1)

keys generated by the default KDF
(password for all iteration is “my_secret_password”

Prompt : openssl enc -aes-256-cbc -k "my_secret_password" -P -md sha1

enc: This subcommand is used for encryption and decryption operations.

-aes-256-cbc: This is the encryption algorithm and mode specified in the command. It uses AES (Advanced Encryption Standard) with a key size of 256 bits in Cipher Block Chaining (CBC) mode. AES is a widely used symmetric encryption algorithm known for its security and efficiency.

-k "my_secret_password": This option specifies the passphrase or password used to derive the encryption key. In this case, "my_secret_password" is the password used for key derivation.

-P: This option is used to print the salt, key, and IV (Initialization Vector) values to the standard output. The salt and key are used in the key derivation process, while the IV is used in the encryption process to ensure that even if the same plaintext is encrypted multiple times, the ciphertext will be different.

-md sha1: This option specifies the message digest algorithm used in key derivation. In this case, SHA-1 (Secure Hash Algorithm 1) is used. SHA-1 is a cryptographic hash function. However, it's important to note that the use of SHA-1 for key derivation is considered deprecated and less secure than newer alternatives.

When you run this command, it does the following:

It derives an encryption key using the provided password ("my_secret_password") and a randomly generated salt. The salt adds randomness to the key derivation process, making it more secure.

The salt, key, and IV values are displayed on the screen. These values are essential for encryption and decryption, and they are unique for each execution due to the use of a random salt.

The derived key can be used for encryption and decryption using the specified AES-256-CBC algorithm and mode.

It's important to note that the use of SHA-1 for key derivation is considered deprecated, and we can use more secure options, like using PBKDF2 with a higher iteration count, are recommended for key derivation in security-critical applications.

keys generated by the default KDF :

prompt that is used in OPENSSL : openssl enc -aes-256-cbc -k "my_secret_password" -P -md sha1

password for all iterations : "my_secret_password"

Iteration 1

salt=4D5B2E8F188AD5F9

key=0BAEEA23DE8EE441DE9BD1B0B0A00C4C331A2DDD80D585980154DAA491DAA097

iv =676196DBED917F89BBD8C54EDF5C1AC7

Iteration 2

salt=0CBED2F2F625892B

key=FE658886651D02E9D72B8D07970B1DE952E883941DC35DCE92AF1CF09B05DA1B

iv =BC316CEB97C5EC9B09FCEC6E9B9EC57D

Iteration 3

salt=EFFB32AE934EDA62

key=217286307E0583BD0FC1467EC095C0F667528EE366B0EFD8E90BEF952FA4CA47

iv =3A7C97DDD58E8BA01FE67B5F0B77B871

Iteration 4

salt=E2A75DB357EE6E33

key=CB8D3CCA653FD5529CE8EC601AF369BD50E5DA48E282258B98D5097BD68AB542

iv =A5BEADFB035260448D0F6A9B7F37630E

Iteration 5

salt=CA54C364B3F39FB7

key=0A01A5ED5FFDA6F149519A23A7B6D20381038F652AF77B1E3139F8A667F1762D

iv =CE6567D3A86A7FC96850787AE94A19A7

Iteration 6

salt=A8ACDEAAEA5BB496

key=072B4D135DBBCCA98758302228CDD2C279B2F7162AF8ED73CB8BB2DAEEBCA6BE

iv =686EB64AC26DA8F69F315361A65F23EF

Iteration 7

salt=4B46A8B3545C533E

key=89130888703AD270565E977590522EDB1AA9106D55168EF1112792B127E369BB

iv =05C2649A81348484592E62F4E7D4D0A9

Iteration 8

salt=A4D793C63ACBF4A3

key=CE64FC890E8670AE488D92E0AB38634F9DCF1A54E53BC03C1C3333667735BFA7

iv =6D41ADF4A53520B649E3D6D3107FCB14

Iteration 9

salt=D0CFF79980D2BFFB

key=184AC8CDC7E093F24933CC52A86901B01E00DAAB9E9D7E3CC01B1CED2778B7D7

iv =5A9AFCF39FB064477157B7F0E84E96B4

Iteration 10

salt=055E013BAA18BCB6

key=92499E62F86A75F14DC0CA4AA44A3FC0E16B959A1308682DCD78AFD96897B81D

iv =F70A06E07CBA3551D7C94091D8B41AF9

Iteration 11

salt=7CB9C8AFE6D4F854

key=82EEC1E5DB62F890F33A0EFC5F2762A7ADBF5C2638EE78DDC35BB350BB9A7F0C

iv =157E82FD36AB118E5BBFD209881DF9C8

Iteration 12

salt=7355481E4F3A4DB8

key=32E341566BEB00CA116BA942A8B690603BE2A779B3F912B2879FE04D4241EC9C

iv =F69E196B63CAA660ECE18B419B33A888

Iteration 13

salt=D9E555CD27407DE4

key=0F0869121E4DD7C0E16FBDB481CDCA3A65B994D47F64AD438883F500C2D5A51C

iv =F91ED03C2C8AB89CBC3C14A271912DB6

Iteration 14

salt=368650EF4562903F

key=FDCB99F083055D1B1672927058CE81DDE10CBFAFFA93BB96D59DA3664F7DB937

iv =F5AB4F29F93A5902F674A01DE1E968D0

Iteration 15

salt=18DC3BF8F461E565

key=E02B5546A70DA3856FE56D2F52103039AAF0957004B3BC179DB7323870612EA4

iv =644A5412BC3331833DF20DA7EC2FADAC

Iteration 16

salt=BDED74E345A86B58

key=292B2BD80779DACAD078900FE49B4544E68DA9644E6E5B6E86001B941CEC3AA8

iv =B08A0DA916EB06B334378BE42344CC95

Iteration 17

salt=5A678C9C9C484A24

key=4304C9E0C3D75DE6752CB03C04198BEB2E59CC1C153111C1A66005F326912D87

iv =E1F7A71ACB785A64978AF0500837473D

Iteration 18

salt=962DA76C34EA8F6D

key=3BB03F61FA339A27FA32542ADF2F49E6DD60779D6DA667BCEAB1871417D97C7B

iv =A6A6D08EB350CA45379E0831500869C6

Iteration 19

salt=F65B68538FE1D07F

key=500EA900AE54F0CC3FAA8B053BE4E170F8D6E3758B933390189762B99D98DB08

iv =4112F75EE449BCED3F1F71611392B5AB

Iteration 20

salt=5C90DB6F9763E57B

key=955001D7DD43D83B715B5701E0BF936BE75C617181DB4CBA4ADA001359A6541A

iv =06DD0176EFE5B83B153B953C39C87520

keys generated by the advanced KDF

password for all iteration is : “my_secret_password”

prompt : openssl enc -aes-256-cbc -k "my_secret_password" -P -md sha3-256 -pbkdf2 -iter 100000

-md sha3-256: This option specifies the message digest algorithm used during the key derivation process. In this case, SHA-3-256 (Secure Hash Algorithm 3 with a 256-bit output) is used. SHA-3 is a cryptographic hash function known for its security properties.

-pbkdf2: This option specifies that the PBKDF2 (Password-Based Key Derivation Function 2) algorithm should be used for key derivation. PBKDF2 is designed to derive a secure encryption key from a password and a salt, with the added benefit of a customizable number of iterations to make it computationally expensive for attackers to guess the password.

-iter 100000: This option specifies the number of iterations used in the PBKDF2 key derivation process. In this case, 100,000 iterations are used, which makes the key derivation process more computationally intensive and secure.

When you run this command, the following happens:

It derives an encryption key using the provided password ("my_secret_password"), a randomly generated salt, SHA-3-256 as the hashing algorithm, PBKDF2 as the key derivation function, and 100,000 iterations.

This command uses a strong combination of cryptographic techniques, including AES encryption, SHA-3 for hashing, PBKDF2 for key derivation, and a high number of iterations to enhance the security of the key derivation process. It is recommended for secure data encryption.

The keys generated by advanced KDF :

Iteration 1

salt=6BA0FEE3A99D66C3

key=0F04D4430B2E90159195D886C4B8814832E430928D7005034007B2C90C0FD9E0

iv =EA3A50B88D7F1128AE81579913B2E1C2

Iteration 2

salt=0BB5037F020ACE49

key=98C35775F5E405F787EEC2EB84E598D2957894832413474 7E9E50D8B01A04C3A

iv =D9BA4796D36C7679BF06498DA604EBD0

Iteration 3

salt=3622918EF3C62E12

key=4F3E8B28A3BCF7601F5E1A43E0A5282FF5BC4ADDB14E4B7B43D2C2B73884722B

iv =FFF830F90F2D4A023F90191366F33D8E

Iteration 4

salt=7331406323F37E83

key=FA9F72EC6539983A3D296CAE65E89814F538EEEDE120DF2347028E637E2A9A00

iv =8A0A1FCDF32AC2F072C0EDA350862A50

Iteration 5

salt=9FAE923B0FAC6B37

key=6621F1C4FAC82177BB9301285CAD3516E7F30B4C989B933254DEF6FFD132271C

iv =C424C79767A8BCF51951914FFEF46C91

Iteration 6

salt=20B8E048A70507F0

key=350E5E6DE167ADEAB5DBF10077A63060B82A3A74A5FF31B11B8BE4EA6C43139D

iv =EEA3B9AF0AC966C90D96FE91A37133E0

Iteration 7

salt=775209FC8554B68A

key=469B2DA808FE9645A6AE779443A9FB90D1A8A4F342D0FC80A465E0554E38628D

iv =24CD7B297CC2E8FF2457E9195C38BBAD

Iteration 8

salt=5AE818B7E4AA9C56

key=5FBC557D2D0D959F67380C26A39B17C586D9989EB554EFD42AE9725F99AB4BA3

iv =00F18001A0D97F607E42FB7CAF204BE0

Iteration 9

salt=E69ECE5636473C64

key=AABDF862EDD9A87BAD04FEF54727424C968CB6B30BB16C82150AAB1B9602FF8D

iv =480D9168C4A71317A334AFF4997B5F8E

Iteration 10

salt=E4F43663FBB66EF0

key=AE050AFBB160E0259330114EFD4E45B5F2417DC925288ABD419E7296C6435A68

iv =9EF07EDF9CAED830D23B34E0443FC1BC

Iteration 11

salt=2A5C705749C2AAA4

key=3CFBFAE4688A898661C84B013898D13AAC41C9B3E2615D6C872BAF11A3414CA7

iv =6BC4063280C00B384142B4D0CEF19AA8

Iteration 12

salt=18BEF849BD7A28F7

key=A34F6CD8B09340A8EE12B3239817F2025E4C3D434A370044425B0A49094FCE04

iv =5DF7EC2ADE356F967BEE0BDD2454F885

Iteration 13

salt=E0491FE1B9346F0B

key=DB4615684606A5CDC1EEB0BAD3B143ADBB0900497E346C56D8AFE2D1E8E3A465

iv =0F0F4EFA7E1211FEF62773243CAFD8F5

Iteration 14

salt=86ACD5A14F67F724

key=6CCC6F166F47667B75F67631C97A12F045ADD10F81AC21DE4A1269BBB8C5C1B3

iv =E9D667A3CA709844F7A0C156757D660E

Iteration 15

salt=22B685E7ACB157FC

key=C7958491B178781726FDF539E2C7D58103ACDAAEB6A3000819FDDF9AA79884E8

iv =B88EC92356AC3D004946EC468A8D2773

Iteration 16

salt=2A7DFBAB055B0C01

key=1FF0AB6857BF002824A2BB977F5A2E7C0FA1257CC1CE13EB5685F64B72DDC835

iv =3B72E2A1572DD0526729EB6F4266D0F2

Iteration 17

salt=96763F0636F729C5

key=266DE6D1D341592369BA271C00EF3E35E170FBE79E5A72C2F419CC7E3FDC1470

iv =40E56D0AF02AA83C3784D4AEBF83AB7E

Iteration 18

salt=29897A644A6CEB44

key=A176DA30A1847F2E8414803259754B446B2CA4D90398429875FD0B5262218BBE

iv =030E544C8EBAD460F55CE55DD4619FE3

Iteration 19

salt=994BF92969C7E428

key=E04333F2D190E04F2E43E1C20C2D1EAE905712F2458A31A4190005EC35DD3066

iv =7549E0A3E67FDCAC6EBB9B0F0B8787D0

Iteration 20

salt=6C9D8B7BA678E977

key=E07E6123CD0590C68614000C38158331DDD0F2475EBE1D34C1D12A4FF57B344C

iv =98736754D354DCC571951698EE478D09

2)Key derivation with default KDF and the same salt :

```
C:\Users\Asus>openssl enc -aes-256-cbc -k "my_secret_password" -S 88ED40A3EF2EF15B -P -md sha1
*** WARNING : deprecated key derivation used.
Using -iter or -pbkdf2 would be better.
salt=88ED40A3EF2EF15B
key=BC15330B61CBBDCF9E44CCAE106E59D14C75291C95D7111313B8E3505E0AD913
iv =08C54061540DCDB400BCC1DA7810B322

C:\Users\Asus>openssl enc -aes-256-cbc -k "my_secret_password" -S 88ED40A3EF2EF15B -P -md sha1
*** WARNING : deprecated key derivation used.
Using -iter or -pbkdf2 would be better.
salt=88ED40A3EF2EF15B
key=BC15330B61CBBDCF9E44CCAE106E59D14C75291C95D7111313B8E3505E0AD913
iv =08C54061540DCDB400BCC1DA7810B322
```

Key derivation with advanced KDF and with the same salt :

```
C:\Users\Asus>openssl enc -aes-256-cbc -k "my_secret_password" -S
2B7AE8432E7ABC73 -P -md sha3-256 -pbkdf2 -iter 100000
salt=2B7AE8432E7ABC73
key=DF07150940FEEB26E68F45F471B301818D2BC1708895DA7AD2C
326B1527333E5
iv =6BBCA61312A2D31405073119C3F841CA
```

```
C:\Users\Asus>openssl enc -aes-256-cbc -k "my_secret_password" -S
2B7AE8432E7ABC73 -P -md sha3-256 -pbkdf2 -iter 100000
salt=2B7AE8432E7ABC73
key=DF07150940FEEB26E68F45F471B301818D2BC1708895DA7AD2C
326B1527333E5
iv =6BBCA61312A2D31405073119C3F841CA
```

If you use the same password and the same salt for key derivation, the Key Derivation Function (KDF), such as PBKDF2 in this case, will generate the same key. This is a fundamental property of KDFs – when given the same input (password and salt), they produce the same output (key) to ensure consistency.
If you run this command multiple times with the same password and salt, you will obtain the same key, as long as the other parameters, such as the number of iterations and the hashing algorithm, remain the same. This consistency is by design and ensures that the same inputs always produce the same key. It's essential for predictable and consistent encryption and decryption processes.

3)ciphertext without salt :

```
 C:\Users\Asus>echo YourPlainText | openssl enc -aes-256-cbc -K
217286307E0583BD0FC1467EC095C0F667528EE366B0EFD8E90BEF952FA
4CA47 -iv 5A9AFCF39FB064477157B7F0E84E96B4  -e -a
```

```
BQ7dkIX958iBzehjlP21E7cBMeOEhL8h79nzskq76aI=
```

Another ciphertext with same plaintext and same key but with salt :

```
C:\Users\Asus>echo YourPlainText | openssl enc -aes-256-cbc -K
217286307E0583BD0FC1467EC095C0F667528EE366B0EFD8E90BEF952FA
4CA47 -iv 5A9AFCF39FB064477157B7F0E84E96B4 -S
A4D793C63ACBF4A3 -e -a
```

```
BQ7dkIX958iBzehjlP21E7cBMeOEhL8h79nzskq76aI=
```

Another ciphertext with same plaintext and same key but different salt :

```
C:\Users\Asus>echo YourPlainText | openssl enc -aes-256-cbc -K
217286307E0583BD0FC1467EC095C0F667528EE366B0EFD8E90BEF952FA
4CA47 -iv 5A9AFCF39FB064477157B7F0E84E96B4 -S 775209FC8554B68A -
e -a
```

```
BQ7dkIX958iBzehjlP21E7cBMeOEhL8h79nzskq76aI=
```

We always got the same ciphertext :

We can see salt doesn’t effect on ciphertext generation but salt has an important role in key derivation , but if I use different iv for the same plaintext and the same key I wil get different ciphertext .

Final report :

The salt is a randomly generated value that is unique for each encryption operation. It is combined with the user's password to create variability. This

means that even if two users have the same password, they will end up with different keys due to the different salts used in their respective key derivations.

The salt serves as a "spice" to make each key derivation unique. It is used to create different keys even when the same password is used, and it prevents attackers from taking shortcuts.

AES-256 encryption uses the 256-bit key length to encrypt as well as decrypt a block of messages. There are 14 rounds of 256-bit keys, with each round consisting of processing steps that entail substitution, transposition, and mixing plaintext to transform it into ciphertext.

The encryption has a key size of 256 bits, which is generated by KFD(default/advanced)

AES encryption is a symmetric cryptography algorithm. This means that the encryption and decryption process uses the same key for both processes

Key Expansion : The algorithm takes a single key up during the first stage. This is later expanded to multiple keys used in each round.

Key Length : The number of rounds of encryption to be carried out depends on the key length being used to encrypt data. The 256-bit key size has 14 rounds.

Byte Data : The AES encryption algorithm operates on byte data instead of bit data. This means that it treats the 128-bit block size as 16 bytes during the encryption process . AES uses block ciphers, where the plaintext is divided into sections called blocks. AES uses a 128-bit block size, whereby data is divided into 4-by-4 arrays that contain 16 bytes. Each byte contains 8 bits, with the total bits in every block being 128. In AES, the size of encrypted data remains the same. This means that 128 bits of plaintext yield 128 bits of ciphertext.

In all encryption, each unit of data is replaced by a different unit according to the security key used. AES is a substitution-permutation network that uses a key expansion process where the initial key is used to come up with new keys

called round keys. The round keys are generated over multiple rounds of modification. Each round makes it harder to break the encryption. The AES-256 encryption uses 14 such rounds.

AES works by having the initial key added to a block using an exclusive or (XOR) cipher. This is an operation that is built into processor hardware. In the block, each byte of data is substituted with another, following a predetermined table. The rows of the 4-by-4 array are shifted, with the bytes in the second row being moved one space to the left. Bytes in the third row are moved two spaces, and the ones in the fourth row moved three spaces. The columns are then mixed, combining the four bytes in each column, and the round key is added to the block. The process is repeated for each round, yielding a ciphertext that is completely different from the plaintext.