

# REPORT

## 키 생성



과목명 :		컴퓨터보안			
담당교수 :		정준호		교수님	
제출일 :	2021	년 06	월 02	일	
공과	대학	컴퓨터공학		과	
학번 :	2016112154	이름:		정동구	

# 1. 키 생성

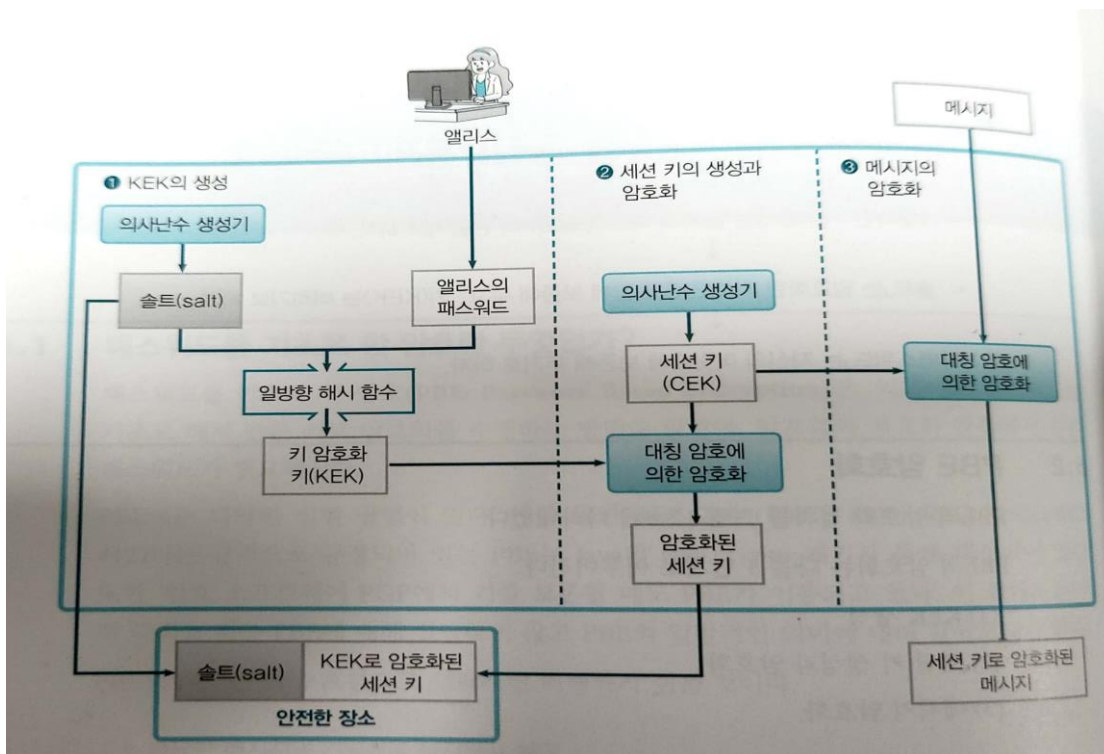
## 난수를 이용한 키 생성

난수를 사용하는 것이 키를 만드는 가장 좋은 방법이다. 키는 “다른 사람이 추측하기 어렵다”는 성질이 필요하기 때문이다. 때문에 암호용으로 이용하는 의사난수 생성기는 반드시 암호용으로 설계되어 있는 것을 사용해야 한다. 암호용으로 설계되지 않은 의사난수 생성기를 이용 할 경우 “예측 불가능”이라는 성질을 갖지 않기 때문에, 반드시 암호용으로 설계된 의사 난수 생성기를 이용하여야 한다.

## 패스워드를 이용한 키 생성

인간이 기억 할 수 있는 패스워드를 이용하여 키를 만드는 방법이다. 하지만 이 때 패스워드를 키로 직접 이용하는 일은 거의 없다. 일반적으로는 패스워드를 일방향 해시 함수를 통해서 얻어진 해시값을 키로 사용한다. 패스워드로 키를 생성할 때에는 사전 공격을 방지하기 위해 솔트(salt)라고 불리는 난수를 부가해서 일방향 해시 함수에 입력 한다. 이러한 방식을 패스워드를 기초로 한 암호(PBE)라 한다.

## PBE



[그림 1.PBE의 암호화 과정]

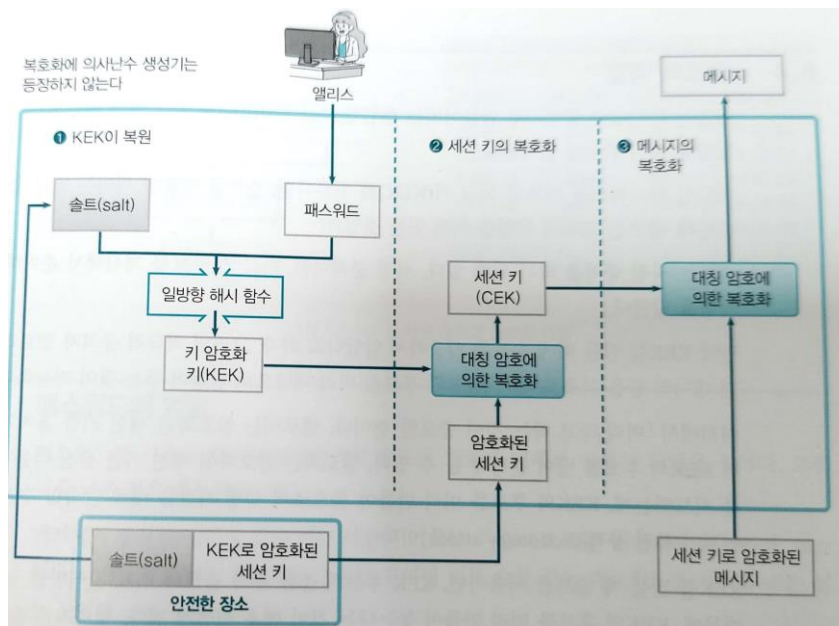
PBE 의 암호화는 다음 과 같은 과정을 거쳐서 일어난다.

1. KEK 생성
2. 세션 키 생성과 암호화
3. 메시지 암호화

KEK 는 우선 의사난수 생성기를 통해 솔트라는 난수를 생성한다. 이 솔트와 사용자가 입력한 패스워드를 합친 후 일방향 해시 함수에 입력하여 생성한다. 솔트는 사전공격을 방지하기 위해 사용한다.

세션 키는 통신 때 마다 한번만 사용하는 키로 솔트와 마찬가지로 의사난수 생성기를 통하여 생성한다. 세션키는 메시지를 암호화 하기 위한 키(CEK)가 된다. 세션 키는 생성한 KEK 를 통하여 암호화를 한 후 솔트와 함께 보관한다. KEK 는 보관 할 필요가 없는데 이는 솔트와 패스워드만 있으면 언제든지 복원이 가능하기 때문이다.

메시지의 암호화는 세션 키(CEK)를 통하여 대칭암호를 이용하여 암호화 한다.



[그림 2. PBE 복호화]

PBE 의 복호화는 다음과 같은 과정을 거친다

1. KEK 복원
2. 세션 키 복호화
3. 메시지 복호화

우선 패스워드와 보관해둔 salt 를 통해 KEK 를 복원한다. 이후 보관해둔 KEK 로 암호화 한 세션키를 가져와 복호화한다. 복호화한 세션키를 이용하여 메시지를 복호화한다.

## 2. 코드 및 결과

```
import hashlib #해시 함수
import secrets #CSPRNG
import sys

passwd = input('enter password :')
salt = secrets.token_hex(8)
CEK = secrets.token_hex(8)
passwd += salt.upper()

encoded_string = passwd.encode()
KEK = hashlib.shake_128(encoded_string).hexdigest(8)

#print(KEK+" "+session_key )
with open("KEK.txt", 'wt',encoding = 'utf-8') as f:
    f.write(KEK.upper())

with open("CEK.txt", 'wt',encoding = 'utf-8') as f:
    f.write(CEK.upper())

with open("salt.txt", 'wt',encoding = 'utf-8') as f:
    f.write(salt.upper())
```

[key.py]

키를 생성하는 파이썬 프로그램이다. 해싱함수는 hashlib 의 shake\_128 을 이용하여 8 바이트 크기의 해시값을 얻어내었다. 8 바이트의 해시값을 얻은 이유는 DES 기반의 CTR 암호를 이용할 것이기 때문에 DES 의 키 길이인 8 바이트를 맞추기 위해서이다.

우선 패스워드를 입력 받고 secrets 라이브러리에서 제공하는 암호용 의사난수 생성기를 이용하여 salt 를 생성한다. 두 문자열을 합친 후 shake\_128 을 이용하여 8 비트 길이의 해시값을 생성하였다. 직접 구현한 DES 가 대문자를 인식하여 16 진수를 나타내기 때문에 패스워드를 제외한 모든 값을 대문자로 변환하여 이용하였다. KEK 는 원래 저장 할 필요가 없지만, 복호화 과정에서 KEK 가 제대로 구해졌는지 확인 하기 위해 저장하였다.

```
import hashlib #해시 함수
import secrets #CSPRNG
import sys

passwd = input('enter password :')

with open("salt.txt", 'rt',encoding = 'utf-8') as f:
    salt = f.readline()
```

```

salt = salt.strip()


passwd +=salt

encoded_string = passwd.encode()
KEK = hashlib.shake_128(encoded_string).hexdigest(8)
print(salt)
print(KEK.upper())

```

[KEK.py]


복호화시 이용할 KEK 를 구하기 위한 KEK.py 이다. 패스워드를 입력받고, 저장해둔 salt 를 읽어와 마찬가지로 shake\_128 을 통해 해시값을 얻는다.

 salt.txt - Windows 메모장

파일(F) 편집(E) 서식(O) 보기(V) 도움말(H)

FBC7E33C7E92F089

[생성된 salt]

 CEK.txt - Windows 메모장

파일(F) 편집(E) 서식(O) 보기(V) 도움말(H)

4D79C425A027993B

[생성된 CEK]

```
(venv) PS D:\repos\security\Key> python ./KEK.py
enter password :ComputerScience
FBC7E33C7E92F089
3FEE3AE4E182BC07
(venv) PS D:\repos\security\Key> 
```

KEK.txt - Windows 메모장

파일(F) 편집(E) 서식(O) 보기(V) 도움말(H)

3FEE3AE4E182BC07

저장해둔 KEK 와 복호화를 위해 직접 구한 KEK 를 비교하였을 때 일치함을 알 수 있다.

```
std::ifstream readFile("test1.txt");//??
std::string plain_text;
std::getline(readFile, plain_text);

std::ifstream readKEK("../KEK.txt");
std::string KEK;
std::getline(readKEK, KEK);

std::ifstream readCEK("../CEK.txt");
std::string CEK;
std::getline(readCEK, CEK);

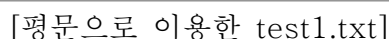
std::string CEK_encrypted;
DES EncCEK(CEK, KEK, MODE::ENCRYPTION);
CEK_encrypted = EncCEK.encryption();

std::ofstream writeCEK("CEK_encrypted.txt");
writeCEK << CEK_encrypted;


CTR ctr(plain_text, CEK);
ctr.encrypt();
std::string enc_str = ctr.getEnc();
std::ofstream writeEnc("enc_str.txt");
writeEnc << enc_str;
ctr.decrypt();
```

```
[main.cpp]
```

평문은 test1.txt 파일을 읽어와 생성하고 DES 기반의 CTR 블록 함수로 암호화를 진행하였다. DES 를 위한 Key 는 앞서 생성한 CEK 를 이용한다. 암호화를 마친 후 생성된 암호문은 enc\_str.txt 파일로 저장하였다. 복호화가 제대로 이루어지는지 보기 위함이다. CTR 클래스 안에 암,복호화가 전부 존재하므로 따로 암호화한 CEK 를 복호화하여 다시 입력하는 과정은 생략 하였다. CTR 을 통해 복호화하고 결과를 dec\_str.txt 파일에 저장하였다. 평문과 복호화된 문자열을 비교하였을 때 일치하면 일치한다는 문구를, 복호화가 제대로 되지않아 문자열이 다를경우 다르다는 문구를 출력하게 하였다.



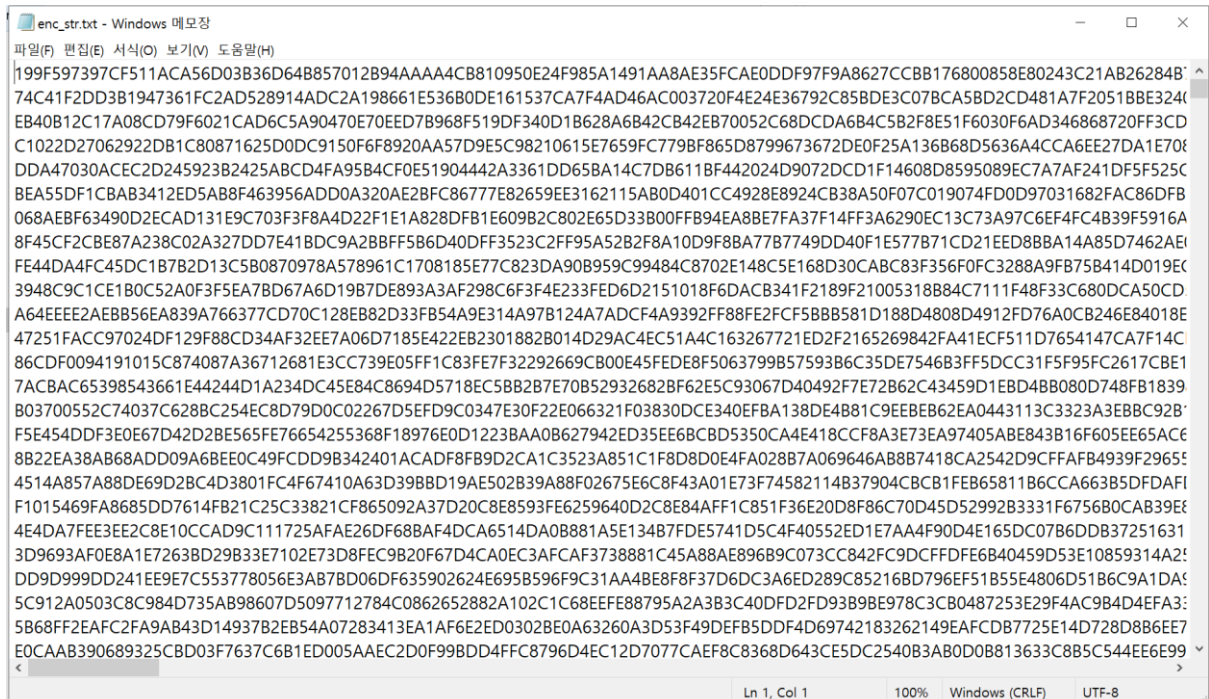


 CEK\_encrypted.txt - Windows 메모장

파일(F) 편집(E) 서식(O) 보기(V) 도움말(H)

A59B9489366530C5

[암호화된 CEK]



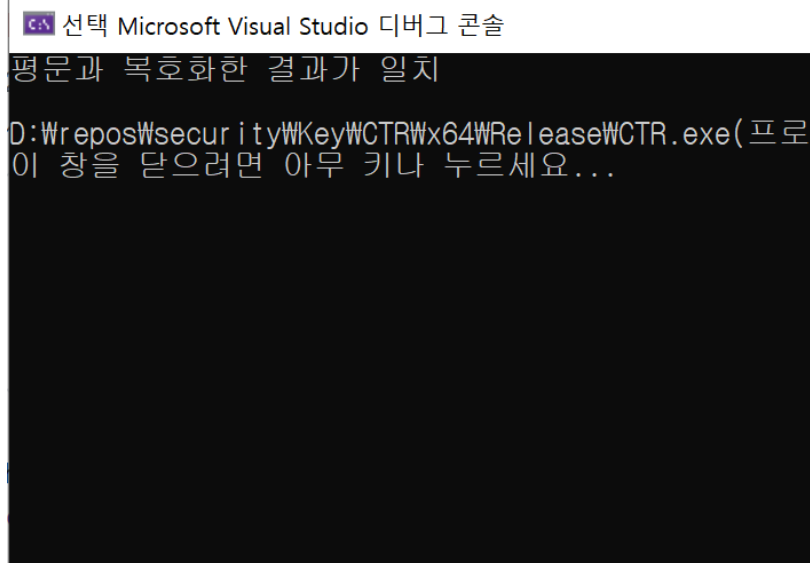
```
199F597397CF511ACA56D03B36D648857012B94AAAA4CB810950E24F985A1491AA8AE35FCAE0DDF97F9A8627CCBB176800858E80243C21AB26284B74C41F2DD3B1947361FC2AD528914ADC2A198661E536B0DE161537CA7F4AD46AC003720F4E24E36792C858DE3C07BCA5BD2CD481A7F2051BBE324CEB40B12C17A08CD79F6021CAD6C5A90470E70EED7B968F519DF340D1B628A6B42CB42EB70052C68DCDA6B4C5B2F8E51F6030F6AD346868720FF3CDC1022D27062922DB1C80871625D0DC9150F6F8920AA57D9E5C98210615E7659FC779BF865D8799673672DE0F25A136B68D5636A4CCA6EE27DA1E708DDA47030ACEC2D245923B2425ABCD4FA95B4CF0E51904442A3361DD65BA14C7DB611BF442024D9072DCD1F14608D8595089EC7A7AF241DF5F525CBEA55DF1CBAB3412ED5AB8F463956ADD0A320AE2BFC86777E82659EE3162115AB0D401CC4928E8924CB38A50F07C019074FD0D97031682FAC86DFB068AEBF63490D2ECAD131E9C703F3F8A4D22F1E1A828DFB1E609B2C802E65D33B00FFB94EA8BE7FA37F14FF3A6290EC13C73A97C6EF4FC4B39F5916A8F45CF2CBE87A238C02A327DD7E41BDC9A28BFF586D40DFF3523C2FF95A52B2F8A10D9F8BA77B7749DD40F1E577B71CD21EED8BBA14A85D7462AE1FE44DA4FC45DC1B7B2D13C5B0870978A578961C1708185E77C823DA90B959C99484C8702E148C5E168D30CABC83F356F0FC3288A9FB75B414D019EC3948C9C1CE1B0C52A0F3F5EA7BD67A6D19B7DE893A3AF298C6F3F4E233FED6D2151018F6DACB341F2189F21005318B84C7111F48F33C680DCA50CD A64EEEE2AEBB56EA839A766377CD70C128EB82D33FB54A9E314A97B124A7ADCF4A9392FF88FE2FCF5BBB581D188D4808D4912FD76A0CB246E84018E47251FACC97024DF129F88CD34AF32EE7A06D7185E422EB2301882B014D29AC4EC51A4C163267721ED2F2165269842FA41ECF511D7654147CA7F14C86CDF0094191015C874087A36712681E3CC739E05FF1C83FE7F32292669CB00E45FDE8F5063799B57593B6C35DE7546B3FF5DCC31F5F95FC2617CBE17ACBAC65398543661E44244D1A234DC45E84C8694D5718EC5BB2B7E70B52932682BF62E5C93067D40492F7E72B62C43459D1EBD4B8080D748FB1839B03700552C74037C628BC254EC8D79D0C02267D5EFD9C0347E30F22E066321F03830DCE340EFBA138DE4B81C9EEEBE62EA0443113C3323A3EBBC92B F5E454DDF3E0E67D42D2BE565FE76654255368F18976E0D1223BAA0B627942ED35EE68CBD5350CA4E418CCF8A3E73EA97405ABE843B16F605EE65AC68B22EA38AB68ADD09A6BEE0C49FCDD9B342401ACADF8FB9D2CA1C3523A851C1F8D8D0E4FA028B7A069646AB8B7418CA2542D9CFFAFB4939F296594514A857A88DE69D2BC4D3801FC4F67410A63D398BD19AE502B39A88F02675E6C8F43A01E73F74582114B37904CBCB1FEB65811B6CCA663B5DFDAF1F1015469FA8685DD7614FB21C25C33821CF865092A37D20C8E8593FE6259640D2C8E84AFF1C851F36E20D8F86C70D45D52992B3331F6756B0CAB39E84E4DA7FEE3EE2C8E10CCAD9C111725AFAE26DF68BAF4DCA6514DA08881A5E134B7FDE5741D5C4F40552ED1E7AA4F90D4E165DC07B6DB372516313D9693AF0E8A1E7263BD29B33E7102E73D8FEC9B20F67D4CA0EC3AFCAF3738881C45A88AE896B9C073CC842FC9DCFFDFE6B40459D53E10859314A29DD9D999DD241EE9E7C553778056E3AB78D06DF635902624E695B596F9C31AA4BE8F8F37D6DC3A6ED289C85216BD796EF51B55E4806D51B6C9A1DA95C912A0503C8C984D735AB98607D5097712784C0862652882A102C1C68EEFE88795A2A3B3C40DFD2FD93B9BE978C3CB0487253E29F4AC9B4D4EFA358B68FF2EAF2FA9AB43D14937B2EB54A07283413EA1AF6E2ED0302BE0A63260A3D53F49DEFB5DDF4D69742183262149EAFCD87725E14D728D8B6EE7E0CAAB390689325CBD03F7637C6B1ED005AAEC2D0F99BDD4FFC8796D4EC12D7077CAEF8C8368D643CE5DC2540B3AB0D0B813633C8B5C544EE6E99
```

[암호문]



```
dec_str.txt - Windows 메모장
파일(F) 편집(E) 서식(O) 보기(V) 도움말(H)
93E41C6E20911B9B36BC7CE94EDC677E32D83BB6F3AD985FD4BC655B3D9ACBE20A0E086D2926ED9CDF424060B4072B12621D1BFA0A9292D9C0E595
45927B6C6233940629442DC0E14830B276283C57E7E2B26C69E6A8D572A726C8EF00ACBFAF0DA94B061D258C8BA7389AEA241EF331BF02A98521C5
E750CB7BB4D61C711BCBCD9884B30AF6C9792E07DB27DD79FD0FFB4EB7B592AEC5F23FF235E365FA3DD80AA762564512723686C35B935E7B61BFA3(
880F2B79067994EC0E427E7F172FD33A0DD910B8BF6CF78682283DC8ECB30D84AAE5D344BBCA22A97394FEB34906539AF9DABCB8831CBDCC391D2/
2ABD7B88480B1D57F0C91F47BA9BAC7E40291258A39011931751703122C0786A80D7678742A1EE57AA6FD3DE10B66012716EE29CB8850B1D027AA0E
CB6BCB869AAE95C2E350B02E5DF766B28479040897D53CA0AB7AC3BB67DEF44066D9FBD9D877AA06D02AC8EAF7557D99F8F22881D30D696EEBE73/
6D292B95DC411D3DD6D861F5E0623FF6C8C906A97C1A46CD409406249CEC5F254BCB9F1C5E5E67B507E5AD05DE058B1170875E650B86C8BF8344BEI
0FD77A932EE385A8D85F02DD83DE083A0B2AF759605E60DAD4BD59ADD1EACA0B31BD226EE43523643DA08211A5859899F72B945A230F26008CA13
B086CAA271760E12CBD6A2A4264AD27E4F7AE9FA54938BE769E69C1716F736E116A0B6A17A1BDF13647C773C8D65A5117FBFC93E5B778451540FCC6
52342AA0B319868DBD6D538CB9A5AB8292CAEBAA38C8A5F4FD0FDE904805A1C7FCA25AF3F0E29CC29B375B475404A3990743FF2283F0E3A12D7C4/
F3F27ABF05BC0EF8A0E4F535C1174F6D61ADD4B2C0CCF018228211A80130CADD194EE3686C95881D1F240633BB4B0118ED73507AB7941F2F5D9D9I
95A0DABD575E8763926CA53BF8D4E3A197ACFFB0041EA1D17426483B5107883B78682881CA0143008BD257E1264CD99067C6B8BD3F2AF43CE3650E
365E2ACB99E10FDE85E3460283F8177E5DCBC19CF486042AAB7BB70DFA2EE3699D8816CA9286D1EF3E880A9AE913DB118D0091D00B6A0E94A794D7/
D80D79CAEB848749787AE7EA2654E0B2911BB24CE8BB2E374094FA763F3C5E4F727BA91D285D9D9E6544FFA5C0B3E8990594D7C423E35CE57F016EFF
7ACBD9D83D2600C47AF197B1B9C0AAF6D47BB4EDCCFF4944D4BD3CF0644AC925686D4D5FAE34594D9C0FD4C1A763F5118C280CA95C6BA36486E
1B7929D77FC9883F6D7838995C3C733A18CBA69DB034535169D68F69A95735084D5FD1A2341A16FCC2CAC8DC7E12039904CC429D84E4198710CB7(
CD2789E5C15CF0AA5F00E960FFA74C7E6B1B983DA4797D6EFD00C2E3EE55A0E1235175E4CAF1D2ABF995ADF855C210118C517872AC6D77D8E928F2/
6EE5D9E403FE791542878A47820316B2AF6C9AE88AD987B8229055C14631BC70943093741C88E5A3F5082033C722D9913E5AE66D4D6D529B19679D/
009428F25591F180350E2B1F257FEFF5E2CC8B8E7CE2B288174258D5597087ADEE369D79D7AE4B09661C772F03122B119B89E45BFC5E347A8AF30092C
A14288F1A73479FB2785DCF6C8EA8839261C8D3F6127DC95AB7B9B4F9E8EE283C42830CC6D8507B89CD75C3AEAC13899121D1A3F24D792CB635087
43F0D80FEAD7F2661A1C7CCE5B56827D6A6C7FDF456BF7A24094EDC8C38C5D69A92AC41EE36CC367C3A23155C17145119AA15F235C50F01C3BCD1
E5BE380D3C697AD11C942DA5FEC25BB1ADBC6180399011BFD4BE20320899C94F9F1C685179438016FA6D2561992153991236850874C95F6D042A95B
866C881C8E0CF24C0F1BCE7D912D24F5E11D63201DD53BC69D763BB4DA73425740EFC30F194CC520280A8C70C0601199DABBFACAC41ADBEC88/
381BD81AC0AF7BB7F1926F542499EE39346D55D1011A56D9FE00B62572B5AF0B5AF180E585F0087457F4FF9847707D99116EF1D1D4CA0B0FA5E5A225
D9C937291241F322E429102CC705B77D78BD5671F55E60E68229F9AEB7B31BE130F324381BD7C5239DBFD4B32E208B11A8F227C5FC436A507E5229E3
<
Ln 1, Col 1 100% Windows (CRLF) UTF-8
```

[복호문]



복호화 결과 기존 평문과 일치함을 볼 수 있다.

DES,CTR 을 포함한 전체 코드 : [security/Key at master · dsaf2007/security \(github.com\)](https://github.com/dsaf2007/security)

### 3. 소감

난수 혹은 패스워드 기반의 키를 생성하는 과제였는데, PBE 암호를 구현하기 위해서는 둘 다 이용하는 과정이 있었기 때문에 별도로 하지 않고, PBE 를 하면서 한번에 수행하였다. 난수를 이용하고 Key 를 생성하는 과정은 파이썬으로 구현하였다. C++에서 기본을 제공하는 메르센 트위스트 의사 난수 생성기는 “예측 불가능”이라는 암호용 난수생성기가 갖추어야 할 조건을 만족하지 않기 때문에 파이썬에서 제공하는 암호용 난수 생성 라이브러리를 이용하였다. 생성한 키를 바탕으로 암호화 하는 과정은 이전 과제에서 구현한 블록 암호인 DES 기반의 CTR 암호를 이용하였다. 해당 암호를 위해 생성하는 키인만큼 8 바이트 크기를 가지기 때 큰 키 공간 내에서 다양한 키를 생성하는 시간을 측정하는 것이 크게 의미있어 보이지 않아 따로 수행하지 않았다. 또한 충돌이 일어나지 않는것으로 알려진 일방향 해시함수를 이용하였기 때문에 주기성 또한 확인할 필요성을 별도로 느끼지 못하여 수행하지 않았다. 후에 내가 직접 구현한 일방향 해시 함수를 통해 키 생성을 수행 할 경우에는 위의 과정을 확인 해 보아야겠다.