	0.7502	5.3329

Errors of call option's results					
decimal point	d_1	d_2	$N(d_1)$	$N(d_2)$	call price
10^{-2}	0.0493	0.0478	0.0017	0.0087	0.4090
10^{-3}	0.0087	0.0064	0.0236	0.0116	0.5499
10^{-4}	0.0006	0.0005	0.0275	0.0153	0.5730

Define $E_N = \sum_{k=0}^{N} \frac{(-1)^k}{2^k k!} x^{2k}$. We use E_1 that computing for $e^{-\frac{{d_1}^2}{2}}$.

Results of computations using encrypted volatility					
decimal point	$\triangle_C$	Γ	θ_C	ν	ρ_C
10^{-2}	0.7808	0.0572	-4.7785	9.8579	13.8123
10^{-3}	0.8027	0.0494	-4.5824	8.4182	14.2018

Errors of greek's results					
decimal point	$\triangle_C$	Γ	θ_C	ν	ρ_C
10^{-2}	0.0017	0.0073	0.2195	1.0445	0.1697
10^{-3}	0.0236	0.0005	0.0234	0.3952	0.2198

- **Performance Time** (second)

(i) decimal point : 10^{-2}

	Call	$\triangle_C$	Γ	θ_C	ν	ρ_C
1st	29.9997	30.1160	48.2636	51.1944	40.7832	36.2859
2nd	29.8878	30.8673	48.1047	51.0545	40.7643	36.3551
3rd	29.9369	30.8317	48.1274	51.0873	40.7764	36.3360

(ii) decimal point : 10^{-3}

	Call	$\triangle_C$	Γ	θ_C	ν	ρ_C
1st	29.9645	30.8960	48.2613	51.2568	40.4680	36.5233
2nd	29.8907	30.8747	48.3308	51.1356	40.6355	36.4254
3rd	29.9223	30.9309	48.2908	51.4494	40.4987	36.4463

```
//Copyright 2014. Hyung Tae Lee, Min Woo Kwon.

#include<iostream>
#include <math.h>
#include <stdio.h>
#include <NTL/ZZ.h>
#include <NTL/RR.h>
#include <NTL/vector.h>
#include <time.h>
#include <vector>
#include <fstream>

NTL_CLIENT
```

```
#define pi 4.0*atan(1.0)
#define u64 unsigned long
#define u32 unsigned int
#define u16 unsigned short
#define u8 unsigned char
#define NumComp 2
#define NumTest 1
#define NumPrime 2
#define lambda 20
#define rho 40
#define eta 512
#define gamma_eta 40958
#define gamma 2097152
#define log_Q 64

ZZ Mod_Inv(ZZ b, ZZ p);
void Encrypt(ZZ* X, ZZ* Y, ZZ(*Z)[2], ZZ* W, ZZ a, ZZ b);
void Decrypt(ZZ* x, ZZ* y, ZZ z, ZZ w);
void integeration(double x, int n, ZZ* output);
void abbreviate(ZZ numerator, ZZ denominator, ZZ* output);
void abbreviate3(ZZ* A, ZZ* B, ZZ* C, ZZ* output1, ZZ* output2, ZZ* output3);
void add(ZZ* x, ZZ* y, ZZ N, ZZ* output);
void subtract(ZZ* x, ZZ* y, ZZ N, ZZ* output);
void mult(ZZ* x, ZZ* y, ZZ N, ZZ* output);
void division(ZZ* x, ZZ* y, ZZ N, ZZ* output);
void reciprocal(ZZ*x, ZZ*output);
void e(ZZ* x, ZZ N, ZZ* output);
void NN(ZZ* x, ZZ N,int f, ZZ* output);
ZZ FindGCD(ZZ x, ZZ y);
ZZ FindLCM(ZZ x, ZZ y);
using namespace std;

int main(void) {
double TimeTemp; TimeTemp = GetTime();
int i, j; int f = 3; double S = 42; double K = 40;
double r = 0.1; double T = 0.5; double v = 0.2;
ZZ *prime; prime = new ZZ[NumPrime]; ZZ *cofactor; cofactor = new ZZ[NumPrime];
ZZ *inverse; inverse = new ZZ[NumPrime]; ZZ *CRT_prod; CRT_prod = new ZZ[NumPrime];
ZZ *sum; sum = new ZZ[NumPrime]; ZZ intermediate[NumComp][NumPrime];
ZZ *decryption; decryption = new ZZ[NumComp]; ZZ ciphertext[NumComp];
ZZ encV[NumComp]; ZZ *V; V = new ZZ[NumComp]; ZZ N;
ZZ Q = power2_ZZ(log_Q); ZZ two_rho; int flag = 0; N = to_ZZ("1");
```

```
//KeyGeneration
do{ do{ do{
prime[0] = RandomBits_ZZ(eta);
} while (NumBits(prime[0]) != eta);
for (i = 0; i<gamma_eta; i++){
prime[0] = (prime[0] « eta); prime[0] += RandomBits_ZZ(eta);
} RandomPrime(prime[1], eta, 30);
} while (GCD(prime[0], prime[1]) != 1);
N = prime[0] * prime[1];
} while (NumBits(N) != gamma);

for (j = 0; j<NumPrime; j++) {
cofactor[j] = N / prime[j]; inverse[j] = Mod_Inv(cofactor[j], prime[j]);
CRT_prod[j] = MulMod(cofactor[j], inverse[j], N);}
two_rho = power2_ZZ(rho); int numbits_p0 = NumBits(prime[0]);
for (i = 0; i<NumComp; i++){
do{ intermediate[i][0] = RandomBits_ZZ(numbits_p0 + 1);
} while ((intermediate[i][0] == 0) || (intermediate[i][0]>2 * prime[0]));
intermediate[i][0] -= prime[0];
do{ intermediate[i][1] = RandomBits_ZZ(rho + 1);
} while (intermediate[i][1] == 0);
intermediate[i][1] -= two_rho; }

//Encryption
integeration(v, f, V); Encrypt(V, encV, intermediate, CRT_prod, Q, N);

//computation of d1,d2, N1, N2
double temp; temp = log(S / K) / sqrt(T); ZZ A[2]; integeration(temp, f, A);
temp = r*sqrt(T); ZZ B[2]; integeration(temp, f, B);
temp = sqrt(T) / 2; ZZ C[2]; integeration(temp, f, C);
abbreviate3(A, B, C, A, B, C); ZZ d1[2]; ZZ d2[2]; ZZ N1[2]; ZZ N2[2];
d1[0] = ((A[0] + B[0]) * encV[1] * encV[1] + C[0] * encV[0] * encV[0]) % N;
d1[1] = (A[1] * encV[0] * encV[1]) % N;
d2[0] = ((A[0] + B[0]) * encV[1] * encV[1] - C[0] * encV[0] * encV[0]) % N;
d2[1] = (A[1] * encV[0] * encV[1]) % N;
NN(d1, N, f, N1); NN(d2, N, f, N2);
```

```
//computation of call option
ZZ c1[2]; integeration(S, f, c1); abbreviate(c1[0], c1[1], c1);
mult(c1, N1, N, c1); double c_2 = K*exp(-(r*T)); cout « ”c2 = ” « c_2 « endl;
ZZ c2[2]; integeration(c_2, f, c2); abbreviate(c2[0], c2[1], c2);
mult(c2, N2, N, c2); ZZ call; call = c1 - c2;

//Decrption
Decrypt(call, decryption, prime[1], Q);
cout « ”call price = ” « to_RR(decryption[0]) / to_RR(decryption[1]) « endl;

double result_time; result_time = GetTime()-TimeTemp;
return 0;
}

void integeration(double x, int n, ZZ* output) {
long temp[2]; temp[0] = (long)(x*pow(10, n)); temp[1] = (long)pow(10, n);
output[0] = to_ZZ(temp[0]); output[1] = to_ZZ(temp[1]); }

void abbreviate(ZZ numerator, ZZ denominator, ZZ* output) {
ZZ min; if (numerator > denominator) min = denominator;
else min = numerator;
for (ZZ i = min; i > 0; i–) {
if ((denominator%i == 0) && (numerator%i == 0)) {
output[0] = (numerator / i); output[1] = (denominator / i);
break; } } }

void add(ZZ* x, ZZ* y, ZZ N, ZZ* output) {
output[0] = ((x[0] * y[1])+ (x[1] * y[0]))%N; output[1] = (x[1] * y[1]) % N; }

void subtract(ZZ* x, ZZ* y, ZZ N, ZZ* output) {
output[0] = ((x[0] * y[1]) - (x[1] * y[0])) % N; output[1] = (x[1] * y[1]) % N; }

void mult(ZZ* x, ZZ* y, ZZ N, ZZ* output) {
output[0] = (x[0] * y[0]) % N; output[1] = (x[1] * y[1]) % N; }
```

```
void division(ZZ* x, ZZ* y, ZZ N, ZZ* output) {
output[0] = (x[0] * y[1]) % N; output[1] = (x[1] * y[0]) % N; }

void reciprocal(ZZ*x, ZZ*output) { output[0] = x[1]; output[1] = x[0]; }

void e(ZZ* x, ZZ N, ZZ* output) {
ZZ one[2]; one[0] = to_ZZ(1); one[1] = to_ZZ(1);
ZZ half[2]; half[0] = to_ZZ(1); half[1] = to_ZZ(2);
ZZ temp[2]; mult(x, x, N, temp); mult(half, temp, N, temp);
subtract(one, temp, N, temp); output[0] = temp[0]; output[1] = temp[1]; }

void NN(ZZ*x, ZZ N, int f, ZZ* output) {
ZZ half[2]; half[0] = to_ZZ(1); half[1] = to_ZZ(2);
double a = 1 / (sqrt(2 * pi)); ZZ A[2]; integeration(a, f, A);
mult(A, x, N, output); add(half, output, N, output); }

ZZ Mod_Inv(ZZ b, ZZ p) {
ZZ a, q, r, t0, t1, t2; t0 = 0; t1 = 1; a = p; q = a / b; r = a%b;
t2 = t0 - t1*q; while (r != 0){
a = b; b = r; q = a / b; r = a%b;
t0 = t1; t1 = t2; t2 = t0 - t1*q; }
if (t1<0) t1 += p; return t1; }

void Encrypt(ZZ* X, ZZ* Y, ZZ(*Z)[2], ZZ* W, ZZ a, ZZ b) {
ZZ enc_intermediate[NumComp][NumPrime];
for (int i = 0; i < NumComp; i++) {
enc_intermediate[i][0] = Z[i][0]; enc_intermediate[i][1] = Z[i][1];
enc_intermediate[i][1] = enc_intermediate[i][1] * a + X[i];
for (int j = 0; j<NumPrime; j++) {
Y[i] += MulMod((enc_intermediate[i][j] % b), W[j], b); Y[i] %= b; }
if (Y[i] >(b / 2)){ Y[i] -= b; } } }

void Decrypt(ZZ* x, ZZ* y, ZZ z, ZZ w) {
for (int i = 0; i<NumComp; i++) {
y[i] = (x[i]) % z; if (y[i]>(z / 2)){ y[i] -= z; }
```

```
y[i] = y[i] % w; } }

void abbreviate3(ZZ* A, ZZ* B, ZZ* C, ZZ* output1, ZZ* output2, ZZ* output3) {
ZZ m1, m2, m3, g1, g; m1 = FindGCD(A[0], A[1]);
m2 = FindGCD(B[0], B[1]); m3 = FindGCD(C[0], C[1]);
g1 = FindGCD(m1, m2); g = FindGCD(m3, g1);
output1[0] = A[0] / g; output1[1] = A[1] / g; output2[0] = B[0] / g; output2[1] = B[1] / g;
output3[0] = C[0] / g; output3[1] = C[1] / g; }

ZZ FindGCD(ZZ x, ZZ y) {
ZZ min; ZZ z; if (x >= y) min = y; else min = x;
for (ZZ i = min; i > 0; i–) {
if ((x%i == 0) && (y%i == 0)) { z = i; break; } else z = 1; } return z; }

ZZ FindLCM(ZZ x, ZZ y) { ZZ g = FindGCD(x, y); return g*(x / g)*(y / g); }
```

4.3 Discussion

Decimal point changes from 10^{-2} to 10^{-3}, then error of $d_1, d_2, \Gamma, \theta_c, \nu$ decreasing. But, error of $N(d_1), N(d_2), \triangle_C, \rho_c$ incresing, since a limitation of $N_0(x)$. So, there are so many assignments accumulated that we need to work on.

- How to computation about $N_m(z) = \frac{1}{2} + \frac{1}{\sqrt{2\pi}} \sum_{k=0}^{m} \frac{(-1)^k y^{2k+1}}{2^k k!(2k+1)}$ for $m = 1, 2, \cdots$?
  And $E_N = \sum_{k=0}^{N} \frac{(-1)^k}{2^k k!} x^{2k}$ for $N = 2, 3, \cdots$?

- For any k, can we handle decimal point 10^{-k}?
  i.e, How do we deal with large bit integers?

References

[1] Jung-Hun Kim. *Financial Mathematics*. Kyo Woo Sa, 2007.

[2] J.Hull. *Options, Futures and Other derivative securitues*. Fifth edition, 2002.

[3] Jae-Ho Jo, Jong-Won Park, Kyu-Sung Jo. *Futures · Options · Swaps*. Dasanbooks, 2009.

[4] Lawrence C. Evans. *Partial Differential Equations*. American Mathematical Society, 1998.

[5] Craig Gentry. *Fully Homomorphic Encryption Using Ideal Lattices*. In STOC, pages 169-178, 2009.

[6] Craig Gentry. *A Fully Homomorphic Encryption Scheme*. PhD thesis, Stanford University, 2009. crypto.stanford.edu/craig.

[7] Z. Brakerski, C. Gentry, and V. Vaikuntanathan. *Fully homomorphic encryption without bootstrapping*. In Innovations in Theoretical Computer Science (ITCS'12), 2012. Available at http://eprint.iacr.org/2011/277.

[8] Jin-su Kim , Moon-Sung Lee, Aaram Yun and Jung-Hee Cheon. *CRT-based Fully homomorphic Encryption over the integers*. SNU, 2013. Available at http://eprint.iacr.org/2013/057.pdf

[9] M. v. Dijk, C. Gentry, S. Halevi, and V. Vaikuntanathan. *Fully homomorphic encryption over the integers*. In EUROCRYPT, pages 24-43, 2010. Full version in http://eprint.iacr.org/2009/616.pdf.

[10] NTL: A Library for doing Number Theory. http://www.shoup.net/ntl/

[11] HElib : A Library of implements Hoomorphic Encryption. https://github.com/shaih/HElib

국문초록

이 논문은 옵션의 가격계산을 완전동형암호화에 적용하여 계산하는 방식을 제안한 것이다.

1장에서는 간단한 소개와 적용방식에 대한 내용이다. 2장은 대표적인 옵션 가격 결정 모형인 블랙숄츠 방정식과 그 해를 유도한다. 3장은 서울대학교에서 발표된 CRT-based FHE[8] 와 HElib의 디자인에 사용된 BGV 알고리즘 [7] 에 대해 소개한다.

마지막으로 4장은 [8] 을 NTL[10] 을 이용하여 c++로 구현한 이형태 박사 (Nanyang Technological University) 의 프로그래밍 코드를 변형하여 계산된 결과와 개선 방안에 대해 논의한다.

주요 어휘 : 블랙숄츠 방정식, 옵션 가격, 옵션 그릭, 폴리호모몰픽, 완전동형암호화

학번 : 2011-23199