DICE (Device Identifier Composition Engine)

---

1. 개요

1.1. 목적

- TPM이 실용적이지 않거나 실행할 수 없는 환경에서 보안 기능 제공
- 보드 코드
  https://github.com/tmdals010126/Micro-ROS-Zephyr-TFM/tree/dice
- Agent 코드
  https://github.com/tmdals010126/Micro-XRCE-DDS-Agent/tree/dice

1.2. 주요 개념

- UDS (Unique Device Secret)
  디바이스 별로 고유하게 주어지는 비밀값. DICE의 입력값으로 CDI 생성에 사용

- CDI (Compound Device Identifier)
  UDS와 소프트웨어를 거쳐 계산되는 비밀값.

---

2. High-Level

2.1. 동작 흐름

1. CDI 입력값을 받아 CDI 값을 계산
2. UDS를 이용해 비대칭키 쌍을 유도
3. UDS 값에 접근하지 못하도록 잠금
4. Attestation CDI로부터 비대칭키 쌍을 유도
5. CDI 인증서 생성
6. 2. 의 개인키 삭제

2.2. open-dice

- Google에서 배포한 오픈소스인 open-dice 소스코드를 활용해 구현
- 포함한 코드
  - 헤더 파일 (include)
    dice.h / ops.h / types.h / utils.h
    ops/clear_memory.h / boringssl_multialg/dice/config.h
    default/dice/profile_name.h
  - 소스 코드 파일 (src)
    dice.c / utils.c / clear_memory.c

3. Module-Level

3.1. Nucleo-L552ZE-Q (Client)

3.1.1. zephyr_transport_open

uROS에서 통신 연결을 위해 설정을 하는 함수
메모리 맵에 정의된 LPUART_1 으로 연결하도록 수정

Parameter	Value	설명
struct uxrCustomTransport * transport	transport	통신 설정을 담고있는 구조체

3.1.2. rclc_node_init_default

node create를 Agent에 요청하는 함수

Parameter	Value	설명
rcl_node_t * node	&node	생성한 node를 저장할 구조체
const char * name	"zephyr_int32_publisher"	node name
const char * namespace_	""	node namespace
rclc_support_t * support	&support	메모리 관리 구조체

3.1.3. uxr_read_session_header

recv_message 함수에서 구동되어 Agent로 부터 받은 데이터에서 header를 파싱하는 함수
Nonce값을 파싱하는 구현이 포함. info->nonce에 저장.

Parameter	Value	설명
uxrSessionInfo* info	&session->info	session의 정보를 포함
ucdrBuffer* ub	ub	raw 데이터 버퍼
uint8_t* stream_id_raw	&stream_id_raw	ub에서 파싱된 stream id 가 저장될 포인터
uxrSeqNum* seq_num	&seq_num	메시지 순서를 저장

3.1.4. uxr_flash_output_streams_with_token

node create를 요청할 때 Agent로 session을 통해 보낼 buffer를 생성하는 함수

해당 함수 내에서 3.1.5.hash_firmware함수와 3.1.6.DiceMainFlow함수를 호출하여 Token을 생성

Parameter	Value	설명
uxrSession* session	session	session 정보를 저장한 구조체

3.1.5. hash_firmware

tinyscrypt의 sha256을 이용해 펌웨어를 해시하는 함수

Parameter	Value	설명
uint8_t *output_hash	hash_result	메모리의 펌웨어 영역을 해시한 값을 저장하는 변수

3.1.6. DiceMainFlow

DICE CDI 값과 인증서를 구하는 함수

3.1.5. hash_firmware 함수의 결과인 펌웨어 해시를 input value로 저장하여 인자로 전달

Parameter	Value	설명
void* context	NULL	라이브러리 통합 환경에 따라 설정
const uint8_t current_cdi_attest[DICE_CDI_SIZE]	cdi_buffer[DICE_CDI_SIZE]	현재 attestation CDI 값을 저장
const uint8_t current_cdi_seal[DICE_CDI_SIZE]	cdi_buffer[DICE_CDI_SIZE]	현재 seal CDI 값을 저장
const DiceInputvalues* input_values	&input_values	Dice의 입력 값을 저장한 구조체
size_t next_cdi_certificate_buffer_size	size_of(cert_buffer)	인증서 버퍼의 크기
uint8_t* next_cdi_certificate	cert_buffer	인증서를 저장
size_t* next_cdi_certificate_actual_size	&cert_size	실제 인증서의 크기
uint8_t next_cdi_attest[DICE_CDI_	cdi_buffer	생성한 attestation CDI 값을 저장

Parameter	Value	설명
SIZE]		
uint8_t next_cdi_seal[DICE_CDI_SIZE]	cdi_buffer	생성한 seal CDI 값을 저장

3.1.7. DiceHash

Dice의 input value를 해시하는 함수

Parameter	Value	설명
void* context_not_used	context	해당 함수에서는 context가 사용되지 않음
const uint8_t* input	input_buffer	Dice의 Input value 구조체를 저장
size_t input_size	sizeof(input_buffer)	Input buffer의 크기
uint8_t output[DICE_HASH_SIZE]	attest_input_hash	함수의 결과를 저장

3.1.8. DiceKdf

현재 CDI와 3.1.7. DiceHash의 결과인 해시값을 사용해 CDI를 생성하는 KDF 함수

Parameter	Value	설명
void* context_not_used	context	해당 함수에서는 context가 사용되지 않음
size_t length	DICE_CDI_SIZE	생성될 CDI의 크기
const uint8_t* ikm	current_cdi_attest	현재 attestation cdi 값 저장한 입력 값
size_t ikm_size	DICE_CDI_SIZE	입력한 현재 CDI의 크기
const uint8_t* salt	attest_input_hash	DiceHash 함수의 결과를 저장
size_t salt_size	DICE_HASH_SIZE	attest_input_hash의 크기
const uint8_t* info	(const uint8_t*) “CDI_ATTEST”	CDI에 포함될 추가 정보
size_t info_size	10	info의 크기
uint8_t output	next_cdi_attest	함수의 결과인 attestation cdi를 저장

3.2. ROS2 Agent (Agent)

3.2.1. Processor<EndPoint>::process_create_client_submessage

node 생성 요청 전에 Client에게 Agent가 존재함을 알려주는 과정에서 답장을 보내주는 함수
Nonce를 header에 포함하여 전송하도록 구현

Parameter	Value	설명
InputPacket<EndPoint>& input_packet	input_packet	packet의 정보를 담고 있는 구조체. message, header, source 등의 정보를 포함

3.2.2. SerialAgent::recv_message

시리얼(Serial) 통신을 통해 Client로부터 들어오는 메시지를 수신하고 이를 상위 로직에서 처리할 수 있는 InputPacket 형태로 만드는 함수
DICE Token을 device_token이라는 변수로 저장

Parameter	Value	설명
InputPacket<SerialEndPoint> &input_packet	std::vector<InputPacket<MultiSerialEndPoint>> input_packet	packet의 정보와 메시지를 담는 구조체
int timeout	RECEIVE_TIMEOUT	timeout 시간을 설정
TransportRc &transport_rc	TransportRc transport_rc = TransportRc::ok	처리 성공 / 실패 여부를 저장

3.2.3. std::unique_ptr<Participant> Participant::create

Participant(Client 측 node)를 생성하는 함수
사전에 whitelist 기반 hash를 기반으로 DiceMainFlow를 동일하게 진행하여 결과 비교 검증
검증 실패시 null 포인터를 반환 / 성공시 생성된 Participant를 반환

Parameter	Value	설명
const dds::xrce::ObjectId& object_id	object_id	생성할 Participant의 id (node id)
const std::shared_ptr<ProxyClient>& proxy_client	shared_from_this()	Participant 생성을 요청한 Micro XRCE-DDS 클라이언트를 대리하는

		객체
const dds::xrce::OBJK_PARTICIPANT_Representation& representation	representation	생성하려는 Participant에 대한 정보

3.2.4. DiceMainFlow

3.1.6 DiceMainFlow 와 동일
인자로 사전에 등록된 hash를 사용한다는 점만 차이가 존재

---

4. Source-Level

4.1. Nucleo-L552ZE-Q (Client)

3.1.1. zephyr_transport_open

`micro_ros_dice/modules/libmicroros/microros_transports/serial/microros_transports.c`

uROS에서 통신 연결을 위해 설정을 하는 함수
메모리 맵에 정의된 LPUART_1 으로 연결하도록 수정

```
bool zephyr_transport_open(struct uxrCustomTransport * transport){
    zephyr_transport_params_t * params = (zephyr_transport_params_t*) transport->args;
    char uart_descriptor[20];
    sprintf(uart_descriptor, "LPUART_1");
    params->uart_dev = device_get_binding(uart_descriptor);
    if (!params->uart_dev) {
        printf("Serial device not found\n");
        return false;
    }
    ring_buf_init(&in_ringbuf, sizeof(uart_in_buffer), uart_out_buffer);
    uart_irq_callback_set(params->uart_dev, uart_fifo_callback);
    /* Enable rx interrupts */
    uart_irq_rx_enable(params->uart_dev);
    return true;
}
```

3.1.2. uxr_read_session_header

`micro_ros_dice/modules/libmicroros/micro_ros_src/src/Micro-XRCE-DDS-Client/src/c/core/session/session_info.c`

recv_message 함수에서 구동되어 Agent로 부터 받은 데이터에서 header를 파싱하는 함수

Nonce값을 파싱하는 구현이 포함. info->nonce에 저장. info->nonce는 uint32_t 타입

```
bool uxr_read_session_header(
        uxrSessionInfo* info,
        ucdrBuffer* ub,
        uint8_t* stream_id_raw,
        uxrSeqNum* seq_num)
{
    bool must_be_read = ucdr_buffer_remaining(ub) > MAX_HEADER_SIZE;
    if (must_be_read)
    {
        uint8_t session_id; uint8_t key[CLIENT_KEY_SIZE];
        uxr_deserialize_message_header(ub, &session_id, stream_id_raw,
seq_num, key);

        ////////////////////////// DICE START ////////////////////////
        info->nonce = (uint32_t *)ub->init + 1;
        ////////////////////////// DICE END ////////////////////////

        must_be_read = session_id == info->id;
        if (must_be_read)
        {
            if (SESSION_ID_WITHOUT_CLIENT_KEY > info->id)
            {
                must_be_read = (0 == memcmp(key, info->key,
CLIENT_KEY_SIZE));
            }
        }
    }

    return must_be_read;
}
```

4.1.3. uxr_flash_output_streams_with_token

micro_ros_dice/modules/libmicroros/micro_ros_src/src/Micro-XRCE-DDS-Client/src/c/core/session/session.c

node create를 요청할 때 Agent로 session을 통해 보낼 buffer를 생성하는 함수
해당 함수 내에서 3.1.5.hash_firmware함수와 3.1.6.DiceMainFlow함수를 호출하여 Token을 생성

```
while (uxr_prepare_next_reliable_buffer_to_send(stream, &buffer, &length,
&seq_num))
{
    uxr_stamp_session_header(&session->info, id.raw, seq_num, buffer);
    uint8_t hash_result[DICE_CDI_SIZE];
    uint8_t nonce_buffer[DICE_CDI_SIZE];
    if (!hash_firmware(hash_result)) {
        printf("Firmware hash failed.\n");
    }
    for (int i = 0; i < DICE_CDI_SIZE; i++)
    {
        nonce_buffer[i] = ((uint8_t *)(session->info.nonce))[i%4] ^
hash_result[i];
    }
    uint8_t cdi_buffer[DICE_CDI_SIZE] = {0,};
    uint8_t cert_buffer[2048];
    size_t cert_size;
    DiceInputValues input_values = {};
    memcpy(input_values.code_hash, nonce_buffer, DICE_CDI_SIZE);
    DiceResult result;
    result = DiceMainFlow(NULL, cdi_buffer, cdi_buffer,
            &input_values, sizeof(cert_buffer), cert_buffer,
            &cert_size, cdi_buffer, cdi_buffer);

    if (result == kDiceResultOk) {
        for (int i = 0; i < DICE_CDI_SIZE; ++i) {
            buffer[length + i] = cdi_buffer[i];
        }
    }
    send_message(session, buffer, length + DICE_CDI_SIZE);
}
```

4.1.4. hash_firmware

tinyscrypt의 sha256을 이용해 펌웨어를 해시하는 함수
hash.h 파일에 정의 된 FIRMWARE_START_ADDR, FIRMWARE_SIZE 값을 변경해 hashing 할 범위를 조정할 수 있음

```
int hash_firmware(uint8_t *output_hash)
{
   if (output_hash == NULL) {
```

```
        return 0;
    }

    const uint8_t *firmware_ptr = (const uint8_t *)FIRMWARE_START_ADDR;

    struct tc_sha256_state_struct sha_ctx;
    if (!tc_sha256_init(&sha_ctx)) {
        return 0;
    }

    for (uint32_t offset = 0; offset < FIRMWARE_SIZE; offset += CHUNK_SIZE) {
        size_t len = (FIRMWARE_SIZE - offset > CHUNK_SIZE) ? CHUNK_SIZE :
(FIRMWARE_SIZE - offset);
        if (!tc_sha256_update(&sha_ctx, firmware_ptr + offset, len)) {
            return 0;
        }
    }

    if (!tc_sha256_final(output_hash, &sha_ctx)) {
        return 0;
    }

    return 1;
}
```

4.1.5. DiceMainFlow

현재 cdi 값과 입력값을 이용해 cdi 값을 계산하는 함수
4.1.6. DiceHash 함수와 4.1.7. DiceKdf 함수를 호출해 CDI를 계산
현재 함수 호출 시에는 적용하지 않았으나 current_cdi_attest와 current_cdi_seal의 input으로 UDS를 넣어 사용할 수 있음

```
DiceResult DiceMainFlow(void* context,
                        const uint8_t current_cdi_attest[DICE_CDI_SIZE],
                        const uint8_t current_cdi_seal[DICE_CDI_SIZE],
                        const DiceInputValues* input_values,
                        size_t next_cdi_certificate_buffer_size,
                        uint8_t* next_cdi_certificate,
                        size_t* next_cdi_certificate_actual_size,
                        uint8_t next_cdi_attest[DICE_CDI_SIZE],
                        uint8_t next_cdi_seal[DICE_CDI_SIZE]) {
 // This implementation serializes the inputs for a one-shot hash. On some
 // platforms, using a multi-part hash operation may be more optimal. The
 // combined input buffer has this layout:
 //
-------------------------------------------------------------------------
 // | Code Input | Config Input | Authority Input | Mode Input | Hidden
Input |
```

```
 //
-----------------------------------------------------------------------------
 const size_t kCodeOffset = 0;
 const size_t kConfigOffset = kCodeOffset + DICE_CODE_SIZE;
 const size_t kAuthorityOffset = kConfigOffset + DICE_CONFIG_SIZE;
 const size_t kModeOffset = kAuthorityOffset + DICE_AUTHORITY_SIZE;
 const size_t kHiddenOffset = kModeOffset + DICE_MODE_SIZE;

 DiceResult result = kDiceResultOk;

 // Declare buffers that get cleaned up on 'goto out'.
 uint8_t input_buffer[DICE_CODE_SIZE + DICE_CONFIG_SIZE +
DICE_AUTHORITY_SIZE +
                      DICE_MODE_SIZE + DICE_HIDDEN_SIZE];
 uint8_t attest_input_hash[DICE_HASH_SIZE];
 uint8_t seal_input_hash[DICE_HASH_SIZE];
 uint8_t current_cdi_private_key_seed[DICE_PRIVATE_KEY_SEED_SIZE];
 uint8_t next_cdi_private_key_seed[DICE_PRIVATE_KEY_SEED_SIZE];

 // Assemble the input buffer.
 memcpy(&input_buffer[kCodeOffset], input_values->code_hash,
DICE_CODE_SIZE);
 if (input_values->config_type == kDiceConfigTypeInline) {
   memcpy(&input_buffer[kConfigOffset], input_values->config_value,
          DICE_CONFIG_SIZE);
 } else if (!input_values->config_descriptor) {
   result = kDiceResultInvalidInput;
   goto out;
 } else {
   result = DiceHash(context, input_values->config_descriptor,
                     input_values->config_descriptor_size,
                     &input_buffer[kConfigOffset]);
   if (result != kDiceResultOk) {
     goto out;
   }
 }
 memcpy(&input_buffer[kAuthorityOffset], input_values->authority_hash,
        DICE_AUTHORITY_SIZE);
 input_buffer[kModeOffset] = input_values->mode;
 memcpy(&input_buffer[kHiddenOffset], input_values->hidden,
DICE_HIDDEN_SIZE);

 // Hash the appropriate input values for both attestation and sealing. For
 // attestation all the inputs are used, and for sealing only the authority,
 // mode, and hidden inputs are used.
 result =
     DiceHash(context, input_buffer, sizeof(input_buffer),
attest_input_hash);
```

```
 if (result != kDiceResultOk) {
   goto out;
 }
 result = DiceHash(context, &input_buffer[kAuthorityOffset],
                   DICE_AUTHORITY_SIZE + DICE_MODE_SIZE + DICE_HIDDEN_SIZE,
                   seal_input_hash);
 if (result != kDiceResultOk) {
   goto out;
 }

 // Compute the next CDI values. For each of these the current CDI value is
 // used as input key material and the input hash is used as salt.
 result = DiceKdf(context, /*length=*/DICE_CDI_SIZE, current_cdi_attest,
                  /*ikm_size=*/DICE_CDI_SIZE, attest_input_hash,
                  /*salt_size=*/DICE_HASH_SIZE,
                  /*info=*/(const uint8_t*)"CDI_Attest", /*info_size=*/10,
                  next_cdi_attest);
 if (result != kDiceResultOk) {
   goto out;
 }
 result = DiceKdf(
     context, /*length=*/DICE_CDI_SIZE, current_cdi_seal,
     /*ikm_size=*/DICE_CDI_SIZE, seal_input_hash,
/*salt_size=*/DICE_HASH_SIZE,
     /*info=*/(const uint8_t*)"CDI_Seal", /*info_size=*/8, next_cdi_seal);
 if (result != kDiceResultOk) {
   goto out;
 }

 // Create the CDI certificate only if it is required (i.e.
non-null/non-zero
 // values are provided for the next CDI certificate parameters).
 if (next_cdi_certificate == NULL &&
     next_cdi_certificate_actual_size == NULL &&
     next_cdi_certificate_buffer_size == 0) {
   goto out;
 }

 // Derive asymmetric private key seeds from the attestation CDI values.
 result = DiceDeriveCdiPrivateKeySeed(context, current_cdi_attest,
                                      current_cdi_private_key_seed);
 if (result != kDiceResultOk) {
   goto out;
 }
 result = DiceDeriveCdiPrivateKeySeed(context, next_cdi_attest,
                                      next_cdi_private_key_seed);
 if (result != kDiceResultOk) {
   goto out;
```

```
 }

 // Generate a certificate for |next_cdi_private_key_seed| with
 // |current_cdi_private_key_seed| as the authority.
 result = DiceGenerateCertificate(
     context, next_cdi_private_key_seed, current_cdi_private_key_seed,
     input_values, next_cdi_certificate_buffer_size, next_cdi_certificate,
     next_cdi_certificate_actual_size);

out:
 // Clear sensitive memory.
 DiceClearMemory(context, sizeof(input_buffer), input_buffer);
 DiceClearMemory(context, sizeof(attest_input_hash), attest_input_hash);
 DiceClearMemory(context, sizeof(seal_input_hash), seal_input_hash);
 DiceClearMemory(context, sizeof(current_cdi_private_key_seed),
                 current_cdi_private_key_seed);
 DiceClearMemory(context, sizeof(next_cdi_private_key_seed),
                 next_cdi_private_key_seed);
 return result;
}
```

4.1.6. DiceHash

mbedtls의 sha512 함수로 DICE의 input value를 해시하는 함수
zephyr의 모듈로 존재하는 mbedtls를 사용하기 위해 open-dice의 mbedtls_ops.c 를 사용

```
DiceResult DiceHash(void* context_not_used, const uint8_t* input,
                    size_t input_size, uint8_t output[DICE_HASH_SIZE]) {
 (void)context_not_used;
 if (0 != mbedtls_md(mbedtls_md_info_from_type(MBEDTLS_MD_SHA512), input,
                     input_size, output)) {
   return kDiceResultPlatformError;
 }
 return kDiceResultOk;
}
```

4.1.7. DiceKdf

mbedtls의 hkdf 함수를 이용해 DICE의 키인 CDI를 구하는 함수
DiceHash와 같이 sha512를 이용하고 salt로는 DiceHash의 결과 해시 값을 사용

```
DiceResult DiceKdf(void* context_not_used, size_t length, const uint8_t*
ikm,
                   size_t ikm_size, const uint8_t* salt, size_t salt_size,
                   const uint8_t* info, size_t info_size, uint8_t* output) {
 (void)context_not_used;
```

```
 if (0 != mbedtls_hkdf(mbedtls_md_info_from_type(MBEDTLS_MD_SHA512), salt,
                       salt_size, ikm, ikm_size, info, info_size, output,
                       length)) {
   return kDiceResultPlatformError;
 }
 return kDiceResultOk;
}
```

4.2. ROS2 Agent (Agent)

4.2.1. Processor<EndPoint>::process_create_client_submessage

`src/cpp/processor/Processor.cpp`

node 생성 요청 전에 Client에게 Agent가 존재함을 알려주는 과정에서 답장을 보내주는 함수

Nonce를 header에 포함하여 전송하도록 구현

XRCE-DDS Agent 코드 내부에 존재. rand함수를 이용하여 랜덤값을 채워 header에 추가하여 전송. uint32_t 타입 nonce 값.

```
/*
nonce 값 전송 패킷 생성
*/
nonce = std::rand();
const size_t message_size = status_header.getCdrSerializedSize() +
                            status_subheader.getCdrSerializedSize() +
                            status_agent.getCdrSerializedSize() + 4;

OutputPacket<EndPoint> output_packet;
output_packet.destination = input_packet.source;
output_packet.message = std::shared_ptr<OutputMessage>(new
OutputMessage(status_header, message_size));
output_packet.message->append_raw_uint32(nonce);
output_packet.message->append_submessage(dds::xrce::STATUS_AGENT,
status_agent);

server_.push_output_packet(std::move(output_packet));
```

4.2.2. SerialAgent::recv_message

`src/cpp/transport/serial/SerialAgentLinux.cpp`

시리얼(Serial) 통신을 통해 Client로부터 들어오는 메시지를 수신하고 이를 상위 로직에서 처리할 수 있는 InputPacket 형태로 만드는 함수

DICE CDI를 수신하여 device_token 에 복사하도록 수정

```
if (0 < bytes_read)
{
    input_packet.message.reset(new InputMessage(buffer_,
static_cast<size_t>(bytes_read)));
    input_packet.source = SerialEndPoint(remote_addr);
    rv = true;
    memcpy(device_token, input_packet.message->get_buf() +
input_packet.message->get_len()-DICE_CDI_SIZE, DICE_CDI_SIZE);
    uint32_t raw_client_key;
    if (Server<SerialEndPoint>::get_client_key(input_packet.source,
raw_client_key))
    {
        UXR_AGENT_LOG_MESSAGE(
            UXR_DECORATE_YELLOW("[==>> SER <<==]"),
            raw_client_key,
            input_packet.message->get_buf(),
            input_packet.message->get_len());
    }
}
return rv;
```

4.2.3. std::unique_ptr<Participant> Participant::create

`src/cpp/participant/Participant.cpp`

Participant(Client 측 node)를 생성하는 함수
사전에 등록한 펌웨어 hash값을 바탕으로 DiceMainFlow함수를 실행한 결과와 수신한 device_token 값을 비교하여 정상적인 펌웨어인지 검증
검증 실패시 null 포인터를 반환 / 성공시 생성된 Participant를 반환
만약 device에서 UDS라는 디바이스 고유값을 사용한다면 동일하게 해당 코드에서도 적용해야함.

```
DiceResult result;
uint8_t cdi_buffer[DICE_CDI_SIZE] = {0,};
uint8_t cert_buffer[2048];
size_t cert_size;

uint8_t device_hash[DICE_HASH_SIZE] =
{0x00,0xa0,0x99,0x81,0x6c,0xb8,0x32,0x98,0x9e,0x3b,0xb1,0x49,0x6b,0x24,0
x63,0xf1,0x63,0xed,0x7a,0x3c,0x7d,0xec,0x03,0x10,0xe4,0xc9,0xb5,0x86,0x7
8,0x5b,0x35,0xc1};

uint8_t nonce_buffer[DICE_CDI_SIZE] = {0,};
for (int i = 0; i < DICE_CDI_SIZE; i++)
{
    nonce_buffer[i] = device_hash[i] ^ ((nonce >> (8 * (i % 4))) &
0xFF);
```

```
}

DiceInputValues input_values = {};
memcpy(input_values.code_hash, nonce_buffer, DICE_CDI_SIZE);
                                    // This should be populated with
actual authority, config, and code hashes.
printf("nonce: %d\n", nonce);
result = DiceMainFlow(/*context=*/NULL, cdi_buffer, cdi_buffer,
                            &input_values, sizeof(cert_buffer),
cert_buffer,
                            &cert_size, cdi_buffer, cdi_buffer);
printf("cdi_attest: ");
for (int i = 0; i < DICE_CDI_SIZE; i++)
{
    printf("%02X ", cdi_buffer[i]);
}
printf("\n");

if (result == kDiceResultOk)
{
    printf("DiceMainFlow successfully executed.\n");
    // You can now use cdi_attest and cdi_seal for attestation and
sealing.
}
else
{
    printf("DiceMainFlow failed with error code: %d\n", result);
}
printf("device_token: ");
for (int i = 0; i < DICE_CDI_SIZE; i++)
{
    printf("%02X ", device_token[i]);
}
printf("\n");
for (int i = 0; i < DICE_CDI_SIZE; i++)
{
    if(device_token[i] != cdi_buffer[i])
    {
        printf("DICE Remote Attestation Failed\n");
        return (nullptr);
    }
}
created_entity =
proxy_client->get_middleware().create_participant_by_bin(raw_object_id,
participant_xrce);
break;
```

---

5. 참고 자료

- Trusted Computing Group
  https://trustedcomputinggroup.org/what-is-a-device-identifier-composition-engine-dice/
- Google의 open-dice Github
  https://github.com/google/open-dice
