Journal of Korea Multimedia Society Vol. 20, No. 12, December 2017(pp. 1913-1927)
https://doi.org/10.9717/kmms.2017.20.12.1913

고대 및 현대 암호 방식을 결합한 초안전 문서 보안 프로그램의 설계 및 구현

유연수†, 이상곤††

Design and Implementation of a Security Program for Supersafe Document Using Ancient and Modern Cryptography

Yeonsoo You†, Samuel Sangkon Lee††

ABSTRACT

Encryption technology is to hide information in a cyberspace built using a computer and to prevent third parties from changing it. If a malicious user accesses unauthorized device or application services on the Internet of objects, it may be exposed to various security threats such as data leakage, denial of service, and privacy violation. One way to deal with these security threats is to encrypt and deliver the data generated by a user. Encrypting data must be referred to a technique of changing data using a complicated algorithm so that no one else knows the content except for those with special knowledge. As computers process computations that can be done at a very high speed, current cryptographic techniques are vulnerable to future computer performance improvements. We designed and implemented a new encryption program that combines ancient and modern cryptography so that the user never knows about data management, and transmission. The significance of this paper is that it is the safest method to combine various kinds of encryption methods to secure the weaknesses of the used cryptographic algorithms.

Key words: Ancient Cryptography, Modern Cryptography, Shift Encryption, Poly-alphabetic Substitution, Transposition Cipher, Nihilist Encryption, DES and AES Encryption, MVVM

1. 서 론

오랜 기간 동안 암호는 정보를 안전하게 보관하고 전송하는데 중요한 역할을 하여 왔다. 고대 암호의 역사를 살펴보면 옛 부터 많은 사람들이 암호의 중요성에 대하여 토론하여 비밀스런 정보 전달 체계를 마련하여 왔다는 것을 알 수 있다. 이러한 암호화 기술은 현대 암호 체계를 만드는데 기여하였으며, 근대의 컴퓨터 기술이 급속도로 발전함에 따라 특별한 체계를 마련하여 보관해야 하는 정보량도 비약적으로 증가되고 있다. 예전에는 주로 특정기관에서 정보 보관이나 혹은 기관과 기관 사이의 정보 전송에 암호화를 이용하였지만, 인터넷 기술의 보급으로 개인들끼리의 정보 전송에도 암호화 기술이 필요하게 되었다. 그러나 현재까지도 국민 개개인이 암호 알고리즘을 통해 자신만의 암호문을 손쉽게 생성할 수 있는 프로그램이 보편화 되어 있지 않다.

암호화란 자신의 정보를 허가받지 않은 제3자가

※ Corresponding Author : Samuel Sangkon Lee, Address: (55069) Chonjam-ro 303, Wansan-gu, Jeonju, South Korea, TEL : +82-63-220-2934, FAX : +82-63-220-2056, E-mail : samuel@jj.ac.kr
Receipt date : Jul. 15, 2017, Revision date : Oct. 9, 2017
Approval date : Nov. 6, 2017

† Dept. of Computer Science & Engineering, Jeonju University
(E-mail : powinz00@naver.com)
†† Dept. of Computer Science & Engineering, Jeonju University

취득하거나 열람할 수 없도록 하는 것에 목적이 있다. 그러나 일반인이 자신의 자료를 암호화하는 방법이나 다른 어떤 암호화 방법을 이용하여야 안전하고 또 이를 어떻게 암호화 하여야 하는지 모르기 때문에 개인적인 목적의 문서 암호화는 쉽지 않은 실정이다. 이러한 이유로 중급 수준의 이용자조차도 타인에게 비밀화 하려는 문서를 암호화 하지 않은 채 저장하고 있는 형편이다. 또한 현재 사용자들이 이용하는 암호화는 정부나 검증된 기관에서 권장하고 지원하는 암호화 프로그램을 이용한다. 이와 같이 제공되는 암호화 방법은 대부분이 공개된 알고리즘만을 이용하기에 성능이 좋은 컴퓨터를 사용하면 해독할 수 있다. 본 논문에서는 사용자의 데이터 전송 시 제3자가 결코 해독할 수 없도록 고대 암호와 현대 암호를 융합한 새로운 암호화 시스템을 설계하고 이를 구현하여 누구나 암호화 프로그램을 쉽게 사용 가능하도록 하는데 목적이 있다.

현대에 암호화 기술이 발달했다는 것은 암호화 기술이 제3자에 의해 해독될 수 있다는 것을 의미하기도 한다. 따라서 현재 사용하는 암호의 취약성이 발견되면 더 나은 암호화 기법이 개발될 것이다. 그러한 노력에 의하여 기존의 암호화 방식은 고대부터 현대에 이르기까지 지속적으로 발전하여 왔다. 처음에는 단순히 글자를 다른 글자로 대체하여 바꾸어 주는 치환(置換) 암호 방식으로 시작하였지만, 그 후 글자의 순서와 위치를 동시에 변화시켜 다른 위치로 옮겨 놓은 전치(轉置) 암호가 개발되었다. 현대에는 컴퓨터의 놀라운 성능을 사용하면 고대 암호의 해독은 쉽게 가능하다. 그래서 이를 개선하는데 수학적인 알고리즘을 적용하는 암호들이 생겨나게 되었다. 컴퓨터의 성능이 높아질수록 암호화 방식 또한 어려워졌을 뿐 아니라 해독 기술도 발달하였다고 볼 수 있다. 발달된 암호화 기술은 컴퓨터의 성능에 의존적인 방향으로 진화하고 있다. 컴퓨터가 사람이 할 수 있는 계산을 매우 빠른 속도로 처리하게 됨으로서 이를 이용하는 암호 기술은 굉장하지만 컴퓨터의 성능이 개선되면 현재 사용하는 암호 기술도 취약해질 수밖에 없다는 결론에 이르게 된다. 이러한 점이 컴퓨터 성능에 의존적인 암호화 알고리즘의 한계라고 볼 수 있다. 암호화에 사용되는 알고리즘이 어려우면 어려울수록 좋은 컴퓨터의 성능이 필요하게 되고, 이를 해독하기 위해서는 더 높은 사양의 컴퓨터가 필요하게 된다.

현재 암호화 방식이 대부분 수학적인 암호화 알고리즘에 의존적이지만, 해당 알고리즘은 대부분 공개되어 있다. 암호화 방식이 공개되어 있다는 것은 누구나 마음만 먹으면 해독하는 알고리즘을 구현할 수 있다는 것이다. 물론 공개된 알고리즘이라 할지라도 쉽게 해독하지 못할 만큼의 안정성도 입증되어 있기 때문에 암호 알고리즘은 계속 사용되고 있다. 그러나 검증된 알고리즘이라 하여도 컴퓨터 성능이 미래에 계속해서 좋아지면 영원히 안전할지 알 수는 없는 일이다. 이를 해결하는 방법 중 하나로 암호화 하는 키의 길이를 증가시키는 방법이 있다. 하지만 키의 길이가 128 비트인 암호화 알고리즘을 사용하는 경우라도 양자 컴퓨터를 사용하여 계산한다고 하면 해독이 불가능한 것만은 아니다. 물론 아직 양자 컴퓨터가 일반 사용자에게 보급되고 있지 않았지만 컴퓨터의 성능이 발달할수록 기존에 사용되던 암호가 해독될 위험성을 무시할 수만은 없는 실정이다[1].

덧붙여 현재 이슈가 되는 암호화 기술 중에 생체 인식을 사용한 인증 방식의 경우 충분히 많은 공격자가 연합하여 생체 인증을 시도할 경우, 위장 인증이 성공할 수 있다는 약점도 존재한다[2]. 그렇기에 생체 인증을 사용하는 암호화의 경우는 보통 다른 암호화 방식을 혼용하기 마련이다. 즉, 현재 사용하는 암호 알고리즘의 취약점을 보완하기 위해 여러 종류의 암호화 방식을 융합하는 방법이 가장 안전하다는 것이다. 이러한 아이디어가 본 논문의 주요 개념이다.

본 논문에서는 기존의 알고리즘을 활용하지만 기존 알고리즘에만 의존하지 않고, 미래에 해독될 가능성을 낮추는 데에도 주안점을 두었다. 이것이 바로 고전 암호화 방식과 현대 암호화 방식의 결합인 것이다. 알고리즘의 성능을 검증하는 가장 좋은 방법은 알고리즘을 공개하여 시험해 보는 것이다. 이렇게 검증된 알고리즘을 사용자가 원하는 대로 선택하여 고대 암호와 현대 암호 방식을 혼합하여 다중 암호화하는 방식으로 문서를 보관할 수 있다. 물론 복호화 하기 위해서는 사용자가 선택한 최대 10가지를 융합한 암호화 방식을 역순으로 알아야 하고, 더불어 각각의 키값도 알고 있어야만 한다. 이러한 방식은 기존 알고리즘에 의존적인 암호화 방식에서 사용자가 암호 알고리즘을 선택하여 다중 암호화를 한다는 점에서 기존의 방식들에 비해 암호화 강도가 크게 높아지게

된다. 다음 절에서는 각각의 암호화 방식에 대하여 설명한다.

2. 각 암호화 방식의 이론적 배경

2.1 고대 암호

고대의 암호에 속하는 시프트(Shift) 암호화 방식은 영문 알파벳 26 글자를 기준으로 키(key)가 되는 시프트 값(숫자) 만큼 우측 혹은 좌측으로 이동하여 암호문을 만드는 비교적 간단한 방식이다. 예를 들어 다음의 Table 1에서 영문 알파벳 26 글자를 우측으로 2만큼 이동하여 시프트 한 경우를 나타낸 것이다. 이 방식을 이용하여 암호화 하려면 입력문 전체에 순차적으로 키를 적용함으로서 시프트 암호가 완성된다.

본 논문에서는 위에서 설명한 시프트 암호화 방식을 사용하여 우리나라에서 사용하는 완성형 한글 11,172 글자와 특수 문자(44) 및 영문 대/소문자(26+26) 총 96 글자(44+26+26)를 포함할 수 있도록 확장하였다. 시중에서 출간된 관련 도서는 영문자만을 사용하여 그 구현 원리를 설명하고 있으나, 본 논문에서는 한글도 포함하여 암호화 및 복호화 과정을 국민들이 살펴 볼 수 있도록 하였다. 특수 문자 및 영문 대/소문자, 완성형 한글을 아스키 코드와 유니코드의 숫자 값으로 변환한 후, 숫자 값 사이의 공백을 제거하고 시프트 암호화 방식을 적용하였다. 그 후 암호화 이전에 제거하였던 숫자 값 사이의 공백에 해당하는 값만큼 대입 연산을 하여 입력이 가능한 범위의 글자만 정상적으로 출력되도록 구현하였다. 양의 정수를 키값으로 입력할 시에는 우측(right)으로 쉬프트 하고, 음의 정수를 입력할 시에는 좌측(left)으로 쉬프트 되도록 설계한 점이 본 연구의 특징이다.

다표식 환자 암호화(Polyalphabetic Substitution) 방식을 학자에 따라서는 '다중 문자 암호(polyalphabetic substitution, 대표적인 것은 Vigenere cipher (비제네르 암호))'라 한다. 이 암호화 방법도 앞서 설명한 기존 연구와 동일하게 고전적인 암호화 방식에 속하는데, Table 2에서와 같이 일반적으로 영어 알파

Table 1. Example of Shift Encryption

A	B	C	D	E	F	G	H	…
c	d	e	f	g	h	i	j	…

Table 2. Example of Polyalphabetic Substitution

	A	B	C	D	E	F	G	…
A	A	B	C	D	E	F	G	…
B	B	C	D	E	F	G	H	…
C	C	D	E	F	G	H	I	…
D	D	E	F	G	H	I	J	…
E	E	F	G	H	I	J	K	…
F	F	G	H	I	J	K	L	…
G	G	H	I	J	K	L	M	…
…	…	…	…	…	…	…	…	…

벳 26개를 가로와 세로(26 × 26)의 크기 블록에 열과 행마다 한 글자씩 이동되도록 채운 후에 암호화에 사용되는 키와 암호화의 대상이 되는 입력문을 각각 가로와 세로 좌표로 구성하여 문서를 암호화 한다[3].

다표식 환자 방식의 암호화는 일반적으로 영문 알파벳 26 글자를 기준으로 앞의 쉬프트 암호와 마찬가지로 완성형 한글 11,172 글자와 특수문자 및 영문 대/소문자 등 96(=44+52) 글자를 추가하였다. 입력문의 글자를 각각 유니코드 값으로 변환하여 프로그램 내에서 사용되지 않는 코드 값을 제외하고 연속된 숫자열로 정렬한다. 동일한 연산을 키값에도 적용한 후, 연산된 입력문에 값을 더하고 이를 다시 문자로 복원한다. 이 복원된 문자가 암호문이 되는 것이다. 요약하면 본 논문에서는 영문자만으로 제작된 기존의 다표식 환자 방식의 암호문의 원리를 그대로 사용하면서도 추가적으로 한글 및 영문과 특수 문자까지 확장하여 암호를 생성하는 알고리즘을 구현하여 우리 국민들이 작성하는 문서 내의 모든 텍스트를 암호화 할 때 모든 문자가 정상적으로 암호화 되도록 프로그램을 개발한 점이 본 연구의 특징이다.

다표식 환자는 고전 암호에 속하는 표준 전치 암호로 블록을 이용한 블록 암호화 방식[4]으로 분류할 수 있다. 암호화하려는 입력문을 입력된 키값(숫자) 만큼의 열을 가진 블록에 대입(행 단위)하여 이를 열 단위로 출력하여 암호문을 만든다. 아래의 Table 3에서 입력문을 행 단위로 입력하면 "stan"이 되고, 출력은 세로로 읽어 "sdtsinh"와 같이 암호화 한다.

위의 Table 3에서 보는 바와 같이 키가 4인 경우의 출력문은 "sdtsinh tarptce araoiir ndnsop"인 반면에 키가 6인 경우(아래의 Table 4 참고)는 출력문이 "srsih tdpoe atonr nrsc daii antp" 로 되어 키의 블록 크기에 따라 전혀 다른 암호문을 생성할 수 있다. 이

Table 3. Example of STC (When Key=4)

s	t	a	n
d	a	r	d
t	r	a	n
s	p	o	s
i	t	i	o
n	c	i	p
h	e	r	

• Input : Standard Transposition Cipher

Table 4. Example of STC (When Key=6)

s	t	a	n	d	a
r	d	t	r	a	n
s	p	o	s	i	t
i	o	n	c	i	p
h	e	r			

와 같이 키의 크기가 달라지면 전혀 다른 암호문이 생성된다는 점에서 암호의 강도를 높이는 장점이 있다. 다시 말하면 키값이 4일 경우, 4×Y의 블록을 생성한 뒤, 암호화 할 입력문을 대입한다. 그 후 해당 블록에서 열 단위(세로열)로 출력문을 작성하여 암호문을 완성한다. 반면에 Table 4와 같이 키값이 6일 경우에도 앞과 동일한 방법이지만 6×Y 크기의 블록을 만든 뒤에 열 단위(세로열)로 출력문을 작성하여 키값에 따라 서로 다른 암호문을 생성할 수 있다는 것이다.

이러한 방식을 “표준 전치 암호(Standard Transposition Cipher)”라 부르며, 본 논문에서 설계한 프로그램에서는 위에서 언급한 바와 같은 방식의 전치 암호를 사용할 수 있도록 하였다. 다만 본 논문은 표준 전치 암호도 다른 암호들과 마찬가지로 영문은 물론 완성형 한글과 특수 문자가 혼재하는 문서에도 에러 없이 적용할 수 있도록 설계하였다.

이에 반해 열쇠형 전치 암호(Key Transposition Cipher)는 표준 전치 암호와 같이 블록 암호 방식을 사용한다는 점에서는 이전의 방식과 유사한 점이 있으나, 입력되는 키값이 숫자로 한정되었던 기존의 표준 전치 암호화에 비해 본 논문에서 설계한 프로그램은 문자형 키값도 가능하도록 하여 해석 강도를 한층 높여 개선된 암호화 방식의 형태이다. 열쇠형 전치 암호는 숫자와 문자 혹은 특수 문자도 키값으로 사용할 수 있는 것이 장점이다. 암호화 방식은 입력되는

Table 5. Example of KTC

키 값	t	r	a	n	s
입력문	k	e	y	t	r
	a	n	s	p	o
	s	i	t	i	o
	n	c	i	p	h
	e	r			

• Input : Key Transposition Cipher
• Key Value : Trans

키값(문자)의 길이만큼 열을 가진 블록을 생성하여 표준 전치 암호화 같이 입력문을 대입하고 키값(문자)을 알파벳 순서와 동일한 숫자로 치환하여 숫자가 작은 열부터 출력하여 암호문을 완성한다.

열쇠형 전치 암호는 위의 Table 5와 같이 키값의 길이만큼의 열을 갖는 블록을 생성한 후에 암호화할 입력문을 대입한다. 그 후 키값을 알파벳 순서(Table 5의 경우, a > n > r > s > t)로 열 단위로 출력하여 암호문을 완성한다. Table 5의 열쇠형 전치 암호의 예에서와 같이 출력하는 열의 순서를 알파벳 순으로 결정함으로서 표준 전치 암호에 비하면 암호화 강도가 높아진 것으로 예측할 수 있다. 본 논문에서 구현한 프로그램에서는 이와 같은 방식의 열쇠형 전치 암호도 구현하였다.

고전 암호 중에서 니힐리스트(Nihilist) 암호는 전치 암호의 일종으로 블록 암호화 방식에 속한다. 열쇠형 전치 암호와 유사한 암호화 방식을 사용하지만 차이점은 Table 6에서와 같이 열 단위로 키값을 적용한 후에 다시 행 단위로 키값을 적용(알파벳 순서에 따라)하여 출력하여 암호문을 완성한다. 따라서 표준 전치 암호나 열쇠형 전치 암호보다 높은 강도의 암호문이 생성된다. 이상과 같이 다섯 가지(Shift, PS; Polyalphabetic Substitution, STC; Standard Transposition Cipher, KTC; Key Transposition Cipher,

Table 6. Encryption of Nihilist

	c	a	k	e
c	n	i	h	i
a	l	i	s	t
k	c	i	p	h
e	e	r		

• Input : Nihilist Cipher
• Key Value : Cake
• Output : iiri lnec tih shp

Nihilist)의 고대 암호에 대한 구현 원리를 마치고 다음 절에서는 현대 암호에 대해 설계한 내용을 설명한다.

2.2 현대 암호

미국의 국립표준국(National Bureau of Standards)은 1975년에 데이터 암호화를 위해 DES(Data Encryption Standard)라는 일종의 블록[4] 암호화를 표준으로 제정하였다. 미국 NBS(National Bureau of Standards, 현재 NIST)에서 국가 표준으로 정한 암호이다. 이 기술은 56 비트의 키(key)로 사용하는 대칭키(symmetric-key) 암호화 방법[1]으로, 한동안은 별 문제없이 이용되었으나 이 기술은 키의 길이가 너무 짧고, 백도어(backdoor)가 포함되어 있다면 특수한 방법으로 해독할 수 있다는 문제점을 가지고 있었다. 따라서 얼마 지나지 않아 DES는 취약한 것으로 알려져 현재는 AES(Advanced Encryption Standard) 암호화 방식으로 대체 되었다.

AES-256은 본래 이름인 라인달(Rijndael) 암호라 하며, 현대 암호의 일종이다. 미국의 국가 표준 암호화 방식인 AES-256은 앞에서 언급한 바와 같이 기존의 국가 표준 암호화 방식으로 사용되었던 DES의 취약점을 보안하기 위해 고안되었는데, 현재 사용되는 암호화 방식 중 가장 강력한 암호화의 하나로 알려져 있다[5]. AES는 DES 암호화 기술이 가지고 있던 문제점을 해결하기 위해 새롭게 개발된 128 비트 블록 암호화 방식이다. AES 기술 역시 암호화와 복호화 과정에 같은 키를 이용하는 대칭키 방식을 이용하지만, 128 비트의 블록을 128, 196, 혹은 256 비트의 키 길이로 처리할 수 있다는 것이 DES와의 가장 큰 차이점이다.

우리가 가장 많이 이용하는 근거리 무선통신 기술인 와이파이의 경우도 암호화 방식 중의 하나로 AES를 이용하고 있으며, 블루투스의 경우도 3.0 표준까지는 SAFER+ 암호화 기술을 이용하였으나, 블루투스 4.0부터는 AES 표준을 이용하고 있다. 본 논문에서는 현대의 암호 방식인 DES와 AES를 구현하였으며, 보다 효과적이고 정확한 암호화 프로그램의 개발과 편리한 구현을 위해 C# 라이브러리의 System.Security.Cryptography를 이용하였다. 이 라이브러리는 인코딩/디코딩 해시, 난수 생성, 메시지 인증 등의 여러 작업뿐만 아니라, 데이터를 포함한 암호화 방식을 손쉽게 구현할 수 있는 방법을 제공한다. 추가적으로 본 프로그램에서는 AES-256을 적용하여 256 비트의 키값을 적용하였다.

3. 암호화 방식의 설계

본 시스템은 고전 암호와 현대 암호를 혼합하여 문서를 암호화하고 암호문을 저장하고 반대로 저장된 암호문을 복호화 하여 본래의 문서로 저장할 수 있는 기능의 구현을 목표로 하였다. 아래의 Fig. 1에 전체 시스템 구조도를 제시하였다. 기능의 요구 사항은 다음과 같다. 문서 불러오기의 기능은 텍스트 파일을 불러와서 화면에 출력할 수 있도록 하고, 암호화 방식의 선택은 암호화에 사용할 암호화 방식을 최대 10개까지 중복 선택하여 혼합된 암호 방식의 선택이 가능하도록 하였다. 이러한 혼합 방식은 고대와 현대 암호 총 일곱 종류를 최대 10개까지 중복을

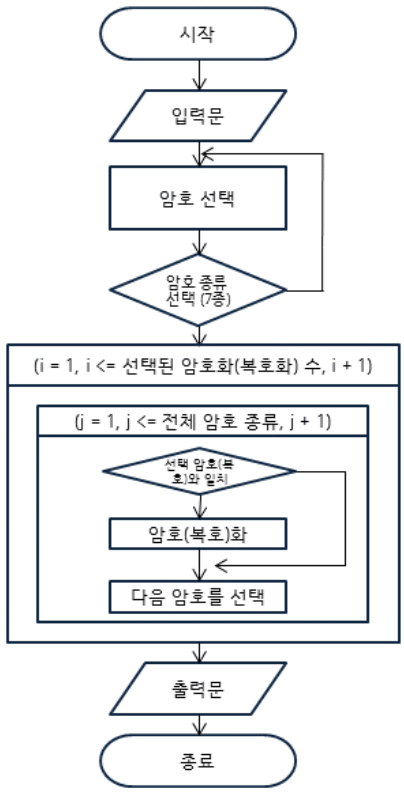

Fig. 1. Overall Control.

1) 위키백과 (DES), https://ko.wikipedia.org/wiki/DES_(암호)

Table 7. Specification of Use-case "Actor"

액 터	설 명
사용자	• 사용자는 프로그램을 사용하는 주체를 의미한다. • 사용자는 프로그램을 사용하여 텍스트를 불러오고 저장할 수 있다. • 사용자는 프로그램을 사용하여 문서 암호화 방식을 선택, 추가할 수 있다. • 사용자는 프로그램을 사용하여 문서 암호화를 할 수 있다.

Table 8. Description of Use-case

Use-case	Description
Open	Read the document file and print the text on the screen.
Selection	Select and add the encryption method to use for encryption.
Key Registration	Register the key value to be used for encryption.
Encryption	Encrypt the document using the selected method.
Save	And stores the encrypted / decrypted document as a file.
Description	Decrypt the encrypted document.

허용하여 융합한 암호화 방식이다. 본 논문은 이와 같이 여러 종류의 암호화를 다양한 형태로 조합할 수 있는 점에서 다른 논문과의 차별성이 있다.

암호화에 사용할 키값의 입력은 암호의 종류에 따라 사용할 키값을 각각 입력할 수 있어야 한다. 여기서 암호 알고리즘의 종류나 키 길이는 시스템의 안전성 수준을 만족하는 중요한 요소이다. 특히 KISA(한국인터넷진흥원)는 개인키의 길이를 160 비트 이상의 사용과 사용하는 유효 기간을 최대 2년으로 권고하고 있다. 일곱 가지의 암호화 방식(Shift, PS (Polyalphabetic Substitution), STC(Standard Transposition Cipher), KTC(Key Transposition Cipher), Nihilist, DES, AES-256)을 지원하고 있다. 또한 위의 일곱 가지를 복합적으로 적용 가능하며, 그 적용 순서도 바꿀 수도 있다. 이 적용 순서를 어떻게 하느냐에 따라 암호문의 저항성도 크게 달라진다. 암호문 출력(암호화된 텍스트를 화면에 출력), 암호문 저장(암호문을 텍스트 파일로 저장 가능), 암호화 된 문서의 복호화(암호문을 복호화 하여 원문을 복원) 등의 기능을 모두 내장하고 있다.

소프트웨어공학적 입장에서 본 프로그램의 품질 요구 사항은 다음과 같다. 입력 및 출력(입력문 및 암호문의 불러오기와 저장하기가 정상적인 기능의 수행이 되고 데이터 손실이 없어야 한다), 암호화의 종류에 따라 정상적인 암호화가 되어야 한다. 만약 원문이 조금이라도 수정되면 복호화가 불가능하도록 설계하였다. 복호화 기능은 암호문의 복호화 시에 정상적인 복호화가 되어야 한다. Fig. 1에 본 프로그램의 전체 암/복호화 과정의 순서도를 소개하였다.

다음으로 유스케이스 액터 명세와 개요 명세에 대하여 설명한다. 본 프로그램은 Table 7과 같이 하나의 액터인 '사용자'가 있다.

사용자의 시각에서 보면 시스템의 범위와 기능을 쉽게 설명하기 위해 유스케이스(use-cases)를 사용하는데 우리말로 '쓰임새'라고 한다. 유스케이스는 두 가지를 동시에 사용하는 혼돈을 막기 위해 말 그대로 '쓰이는 용도'라 말한다. 유스케이스는 시스템이 사용되는 용도를 모아서 다용도 시스템을 만드는 것이다. 이와 같이 유스케이스들을 모아 시스템으로 연결시켜 보여 주면 개발 과정을 사용자가 쉽게 이해할 수 있다. Table 8과 같이 본 프로그램의 유스케이스에는 '문서 불러오기(Open)', '암호 선택/추가(Selection)', '키등록(Key Registration)', '문서 암호화(Encryption)', '문서 저장하기(Save)', '문서 복호화(Description)' 등이 추가되어 있다. 다음에 Table 9는 유스케이스의 시나리오를 제시하였다.

시퀀스 다이어그램은 웹 시퀀스 다이어그램(Web Sequence Diagrams[2])을 사용하여 작성하였다. 여기서 시퀀스 다이어그램이란 상호 작용의 오브젝트 간 메시지 교환의 순서를 설명하고, UML(Unified Modeling Language) 다이어그램을 제시하는 것이다. 특히 UML은 소프트웨어를 만들기 위해 어휘와

2) https://www.websequencediagrams.com

Table 9. Scenario of Use-case

3.3.1 문서 불러오기(Open)			3.3.2 암호 선택/추가(Selection)		
A	개요	사용자는 프로그램 내에서 문서 파일을 불러올 수 있다.	A	개요	1. 사용자는 문서 암호화 방식을 선택 / 추가할 수 있다.
B	릴레이션	2. - Initiator : 사용자 3. - Supporters : 없음 4. - Pre-condition : 없음 5. - Post-condition : 사용자가 선택한 문서 파일의 텍스트가 화면에 출력된다.	B	릴레이션	6. - Initiator : 사용자 7. - Supporters : 없음 8. - Pre-condition : 없음 9. - Post-condition : 사용자가 선택한 암호화 방식이 추가된다.
C	기본 흐름	C-1. 프로그램 내에서 파일 불러오기를 선택한다. C-2. 불러오려는 문서 파일을 선택한다. C-3. 문서 파일의 텍스트가 화면에 출력된다.	C	기본 흐름	C-1. 암호 목록 콤보 박스에서 사용할 암호 종류를 선택한다. C-2. Select 버튼을 클릭하여 암호를 추가한다.
D	대안 흐름	없음	D	대안 흐름	없음
E	예외 흐름	파일이 선택되지 않은 경우 1. 파일 선택을 취소하고 초기 프로그램으로 돌아간다.	E	예외 흐름	최대 선택 개수를 초과할 시 오류 메시지를 출력한다.
3.3.3 키 값 등록(Key Registration)			3.3.4 문서 암호화(Encryption)		
A	개요	사용자는 선택된 암호 방식에 따라 사용할 키값을 등록할 수 있다.	A	개요	사용자는 선택된 암호 방식에 따라 문서를 암호화할 수 있다.
B	릴레이션	- Initiator : 사용자 - Supporters : - Pre-condition : 암호 선택 / 추가 - Post-condition : 암호화에 사용될 키값이 등록된다.	B	릴레이션	- Initiator : 사용자 - Supporters : - Pre-condition : 문서 불러오기, 암호 선택 / 추가 - Post-condition : 암호화된 문서가 화면에 출력된다.
C	기본 흐름	C.1. 추가된 암호 버튼을 클릭한다. C.2. 키값을 입력한다.	C	기본 흐름	C1. Convert 버튼을 클릭하여 암호문을 변환한다. 대안 흐름과 예외 흐름은 없다.
D	대안 흐름	없음	D	대안 흐름	키값이 입력되지 않은 경우 키 입력창을 표시한다.
E	예외 흐름	규칙에 맞지 않는 키값 입력 시 오류 메시지를 출력한다.	E	예외 흐름	사용할 암호가 선택되지 않았거나, 키값이 입력되지 않았을 시 오류 메시지를 출력합니다.
3.3.5 문서 저장하기(Save)			3.3.6 문서 복호화(Description)		
A	개요	사용자는 암호화 / 복호화 된 문서를 파일로 저장할 수 있다.	A	개요	사용자는 선택된 암호 방식에 따라 문서를 복호화할 수 있다.
B	릴레이션	- Initiator : 사용자 - Supporters : - Pre-condition : 암호화 / 복호화 - Post-condition : 문서가 파일로 저장된다.	B	릴레이션	- Initiator : 사용자 - Supporters : - Pre-condition : 문서 불러오기, 암호 선택 / 추가 - Post-condition : 복호화 된 문서가 화면에 출력된다.
C	기본 흐름	C.1. 저장하기를 선택합니다. C.2. 원하는 경로에 파일 이름을 입력한다. C.3. 저장 버튼을 클릭한다.	C	기본 흐름	C.1. 암호화 방식에 따라 키값을 등록한다. C.2. Restore 버튼을 클릭하여 암호문을 변환한다.
D	대안 흐름	없음	D	대안 흐름	없음
E	예외 흐름	E1. 저장을 취소하였을 경우 1. 초기 화면으로 돌아간다.	E	예외 흐름	암호문이 변경되었거나 키값이 다를 경우 오류 메시지를 출력한다.

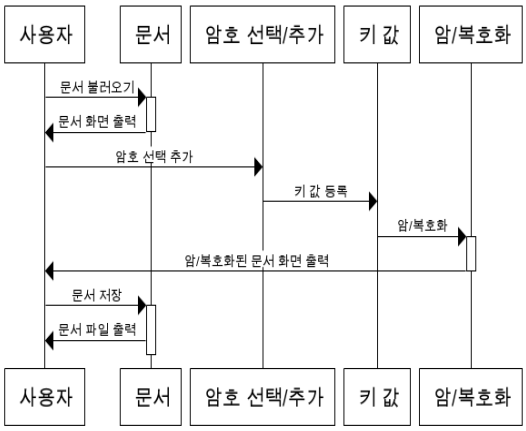

Fig. 2. Sequence Diagram.

규칙을 두어 시스템을 개념적, 물리적으로 표현하는 모델에 사용된다. 이것은 시스템의 구조적 문제와 프로젝트 팀 내의 의사소통 그리고 소프트웨어 구조의 재사용 문제를 해결하는데 도움이 된다. 본 논문의 프로그램에서 사용된 설계 내용을 다음의 Fig. 2와 같이 시퀀스 다이어그램을 설계하였다.

4. 구 현

4.1 개발 환경 및 구현 툴

개발 환경은 다음의 Table 10과 같다. 본 프로그램은 윈도우 환경에서 구동되는 응용 프로그램으로 윈도우 어플리케이션에 최적화된 언어와 툴을 사용하였으며, 윈도우즈 10 환경에서 진행했다. 윈도우 어플리케이션에 최적의 언어인 C#으로 개발하고, 디자인은 WPF(Windows Presentation Foundation)를 사용하여 XAML(eXtensible Application Markup Language)로 개발하였다. 개발 툴은 Visual Studio 2017 Community RC을 사용하였다. 실행 환경은 가상 머신인 .NET Framework 4.6 이상 버전에서 정상 작동하도록 설계하였다.

Table 10. Development Environment

Development Environment	WINDOWS 10
Development Language	C#, WPF(XAML)
Development Tools	Visual Studio 2017 Community RC
Execution Environment	.NET Framework 4.6

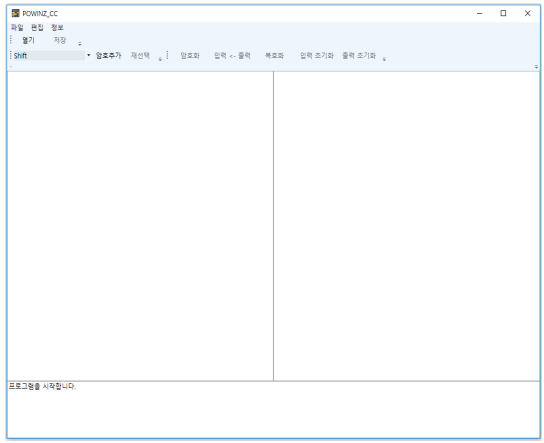

Fig. 3. Screen of Program Execution.

4.2 사용자 인터페이스

화면 디자인은 사용자가 쉽게 사용할 수 있는 익숙한 디자인으로 구성하였다. 상단에 메뉴를 배치하고, 그 아래 툴바를 이용하여 메뉴에 빠르게 접근하도록 하였다. 프로그램의 중앙에는 입력문이 출력되는 창과 변환된 암호문이 출력되는 창으로 구분하였고, 사용자가 프로그램의 진행 여부를 확인할 수 있도록 하단에 로그 창을 Fig. 3과 같이 배치하였다.

메뉴 바에는 파일과 정보의 두 가지 메뉴를 구성하였으며, 파일 메뉴의 하위 메뉴로 열기/저장/종료 버튼을 구성하였다. 툴바에는 파일 입출력과 관련된 기능을 묶어 구성하였다. 두 번째 툴바에는 콤보 박스를 사용하여 암호 목록을 나열하였다. 선택과 클리어 버튼으로 암호를 선택하고 선택 목록을 초기화할 수 있는 기능도 함께 구현하였다. 세 번째 툴바는 암호 변환과 관련된 기능으로 구성하였다. 각각 암호화, 입력 <- 출력, 복호화, 입력 초기화, 출력 초기화 버튼 등 다섯 가지 메뉴로 구성하였다.

4.3 구조 설계 방법

4.3.1 아키텍처 모델

프로그램 개발 시 중요한 디자인 패턴은 아래의 Fig. 4와 같이 MVVM(Model-View-ViewModel)[3] 모델을 사용하여 개발하였다. 해당 모델은 기존의 MVC(Model-View-Controller)나 MVP(Model-View-Presenter) 패턴에 비해 UI 접근성이나 프로그램 소

3) MSDN, MVVM의 예, https://msdn.microsoft.com

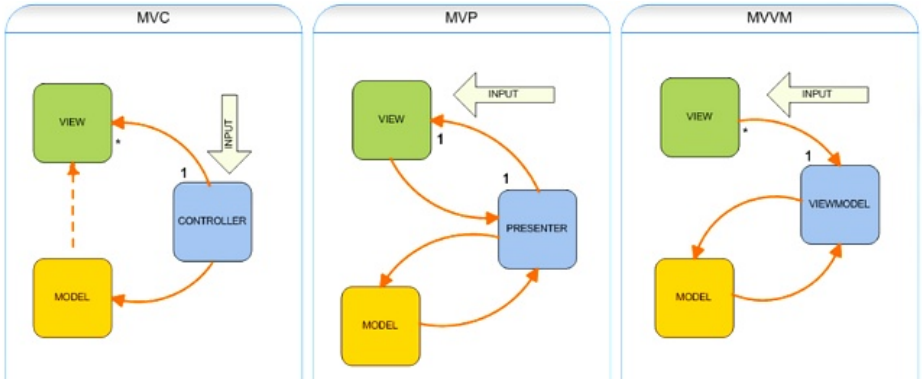

Fig. 4. Design Patterns.

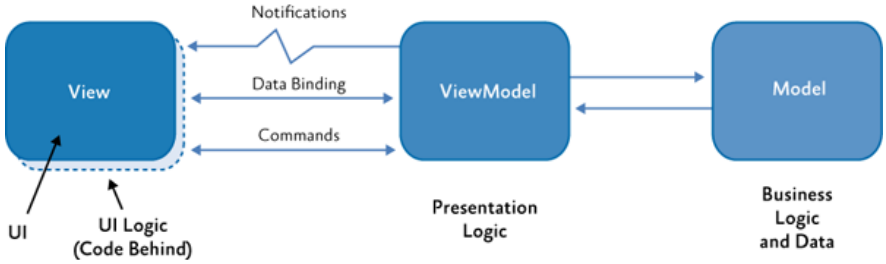

Fig. 5. Example of MVVM.

스 코드의 해석에 효율적이다. 그림에서 INPUT은 사용자의 입력이나 혹은 이벤트를 의미하고, 실선은 일반적인 형태의 정보 교환을 의미하고, 반면에 점선은 정보 교환이 가능하나 모델 간의 종속성과 보안상의 이유로 권장하지 않는 것을 의미한다. 또한 * 의 의미는 다수(many)를, 1 의 의미는 일(one)의 관계를 의미하므로, 왼쪽의 MVC는 다수 대 일의 관계(many-to-one relationship)를 의미하며, MVP는 일과 일의 관계(one-to-one relationship)를 의미하며, 마지막으로 MVVM은 일 대 다수의 관계(one-to-many relationship)를 의미한다.

기존의 MVC 패턴은 컨트롤러(Controller)가 뷰(View)와 모델(Model)에 직접 접근하여 제어되고, 모델에서는 뷰에 간접적으로 접근할 수 있었다. MVP[4] 패턴에서는 뷰와 모델이 프리젠터(Presenter)를 통하여 접근하였다. 그러나 본 프로그램에서 채택한 MVVM 패턴은 아래의 Fig. 5와 같이 뷰와 모델 그리고 뷰모델(ViewModel) 등 세 가지 구조로 설계되어 뷰에서는 디자인을 설계하고, 모델에서는 데이터를 처리하고, 그리고 뷰모델에서는 모델의 정보를 이용하여 데이터 바인딩으로 뷰를 제어한다. 그림의 왼쪽에서 뷰(View)를 설계할 때 XAML로 구성한 UI(User Interface)와 개발에 용이한 언어로 구성된 코드를 뒷면(Code Behind)에 설계하여 사용자 인터페이스 로직(User Interface Logic)을 구성한다. 그림 가운데의 지시(Notification)는 뷰모델의 지시에 의해 뷰가 동작되고, 이러한 동작이 행해지면 뷰와 뷰모델 사이의 데이터 바인딩(Data Binding)의 정보교환이 이루어진다. 또한 이와 동시에 명령어(commands)가 실행된다. 프리젠테이션 로직(Presentation Logic)에 해당되는 뷰모델은 뷰에 표시되는 전반적인 부분을 제어하며, 그림 오른쪽의 비즈니스 로직과 데이터(Business Logic and Data)에 해당하는 모델은 정보의 저장, 수정, 삭제 등의 관리를 담당한다.

따라서 앞의 두 가지 디자인 패턴보다 MVVM 패턴은 현대적이며, 도메인 로직과 프리젠테이션 층 사이를 분명하게 분리 제공하여 사용자 입장에서 얻은 뷰와 모델을 분리하여 효율성을 달성할 수 있었다. 또한 MVVM은 코드의 깨끗한 분리가 되어 유지 보

4) 정규 표현식, RegExr v2.1, Regular Expression: Learn, Build, and Test Reg. Ex., http://regexr.com

수성이 좋고, 외부 및 내부 의존성으로 코어 로직 부분의 분리 다시 말하면 핵심 로직에 대한 단위 테스트가 용이하며 프로그램 테스트가 용이하고, 코드 재사용이나 코드 교체 혹은 아키텍처의 적당한 장소에 코드의 추가가 좋으며, 동시에 코드 숨김으로 추상화를 실현할 수 있다. 이상과 같이 POWINZ_CC[5](프로그램의 이름)에서는 MVVM 모델을 사용하여 보다 효율적인 프로그램을 설계하였다.

위의 Fig. 6은 본 시스템의 구조도를 나타낸다. 먼저 뷰로 사용될 메인 윈도우(Main Window)를 만들고, WPF의 XAML[6]을 사용하여 소스 코드를 작성하였다. 뷰모델이 될 메인 윈도우 뷰 모델(Main Window View Model)을 만들고, 메인 윈도우와 데이터 바인딩을 통하여 메인 윈도우 뷰 모델에서 데이터의 변경이 일어나면 뷰에 바로 반영될 수 있도록 구현하였다. 또한 명령(Command)를 사용하여 뷰와 뷰모델 간의 사용자 명령이 전달되어 각각의 기능이 잘 동작할 수 있도록 구성하였다. 메인 윈도우 뷰 모델에서는 모델에 해당하는 텍스트 파일을 불러오거나 저장할 수 있도록 하였다. 또한 암호화 기능을 사용하기 위해 그림 오른쪽 Cipher에 구현되어 있는 암호화 알고리즘 파일에 접근할 수 있도록 구현하였다.

4.3.2 정규식 및 문자열 연산의 설계

본 프로그램은 문자열을 처리하기 위하여 정규식(regular expression)과 문자열 내부 연산을 4.3.1절의 각주 3을 이용하여 사용하였다. 정규식은 입력되는 텍스트를 전처리(pre-processing) 하는데 효과적

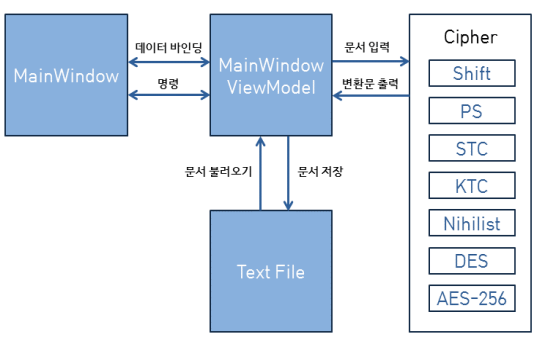

Fig. 6. System Architecture.

이다. 정규식에서는 완성형 한글과 영문자(특수 문자 포함)만 입력될 수 있도록 허용한다. 전치 암호 방식의 경우, 입력문에서는 문자열 내부 연산이 필요하지 않지만, 키값은 내부 연산이 필요하다. 특히 치환 암호 방식의 입력문이든 키값이든 모두 문자열 내부 연산이 필요하다.

시프트(shift) 암호화 방식과 표준 전치 암호(STC) 방식에서 입력문은 어떤 형태든 상관없지만 키값으로 숫자만 허용되기 때문에 입력된 키값을 숫자형인 정수(int)형으로 변환하여 연산해야 한다. 또한 시프트 암호의 키값은 두 가지 방향을 나타내는 플러스(+)/마이너스(-) 값을 입력 받고, 입력문은 아스키 코드 값이나 유니코드 값으로 변환한 뒤 내부적인 연산을 수행한다. 이 내부 연산은 암호화 과정이 끝난 후 암호문을 출력할 때 반대로 숫자를 문자로 치환하여 출력하도록 구현되었다. 또한 암호화 과정 뿐 아니라 복호화 과정에도 동일하게 처리된다.

4.4 프로그램의 구현 및 기능 설명

본 절에서는 메뉴 구성을 시각적으로 나타내 사용자의 이해를 돕고자 한다. Fig. 7과 같이 파일 열기/저장은 상단 메뉴의 파일 메뉴를 클릭하면 나타나는 하위 메뉴에서 Fig. 7과 같이 확인할 수 있다. 또한 상단의 첫 번째 툴바에서도 열기/저장 버튼을 클릭하면 파일 열기/저장 다이얼로그 박스가 출력된다. 이 다이얼로그를 사용하여 텍스트 파일을 불러오거나 저장할 수 있다.

아래의 Fig. 8과 같이 두 번째 툴바의 콤보 박스와 각각의 버튼은 암호화에 사용할 암호화 방식을 선택할 수 있는 메뉴로 구성되어 있다. 콤보 박스를 누르면 Shift, PS, STC, KTC, Nihilist, DES, AES-256

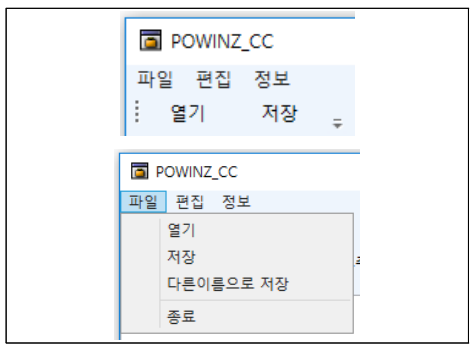

Fig. 7. File Open or Save.

5) POWINZ는 본 논문 저자의 필명을 사용하였으며, CC는 Cipher Converter의 의미이다.
6) MSDN, WPF의 예, https://msdn.microsoft.com

등과 같이 일곱 가지로 구성된 메뉴가 나타난다. 암호 선택 전에는 열기/저장 툴 바 메뉴의 밑이 공백으로 비어 있다. Fig. 9는 네 가지 암호를 선택한 예를 보여 주는데, 현재 열기한 문서를 Shift, PS, STC, KTC의 순서로 암호화한 화면이다. 콤보 박스에서 특정 암호를 선택하고, 선택(Select) 버튼을 선택하면 해당 암호가 적힌 버튼이 추가되는 것을 볼 수가 있다. 암호화 방식의 조합은 최대 10개까지 선택이 가능하며, 향후 선택 방법의 변경(추가/수정/삭제)도 가능하도록 업그레이드 할 예정이다. 선택된 암호화 방식은 암호화 혹은 복호화 과정에서도 순서대로 연산된다. 재선택 버튼을 클릭하면 현재 선택되어 추가된 암호화 방법을 모두 클리어하고 현재까지 선택된 암호를 출력하는 툴바도 동시에 초기화 한다.

암호 선택으로 사용할 암호화 방식을 충분히 선택하였다면 다음으로 선택한 암호의 키(key) 값을 등록하여야 한다. 추가된 암호 버튼을 클릭하면 Fig. 10과 같이 키값을 등록할 수 있는 다이얼로그가 화면에 표시된다. Shift 암호의 경우, 음의 정수를 입력하면 왼쪽으로, 양의 정수를 입력하면 오른쪽으로 Shift 연산을 진행한다.

Shift 암호화 방식과 STC 암호화 방식은 Fig. 11과 같이 키값으로 숫자만 입력이 가능하다. 만약 숫자 이외에 문자가 입력이 되는 경우 정규식을 이용하여 값을 비교한 후, 오류 메시지를 출력한다. 사용자는 이 오류에 대하여 적절하게 대응하여야 한다. 세 번째 툴바는 Fig. 12와 같이 암호화 및 복호화에 사용하는 기능들로 구성되어 있다.

암호화 버튼은 암호화를 실제 수행하는 버튼이다. 앞에서 서술한 3절의 Table 8과 일치시키기 위해 괄호 안에 표기하였다. 입력 문서를 입력하거나 문서 불러오기(Open)를 한 상태에서 암호화 방식을 선택(Selection)하고, 암호의 키값을 적당한 암호 방식에 따라 입력한 후(Key Registration)에 암호화 버튼을 선택하면 암호화(Encryption)가 진행된다. 암호화가 진행되는 중에는 프로그레스 바(progress bar)가 나타난다. 암호화 진행 정도에 따라 값이 증가하도록 구현되었으며, 암호화가 완료되면 이 바는 사라진다. [입력 ← 출력] 버튼은 암호화/복호화 된 텍스트를

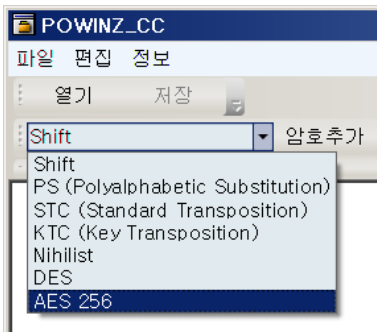

Fig. 8. Selection of Encryption Toolbar.

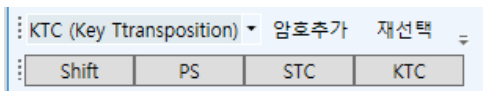

Fig. 9. How to Add Encryption Method.

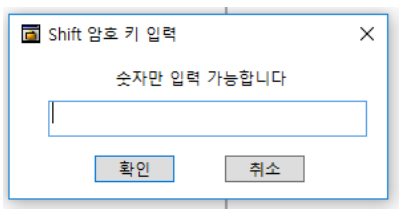

Fig. 10. Dialog of Key Input.

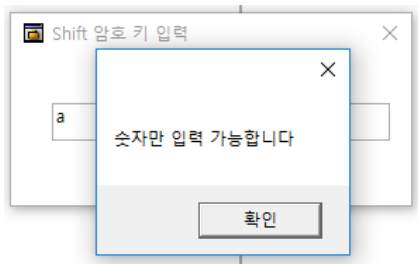

Fig. 11. Input Messages of Key Value in Shift and STC.

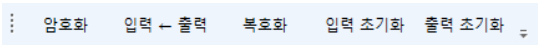

Fig. 12. Toolbar for Encryption.

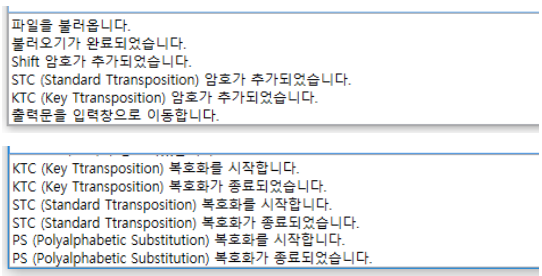

Fig. 13. Example of Log Message for All Steps.

출력 창에서 입력 창으로 이동시키는 기능을 한다. 입력 창으로 텍스트가 이동되면서 출력 창의 텍스트는 지워지도록 구현하였다. 복호화(Description)는 암호화 방법의 역순으로 수행된다. 본 프로그램에서는 암호화 할 때 입력한 암호화 방식의 순서와 키값을 동일하게 입력하면 암호화의 역순으로 복호화를 진행한다.

입력 초기화 버튼은 입력창의 문자를 모두 지운다. 이 버튼은 입력 창의 텍스트가 길어질 경우 직접 모든 텍스트를 삭제하는데 번거로울 수 있기 때문에 사용자의 편의를 위해 추가한 버튼이다. 출력 초기화 버튼은 출력 창의 텍스트를 모두 지운다. 출력 창은 사용자가 입력하거나 수정, 삭제할 수 없도록 되어 있어 텍스트를 지우고 싶을 때에는 이 버튼만을 사용하여야 한다.

본 프로그램에서 사용하는 모든 기능은 Fig. 13과 같이 로그를 출력하여 사용자가 자신이 사용한 기능을 확인할 수 있다. 처음 프로그램이 시작되면 '프로그램을 시작합니다'라는 로그가 등록된다. 프로그램의 기능을 사용할 때마다 '파일을 불러옵니다', '불러오기가 완료되었습니다', 'Shift 암호가 추가되었습니다', 'STC(Standard Transposition Cipher) 암호가 추가되었습니다' 등의 로그 메시지가 추가되고 사용자는 자신이 선택한 기능을 확인할 수 있다. 위의 Fig. 13과 같이 로그 메시지는 지속적으로 표시된다.

다음으로 본 논문에서 구현한 프로그램의 사용 예를 설명한다. 먼저 파일 불러오기를 통하여 암호화할 평문(plain text)을 불러온다. 불러온 텍스트가 프로그램의 입력 창에 출력되면 암호화할 방식을 선택하여 추가한다. 사용할 암호화 방식을 여러 개 추가한 후에 추가된 암호 버튼을 클릭하여 키값을 등록하도록 한다. 모든 키값을 등록한 후에 암호화 버튼을 눌러서 Fig. 14와 같이 입력문을 암호화 할 수 있다. 모든 프로그램 동작에는 로그가 표시되어 암호화의 과정에도 어떤 암호화 방식이 현재 실행 중인지 표시된다. 모든 암호화가 완료되면 출력 창에서 암호화된 텍스트를 확인할 수가 있다. 그리고 출력창에 표시된 텍스트를 저장하여 암호화 된 파일의 형태로 안전하게 보관할 수 있다.

복호화에 암호문을 이용할 시에도 암호화와 같은

Fig. 14. Encryption of Text Files.

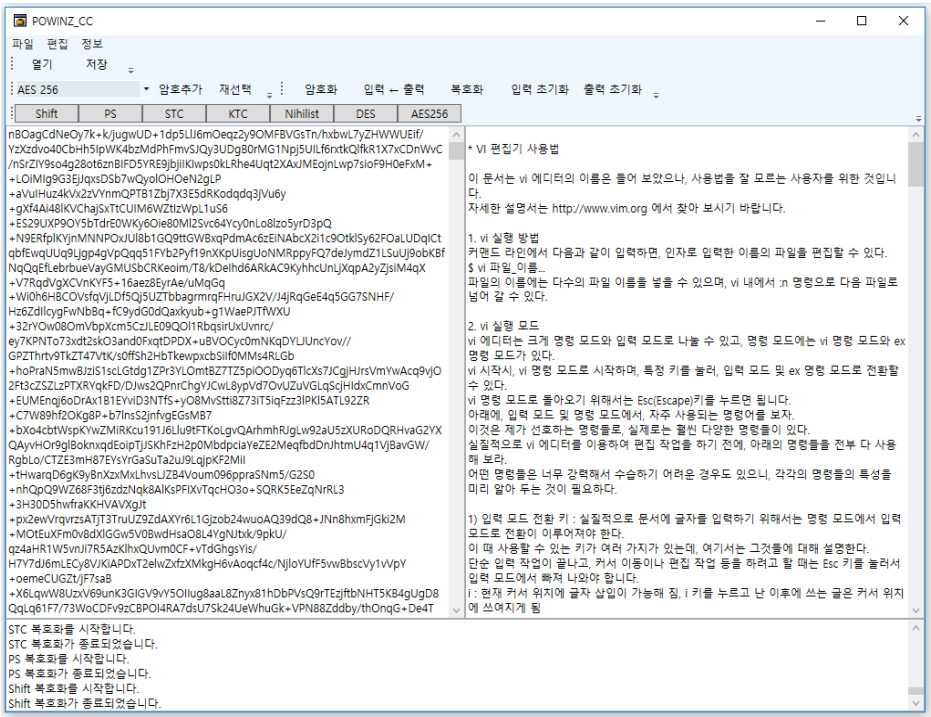

Fig. 15. Description of Encrypted Text.

방법으로 파일 불러오기를 이용하여 텍스트 파일을 불러온다. 불러온 텍스트 파일은 입력창에서 그 내용을 확인할 수 있다. 그 후 암호화에 사용했던 암호를 순서대로 추가하여 준다. 암호화에서 선택했던 것과 동일한 과정으로 키값 등록을 하도록 한다. 모든 입력 값이 설정되면, 복호화 버튼을 클릭함으로서 Fig. 15와 같이 복호화를 시작한다. 암호화 할 때와 마찬가지로 로그 창에서 복호화가 진행되는 과정이 표시된다. 복호화는 암호화 때와는 반대의 순서로 진행되며, 복호화가 완료되면 복원된 문서가 출력창에 표시된다. 이 또한 파일 저장을 통하여 복원된 텍스트를 저장할 수 있다.

5. 결 론

본 프로그램을 사용하여 텍스트를 암호화할 때 AES-256의 암호화 알고리즘을 사용하여 1회 암호화 하였을 경우와 혹은 2회 암호화 하였을 경우, 추가적으로 다양한 고대 암호와 융합하여 암호화 하였을 경우는 암호화 강도에 상당한 차이가 있을 것으로 예상된다. 이는 컴퓨터 성능에 의존적인 기존의 암호화 기법과는 달리 고대 암호와 현대 암호의 융합을 통하여 보다 안전하고 강도 높은 암호를 만들 수 있음을 의미한다.

이와 같이 여러 암호를 융합하여 암호화된 자료는 기존의 AES-256을 해독할 때 사용하던 해독 알고리즘으로는 원문을 복원할 수 없으며, 암호화에 사용한 모든 암호화 기법을 역순으로 적용하여야만 해독이 가능하다. 하지만 사용자가 여러 종류의 암호화 기법을 다양한 순서로 조합하여 자신만의 암호화 알고리즘을 만들어 사용한다는 사실은 암호화된 자료를 해독할 때에 고려해야 할 경우의 수가 기하급수적으로 증가하고, 자료의 보안 강도 또한 상당히 높아질 수 있음을 의미한다. 이것은 미래의 컴퓨터 성능이 크게 좋아지더라도 쉽게 복호화가 불가능하다는 것을 의미한다. 이와 같이 본 논문에서 제시하는 암호화 방식은 하나의 알고리즘에 의한 암호화가 아니라, 이미 검증된 여러 형태의 기법이 융합되었으므로 안전한 암호화가 가능하다. 미래에는 이와 같이 결합된 암호 방식의 안정성과 견고성[6-8], 저항성[9-11], 효율성[12-15]이 어떻게 좋아지는지를 계속하여 연구하고자 한다. 또한 기관이나 기업은 물론 일반인이 사용

하는 전자기기[16]에도 손쉽게 문서의 안전한 암호화가 실현되었다는 점에서 본 논문은 의의가 있다.

REFERENCE

[1] C. Paar, J. Pelzl, and B. Preneel, *Understanding Cryptography : A Textbook for Students and Practitioners*, Korean Publishing Company Green, 2013. Seoul.

[2] R.E. Smith, *Authentification from Passwords to Publish Keys,* Addison-Wesley, 2012. Boston.

[3] M. Lee, H. Yeom, Y. Song, and J. Kim., *Introduction to Information Protection,* Korean Publishing Company Hongreung, 2005. Seoul.

[4] H. Seo, “Implementing Software Passwords over the Internet,” *Communications of the Korean Institute of Information Scientists and Engineers,* Vol. 35, No. 1, pp. 8-15, 2017.

[5] N. Ferguson and B. Schneier, *Practical Cryptography,* Korean Publishing Company SciTech Media, 2004. GoYang.

[6] B.H. Kim, T.K. Kim, and J.H. Kim, "FPGA Implementation of the AES Cipher Algorithm by Using Pipelining," *KIISE Transactions on Computing Practices,* Vol. 8, No. 6, pp. 717-726, 2002.

[7] H.W. Nam, D.H. Kim, and N.S. Park, "Implementation of ARIA Encryption Algorithm based on WebCL," *Proceeding of Annual Meeting and Proceedings of Korea Computer Conference,* Vol. 42, No. 2, pp. 49-51, 2015.

[8] M.B. Song, M.K. Ko, and Y.M. Jung, "Hardware Using of the SEED Algorithm," *Proceedings of Korea Information Processing society,* Vol. 7, No. 2, pp. 1453-1456, 2000.

[9] K.H. Song, H.C. Kang, and J.C. Sung, "An Efficient New Format Preserving Encryption Algorithm to Encrypt the Personal Information," *Journal of the Korea Institute of Information Security and Cryptology,* Vol. 24, No. 4, pp. 753-763, 2014.

[10] S.J. Jang, "Design of the File Security Function Using Encryption Algorithm in the Windows Operating System," *The Korea Institute of Information and Communication Engineering,* Vol. 17, No. 3, pp. 612-618, 2013.

[11] T.H. Kim, M.Y. Jang, and J.W. Chang, "Hilbert-Curve Based Multi-Dimensional Indexing Key Generation Scheme and Query Processing Algorithm for Encrypted Databases," *Journal of Korea Multimedia Society,* Vol. 17, No. 10, pp. 1182-1188, 2014.

[12] M. Yoon, A.R. Cho, and J.W. Chang, "A Bitmap Encryption Scheme and a GPU-based Query Processing Algorithm for Spatial Database Outsourcing," *Journal of Korean Institute of Information Scientists and Engineers,* Vol. 41, No. 2, pp. 71-82, 2014.

[13] K.R. Lee, I.S. You, and K.B. Yim, "An Analysis of Agility of the Cryptography API Next Generation in Microsoft : Based on Implementation Example of Applying Cryptography Algorithm HAS-160 in South Korea," *Journal of the Korea Institute of Information Security and Cryptology,* Vol. 25, No. 6, pp. 1327-1339, 2015.

[14] K.R. Lee, I.S. You, and K.B. Yim, "An Analysis of a Structure and Implementation of Error-Detection Tool of Cryptography API-Next Generation in Microsoft," *Journal of the Korea Institute of Information Security and Cryptology,* Vol. 26, No. 1, pp. 153-168, 2016.

[15] T.W. Kwon, H.M. Kim, and S.H. Hong, "SEED and ARIA Algorithm Design Methods Using GEZEL," *Journal of the Korea Institute of Information Security and Cryptology*, Vol. 24, No. 1, pp. 15-29, 2014.

[16] B.H. Jeon, S.H. Shin, K.H. Jung, J.H. Lee, and K.Y. Yoo, "Reversible Secret Sharing Scheme Using Symmetric Key Encryption Algorithm in Encrypted Images," *Journal of Korea Multimedia Society,* Vol. 18, No. 11, pp. 1332-1341, 2015.

유 연 수

2006년 전주대학교 컴퓨터공학과 입학
2017년 전주대학교 컴퓨터공학과 재학중
관심분야 : 자연언어처리, 한국어 정보처리, 정보검색, 문서분류, 문서요약

이 상 곤

1994년 전주대학교 영어영문학과 (학사)
1996년 전북대학교 컴퓨터과학과 (학사)
1998년 전북대학교 전산통계학과 (이학석사)
2001년 일본 도쿠시마대학교 지능정보공학과(공학박사)
2002년~현재 전주대학교 공과대학 컴퓨터공학과 교수
관심분야 : 자연언어처리, 한국어 정보처리, 이메일문서처리, 한글공학, 정보검색, 문서분류, 문서요약