시큐리티 플랫폼 2021 인턴
CAVP 검증 Tools 개발

2021.07.27 김은주

01

목 표

목 표

axiocrypto의 암호 알고리즘 구현 정확성을 검증하는 CAVP Tools 개발

/dorsalstream/sys/axiocrypto_shellcmd/tools/kcmvp의 파일들을 모두 검증

1. Hash (ShortMsg, LongMsg, Monte)
2. HMAC-SHA256
3. PBKDF(HMAC-SHA256)
4. ARIA-CBC/CTR/GCM (KAT, MCT, MMT, AD, AE)
5. LEA-CBC/CTR/GCM (KAT, MCT, MMT, AD, AE)
6. ECDH-P256(KAKAT, KPG, PKV)
7. Hash-DRBG(no PR, use PR) KAT
8. ECDSA-P256-SHA256 (KPG, PKV, SGT, SVT)

02

axiocrypto_shellcmd 수정

cmd_hash.c 수정

```c
int cmd_hash(int argc, char *argv[])
{
    int ret;
    struct psz_t *msg;
    uint8_t digest[32];
    (void)argc;
    (void)argv;
    msg = ac_getenv("MSG");

    if (NULL == msg->p) {
        printf("MSG not set\n");
        usage_hash();
        return 0;
    }

    ret = axiocrypto_hash(HASH_SHA_256, msg->p, msg->sz, digest, 32);
    if (ret < 0) {
        printf("FAIL: axiocrypto_hash, ret = %d, %s\n", ret, axiocrypto_strerror(ret));
        return ret;
    }
    dumpbinary("MD", digest, 32, 1);
    psz_set_zero(msg);

    printf("OK\n");
    return 0;
}
DEFINE_SHELL_CMD(hash, "hash operation ", cmd_hash);
```

입력 길이가 0 byte일 때
처리하는 if문 추가

cmd_setenv.c 수정

```c
int acset_hexstr(struct psz_t * v, char *hexstr)
{
    if (v->p) {
        free(v->p);
        v->p = NULL;
        v->sz = 0;
    }
    v->sz = (strlen(hexstr) + 1) / 2;
    v->p = calloc(v->sz, sizeof(uint8_t));
    ac_assert_alloc_not_fail(v->p);
    return read_hexstring(hexstr, v->p, v->sz, &v->sz);
}
```

기존엔 함수 안에서 sz라는 변수를 새로 만들어서 했지만,
구조체 v의 sz를 그대로 사용하는 방식으로 변경

cmd_enc.c 수정

```c
if (ret == -8){
    printf("Invalid\n");
    goto cleanup;
}

else if (ret != CRYPTO_GCM_ACCEPT) {
    printf("FAIL: axiocrypto_sym_dec_GCM, %d %s\n",
                    ret, axiocrypto_strerror(ret));
    goto cleanup;
}
```

axiocypto_sym_dec_GCM = -8이 되는 경우,
Invalid 결과를 반환해야 해서 추가

cmd_pbkdf.c 수정

```c
static void usage_pbkdf(void)
{
    printf(
"usage: pbkdf {password} [Klen] [iteration]\n"
"        iteration: integer >= 1000, default 1000\n"
"        supported length: key 256bit\n"
"\n"
"\tEnv.variable SALT is used\n"
"\tAPIs used: axiocrypto_pbkdf()\n"
            );
}
```

```c
int cmd_pbkdf(int argc, char *argv[])
{
    int ret;
    uint8_t *pw;
    uint32_t iter = 1000;
    uint32_t pwsz;
    struct psz_t *salt;
    uint8_t key[64] = {0,};
    uint32_t klen = 0;

    salt = ac_getenv("SALT");
    pw = (uint8_t *)argv[1];
    pwsz = strlen(argv[1]);
    klen = atoi(argv[2]) >> 3;

    ret = axiocrypto_pbkdf
          (pw, pwsz, salt->p, salt->sz,
                     iter, key, klen);
```

항상 32-byte를 출력하는 것에서 Key length를
입력받아 그 길이만큼만 출력하는 것으로 변경

03

CAVP 코드 : HASH, HMAC

HASH CAVP Python

def main

① 검사할 파일 열기

② 저장할 파일 생성(.req)

③ algo_test() 생성

④ 파일 이름에 따라서 정보 저장

⑤ test type에 따라 진행

(1) long or short

(2) monte

⑥ 파일에 쓰기

```
def main(filename, comport):
① f = open(filename, "r")
  lines = f.readlines()
  print("==============================")
  print(filename)
② outfile = filename.replace(".txt",".req")
  print(outfile)
  w = open(outfile, "a")
  print("==============================")
③ test = algo_test()

④ if "sha2" in filename.lower():
      if "256" in filename.lower():
          test.algo = "sha256"
  if "short" in filename.lower():
      test.type = "short"
  elif "long" in filename.lower():
      test.type = "long"
  elif "monte" in filename.lower():
      test.type = "monte"

  ret = []
  for line in lines:
      words = line.split()
⑤     if (test.type == 'long') | (test.type == 'short'):…

      elif test.type == 'monte':…
```

class algo_test():
- total : 테스트 개수
- len : Msg 길이
- msg : 메세지
- algo : 알고리즘
- type : 검사 유형

HASH CAVP Python

def main

④ test type에 따라 진행

(1) long or short

SHA2(256)LongMsg.txt - Windows 메모장

파일(F) 편집(E) 서식(O) 보기(V) 도움말(H)

```
L = 32

Len = 1304
Msg = D3696760070A0D143CBC3EE3
MD = 076078B6482ADF7033FA165D5

Len = 2096
Msg = DFDD983718A7127A7B40471C
MD = D24F0ED53BF2B9F4E893664BFA
```

테스트 개수

Msg 길이

```
words = line.split()
if (test.type == 'long') | (test.type == 'short'):
    if len(words) == 0:
        continue
    elif words[0].upper() == 'L':
        test.total = words[2]
        w.write("L = {0}".format(test.total))
    elif words[0].upper() == "LEN":
        test.len = words[2]
    elif words[0].upper() == "MSG":
        test.msg = words[2]
        ret = test.run(comport)
        w.write("\n\nLen = {0}\n".format(test.len))
        w.write("Msg = {0}\n".format(line.split()[2]))
        ret_md = ''.join(ret.split())[2:]
        w.write("MD = {0}".format(ret_md.upper()))
    else:
        continue
```

HASH CAVP Python

def main

④ test type에 따라 진행

(2) monte

SHA2(256)Monte.txt - Windows 메모장

파일(F) 편집(E) 서식(O) 보기(V) 도움말(H)

```
L = 32

Seed = D6652F60526035A4BCB8AA0F

COUNT = 0
MD = A2BC5ED343F3C473D2AAADA9

COUNT = 1
MD = 5C60DA197AE822D1EB0F1EF14E
```

테스트 개수

```
elif test.type == 'monte':
    if len(words) == 0:
        continue
    elif words[0].upper() == 'L':
        test.total = words[2]
        w.write("L = {0}\n\n".format(test.total))
    elif words[0].upper() == "SEED":
        test.msg = words[2]
        w.write("Seed = {0}".format(test.msg))
        ret = test.run(comport)
        for cnt in range(0, 100):
            w.write("\n\nCOUNT = {0}\n".format(cnt))
            w.write("MD = {0}".format(ret[cnt+1].upper()))
        break
```

HASH CAVP Python

def run

```python
def run(self, comport):
    if self.algo != "sha256":
        return

    if self.type == "short":
        if self.len == '0':
            ac_run("set msg = _", comport)
        else:
            ac_run("set msg = {0}".format(self.msg), comport)
        calc = ac_run("hash", comport)
        print("calc = {0}\n".format(calc))
        return calc
```

0 Byte일 때

```python
elif self.type == "long":
    i = 0
    print("Msg = {0}...".format(self.msg[:150]))
    ac_run("hash2 init 100", comport)
    while True:
        shortmsg = self.msg[i*900:(i+1)*900]
        if len(shortmsg) == 0:
            break
        ac_run("hash2 update 100 {0}".format(shortmsg), comport)
        i = i + 1
    calc = ac_run("hash2 final 100", comport)
    return calc

elif self.type == "monte":
    ret_list = []
    print("seed = {0}".format(self.msg))
    ac_run("set seed {0}".format(self.msg), comport)
    ret_list = ac_run_monte("hash2 monte", comport)
    return ret_list
```

HASH CAVP Python

def ac_run

```python
def ac_run(cmd, comport):
    global comp
    init_serial(comport)
    ac_flush(comp)
    comp.write('{0}\n'.format(cmd).encode('ascii'))
    retline, ctline = "", None

    while(True):
        line = comp.readline()
        if cmd.startswith("hash") or cmd.startswith("hash2 final"):
            l = print_result(line)
            if (l):
                retline = l
                ll = l.lstrip()
                if ll.startswith("MD"):
                    ctline = ll.rstrip()
        if line == b'' or line.startswith("> \n".encode()):
            break
    if (ctline):
        return ctline
    return retline
```

결과를 바로 return

def ac_run_monte

```python
def ac_run_monte(cmd, comport):
    global comp
    init_serial(comport)
    ac_flush(comp)
    comp.write('{0}\n'.format(cmd).encode('ascii'))

    ret = []
    while(True):
        line = comp.readline()
        if cmd.startswith("hash") or cmd.startswith("hash2 final"):
            l = print_result(line)
            if line.startswith(">".encode()) or line.startswith("> \n".encode()):
                break
            elif l == None or l.startswith("COUNT"):
                continue
            else:
                ret.append(''.join(l.split())[3:])
                print("l = {0}".format(''.join(l.split())))
    return ret
```

결과를 list에 저장 후 return

HASH CAVP Python

한 개의 Input에 한 개의 Output

```
> hash2 init 100
OK
> hash2 update 100 DFDD983718A7127A7
D96EF356307020CEF74CAC9B758A6E0B5DAF0D1B5A2829FFEEB788434D938E37D0DB07EE378DFE9FA6C973
97400F86A77D41A63A827B326F9AEF03B996F11F074301163FD2F89DD2B13D261C1236985D3B6988186EA5
24F0072FA75BCEDB8F16AC2E2BD69DF7B616F3484692BC5C31629FEC354F5471CEC83D9F4FBCED0C6A3C08
2BB61CA2E165139671C6CA0
OK
> hash2 final 100
        MD d24f0ed5 3bf2b9f4  e893664b fa6da335  21ef0e3b 3a779579  7c63b41c 1bed85a0
OK
```

한 개의 Input에 여러 개의 Output

```
> set seed D6652F60526035A4BCB8AA0F6
32B
OK
> hash2 monte
COUNT = 0
     MD =  a2bc5ed3 43f3c473  d2aaada9 7b8e39d9  520442cf cad2f768  8f2f943c 622b738a
COUNT = 1
     MD =  5c60da19 7ae822d1  eb0f1ef1 4ee48f6a  08002142 6f59837b  cf1c1ae8 dcca9369
COUNT = 2
     MD =  bb7e7f66 22f1fe08  0318f0f0 e9d3577b  6373bfb7 4c077286  15e83e98 0623d0b5
```

HMAC CAVP Python

def main

① 검사할 파일 열기

② 저장할 파일 생성(.req)

③ algo_test() 생성

④ 파일 이름에 따라서 정보 저장

⑤ 파일에 쓰기

```
test.algo = "sha256"
cnt = 0
new_ret = None
for line in lines:
    words = line.split()
    if len(words) == 0:                continue
④   elif words[0].upper() == 'L':
        test.total = words[2]
        w.write("L = {0}".format(test.total))
    elif words[0].upper() == "KLEN":    test.klen = words[2]
    elif words[0].upper() == "TLEN":    test.tlen = words[2]
    elif words[0].upper() == "KEY":     test.key = words[2]
    elif words[0].upper() == "MSG":     test.msg = words[2]
    elif words[0].upper() == "MAC":
        test.hmac = words[2]
        ret = test.run(comport)
```

class algo_test():
- key : 키
- total : 테스트 개수
- klen : Msg 길이
- tlen : Mac 길이
- msg : 메세지
- algo : 알고리즘
- type : 검사 유형

HMAC CAVP Python

def main

⑤
```
w.write("\n\nCOUNT = {0}\n".format(cnt))
w.write("Klen = {0}\n".format(test.klen))
w.write("Tlen = {0}\n".format(test.tlen))
w.write("Key = {0}\n".format(test.key))
w.write("Msg = {0}\n".format(test.msg))

    if test.tlen == '16':
        new_ret = ''.join(ret.split()[1:5]).upper()
    elif test.tlen == '24':
        new_ret = ''.join(ret.split()[1:7]).upper()
    elif test.tlen == '32':
        new_ret = ''.join(ret.split()[1:]).upper()
    w.write("Mac = {0}".format(new_ret))

    cnt += 1

else:
    continue
```

def run

```
def run(self, comport):
    ac_run("init")
    ac_run("set msg = {0}".format(self.msg), comport)
    ac_run("set key = {0}".format(self.key), comport)
    calc = ac_run("hmac", comport)
    print("msg = {0}".format(self.msg))
    print("key = {0}".format(self.key))
    return calc
```

04

CAVP 코드 : PBKDF

PBKDF CAVP Python

def main

① 검사할 파일 열기

② 저장할 파일 생성(.req)

③ algo_test() 생성

④ 파일 이름에 따라서 정보 저장

⑤ 파일에 쓰기

```
for line in lines:
    words = line.split()
    if len(words) == 0:
        continue
    elif words[0].startswith("PRF", 1):
        w.write("\n{0}".format(line))
    elif words[0].startswith("Iteration", 1):
        test.iter = words[2][:-1]
        w.write("{0}".format(line))
        cnt = 0
    elif words[0].upper() == "PASSWORD":
        test.pwd_raw = words[2]

        index1 = test.pwd_raw.find('"')
        index2 = test.pwd_raw.find("'", 0)

        if index1 != -1 and index2 != -1:
            test.pwd = '\\' + test.pwd_raw[:index1] + '\\' + test.pwd_raw[index1:]
        elif index1 != -1:
            test.pwd = test.pwd_raw[:index1] + '\\' + test.pwd_raw[index1:]
        elif index2 != -1:
            test.pwd = '\\' + test.pwd_raw
        else:
            test.pwd = test.pwd_raw
```

class algo_test():
- iter : 반복 횟수
- salt : 입력 솔트 값
- pwd_raw
  : 파일 그대로의 pwd
- pwd : password
- klen : Key 길이
- mk : 결과

PBKDF CAVP Python

def main

```
elif words[0].upper() == "SALT":
    test.salt = words[2]
elif words[0].upper() == "KLEN":
    test.klen = words[2]
elif words[0].upper() == "MK":
    test.mk = words[2]
    ret = test.run(comport)
    new_ret = ''.join(ret.split()[1:]).upper()
⑤   w.write("\nCOUNT = {0}\n".format(cnt))
    w.write("Password = {0}\n".format(test.pwd_raw))
    w.write("Salt = {0}\n".format(test.salt))
    w.write("KLen = {0}\n".format(test.klen))
    w.write("MK = {0}\n".format(new_ret))
    cnt += 1

else:
    continue
```

def run

```
def run(self, comport):
    ac_run("set salt = {0}".format(self.salt), comport)
    if self.iter == '1000':
        calc = ac_run("pbkdf {0} {1}"
                      .format(self.pwd, self.klen), comport)
    else:
        calc = ac_run("pbkdf {0} {1} {2}"
                      .format(self.pwd, self.klen, self.iter), comport)
    print("Calc = ", calc)
    return calc
```

기본 iteration은 1000으로 고정

입력 받은 iteration이 있을 경우 else문으로 실행

PBKDF CAVP 실행 시간

		0.1	0.5	1	20	30	40	50	60	70	80	90
1000	128	X	O	-	-	-	-	-	-	-	-	-
	192	X	O	-	-	-	-	-	-	-	-	-
	256	X	O	-	-	-	-	-	-	-	-	-
	512	X	X	O	-	-	-	-	-	-	-	-
100000	128	X	X	X	X	X	X	O	-	-	-	-
	192	X	X	X	X	X	X	O	-	-	-	-
	256	X	X	X	X	X	X	O	-	-	-	-
	512	X	X	X	X	X	X	X	X	X	X	O

< 시간에 따른 실행 결과 >

PBKDF CAVP - run 함수 수정

```
def run(self, comport):
    ac_run("set salt = {0}".format(self.salt), comport, 0.1)
    tout = 0.0
    if self.iter == '1000':
        if self.klen == "512":
            calc = ac_run("pbkdf {0} {1}".format(self.pwd, self.klen), comport, 1.0)
        else:
            calc = ac_run("pbkdf {0} {1}".format(self.pwd, self.klen), comport, 0.5)
    else:
        if self.klen == "512":
            calc = ac_run("pbkdf {0} {1} {2}".format(self.pwd, self.klen, self.iter), comport, 90.0)
        else:
            calc = ac_run("pbkdf {0} {1} {2}".format(self.pwd, self.klen, self.iter), comport, 50.0)
    print("Calc = ", calc)
    return calc
```

기본 iteration은 1000으로 고정 / 입력 받은 iteration이 있을 경우 else문으로 실행

klen에 따라 실행 시간을 다르게 적용

05

CAVP 코드 : ARIA, LEA

ARIA CAVP Python

def main

① 검사할 파일 열기

② 저장할 파일 생성(.req)

③ algo_test() 생성

④ 파일 이름에 따라서 정보 저장

⑤ 운영 모드에 따라서 진행

(1) CBC/CTR (2) GCM

⑥ 파일에 쓰기

```
test = algo_test()
test.algo = "aria"

mode_list = ["gcm", "cbc", "ctr"]
for mode in mode_list:
    if mode in filename.lower():
        test.mode = mode

keysz_list = ["128", "192", "256"]
for sz in keysz_list:
    if sz in filename.lower():
        test.keysz = sz

test_list = ["kat", "mct", "mmt", "ae", "ad"]
for tname in test_list:
    if tname in filename.lower():
        test.operationcmd = tname
```

class algo_test():
- algo : 알고리즘
- iv : 입력 iv 값
- aad : GCM 입력 Adata 값
- tag : GCM 입력/출력 tag 값
- taglen : tag 길이
- pt : 평문
- ct : 암호문
- mode : 운영모드
- key : 입력 키
- keysz : 키 사이즈
- operationcmd : 검사 유형

ARIA CAVP Python

```python
for line in lines:
    words = line.split()
    if len(words) == 0:
        continue

    elif words[0] == "[KeyLen":
        test.keysz = int(words[2].split(']')[0])
        cnt = 0
        w.write("{0}".format(line))
    elif words[0] == "[IVLen":
        w.write("{0}".format(line))
    elif words[0] == "[PTLen":
        w.write("{0}".format(line))
    elif words[0] == "[AADLen":
        w.write("{0}".format(line))
    elif words[0] == "[TagLen":
        test.taglen = int(words[2].split(']')[0])
        w.write("{0}\n".format(line))

    elif words[0].upper() == "KEY":
        test.key = words[2]

    elif words[0].upper() == "IV" or words[0].upper() == "CTR":
        test.iv = words[2]
```

비어 있는 값일 때
_로 대체

```python
elif words[0].upper() == "PT":
    try:
        test.pt = words[2]
    except:
        test.pt = "_"

elif words[0].upper() == "TAG" or words[0].upper() == "T":
    try:
        test.tag = words[2]
    except:
        test.tag = "_"

elif words[0].upper() == "C" or words[0].upper() == "CT":
        try:
            test.ct = words[2]
        except:
            test.ct = "_"

elif words[0].upper() == "ADATA":
    try:
        test.aad = words[2]
    except:
        test.aad = "_"
```

ARIA CAVP Python - KAT or MMT

```python
elif words[0].upper() == "C" or words[0].upper() == "CT":
    if test.operationcmd == "mmt" or test.operationcmd == "kat":
        ret = test.run(comport) # ret = CT
        print("CT = {0}\n\n".format( ''.join(ret.split()[1:]).upper() ))

        w.write("KEY = {0}\n".format(test.key))
        w.write("IV = {0}\n".format(test.iv))
        w.write("PT = {0}\n".format(test.pt))
        w.write("CT = {0}\n\n".format( ''.join(ret.split()[1:]).upper() ))
```

def run

```python
def run(self, comport):
    if self.algo != "aria":
        return
    ac_run("init", comport)
    ac_run("set aria {0} {1}".format(self.keysz, self.mode), comport)
    if self.mode == "cbc" or self.mode == "ctr":
        ret = self.cmd_run(comport)
        return ret

    elif self.mode == "gcm":
        ret = self.cmd_run_gcm(comport)
        return ret
```

def cmd_run

```python
def cmd_run(self, comport):
    ac_run("set iv {0}".format(self.iv), c
    ac_run("set pt {0}".format(self.pt), comport)
    self.print_key(self.key)

    if self.operationcmd == "kat" or self.operationcmd == "mmt":
        ac_run("putkey3 100 aria {0}".format(self.key), comport)
        ret = ac_run("enc 100", comport)
    elif self.operationcmd == "mct":
        ac_run("set key {0}".format(self.key), comport)
        ret = ac_run_monte("enc 100 mct", comport)

    return ret
```

ARIA CAVP Python - MCT

```python
elif words[0].upper() == "C" or words[0].upper() == "CT":
    if test.operationcmd == "mmt" or test.operationcmd == "kat": ...
    elif test.operationcmd == "mct":
        ret = test.run(comport) # [key, iv, pt, ct]
        ret[0][0] = test.key
        ret[1][0] = test.iv
        ret[2][0] = test.pt
        for i in range(0, 100):
            w.write("COUNT = {0}\n".format(i))
            w.write("KEY = {0}\n".format(ret[0][i].upper()))
            if test.mode == "cbc":
                w.write("IV = {0}\n".format(ret[1][i].upper()))
            elif test.mode == "ctr":
                w.write("CTR = {0}\n".format(ret[1][i]))
            w.write("PT = {0}\n".format(ret[2][i].upper()))
            w.write("CT = {0}\n\n".format(ret[3][i].upper()))
        break
```

def run

```python
def run(self, comport):
    if self.algo != "aria":
        return
    ac_run("init", comport)
    ac_run("set aria {0} {1}".format(self.keysz, self.mode), comport)
    if self.mode == "cbc" or self.mode == "ctr":
        ret = self.cmd_run(comport)
        return ret

    elif self.mode == "gcm":
        ret = self.cmd_run_gcm(comport)
        return ret
```

def cmd_run

```python
def cmd_run(self, comport):
    ac_run("set iv {0}".format(self.iv), comport)
    ac_run("set pt {0}".format(self.pt), comport)
    self.print_key(self.key)

    if self.operationcmd == "kat" or self.operationcmd == "mmt":
        ac_run("putkey3 100 aria {0}".format(self.key), comport)
        ret = ac_run("enc 100", comport)
    elif self.operationcmd == "mct":
        ac_run("set key {0}".format(self.key), comport)
        ret = ac_run_monte("enc 100 mct", comport)

    return ret
```

ARIA CAVP Python - MCT

```python
# mct test
def ac_run_monte(cmd, comport):
    global comp
    init_serial(comport)
    ac_flush(comp)
    comp.write('{0}\n'.format(cmd).encode('ascii'))
    ret, ret_key, ret_iv, ret_pt, ret_ct = [], [], [], [], []

    while(True):
        line = comp.readline()

        if cmd.startswith("enc") or cmd.startswith("enc 100 mct"):
            l = print_result(line)
            print("line = {0}\n".format(l))
            if line.startswith(">".encode()) or line.startswith("> \n".encode()):
                break
            elif l == None:
                continue
```

```python
            else:
                ll = l.lstrip()
                if ll == None or ll.startswith("COUNT"):
                    continue
                elif ll.startswith("KEY"):
                    ret_key.append(''.join(ll.split())[3:])
                elif ll.startswith("IV"):
                    ret_iv.append(''.join(ll.split())[2:])
                elif ll.startswith("CTR"):
                    ret_iv.append(''.join(ll.split())[3:])
                elif ll.startswith("PT"):
                    ret_pt.append(''.join(ll.split())[2:])
                elif ll.startswith("CT"):
                    ret_ct.append(''.join(ll.split())[2:])

    ret = [ret_key, ret_iv, ret_pt, ret_ct]
    return ret
```

```
> enc 100 mct
COUNT = 0
       KEY 7c950d07 e6149892  07ac2241 4d232737
        IV 9dd562ce 3d07d989  f278194b 6539c3c6
        PT cbbf4735 c537f04e  85192172 3300de28
        CT 6a19181f e0f022a0  56fe9804 a0761df7
COUNT = 1
       KEY 168c1518 06e4ba32  5152ba45 ed553ac0
        IV 6a19181f e0f022a0  56fe9804 a0761df7
```

- 한 개의 Input에 여러 개의 Output이기 때문에 List에 저장
- Output 그대로 파일에 쓰기 위해서 결과뿐만 아니라 나머지도 저장
- ret은 2차원 list

ARIA CAVP Python - AE

```
COUNT = 0
Key = D7DFBF631ABB798ED80C27E7E2EEBF9E
IV = 7C2745B96015EDAD05E5802C2006003F
PT =
Adata =
```

ARIA128(GCM)AE.txt

```python
elif words[0].upper() == "ADATA":
    if test.operationcmd == "ae":
        ret = test.run(comport) # ret = [CT, Tag]
        print("C = {0}\n".format(ret[0]))
        print("T = {0}".format(ret[1]))

        w.write("COUNT = {0}\n".format(cnt))
        w.write("Key = {0}\n".format(test.key))
        w.write("IV = {0}\n".format(test.iv))
        if test.pt == "_":
            w.write("PT = \n")
        else:
            w.write("PT = {0}\n".format(test.pt))

        if test.aad == "_":
            w.write("Adata = \n")
        else:
            w.write("Adata = {0}\n".format(test.aad))

        w.write("C = {0}\n".format(ret[0]))
        w.write("T = {0}\n\n".format(ret[1].upper()))
        cnt += 1
```

def run

```python
def run(self, comport):
    if self.algo != "aria":
        return
    ac_run("init", comport)
    ac_run("set aria {0} {1}".format(self.keysz, self.mode), comport)
    if self.mode == "cbc" or self.mode == "ctr":
        ret = self.cmd_run(comport)
        return ret

    elif self.mode == "gcm":
        ret = self.cmd_run_gcm(comport)
        return ret
```

def run_gcm

```python
def cmd_run_gcm(self, comport):
    ac_run("putkey3 100 aria {0}".format(self.key), comport)
    ac_run("set iv {0}".format(self.iv), comport)
    ac_run("set aad {0}".format(self.aad), comport)
    self.print_key(self.key)
    if self.operationcmd == "ae":
        ac_run("set pt {0}".format(self.pt), comport)
        ret = ac_run_gcm("enc 100 {0}".format(self.taglen), comport)
    elif self.operationcmd == "ad":
        ac_run("set ct {0}".format(self.ct), comport)
        ac_run("set tag {0}".format(self.tag), comport)
        ret = ac_run_gcm("dec 100", comport)

    return ret
```

ARIA CAVP Python- AD

```
COUNT = 0
Key = F177A58888C0E3C9185A7531C3F02AE7
IV = 63045289713CC41F26FD264EDF3B68CF
Adata = 6594E027BFDB48709E52EEA7C5BC50D
C = dc1dcbbebac9
T = F36816CD
```

ARIA128(GCM)AD.txt

```
elif words[0].upper() == "TAG" or words[0].upper() == "T":
    if test.operationcmd == "ad":
        ret = test.run(comport) # ret = PT
        print("PT = {0}\n".format(ret))

        w.write("COUNT = {0}\n".format(cnt))
        w.write("Key = {0}\n".format(test.key))
        w.write("IV = {0}\n".format(test.iv))
        if test.aad == "_":
            w.write("Adata =\n")
        else:
            w.write("Adata = {0}\n".format(test.aad))
        if test.ct == "_":
            w.write("C =\n")
        else:
            w.write("C = {0}\n".format(test.ct))
        w.write("T = {0}\n".format(test.tag))
        if ret == "Invalid":
            w.write("Invalid\n\n")
        elif ret == "":
            w.write("PT =\n\n")
        else:
            w.write("PT = {0}\n\n".format(ret.upper()))
        cnt += 1
```

def run

```
def run(self, comport):
    if self.algo != "aria":
        return
    ac_run("init", comport)
    ac_run("set aria {0} {1}".format(self.keysz, self.mode), comport)
    if self.mode == "cbc" or self.mode == "ctr":
        ret = self.cmd_run(comport)
        return ret

    elif self.mode == "gcm":
        ret = self.cmd_run_gcm(comport)
        return ret
```

def run_gcm

```
def cmd_run_gcm(self, comport):
    ac_run("putkey3 100 aria {0}".format(self.key), comport)
    ac_run("set iv {0}".format(self.iv), comport)
    ac_run("set aad {0}".format(self.aad), comport)
    self.print_key(self.key)
    if self.operationcmd == "ae":
        ac_run("set pt {0}".format(self.pt), comport)
        ret = ac_run_gcm("enc 100 {0}".format(self.taglen), comport)
    elif self.operationcmd == "ad":
        ac_run("set ct {0}".format(self.ct), comport)
        ac_run("set tag {0}".format(self.tag), comport)
        ret = ac_run_gcm("dec 100", comport)

    return ret
```

ARIA CAVP Python - AE, AD

```python
# gcm-ae, ad test
def ac_run_gcm(cmd, comport):
    global comp
    init_serial(comport)
    ac_flush(comp)
    comp.write('{0}\n'.format(cmd).encode('ascii'))
    ret_list = ["null", "null"]

    while(True):
        line = comp.readline()
        if cmd.startswith("dec") or cmd.startswith("dec 100"):
            l = print_result(line)
            if line.startswith(">".encode()) or line.startswith("> \n".encode()):
                break
            elif l == None:
                continue
            else:
                ll = l.lstrip()
                if ll == None or ll.startswith("CT") or ll.startswith("IV") \
                                 or ll.startswith("AAD") or ll.startswith("TAG"):
                    continue
                elif ll.startswith("PT"):
                    ret = ''.join(ll.split())[2:]
                elif ll.startswith("Invalid"):
                    ret = "Invalid"
```

```python
        elif cmd.startswith("enc") or cmd.startswith("enc 100"):
            l = print_result(line)
            if line.startswith(">".encode()) or line.startswith("> \n".encode()):
                break
            elif l == None:
                continue
            else:
                ll = l.lstrip()
                if ll == None or ll.startswith("PT") or ll.startswith("IV") \
                                                     or ll.startswith("AAD"):
                    continue
                elif ll.startswith("TAG"):
                    ret_list[1] = ''.join(ll.split())[3:]
                elif ll.startswith("CT"):
                    ret_list[0] = ''.join(ll.split())[2:]
            ret = ret_list

        if line == b'> ' or line.startswith('>'.encode()):
            break

    return ret
```

- AE : PT만 return, Invalid인 경우도 존재

- AD : TAG와 CT 두 개를 return 해야 하므로 List로 저장

LEA CAVP Python

```python
test.init(comport)
test.algo = "lea"
for line in lines:
    words = line.split()
    if len(words) == 0:
        continue

    elif words[0] == "[Klen":
        test.keysz = int(words[2].split(']')[0])
        cnt = 0
        w.write("{0}".format(line))
    elif words[0] == "[Nlen":
        w.write("{0}".format(line))
    elif words[0] == "[Plen":
        w.write("{0}".format(line))
    elif words[0] == "[Alen":
        w.write("{0}".format(line))
    elif words[0] == "[Tlen":
        test.taglen = int(words[2].split(']')[0])
        w.write("{0}\n".format(line))
```

```python
if test.operationcmd == "ad":
    ret = test.run(comport) # ret = PT
    print("PT = {0}\n".format(ret))

    w.write("COUNT = {0}\n".format(cnt))
    w.write("K = {0}\n".format(test.key))
    w.write("N = {0}\n".format(test.iv))
    if test.aad == "_":
        w.write("A = \n")
    else:
        w.write("A = {0}\n".format(test.aad))
    if test.ct == "_":
        w.write("C = \n")
    else:
        w.write("C = {0}\n".format(test.ct.upper()))
    w.write("T = {0}\n".format(test.tag))
    if ret == "Invalid":
        w.write("INVALID\n\n")
    else:
        w.write("P = {0}\n\n".format(ret.upper()))
    cnt += 1
```

알고리즘 이름과 파일 쓰는 형식 등을 제외하고 ARIA와 유사

BLOCK CAVP Python

파일명에서 알고리즘, 모드, 키 길이, 검사 유형을 저장

```
test = algo_test()
name_list = ["aria", "lea", "gcm", "cbc", "ctr", "128", "192", "256", "kat", "mct", "mmt", "ae", "ad"]
test.algo, test.mode, test.keysz, test.operationcmd = [name for name in name_list if name in filename.lower()]
```

```
def run(self, comport):
    if self.algo != "aria" and self.algo != "lea":
        return

    if self.mode == "cbc" or self.mode == "ctr":
        ret = self.cmd_run(comport)
        return ret

    elif self.mode == "gcm":
        ret = self.cmd_run_gcm(comport)
        return ret

def init(self, comport):
    ac_run("init", comport)
    ac_run("set {0} {1} {2}".format(self.algo, self.keysz, self.mode), comport)
```

06

CAVP 코드 : ECDSA, ECDH

ECDSA CAVP Python

def main

① 검사할 파일 열기

② 저장할 파일 생성(.req)

③ algo_test() 생성

④ 파일 이름에 따라서 정보 저장

⑤ test type에 따라서 진행

(1) KPG (2) PKV

(3) SGT (4) SVT

⑥ 파일에 쓰기

```
def main(filename):
    f = open(filename, "r")
    lines = f.readlines()

    print("=============================")
    print(filename)
    outfile = filename.replace(".txt",".req")
    print(outfile)
    w = open(outfile, "a")
    print("=============================")

    init_serial()
    ac_flush(comp)

    test = algo_test()
    test.algo = "ecdsa"
```

class algo_test():
- algo : 알고리즘
- type : 테스트 종류
- qx : x 좌표 값
- qy : y 좌표 값
- msg : message
- hashed : 해시 값

ECDSA CAVP Python

KPG Test

```python
if "kpg" in filename.lower():
    test.type = "keypair"
    print("test.type = {0}".format(test.type))
    test_keypair(test, lines, w)
elif "pkv" in filename.lower():
    test.type = "pkv"
    test_pkv(test, lines, w)
elif "sgt" in filename.lower():
    test.type = "siggen"
    if "component" in filename.lower():
        test.hashed = True
    test_siggen(test, lines, w)
elif "svt" in filename.lower():
    test.type = "sigver"
    test_sigver(test, lines, w)
```

```python
def test_keypair(test, lines, w):
    for line in lines:
        # print("test_keyparir : line = {0}".format(line))
        words = line.split()
        if len(words) == 0:
            continue
        elif words[0].startswith("["):
            w.write("{0}\n".format(line))
            test.n = 10
            test.run_keypair(w)
        else:
            continue
```

```python
def run_keypair(self, w):
    set_serial_timeout(0.3)
    for i in range(self.n):
        ac_run("genkey 100 ecdsa")
        keys = ac_run_multilineoutput("showkey")
        if len(keys) > 1:
            print("X = {0}".format(keys[0].upper()))
            w.write("X = {0}\n".format(keys[0].upper()))
            w.write("Yx = {0}\n".format(keys[1].upper()))
            w.write("Yy = {0}\n\n".format(keys[2].upper()))
        ac_run("delkey 100")
        print('')
```

ECDSA CAVP Python

PKV Test

```python
if "kpg" in filename.lower():
    test.type = "keypair"
    print("test.type = {0}".format(test.type))
    test_keypair(test, lines, w)
elif "pkv" in filename.lower():
    test.type = "pkv"
    test_pkv(test, lines, w)
elif "sgt" in filename.lower():
    test.type = "siggen"
    if "component" in filename.lower():
        test.hashed = True
    test_siggen(test, lines, w)
elif "svt" in filename.lower():
    test.type = "sigver"
    test_sigver(test, lines, w)
```

```python
def test_pkv(test, lines, w):
    for line in lines:
        words = line.split()
        if len(words) == 0:
            if test.qx != None and test.qy!= None :
                test.run_pkv(test.qx, test.qy, w)
                test.qx = None
                test.qy = None
        elif words[0].upper() == "YX":
            test.qx = words[2]
        elif words[0].upper() == "YY":
            test.qy = words[2]
        elif words[0] == "#":
            print("{0}".format(line))
            w.write("{0}\n".format(line))
        else:
            continue
```

ECDSA CAVP Python

PKV Test

```
def run_pkv(self, qx, qy, w):
    pubkey = ''.join([qx,qy])
    ret = ac_run("putkey3 100 ecdsa _ {0}".format(pubkey))

    w.write("Yx = {0}\n".format(qx))
    w.write("Yy = {0}\n".format(qy))
    print("Qx = {0}".format(qx))
    print("Qy = {0}".format(qy))

    result = "FAIL" if len(ret) > 1 and ret.split()[0] == b'error:' else "PASS"
    w.write("Result = {0}\n\n".format(result[0]))
    print('Result = {0}'.format(result))
```

ECDSA CAVP Python

SG Test

```
if "kpg" in filename.lower():
    test.type = "keypair"
    print("test.type = {0}".format(test.type))
    test_keypair(test, lines, w)
elif "pkv" in filename.lower():
    test.type = "pkv"
    test_pkv(test, lines, w)
elif "sgt" in filename.lower():
    test.type = "siggen"
    if "component" in filename.lower():
        test.hashed = True
    test_siggen(test, lines, w)
elif "svt" in filename.lower():
    test.type = "sigver"
    test_sigver(test, lines, w)
```

```
def test_siggen(test, lines, w):
    for line in lines:
        words = line.split()
        if len(words) == 0:
            if test.msg != None:
                test.run_siggen(w)
                test.msg = None
        elif words[0].upper() == "M":
            test.msg = words[2]
            w.write("M = {0}\n".format(test.msg))
        elif line.startswith("["):
            w.write("{0}".format(line))
        else:
            continue
```

ECDSA CAVP Python

SG Test

```python
def run_siggen(self, w):
    ac_run("set MSG = {0}".format(self.msg))

    write_list = ["Yx", "Yy", "R", "S"]
    set_serial_timeout(0.5)
    ac_run("genkey 100 ecdsa")
    lines = ac_run_multilineoutput("sign 100")

    if len(lines) > 1:
        for i in range(0, 4):
            w.write("{0} = {1}\n".format(write_list[i], lines[i].upper()))
        w.write("\n")

    ac_run("delkey 100")
    print()
```

ECDSA CAVP Python

SV Test

```python
if "kpg" in filename.lower():
    test.type = "keypair"
    print("test.type = {0}".format(test.type))
    test_keypair(test, lines, w)
elif "pkv" in filename.lower():
    test.type = "pkv"
    test_pkv(test, lines, w)
elif "sgt" in filename.lower():
    test.type = "siggen"
    if "component" in filename.lower():
        test.hashed = True
    test_siggen(test, lines, w)
elif "svt" in filename.lower():
    test.type = "sigver"
    test_sigver(test, lines, w)
```

```python
def test_sigver(test, lines, w):
    for line in lines:
        words = line.split()
        if len(words) == 0:
            if test.msg != None:
                test.run_sigver(w)
                test.msg = None
        elif words[0].upper() == "M":
            test.msg = words[2]
        elif words[0].upper() == "YX":
            test.qx = words[2]
        elif words[0].upper() == "YY":
            test.qy = words[2]
        elif words[0].upper() == "R":
            test.r = words[2]
        elif words[0].upper() == "S":
            test.s = words[2]
        elif words[0].startswith("#"):
            w.write("{0}\n".format(line))
        elif words[0].startswith("["):
            w.write("{0}".format(line))
        else:
            continue
```

ECDSA CAVP Python

SV Test

```
def run_sigver(self, w):
    print("MSG = ", self.msg)
    ac_run("set msg {0}".format(self.msg))
    ac_run("set qx = {0}".format(self.qx))
    ac_run("set qy = {0}".format(self.qy))
    ac_run("set r = {0}".format(self.r))
    ac_run("set s = {0}".format(self.s))

    w.write("M  = {0}\n".format(self.msg))
    w.write("Yx = {0}\n".format(self.qx))
    w.write("Yy = {0}\n".format(self.qy))
    w.write("R = {0}\n".format(self.r))
    w.write("S = {0}\n".format(self.s))

    set_serial_timeout(1.0)
    ret = ac_run("verify 101")

    result = "FAIL" if len(ret) > 1 and ret.split()[0] == b'FAIL:' else "PASS"
    w.write("Result = {0}\n\n".format(result[0]))
    print('Result = {0}'.format(result))

    set_serial_timeout(0.1)
    ac_run("delkey 101")
    print()
```

ECDSA CAVP Python

```python
def ac_run(cmd):
    comp = get_serial()
    ac_flush(comp)
    comp.write('{0}\n'.format(cmd).encode('ascii'))
    # print(cmd)
    retline = ""
    while(True):
        line = comp.readline()
        if line == b'OK\n':
            retline = "OK"
        elif line.startswith("error".encode()) or line.startswith("FAIL".encode()):
            retline = line
        # print("ac_run : line = {0}".format(line))
        if line == b'' or line.startswith("> \n".encode()):
            break
    return retline
```

```python
def ac_run_multilineoutput(cmd):
    comp = get_serial()
    ac_flush(comp)
    comp.write('{0}\n'.format(cmd).encode('ascii'))
    ret = []

    while(True):
        line = comp.readline()
        if cmd.startswith("showkey") or cmd.startswith("sign 100"):
            l = print_result(line)
            print("ac_run_multi : line = {0}\n".format(l))
            if line.startswith("    ]".encode()) or line.startswith("OK".encode()):
                break
            if l == None:
                continue
            else:
                ll = l.lstrip()
                if ll.startswith("priv"):
                    ret.append(''.join(ll.split())[4:])
                elif ll.startswith("pub.x") or ll.startswith("pub.y"):
                    ret.append(''.join(ll.split())[5:])
                elif ll.startswith("Qx") or ll.startswith("Qy"):
                    ret.append(''.join(ll.split())[2:])
                elif ll.startswith("R") or ll.startswith("S"):
                    ret.append(''.join(ll.split())[1:])
                else:
                    continue
    return ret
```

def ac_run	한 개의 Input에 한 개의 Output
def ac_run_multi	한 개의 Input에 여러 개의 Output

ECDH CAVP Python

def main

① 검사할 파일 열기

② 저장할 파일 생성(.req)

③ algo_test() 생성

④ 파일 이름에 따라서 정보 저장

⑤ test type에 따라서 진행

(1) KPG (2) PKV

(3) KAKAT

⑥ 파일에 쓰기

```
def main(filename):
    f = open(filename, "r")
    lines = f.readlines()

    print("=============================")
    print(filename)
    outfile = filename.replace(".txt",".req
    print(outfile)
    w = open(outfile, "a")
    print("=============================")

    init_serial()
    ac_flush(comp)

    test = algo_test()
    test.algo = "ecdh"
```

class algo_test():
- algo : 알고리즘
- type : 테스트 종류
- privkey : 개인키
- qx : x 좌표 값
- qy : y 좌표 값
- msg : message
- hashed : 해시 값

ECDH CAVP Python

KPG Test

```python
if "kpg" in filename.lower():
    test.type = "keypair"
    print("test.type = {0}".format(test.type))
    test_keypair(test, lines, w)
elif "pkv" in filename.lower():
    test.type = "pkv"
    test_pkv(test, lines, w)
elif "kakat" in filename.lower():
    test.type = "kakat"
    if "component" in filename.lower():
        test.hashed = True
    test_kakat(test, lines, w)
```

```python
def test_keypair(test, lines, w):
    for line in lines:
        # print("test_keyparir : line = {0}".format(line))
        words = line.split()
        if len(words) == 0:
            continue
        elif words[0].startswith("["):
            w.write("{0}\n".format(line))
        elif words[0].upper() == "D":
            test.privkey = words[2]
            test.run_keypair(w)
        else:
            continue
```

```python
def run_keypair(self, w):
    set_serial_timeout(0.3)
    ac_run("genkey 100 ecdh")
    ac_run("putkey3 100 ecdh {0}".format(self.privkey))
    keys = ac_run_multilineoutput("showkey")
    if len(keys) > 1:
        for i in range(0, 3):
            keys[i] = keys[i][1:] if keys[i][0] == "0" else keys[i]
        w.write("d = {0}\n".format(keys[0].upper()))
        w.write("Qx = {0}\n".format(keys[1].upper()))
        w.write("Qy = {0}\n\n".format(keys[2].upper()))
    ac_run("delkey 100")
    print('')
```

ECDH CAVP Python

PKV Test

```
if "kpg" in filename.lower():
    test.type = "keypair"
    print("test.type = {0}".format(test.type))
    test_keypair(test, lines, w)
elif "pkv" in filename.lower():
    test.type = "pkv"
    test_pkv(test, lines, w)
elif "kakat" in filename.lower():
    test.type = "kakat"
    if "component" in filename.lower():
        test.hashed = True
    test_kakat(test, lines, w)
```

```
def test_pkv(test, lines, w):
    for line in lines:
        words = line.split()
        if len(words) == 0:
            if test.qx != None and test.qy!= None:
                test.run_pkv(test.qx, test.qy, w)
                test.qx = None
                test.qy = None
        elif words[0].startswith("["):
            w.write("{0}".format(line))
        elif words[0].upper() == "QX":
            test.qx = words[2]
        elif words[0].upper() == "QY":
            test.qy = words[2]
        else:
            continue
```

ECDH CAVP Python

PKV Test

```
def run_pkv(self, qx, qy, w):
    pubkey = ''.join([qx,qy])
    ret = ac_run("putkey3 100 ecdsa _ {0}".format(pubkey))

    w.write("Qx = {0}\n".format(qx))
    w.write("Qy = {0}\n".format(qy))
    print("Qx = {0}".format(qx))
    print("Qy = {0}".format(qy))

    result = "FAIL" if len(ret) > 1 and ret.split()[0] == b'error:' else "PASS"
    w.write("Result = {0}\n\n".format(result[0]))
    print('Result = {0}'.format(result))
```

ECDH CAVP Python

KAKAT Test

```
if "kpg" in filename.lower():
    test.type = "keypair"
    print("test.type = {0}".format(test.type))
    test_keypair(test, lines, w)
elif "pkv" in filename.lower():
    test.type = "pkv"
    test_pkv(test, lines, w)
elif "kakat" in filename.lower():
    test.type = "kakat"
    if "component" in filename.lower():
        test.hashed = True
    test_kakat(test, lines, w)
```

```
def test_kakat(test, lines, w):
    for line in lines:
        words = line.split()
        if len(words) == 0:
            if test.privkey != None:
                test.run_kakat(w)
                test.privkey = None
        elif line.startswith("[") or words[0].upper() == "J" or wor
            w.write("{0}".format(line))
        elif words[0].upper() == "RB":
            test.privkey = words[2]
            w.write("rB = {0}\n".format(test.privkey))
        elif words[0].upper() == "KTA1X":
            test.qx = words[2]
            w.write("KTA1x = {0}\n".format(test.qx))
        elif words[0].upper() == "KTA1Y":
            test.qy = words[2]
            w.write("KTA1y = {0}\n".format(test.qy))
        else:
            continue
```

ECDH CAVP Python

KAKAT Test

```
def run_kakat(self, w):
    ac_run("putkey3 100 ecdh {0}".format(self.privkey))
    ac_run("set qx {0}".format(self.qx))
    ac_run("set qy {0}".format(self.qy))
    keys = ac_run_multilineoutput("computekey 100")
    if len(keys) > 1:
        for i in range(0, 2):
            keys[i] = keys[i][1:] if keys[i][0] == "0" else keys[i]
    w.write("KABx = {0}\n".format(keys[0].upper()))
    w.write("KABy = {0}\n\n".format(keys[1].upper()))
    ac_run("delkey 100")
    print()
```

07

CAVP 코드 : Hash-DRBG

Hash-DRBG CAVP Python

def main

① 검사할 파일 열기

② 저장할 파일 생성(.req)

③ algo_test() 생성

④ 파일 이름에 따라서 정보 저장

⑤ test type에 따라서 진행

(1) no PR (2) use PR

⑥ 파일에 쓰기

```
def main(filename, comport):
    f = open(filename, "r")
    lines = f.readlines()
    print("=============================")
    print(filename)
    outfile = filename.replace(".txt",".req")
    print(outfile)
    w = open(outfile, "a")
    print("=============================")

    test = algo_test()
    test.algo = "hash_drbg"
    test.init()
    test.mode = "usepr" if "use" in filename.lower() else "nopr"
    cnt = 0
```

class algo_test():
- algo : 알고리즘
- mode : 테스트 종류
- nonce
- entropyinput
- personalizationstring
- entropyinputreseed
- additionalinputreseed
- additionalinput
- entropyinputpr

Hash-DRBG CAVP Python

def main

```
for line in lines:
    words = line.split()

    if len(words) == 0:
        if test.entropyinput != None:
            ret = test.run(comport)
            w.write("ReturnedBits = {0}\n".format(ret[3:].replace(" ", "").upper()))
            test.print_result_line("ReturnedBits", ret[3:].replace(" ", "").upper() + "\n")
            test.init()

    elif words[0].startswith("[S"):
        if cnt == 0:
            w.write("{0}".format(line))
        else:
            w.write("\n{0}".format(line))

    elif words[0].startswith("["):
        w.write("{0}".format(line))
        cnt = 0
```

```
elif words[0].upper() == "ADDITIONALINPUTRESEED":
    if len(words) > 2:
        test.additionalinputreseed = words[2]
        w.write("AdditionalInputReseed = {0}\n".format(test.additionalinputreseed))
    else:
        w.write("AdditionalInputReseed = \n")

elif words[0].upper() == "ADDITIONALINPUT":
    if len(words) > 2:
        temp = words[2]
        test.additionalinput.append(temp)
        w.write("AdditionalInput = {0}\n".format(temp))
    else:
        w.write("AdditionalInput = \n")
```

Hash-DRBG CAVP Python

def run

```
def run(self, comport):
    n_list = ["entropyinput", "nonce", "personalizationstring", "entropyinputreseed",
    v_list = [self.entropyinput, self.nonce, self.personalizationstring, self.entropyi

    if self.nonce == None:
        return

    for v in v_list:
        if v != None:
            ac_run("set {0} {1}".format(n_list[v_list.index(v)], v), comport)

    if len(self.entropyinputpr) == 2:
        ac_run("set EntropyInputPR0 = {0}".format(self.entropyinputpr[0]), comport)
        ac_run("set EntropyInputPR1 = {0}".format(self.entropyinputpr[1]), comport)

    if len(self.additionalinput) == 2:
        ac_run("set AdditionalInput0 = {0}".format(self.additionalinput[0]), comport)
        ac_run("set AdditionalInput1 = {0}".format(self.additionalinput[1]), comport)

    ret = ac_run("drbg {0}".format(self.mode), comport)
    self.print_result()

    return ret
```

```
n_list = ["entropyinput",
    "nonce",
    "personalizationstring",
    "entropyinputreseed",
    "additionalinputreseed"]
```

```
v_list = [self.entropyinput,
    self.nonce,
    self.personalizationstring,
    self.entropyinputreseed,
    self.additionalinputreseed]
```

08

실행 방법

1. Putty 실행 방법 - README.md 참고

1. 보드 초기화 후에는 init 필요

   *python 파일에는 init 과정이 있는 것도 있지만, 시간이 오래 걸려 다음 과정으로 안 이어짐

2. Set의 경우, 입력 순서 상관 없음

3. 입력 데이터가 없는 경우, input 자리에 _를 넣어 실행하거나,

   실행하지 않아도 됨

4. handle 값은 아무 숫자나 상관 없지만, 거의 100으로 사용

5. PBKDF의 iteration은 기본 1000, 그 이외에 숫자는 입력 필요

   *iter = 1000일 때, PBKDF {pwd} {klen} 1000 or PBKDF {pwd} {klen}
   *iter = 100000일 때, PBKDF {pwd} {klen} 100000

1. Putty 실행 방법 - README.md 참고

6 PBKDF 입력 시 pwd에 맨 처음에 ‘(작은따옴표)로 시작하거나, 중간에 “(큰 따옴표)가 있는 경우 해당 위치 앞에 ₩ 추가 필요

*ex) pwd : ‘a?r”&phz0a => input_pwd : ₩’a?r₩”&phz0a

7 ECDH PKV Test에서, 키 입력 시 “putkey3 {handle} ecdh _ {input_key}” 를 할 경우 오류 발생 => ecdsa로 사용

2. python 실행 방법

1 CAVP Tool 실행

```
python {cavp_file.py} "Test File.req" [COMPORT]
```

ex)
```
>python pbkdf_cavp.py "PBKDF(HMAC-SHA256)KAT.txt" COM3
```

2 파일 확인

```
python compare.py “Right File" “Check File”
```

ex)
```
>python compare.py "PBKDF(HMAC-SHA256)KAT.txt" "PBKDF(HMAC-SHA256)KAT.req"
```

SAME FILE1 AND FILE2	NOT THE SAME
파일 내용 동일	파일 내용 다름