Bcrypt.c

융합보안학과 윤세영

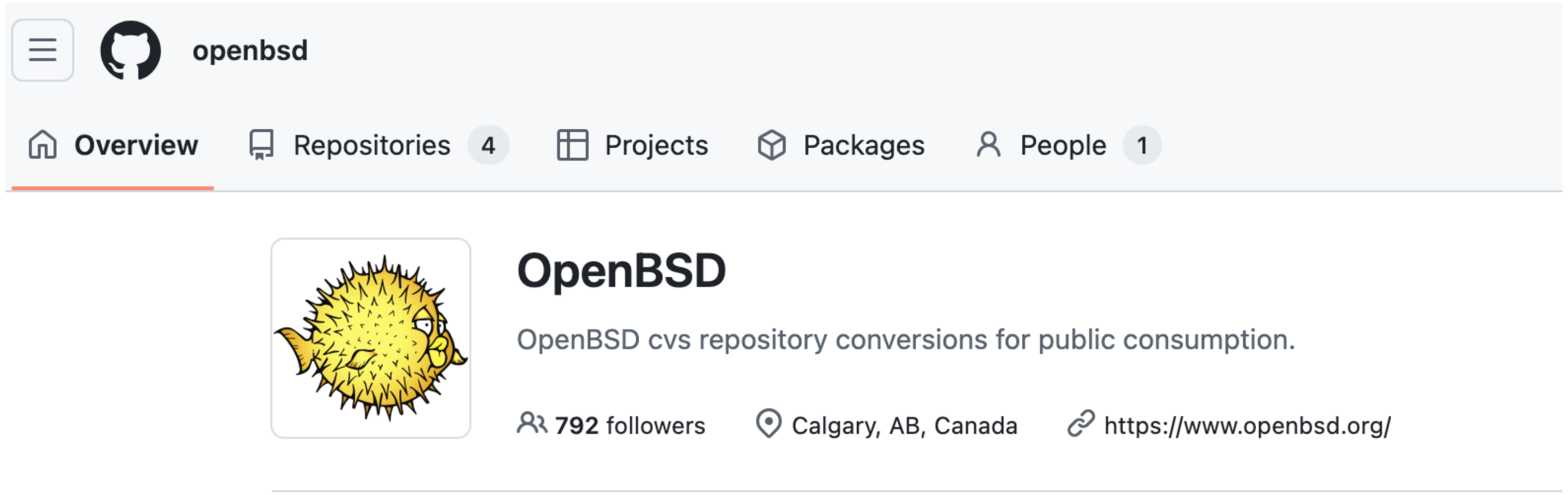
https://youtu.be/LJ4KKabR0Hk

목차

OpenBSD

https://cvsweb.openbsd.org/cgi-bin/cvsweb/src/lib/libc/crypt/bcrypt.c

https://github.com/openbsd/src/blob/master/lib/libc/crypt/bcrypt.c



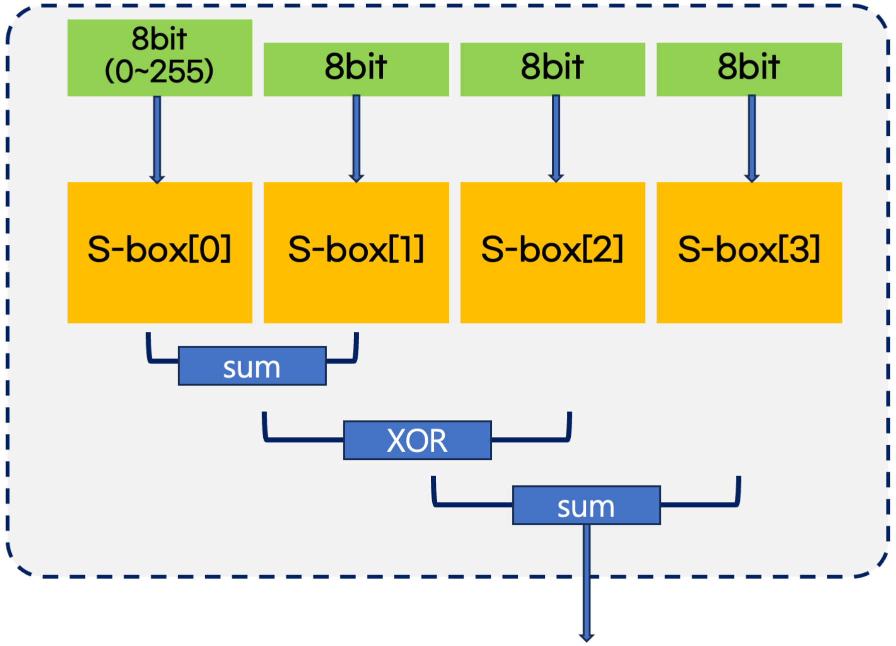
OpenBSD

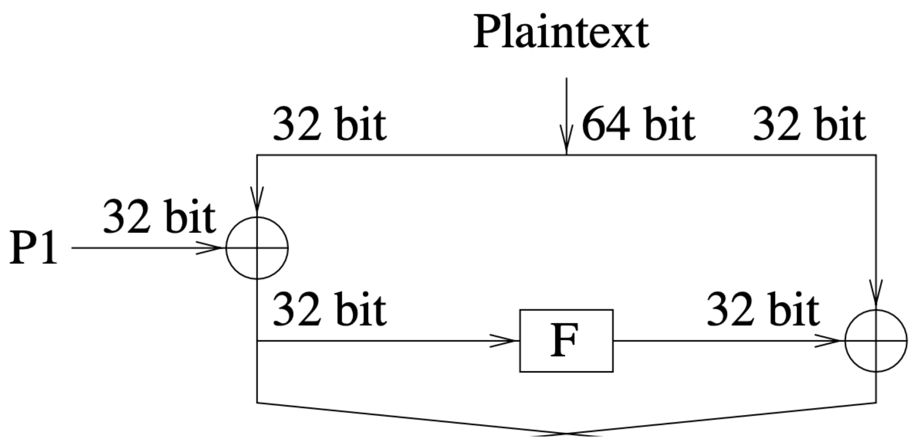
OpenBSD cvs repository conversions for public consumption.

792 followers · Calgary, AB, Canada · https://www.openbsd.org/

OpenBSD - Blowfish.c. Github : https://github.com/openbsd/src/blob/master/lib/libc/crypt/blowfish.c

```
#define F(s, x) ((((s)[        (((x)>>24)&0xFF)]  \
                + (s)[0x100 + (((x)>>16)&0xFF)]) \
                ^ (s)[0x200 + (((x)>> 8)&0xFF)]) \
                + (s)[0x300 + ( (x)      &0xFF)])

#define BLFRND(s,p,i,j,n) (i ^= F(s,j) ^ (p)[n])
```

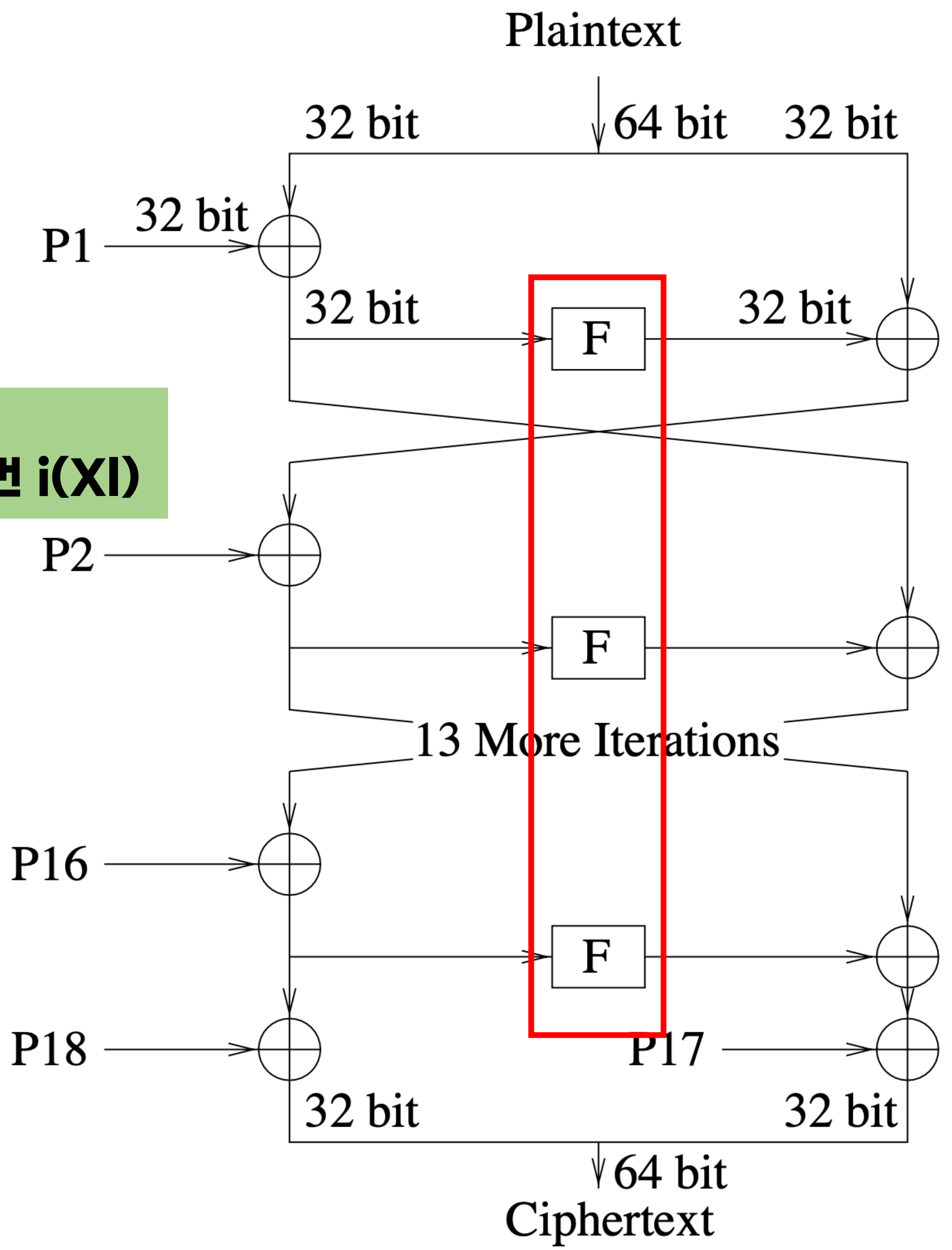




OpenBSD - Blowfish.c. Github : https://github.com/openbsd/src/blob/master/lib/libc/crypt/blowfish.c

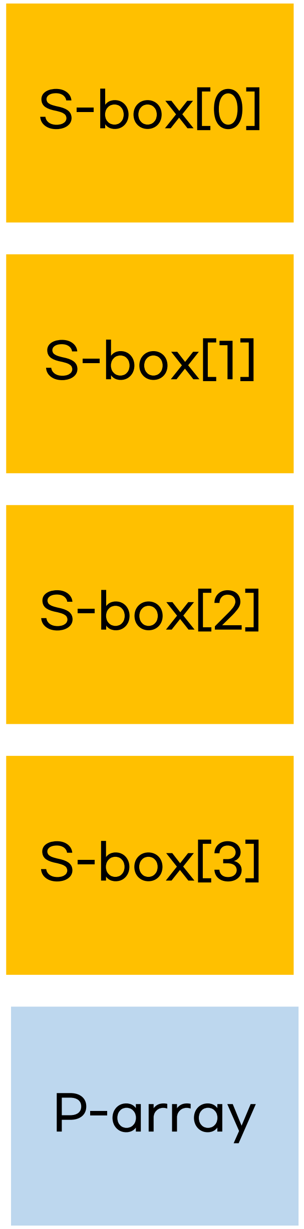
```
void
Blowfish_encipher(blf_ctx *c, u_int32_t *xl, u_int32_t *xr)
{
        u_int32_t Xl;
        u_int32_t Xr;
        u_int32_t *s = c->S[0];
        u_int32_t *p = c->P;

        Xl = *xl;
        Xr = *xr;

        Xl ^= p[0];
        BLFRND(s, p, Xr, Xl, 1); BLFRND(s, p, Xl, Xr, 2);
        BLFRND(s, p, Xr, Xl, 3); BLFRND(s, p, Xl, Xr, 4);
        BLFRND(s, p, Xr, Xl, 5); BLFRND(s, p, Xl, Xr, 6);
        BLFRND(s, p, Xr, Xl, 7); BLFRND(s, p, Xl, Xr, 8);
        BLFRND(s, p, Xr, Xl, 9); BLFRND(s, p, Xl, Xr, 10);
        BLFRND(s, p, Xr, Xl, 11); BLFRND(s, p, Xl, Xr, 12);
        BLFRND(s, p, Xr, Xl, 13); BLFRND(s, p, Xl, Xr, 14);
        BLFRND(s, p, Xr, Xl, 15); BLFRND(s, p, Xl, Xr, 16);

        *xl = Xr ^ p[17];
        *xr = Xl;
}
```

Xl(j) ^ P[0] => j
F(s, j(Xl)) ^ i(Xr) ^ P[n=1] => 다음번 i(Xl)



OpenBSD - Blowfish.c. Github : https://github.com/openbsd/src/blob/master/lib/libc/crypt/blowfish.c

```
void
Blowfish_initstate(blf_ctx *c)
{
	/* P-box and S-box tables initialized with digits of Pi */

	static const blf_ctx initstate =
	{ {
		{
			0xd1310ba6, 0x98dfb5ac, 0x2ffd72db, 0xd01adfb7,
			0xb8e1afed, 0x6a267e96, 0xba7c9045, 0xf12c7f99,
			0x24a19947, 0xb3916cf7, 0x0801f2e2, 0x858efc16,
			0x636920d8, 0x71574e69, 0xa458fea3, 0xf4933d7e,
			0x0d95748f, 0x728eb658, 0x718bcd58, 0x82154aee,
			0x7b54a41d, 0xc25a59b5, 0x9c30d539, 0x2af26013,
			0xc5d1b023, 0x286085f0, 0xca417918, 0xb8db38ef,
			0x8e79dcb0, 0x603a180e, 0x6c9e0e8b, 0xb01e8a3e,
			0xd71577c1, 0xbd314b27, 0x78af2fda, 0x55605c60,
			0xe65525f3, 0xaa55ab94, 0x57489862, 0x63e81440,
			0x55ca396a, 0x2aab10b6, 0xb4cc5c34, 0x1141e8ce,
			0xa15486af, 0x7c72e993, 0xb3ee1411, 0x636fbc2a,
			0x2ba9c55d, 0x741831f6, 0xce5c3e16, 0x9b87931e,
			0xafd6ba33, 0x6c24cf5c, 0x7a325381, 0x28958677,
```

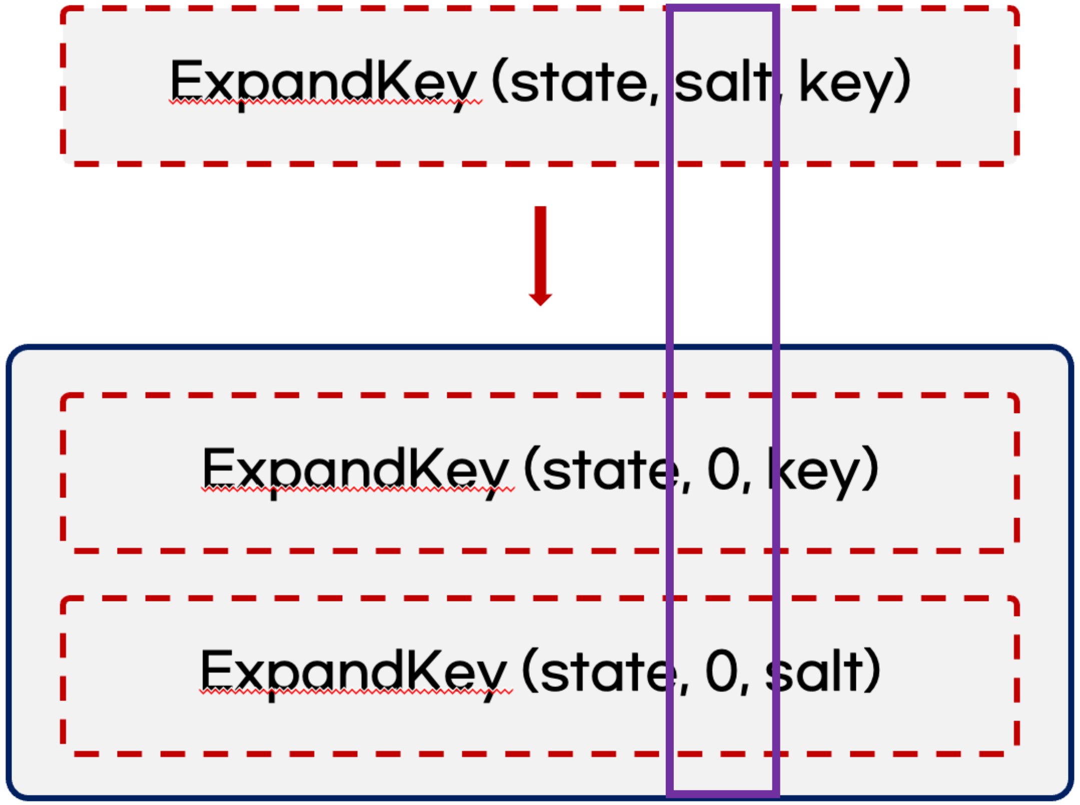


HANSUNG UNIVERSITY

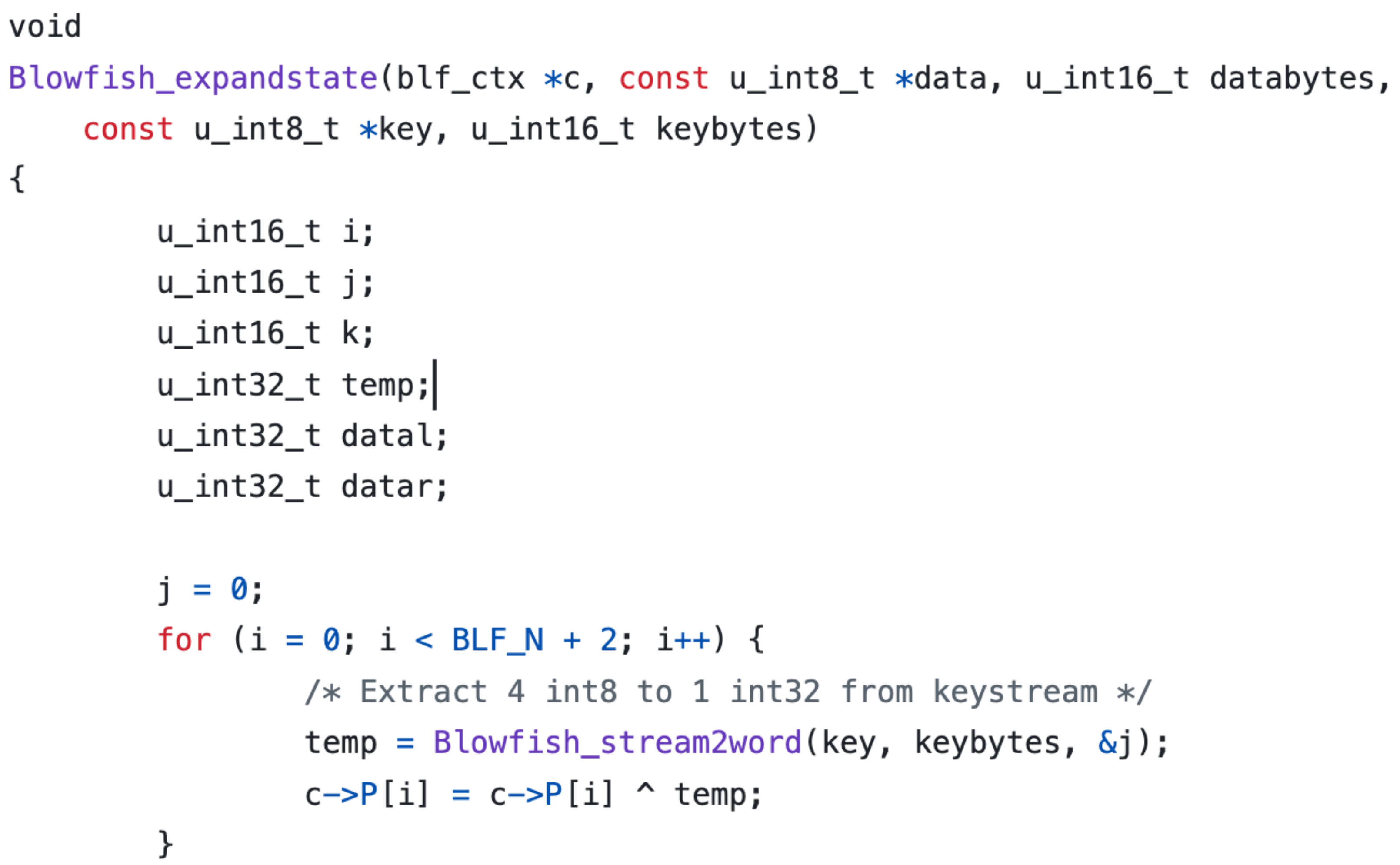
OpenBSD - Blowfish.c. Github : https://github.com/openbsd/src/blob/master/lib/libc/crypt/blowfish.c

```
u_int32_t
Blowfish_stream2word(const u_int8_t *data, u_int16_t databytes,
    u_int16_t *current)
{
        u_int8_t i;
        u_int16_t j;
        u_int32_t temp;

        temp = 0x00000000;
        j = *current;

        for (i = 0; i < 4; i++, j++) {
                if (j >= databytes)
                        j = 0;
                temp = (temp << 8) | data[j];
        }

        *current = j;
        return temp;
}
```

```
0 : 71
1 : 71c7
2 : 71c71c
3 : 71c71c75
```


ExpandKey (state, salt, key)
ExpandKey (state, 0, key)
ExpandKey (state, 0, salt)

OpenBSD - Blowfish.c. Github : https://github.com/openbsd/src/blob/master/lib/libc/crypt/blowfish.c



```c
void
Blowfish_expandstate(blf_ctx *c, const u_int8_t *data, u_int16_t databytes,
    const u_int8_t *key, u_int16_t keybytes)
{
        u_int16_t i;
        u_int16_t j;
        u_int16_t k;
        u_int32_t temp;
        u_int32_t datal;
        u_int32_t datar;

        j = 0;
        for (i = 0; i < BLF_N + 2; i++) {
                /* Extract 4 int8 to 1 int32 from keystream */
                temp = Blowfish_stream2word(key, keybytes, &j);
                c->P[i] = c->P[i] ^ temp;
        }
```

OpenBSD - Blowfish.c. Github : https://github.com/openbsd/src/blob/master/lib/libc/crypt/blowfish.c

```c
    j = 0;
    datal = 0x00000000;
    datar = 0x00000000;
    for (i = 0; i < BLF_N + 2; i += 2) {
        datal ^= Blowfish_stream2word(data, databytes, &j);
        datar ^= Blowfish_stream2word(data, databytes, &j);
        Blowfish_encipher(c, &datal, &datar);

        c->P[i] = datal;
        c->P[i + 1] = datar;
    }

    for (i = 0; i < 4; i++) {
        for (k = 0; k < 256; k += 2) {
            datal ^= Blowfish_stream2word(data, databytes, &j);
            datar ^= Blowfish_stream2word(data, databytes, &j);
            Blowfish_encipher(c, &datal, &datar);

            c->S[i][k] = datal;
            c->S[i][k + 1] = datar;
        }
    }

}
```

HANSUNG UNIVERSITY

OpenBSD - Blowfish.c. Github : https://github.com/openbsd/src/blob/master/lib/libc/crypt/blowfish.c

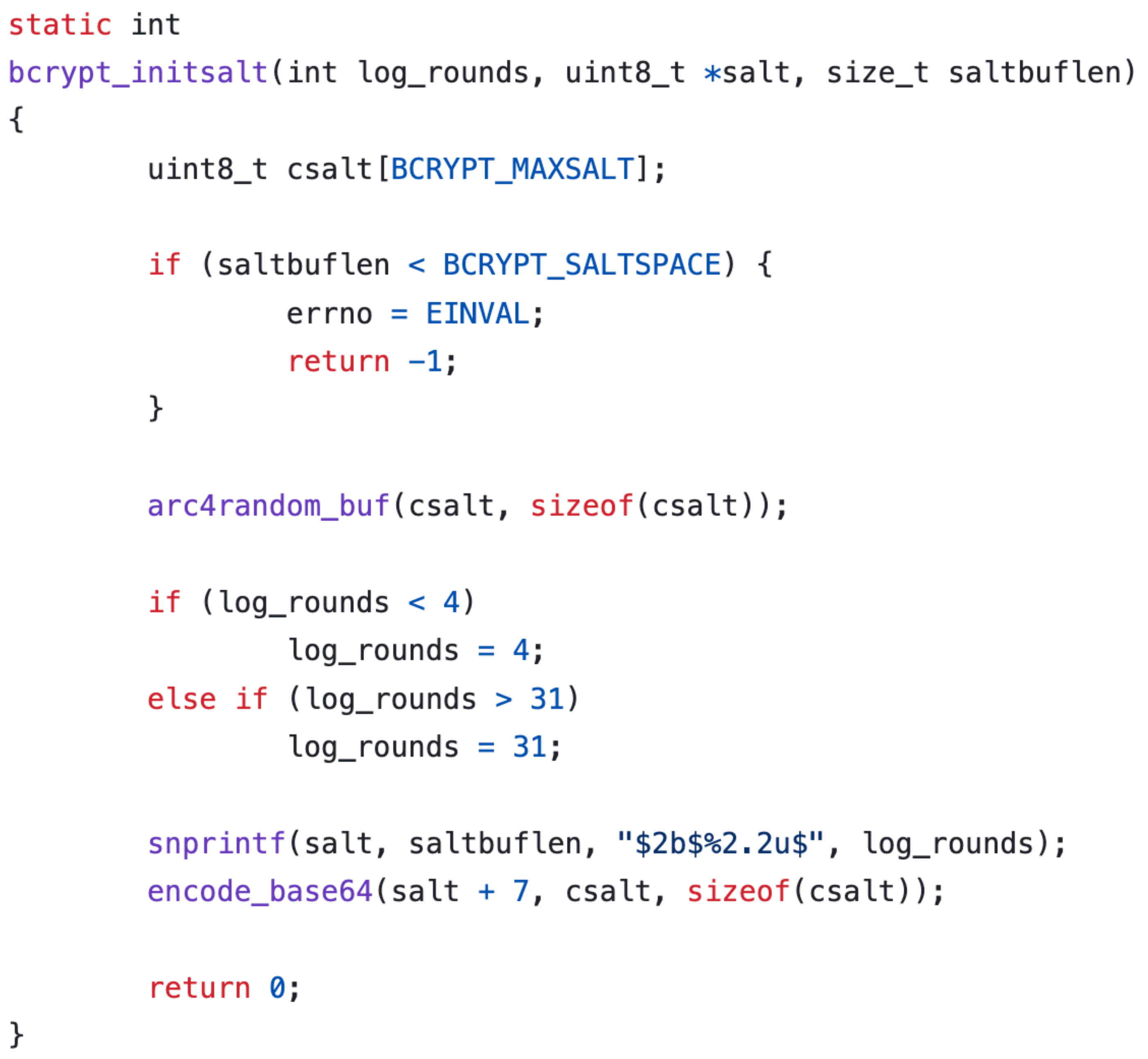
```c
void
Blowfish_expand0state(blf_ctx *c, const u_int8_t *key, u_int16_t keybytes)
{
	u_int16_t i;
	u_int16_t j;
	u_int16_t k;
	u_int32_t temp;
	u_int32_t datal;
	u_int32_t datar;

	j = 0;
	for (i = 0; i < BLF_N + 2; i++) {
		/* Extract 4 int8 to 1 int32 from keystream */
		temp = Blowfish_stream2word(key, keybytes, &j);
		c->P[i] = c->P[i] ^ temp;
	}

	j = 0;
	datal = 0x00000000;
	datar = 0x00000000;
	for (i = 0; i < BLF_N + 2; i += 2) {
		Blowfish_encipher(c, &datal, &datar);

		c->P[i] = datal;
		c->P[i + 1] = datar;
	}

	for (i = 0; i < 4; i++) {
		for (k = 0; k < 256; k += 2) {
			Blowfish_encipher(c, &datal, &datar);

			c->S[i][k] = datal;
			c->S[i][k + 1] = datar;
		}
	}
}
```

HANSUNG UNIVERSITY

OpenBSD - Blowfish.c. Github : https://github.com/openbsd/src/blob/master/lib/libc/crypt/blowfish.c

```c
void
blf_enc(blf_ctx *c, u_int32_t *data, u_int16_t blocks)
{
	u_int32_t *d;
	u_int16_t i;

	d = data;
	for (i = 0; i < blocks; i++) {
		Blowfish_encipher(c, d, d + 1);
		d += 2;
	}
}
```

OpenBSD - Bcrypt.c. Github : https://github.com/openbsd/src/blob/master/lib/libc/crypt/bcrypt.c



```c
static int
bcrypt_initsalt(int log_rounds, uint8_t *salt, size_t saltbuflen)
{
        uint8_t csalt[BCRYPT_MAXSALT];

        if (saltbuflen < BCRYPT_SALTSPACE) {
                errno = EINVAL;
                return -1;
        }

        arc4random_buf(csalt, sizeof(csalt));

        if (log_rounds < 4)
                log_rounds = 4;
        else if (log_rounds > 31)
                log_rounds = 31;

        snprintf(salt, saltbuflen, "$2b$%2.2u$", log_rounds);
        encode_base64(salt + 7, csalt, sizeof(csalt));

        return 0;
}
```

OpenBSD - Bcrypt.c. Github : https://github.com/openbsd/src/blob/master/lib/libc/crypt/bcrypt.c

```c
static int
bcrypt_hashpass(const char *key, const char *salt, char *encrypted,
    size_t encryptedlen)
{
	blf_ctx state;
	u_int32_t rounds, i, k;
	u_int16_t j;
	size_t key_len;
	u_int8_t salt_len, logr, minor;
	u_int8_t ciphertext[4 * BCRYPT_WORDS] = "OrpheanBeholderScryDoubt";
	u_int8_t csalt[BCRYPT_MAXSALT];
	u_int32_t cdata[BCRYPT_WORDS];

	if (encryptedlen < BCRYPT_HASHSPACE)
		goto inval;

	/* Check and discard "$" identifier */
	if (salt[0] != '$')
		goto inval;
	salt += 1;

	if (salt[0] != BCRYPT_VERSION)
		goto inval;
```

HANSUNG UNIVERSITY

OpenBSD - Bcrypt.c. Github : https://github.com/openbsd/src/blob/master/lib/libc/crypt/bcrypt.c

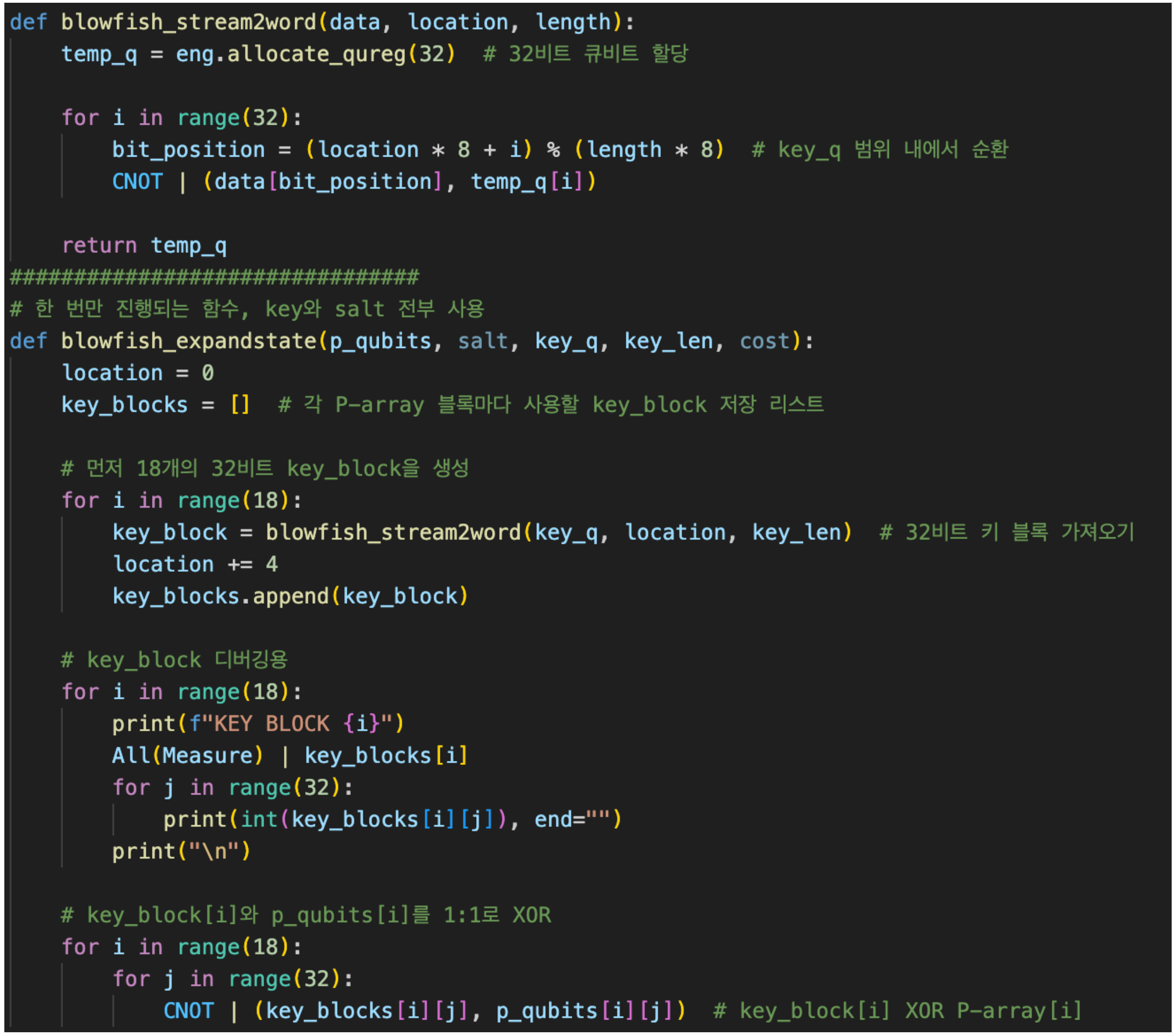
```
/* Check for minor versions */
switch ((minor = salt[1])) {
case 'a':
        key_len = (u_int8_t)(strlen(key) + 1);
        break;
case 'b':
        /* strlen() returns a size_t, but the function calls
         * below result in implicit casts to a narrower integer
         * type, so cap key_len at the actual maximum supported
         * length here to avoid integer wraparound */
        key_len = strlen(key);
        if (key_len > 72)
                key_len = 72;
        key_len++; /* include the NUL */
        break;
default:
         goto inval;
}
if (salt[2] != '$')
        goto inval;
/* Discard version + "$" identifier */
salt += 3;

/* Check and parse num rounds */
if (!isdigit((unsigned char)salt[0]) ||
    !isdigit((unsigned char)salt[1]) || salt[2] != '$')
        goto inval;
logr = (salt[1] - '0') + ((salt[0] - '0') * 10);
if (logr < BCRYPT_MINLOGROUNDS || logr > 31)
        goto inval;
/* Computer power doesn't increase linearly, 2^x should be fine */
rounds = 1U << logr;

/* Discard num rounds + "$" identifier */
salt += 3;
```

HANSUNG UNIVERSITY

OpenBSD - Bcrypt.c. Github : https://github.com/openbsd/src/blob/master/lib/libc/crypt/bcrypt.c

```c
/* Setting up S-Boxes and Subkeys */
Blowfish_initstate(&state);
Blowfish_expandstate(&state, csalt, salt_len,
    (u_int8_t *) key, key_len);
for (k = 0; k < rounds; k++) {
        Blowfish_expand0state(&state, (u_int8_t *) key, key_len);
        Blowfish_expand0state(&state, csalt, salt_len);
}

/* This can be precomputed later */
j = 0;
for (i = 0; i < BCRYPT_WORDS; i++)
        cdata[i] = Blowfish_stream2word(ciphertext, 4 * BCRYPT_WORDS, &j);

/* Now do the encryption */
for (k = 0; k < 64; k++)
        blf_enc(&state, cdata, BCRYPT_WORDS / 2);

for (i = 0; i < BCRYPT_WORDS; i++) {
        ciphertext[4 * i + 3] = cdata[i] & 0xff;
        cdata[i] = cdata[i] >> 8;
        ciphertext[4 * i + 2] = cdata[i] & 0xff;
        cdata[i] = cdata[i] >> 8;
        ciphertext[4 * i + 1] = cdata[i] & 0xff;
        cdata[i] = cdata[i] >> 8;
        ciphertext[4 * i + 0] = cdata[i] & 0xff;
}
```

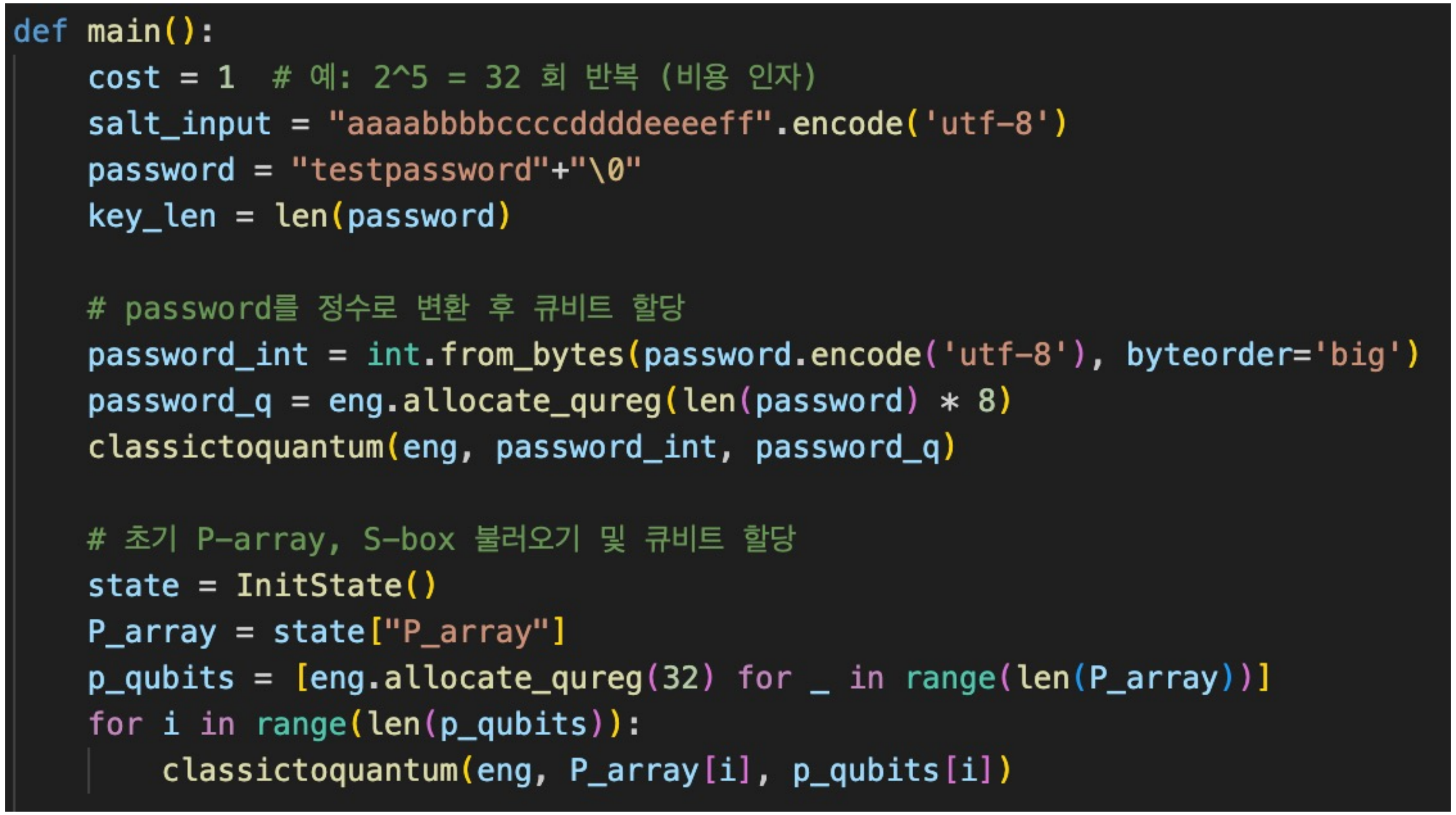
연구 진행 중인 부분



```
def blowfish_stream2word(data, location, length):
    temp_q = eng.allocate_qureg(32)  # 32비트 큐비트 할당

    for i in range(32):
        bit_position = (location * 8 + i) % (length * 8)  # key_q 범위 내에서 순환
        CNOT | (data[bit_position], temp_q[i])

    return temp_q
################################
# 한 번만 진행되는 함수, key와 salt 전부 사용
def blowfish_expandstate(p_qubits, salt, key_q, key_len, cost):
    location = 0
    key_blocks = []  # 각 P-array 블록마다 사용할 key_block 저장 리스트

    # 먼저 18개의 32비트 key_block을 생성
    for i in range(18):
        key_block = blowfish_stream2word(key_q, location, key_len)  # 32비트 키 블록 가져오기
        location += 4
        key_blocks.append(key_block)

    # key_block 디버깅용
    for i in range(18):
        print(f"KEY BLOCK {i}")
        All(Measure) | key_blocks[i]
        for j in range(32):
            print(int(key_blocks[i][j]), end="")
        print("\n")

    # key_block[i]와 p_qubits[i]를 1:1로 XOR
    for i in range(18):
        for j in range(32):
            CNOT | (key_blocks[i][j], p_qubits[i][j])  # key_block[i] XOR P-array[i]
```

HANSUNG UNIVERSITY

연구 진행 중인 부분



```python
def main():
    cost = 1  # 예: 2^5 = 32 회 반복 (비용 인자)
    salt_input = "aaaabbbbccccddddeeeeff".encode('utf-8')
    password = "testpassword"+"\0"
    key_len = len(password)

    # password를 정수로 변환 후 큐비트 할당
    password_int = int.from_bytes(password.encode('utf-8'), byteorder='big')
    password_q = eng.allocate_qureg(len(password) * 8)
    classictoquantum(eng, password_int, password_q)

    # 초기 P-array, S-box 불러오기 및 큐비트 할당
    state = InitState()
    P_array = state["P_array"]
    p_qubits = [eng.allocate_qureg(32) for _ in range(len(P_array))]
    for i in range(len(p_qubits)):
        classictoquantum(eng, P_array[i], p_qubits[i])
```

감사합니다