

수이(Sui)

해당 번역본은 이용자가 디지털자산 관련 정보를 편리하게 확인할 수 있도록
단순 참고용으로 번역된 것이며, 투자 권유를 목적으로 하지 않습니다.
그 내용이 정확하지 않을 수 있으므로 정확한 정보 습득을 위해서는
원문을 참고하시거나 원문 작성 주체 측에 문의하시기 바랍니다.
두나무는 해당 번역본 내용의 진실성 및 정확성을 보장하지 않고,
제공된 정보에 의한 투자 결과에 대하여 어떠한 책임도 지지 않으며,
투자 손실은 투자자에게 귀속됩니다.

수이 스마트 컨트랙트 플랫폼(The Sui Smart Contracts Platform)

미스틴랩스 팀(The MystenLabs Team)

hello@mystenlabs.com

1 서론(Introduction)

수이(Sui)는 지연(latency)이 적은 자산 관리에 중점을 둔 탈중앙화되고 비허가적인 스마트 컨트랙트 플랫폼이다. 수이는 주소가 소유할 수 있는 객체(objects)로 자산을 정의하기 위해 무브(Move) 프로그래밍 언어를 사용한다. 무브 프로그램은 이와 같이 타입화된(typed) 객체에 대한 연산을 정의하며, 여기에는 객체 생성, 새로운 소유자에게의 자산 이전, 자산 변형을 위한 사용자 정의 규칙이 포함된다.

수이는 다른 블록체인 시스템의 검증자(validators)나 채굴자(miners)와 유사한 역할을 수행하는 비허가적인 권한 주체(authorities) 집합에 의해 유지된다. 수이는 권한 주체 간 'Byzantine consistent broadcast' 프로토콜을 사용하여 자산과 관련된 일반 연산의 안전성을 보장해 비잔틴 합의(Byzantine agreement) 대비 더 적은 지연과 더 나은 확장성(scalability)을 달성한다. 비잔틴 합의는 공유 객체(shared objects)의 안전성을 위해서만 활용되는데, 거버넌스 연산, 체크포인팅(check-pointing) 등과 같이 핵심 지연 경로(critical latency path)의 외부에서 수행된다. 스마트 컨트랙트의 실행은 가능한 경우 자연스럽게 병렬화된다. 수이는 읽기 연산을 검증할 수 있는 라이트 클라이언트(light clients)와 모든 상태 변화를 검증할 수 있는 풀 클라이언트(full clients)를 지원한다. 이러한 기능은 신뢰를 최소화하는(trust-minimized) 방식으로 다른 블록체인과의 브리지 구현을 가능하게 한다.

네이티브 자산인 SUI는 모든 연산의 가스 비용 지불에 사용된다. 또한, SUI 보유자는 이를 권한 주체에게 위임하여(epoch 단위로) 수이를 운영할 수 있으며, 주기적으로 위임된 지분에 따라 권한 주체 구성이 재조정된다. 사용된 가스는 권한 주체와 그들의 위임자에게 지분과 수이 운영 기여도에 따라 분배된다.

이 백서는 두 부분으로 구성되어 있으며, 2장에서는 무브 언어를 사용하는 수이 프로그래밍 모델을, 4장에서는 수이의 안전성, 생존성(liveness), 성능을 보장하는 비허가적인 탈중앙화 시스템의 작동 방식을 설명한다.

2 수이 스마트 컨트랙트 프로그래밍(Sui Smart Contract Programming)

수이의 스마트 컨트랙트는 무브[4] 언어로 작성된다. 안전하고 표현력이 뛰어난 무브는 타입(type) 시스템과 데이터 모델을 통해 수이를 확장 가능하게 만드는 병렬 합의 및 실행 전략을 자연스럽게 지원한다.

무브는 원래 페이스북이 디엠(Diem) 블록체인을 위해 개발한 오픈소스 스마트 컨트랙트 언어이다. 이 언어는 플랫폼에 종속되지 않으며, 수이뿐 아니라 oL, StarCoin 등 다른 플랫폼에서도 점차 채택되고 있다.

이 장에서는 무브 언어의 주요 특징을 논의하고, 수이에서 자산을 생성하고 관리하는 방법을 설명한다. 무브의 기능에 대한 보다 자세한 설명은 [Move Programming Language Book](#)에서 확인할 수 있으며, 수이에 특화된 무브 관련 자료는 Sui Developer Portal에서, 수이의 맥락에서의 무브에 대한 형식적 설명은 3장에서 확인할 수 있다.

2.1 개요(Overview)

수이의 전역 상태(global state)는 무브 *패키지(package)*에 의해 생성 및 관리되는 프로그래밍 가능한(programmable) 객체들의 풀(pool)로 구성된다. 패키지는 무브 모듈들의 집합(자세한 내용은 2.1.1절 참조)으로, 무브 함수와 타입을 포함한다. 무브 패키지 자체도 객체이므로, 수이의 객체는 다음 두 가지 범주로 구분된다.

- **구조체 데이터 값(Struct data values):** 무브 모듈에 의해 관리되는 타입 데이터이다. 각 객체는 필드(fields)를 가진 구조체(struct) 값으로, 필드는 기본 타입(예: 정수, 주소), 다른 객체, 또는 비객체(non-object) 구조체를 포함할 수 있다.
- **패키지 코드 값(Package code values):** 원자적 단위로 게시(publish)된 관련 무브 바이트코드 모듈 집합이다. 각 모듈은 동일 패키지 내의 다른 모듈 및 이전에 게시된 패키지의 모듈에 의존할 수 있다.

객체는 자산(예: 대체 가능 토큰, 비대체 토큰), 특정 함수 호출이나 객체 생성 권한을 부여하는 *capability*, 다른 자산을 관리하는 “스마트 컨트랙트” 등을 인코딩할 수 있다. 즉, 그 정의는 프로그래머에게 달려 있다. 사용자 정의 수이 객체 타입을 선언하는 무브 코드는 다음과 같다.

```

struct Obj has key {
    id: VersionedID, // 전역적으로 고유한 ID 및 버전
    f: u64 // 객체는 기본 필드를 가질 수 있음
    g: OtherObj // 필드는 다른 객체를 저장할 수도 있음
}

```

수이 객체를 나타내는 모든 구조체(그러나 모든 무브 구조체 값은 아님)는 id 필드와 key [ability](#)를 반드시 가져야 하며, 이는 해당 값이 수이의 전역 객체 풀(global object pool)에 저장될 수 있음을 의미한다.

2.1.1 모듈(Modules)

무브 프로그램은 구조체 선언과 함수 선언의 리스트로 구성된 모듈들의 집합으로 구성된다. 하나의 모듈은 다른 모듈에서 구조체 타입(struct type)을 가져오거나(import), 그 모듈의 함수를 호출할 수 있다.

한 모듈에서 선언된 값은 다른 모듈로 전달될 수 있다. 예를 들어 위의 예시에서 OtherObj 모듈은 Obj를 정의하는 모듈과 다른 모듈로 정의될 수 있다. 이는 대부분의 스마트 컨트랙트 언어와 달리, 컨트랙트 간에 비구조화된(unstructured) 바이트만 전달되는 것이 아니라 구조화된 타입 데이터가 교차할 수 있음을 의미한다. 무브는 프로그래머가 'robustly safe' 한[14] 코드를 작성할 수 있도록 도와주는 'encapsulation' 기능을 제공하기 때문에 이를 지원할 수 있다. 구체적으로, 무브의 타입 시스템은 위의 Obj와 같은 타입이 오직 그 타입을 선언한 모듈 내부의 함수에 의해서만 생성(create), 제거(destroy), 복사(copy), 읽기(read), 쓰기(write)가 가능하도록 보장한다. 이로써 모듈은 자신이 선언한 타입에 대해 강력한 불변식(invariant)을 정의할 수 있으며, 이는 스마트 컨트랙트 간의 신뢰 경계를 넘어 전달되더라도 계속 유지된다.

2.1.2 트랜잭션과 엔트리포인트(Transactions and Entrypoints)

전역 객체 풀(global object pool)은 객체를 생성, 제거, 읽기, 쓰기하는 트랜잭션을 통해 업데이트된다. 트랜잭션은 연산이 수행되는 각 기존 객체를 입력(input)으로 받아야 한다. 또한 트랜잭션에는 패키지 객체의 버전이 지정된 ID(versioned ID), 해당 패키지 내의 모듈 이름과 함수 이름, 그리고 함수의 인자(입력 객체 포함)가 포함되어야 한다. 예를 들어, 다음 함수를 호출하려면 트랜잭션은 Obj 타입의 세 개의 서로 다른 객체 ID와 정수형 인자 x를 제공해야 한다.

```

public fun entrypoint(
    o1: Obj, o2: &mut Obj, o3: &Obj, x: u64, ctx: &mut TxContext
) { ... }

```

TxContext는 런타임에서 자동으로 채워지는 특수한 매개변수로, 송신자 주소(sender address)와 새로운 객체 생성을 위한 정보를 포함한다.

엔트리포인트(또는 일반적인 무브 함수)의 입력은 타입에 인코딩된 변경 가능성(mutability) 권한에 따라 전달될 수 있다. Obj 입력은 읽기, 쓰기, 이전(transfer), 제거가 가능하다. &mut Obj 입력은 읽기와 쓰기만 가능하며, &Obj 입력은 읽기만 가능하다. 트랜잭션 송신자는 지정된 변경 가능성 권한으로 각 입력 객체를 사용할 수 있는 권한을 가져야 한다. 자세한 내용은 4.4절을 참조하길 바란다.

2.1.3 객체 생성 및 전송(Creating and Transferring Objects)

프로그래머는 엔트리포인트로 전달된 `TxContext`를 사용하여 객체의 새로운 ID를 생성함으로써 객체를 만들 수 있다.

```
public fun create_then_transfer(
    f: u64, g: OtherObj, o1: Obj, ctx: &mut TxContext
) {
    let o2 = Obj { id: TxContext::fresh_id(ctx), f, g };
    Transfer::transfer(o1, TxContext::sender());
    Transfer::transfer(o2, TxContext::sender());
}
```

이 코드는 `OtherObj`와 `Obj` 타입의 두 객체를 입력으로 받아, 첫 번째 객체와 새로 생성된 ID를 사용하여 새로운 `Obj`를 생성한 뒤, 두 `Obj` 객체 모두를 트랜잭션 송신자에게 전송한다. 객체가 한 번 전송되면, 그것은 전역 객체 풀(global object pool)로 이동하며, 트랜잭션의 나머지 코드에서는 더 이상 접근할 수 없다. `Transfer` 모듈은 수이 표준 라이브러리(Sui standard library)의 일부로, 객체를 사용자 주소나 다른 객체로 전송하는 함수들을 포함하고 있다.

만약 프로그래머가 `transfer` 호출 중 하나를 누락했다면, 이 코드는 무브 타입 시스템에 의해 거부된다. 무브는 객체가 허가 없이 생성되거나, 복사되거나, 실수로 소멸되지 않도록 *자원 안전성(resource safety)* [5]을 강제한다. 자원 안전성의 또 다른 예로, 동일한 객체를 두 번 전송하려는 시도 또한 무브 타입 시스템에 의해 거부된다.

3. 수이 프로그래밍 모델(The Sui Programming Model)

이 장에서는 2장에서 비공식적으로 설명된 수이 프로그래밍 모델을 보다 구체적인 의미적 정의로 확장한다. 앞선 장에서는 무브 소스 코드 예시를 보였지만, 여기에서는 무브 바이트코드의 구조를 정의한다. 개발자는 무브 소스 코드를 로컬에서 작성, 테스트, 그리고 형식적으로 검증(formal verification) [10, 16]한 후, 이를 무브 바이트코드로 컴파일하여 블록체인에 게시(publish)한다. 온체인에 게시될 수 있는 모든 무브 바이트코드는 *바이트코드 검증기(bytecode verifier)* [4, 5]를 통과해야 하며, 이 과정에서 타입 안전성(type safety), 메모리 안전성(memory safety), 자원 안전성(resource safety) 등의 핵심 속성을 만족하는지 확인된다.

2장에서 언급했듯이, 무브는 플랫폼 비종속적(platform-agnostic) 언어로서, 핵심 언어를 포크하지 않고도 다양한 시스템의 특정 요구사항에 맞게 조정할 수 있다. 이후의 설명에서, 검은색으로 표시된 부분은 코어 무브(core Move) 언어의 개념을, 주황색으로 표시된 부분은 코어 무브 언어를 확장한 수이 고유 기능을 나타낸다.

3.1 모듈(Modules)

Module =	ModuleName× (StructName → StructDecl)× (FunName → FunDecl) × FunDecl
GenericParam =	[Ability]
StructDecl =	(FieldName → StorableType)× [Ability] × [GenericParam]
FunDecl =	[Type][Type] × [Instr] × [GenericParam]
Instr =	TransferToAddr TransferToObj ShareMut ShareImmut ...

Table 1: Module

무브 코드는 *모듈(modules)*로 구성되며, 그 구조는 표 1(Table 1)에 정의되어 있다. 하나의 모듈은 이름이 지정된 *구조체(struct)* 선언들과 이름이 지정된 *함수(function)* 선언들의 집합으로 이루어진다 (이러한 선언의 예시는 2.1절에 제시되어 있다). 또한, 모듈은 모듈의 *초기화 함수(initializer)* 역할을 하는 특수 함수 선언을 포함한다. 이 함수는 모듈이 온체인에 게시되는 시점에 단 한 번만 호출된다.

구조체 선언(struct declaration)은 이름이 지정된 필드들의 집합으로 구성되며, 각 필드 이름은 저장 가능한 타입(storeable type)에 매핑된다. 구조체 선언에는 선택적으로 ability의 목록이 포함될 수 있다 (저장 가능한 타입과 ability에 대한 설명은 2장을 참조). 또한 구조체 선언은 *일반 매개변수(generic parameters)*와 ability 제약 조건을 포함할 수도 있으며, 이 경우 이를 *일반 구조체(generic struct)* 선언이라 한다. 예를 들어 다음과 같다:

```
struct Wrapper<T: copy> { t: T }
```

일반 매개변수는 구조체 필드를 선언할 때 사용되는 타입을 나타낸다.

이는 구조체 선언 시점에는 아직 알려지지 않은 타입이며, 구조체가 인스턴스화될 때(즉, 구조체 값이 생성될 때) *구체적인(concrete)* 타입이 제공된다.

함수 선언(function declaration)은 매개변수 타입(parameter types) 목록, 반환 타입(return types) 목록, 그리고 함수 본문을 구성하는 명령어(instructions) 목록을 포함한다. 함수 선언 또한 ability 제약 조건을 가진 일반 매개변수 목록을 포함할 수 있으며, 이러한 경우 이를 *일반 함수(generic function)* 선언이라고 부른다. 예를 들어 다음과 같다:

```
fun unwrap<T: copy>(p: Wrapper<T>) {}
```

마찬가지로 구조체 선언과 유사하게, 일반 매개변수는 함수 선언 시점에는 알려지지 않은 타입을 나타내지만, 함수의 매개변수(parameter), 반환값(return value), 그리고 함수 본문(function body)을 선언할 때 사용된다. 구체적인 타입은 함수가 호출될 때 제공된다.

함수 본문에 포함될 수 있는 명령어는 전통적인 무브 명령어 대부분을 포함하지만, 전역 저장(global storage) 명령어(예: `move_to`, `move_from`, `borrow_global`)는 예외이다. 코어 무브 명령어와 그 의미론의 전체 목록은 [14]에서 확인할 수 있다. 수이에서는 영구적 저장(persistent storage)이 코어 무브의 계정 기반 전역 스토리지(account-based global storage) 대신, 수이의 전역 객체 풀을 통해 지원된다.

수이에는 네 가지 객체에 대한 특화 연산이 존재한다. 이들 각각의 연산은 객체의 소유권 메타데이터(ownership metadata)를 변경하고(3.3절 참조), 이를 전역 객체 풀로 반환한다. 가장 단순한 경우, 수이 객체는 수이 사용자의 주소로 전송될 수 있다. 또한 객체는 다른 부모(parent) 객체로 전송될 수도 있는데, 이때 호출자는 자식 객체(child object) 외에도 부모 객체에 대한 변경 가능한 참조(mutable reference)를 제공해야 한다.

객체는 변경 가능한 형태로 공유(mutably shared) 될 수도 있으며, 이 경우 수이 시스템 내 누구나 해당 객체를 읽거나 쓸 수 있다. 마지막으로 객체는 변경 불가능한 형태로 공유(immutably shared) 될 수도 있는데, 이 경우 수이 시스템 내 누구나 객체를 읽을 수 있지만, 누구도 수정할 수는 없다.

이처럼 다양한 형태의 소유권(ownership)을 구분할 수 있는 기능은 수이의 고유한 특징이다. 다른 블록체인 플랫폼에서는 모든 컨트랙트와 객체가 변경 가능한 공유(mutably shared) 상태로 존재한다. 4장에서 설명하겠지만, 수이는 이러한 소유권 정보를 활용하여 모든 트랜잭션의 병렬 실행과, 변경 권한이 공유되지 않은 객체와 관련된 트랜잭션에 대한 병렬 합의를 구현한다.

3.2 타입과 어빌리티(Types and Abilities)

PrimType =	{address, id, bool, u8, u64, ...}
StructType =	ModuleName × StructName × [StorableType]
StorableType =	PrimType ⊔ StructType ⊔ GenericType ⊔ VectorType
VectorType =	StorableType
GenericType =	N
MutabilityQual =	{mut, immut}
ReferenceType =	StorableType × MutabilityQual
Type =	ReferenceType ⊔ StorableType
Ability =	{key, store, copy, drop}

Table 2: Types and Abilities

무브 프로그램은 수이 전역 객체 풀에 저장된 데이터와, 프로그램 실행 중 생성되는 일시적 데이터(transient data) 모두를 다룬다. 이 두 종류의 데이터(객체와 일시적 데이터)는 언어 수준에서 모두 무브 값(value)으로 취급된다. 그러나 모든 값이 동일하게 생성되는 것은 아니며, 타입에 의해 규정된 서로 다른 속성과 구조를 가진다.

무브에서 사용되는 타입은 표 2(Table 2)에 정의되어 있다. 무브는 다른 프로그래밍 언어에서 지원되는 것과 유사한 여러 *기본 타입(primitive types)* 을 지원한다. 여기에는 불리언(boolean) 타입이나 다양한 크기의 부호 없는 정수(unsigned integer) 타입 등이 있다. 또한 코어 무브에는 시스템의 사용자를 나타내는 주소(address) 타입이 존재하며, 이는 트랜잭션의 송신자(sender) 및 (수이의 경우) 객체의 소유자(owner)를 식별하는 데 사용된다. 마지막으로 수이는 수이 객체의 식별자(identity)를 나타내는 *id* 타입을 정의한다 — 자세한 내용은 3.3절에서 다룬다.

*구조체 타입*은 특정 모듈에서 선언된 구조체의 인스턴스(즉, 값)를 나타낸다 (구조체 선언에 대한 정보는 3.1절 참조). *일반 구조체 타입(generic struct type)*은 *일반 구조체 선언(generic struct declaration)*을 나타내며, 여기에 포함된 *저장 가능한 타입* 목록은 구조체 선언 시의 일반 매개변수 목록에 대응한다. 저장 가능한 타입은 *특수 타입(concrete type)* (예: 기본 타입이나 구조체) 또는 *일반 타입(generic type)* 이 될 수 있다. 이러한 타입을 저장 가능하다고 부르는 이유는, 구조체의 필드나 온체인에 영구적으로 저장되는 객체 내에 등장할 수 있기 때문이다. 반면, 참조 타입(reference type)은 그렇지 않다.

예를 들어, `Wrapper<u64>` 구조체 타입은 구체적인(기본) 저장 가능 타입인 `u64`로 매개화된 일반 구조체 타입이며, 이러한 타입은 구조체 인스턴스(즉, 값)를 생성하는 데 사용될 수 있다. 반면, 동일한 일반 구조체 타입은 상위 구조체나 함수 선언의 일반 매개변수로부터 유래한 제네릭 타입(예: `struct Parent<T> { w: Wrapper<T> }`)으로 매개화될 수도 있다. 이러한 타입은 구조체 필드나 함수 매개변수 등을 선언하는 데 사용된다. 구조적으로 일반 타입은 자연수($\mathbb{N}$, 표 5에서 정의됨)로 표현되는 인덱스로, 상위 구조체나 함수 선언의 일반 매개변수 목록을 참조한다.

무브의 *벡터 타입(vector type)*은 동일한 타입의 값들을 가변 길이로 담는 컬렉션을 표현한다. 무브 벡터는 저장 가능한 타입만을 포함할 수 있으며, 그 자체도 저장 가능한 타입이다.

무브 프로그램은 값에 직접 연산을 수행하거나 참조(reference)를 통해 간접적으로 접근할 수 있다. *참조 타입(reference type)*은 참조 대상의 저장 가능한 타입과, 해당 값이 읽기와 쓰기가 모두 가능한지 혹은 읽기 전용인지를 결정하고 강제하는 *변경성 한정자(mutability qualifier)*를 함께 포함한다. 따라서 표 2의 타입 정의에서 볼 수 있듯, 무브의 가장 일반적인 형태의 값 타입은 저장 가능한 타입이거나 참조 타입일 수 있다.

마지막으로, 무브의 *어빌리티*는 특정 타입의 값에 대해 허용되는 동작(예: 복사 가능 여부)을 제어한다. 어빌리티는 구조체 선언 및 일반 타입 매개변수에 제약을 부여하며, 무브 바이트코드 검증기는 복사(copy)와 같은 민감한 연산이 해당 어빌리티를 가진 타입에 대해서만 수행되도록 보장한다.

3.3 객체와 소유권(Objects and Ownership)

TxDigest =	$Com(Tx)$
ObjID =	$Com(TxDigest \times N)$
SingleOwner =	$Addr \uplus ObjID$
Shared =	$\{shared_mut, shared_immut\}$
Ownership =	$SingleOwner \uplus Shared$
StructObj =	$StructType \times Struct$
ObjContents =	$StructObj \uplus Package$
Obj =	$ObjContents \times ObjID \times Ownership \times Version$

Table 3: Objects and Ownership

각 수이 객체는 전역적으로 고유한 식별자(Obj ID, 표 3 참조)를 가지며, 이는 객체가 소유자 간을 오가거나 다른 객체 내부로 들어가거나 나올 때까지 지속적으로 유지되는 객체의 영구적 정체성(persistent identity)을 나타낸다. 이 ID는 객체를 생성하는 트랜잭션에 의해 부여된다. 객체 ID는 현재 트랜잭션의 내용과, 해당 트랜잭션이 지금까지 생성한 객체의 수를 기록하는 카운터(counter)에 대해 충돌 저항성 해시 함수(collision-resistant hash function)를 적용함으로써 생성된다.

트랜잭션(따라서 그 다이제스트(digest) 역시)은 트랜잭션의 입력 객체에 대한 제약 조건으로 인해 항상 고유함이 보장되며, 이에 대해서는 뒤에서 자세히 설명한다.

객체는 ID 외에도 소유권에 대한 메타데이터를 포함한다. 객체는 주소나 다른 객체에 의해 고유하게(unique) 소유될 수도 있고, 읽기/쓰기 권한을 가진 공유(shared) 상태이거나, 읽기 전용(read-only) 공유 상태일 수도 있다.

객체의 소유권은 트랜잭션이 해당 객체를 입력으로 사용할 수 있는지, 그리고 사용할 수 있다면 어떤 방식으로 사용하는지를 결정한다. 일반적으로 고유 소유 객체는 해당 객체의 소유자가 시작한 트랜잭션이거나, 부모 객체(parent object)를 입력으로 포함하는 트랜잭션에서만 사용할 수 있다. 반면 공유 객체는 모든 트랜잭션에서 사용할 수 있지만, 지정된 변경 가능(mutability) 권한 내에서만 접근 가능하다. 전체 설명은 4.4절을 참고하길 바란다.

객체는 두 가지 유형으로 나뉜다: *패키지 코드 객체(package code objects)*와 *구조체 데이터 객체(struct data objects)*. 패키지 객체는 여러 모듈의 리스트를 포함하며, 구조체 객체는 무브 구조체 값과 해당 값의 무브 타입을 포함한다.

객체의 내용은 변경될 수 있지만, 객체의 ID, 객체 유형(패키지 vs 구조체), 그리고 무브 구조체 타입은 변경될 수 없다. 이로써 객체는 강한 타입 안정성(strong typing)과 영구적 정체성을 유지할 수 있다.

마지막으로, 각 객체는 버전을 가진다. 새로 생성된 객체는 버전 0을 가지며, 객체가 트랜잭션의 입력으로 사용될 때마다 그 버전이 증가한다.

3.4 주소와 인증자(Addresses and Authenticators)

Authenticator =	Ed25519PubKey $\uplus$ ECDSAPubKey $\uplus$...
Addr =	Com(Authenticator)

Table 4: Addresses and Authenticators

주소는 수이 사용자의 영구적 정체성을 나타낸다. 단, 하나의 사용자가 임의의 개수의 주소를 가질 수 있음을 유의해야 한다. 객체를 다른 사용자에게 전송하려면, 송신자는 수신자의 주소를 알아야 한다.

앞으로 설명하겠지만, 수이 트랜잭션은 송신자(즉, 트랜잭션을 시작한 사용자)의 주소와, 해당 주소와 일치하는 다이제스트를 가진 *인증자(authenticator)* 를 반드시 포함해야 한다. 주소와 인증자의 분리는 *암호학적 유연성(cryptographic agility)* 을 가능하게 한다. 인증자는 서로 다른 키 길이를 사용하는 서명 체계라도 사용할 수 있으며, 어떤 서명 체계의 공개키(public key)도 인증자가 될 수 있다 (예: 포스트 양자 서명(post-quantum signatures) 지원을 위해). 또한 인증자는 반드시 단일 공개키일 필요는 없다. 예를 들어 K-of-N 다중서명(multisig) 키와 같은 형태일 수도 있다.

3.5 트랜잭션(Transactions)

ObjRef =	ObjID $\times$ Version $\times$ Com(Obj)
CallTarget =	ObjRef $\times$ ModuleName $\times$ FunName
CallArg =	ObjRef $\uplus$ ObjID $\uplus$ PrimType
Package =	[Module]
Publish =	Package $\times$ [ObjRef]
Call =	CallTarget $\times$ [StorableType] $\times$ [CallArg]
GasInfo =	ObjRef $\times$ MaxGas $\times$ BaseFee $\times$ Tip
Tx =	(Call $\uplus$ Publish) $\times$ GasInfo $\times$ Addr $\times$ Authenticator

Table 5: Transactions

수이에는 두 가지 유형의 트랜잭션이 존재한다: 새로운 무브 패키지의 게시(publishing)와, 이전에 게시된 무브 패키지의 호출(calling)이다. 게시 트랜잭션(publish transaction)은 *패키지(package)* 를 포함하는데, 이는 함께 게시될 모듈들의 집합과, 해당 패키지의 모든 모듈이 의존하는 의존성(dependencies)을 포함한다. 이 의존성들은 이미 게시된 패키지 객체를 참조하는 객체 참조(object reference)의 리스트로 인코딩된다. 게시 트랜잭션이 실행될 때, 수이 런타임은 각 패키지에 대해 무브 바이트코드 검증기(bytecode verifier)를 실행하고, 패키지를 그 의존성과 연결하며, 각 모듈의 초기화 함수(module initializer)를 실행한다.

모듈 초기화 함수는 패키지가 구현하는 애플리케이션의 초기 상태(initial state)를 부트스트래핑(bootstrapping)하는 데 유용하다.

호출 트랜잭션(call transaction)의 가장 중요한 인자는 객체 입력이다.

객체 인자는 객체 참조(object reference) 또는 객체 ID로 지정된다.

단일 소유(single-owner) 객체나 변경 불가능한 공유(shared immutable) 객체의 경우 객체 참조를 사용하며, 변경 가능한 공유(shared mutable) 객체의 경우 객체 ID를 사용한다. 객체 참조는 객체 ID,

객체 버전, 그리고 객체 값의 해시로 구성된다. 수이 런타임은 객체 ID와 객체 참조를 전역 객체 풀에 저장된 객체 값으로 해석(resolve)한다. 객체 참조의 경우, 런타임은 참조된 버전이 풀에 저장된 객체의 버전과 일치하는지, 그리고 참조된 해시가 풀에 있는 객체의 해시와 일치하는지를 확인한다. 이 검증을 통해 런타임이 인식하는 객체 상태와 트랜잭션 송신자가 인식하는 객체 상태가 일치함을 보장한다.

또한 호출 트랜잭션은 타입 인자(type arguments)와 순수 값 인자(pure value arguments)를 허용한다. 타입 인자는 호출될 엔트리포인트 함수의 일반 타입 매개변수를 인스턴스화하며, 예를 들어 `send_coin<T>(c: Coin<T>, ...)` 함수의 경우, 일반 타입 매개변수 T를 SUI로 인스턴스화하여 수이 네이티브 토큰을 전송할 수 있다. 순수 값 인자는 기본 타입(primitive types) 및 기본 타입의 벡터를 포함할 수 있지만, 구조체 타입(struct types)은 포함할 수 없다.

호출될 함수는 객체 참조(패키지 객체를 참조해야 함), 해당 패키지 내의 모듈 이름, 그리고 함수 이름을 통해 지정된다. 호출 트랜잭션을 실행하기 위해 수이 런타임은 해당 함수를 해석하고, 타입, 객체, 값 인자를 함수의 매개변수에 할당(bind)한 후, 무브 VM을 사용하여 해당 함수를 실행한다.

게시 및 호출 트랜잭션 모두 가스 측정(gas metering)과 가스 요금(gas fees)이 적용된다. 측정 한도는 최대 가스 예산(maximum gas budget)으로 표현된다. 런타임은 예산이 소진될 때까지 트랜잭션을 실행하며, 예산이 초과되면 트랜잭션은 중단되고 상태 변화는 적용되지 않는다(단, 수수료 공제 및 중단 코드 보고는 수행된다).

가스 요금은 객체 참조로 지정된 *가스 객체*에서 공제된다.

이 객체는 반드시 수이 네이티브 토큰이어야 하며, 즉 그 타입이 `Coin<SUI>`여야 한다. 수이는 [EIP-1559](#) 스타일의 수수료 체계를 사용한다. 프로토콜은 에포크(epoch) 경계마다 알고리즘적으로 조정되는 기본 수수료(Base Fee, Sui 토큰당 가스 단위로 표시)를 정의하며, 트랜잭션 송신자는 선택적으로 팁(Tip, Sui 단위)을 추가할 수도 있다. 정상적인 시스템 부하 상태에서는 팁이 없어도 트랜잭션이 즉시 처리된다. 그러나 시스템이 혼잡할 경우, 더 높은 팁을 제공한 트랜잭션이 우선 처리된다. 가스 객체에서 공제되는 총 수수료는 $(GasUsed * BaseFee) + Tip$ 으로 계산된다.

3.6 트랜잭션 효과(Transaction Effects)

Event =	StructType × Struct
Create =	Obj
Update =	Obj
Wrap =	ObjID × Version
Delete =	ObjID × Version
ObjEffect =	Create ⊔ Update ⊔ Wrap ⊔ Delete
AbortCode =	$\mathbb{N} \times \text{ModuleName}$
SuccessEffects =	[ObjEffect] × [Event]
AbortEffects =	AbortCode
TxEffects =	SuccessEffects ⊔ AbortEffects

Table 6: Transaction Effects

트랜잭션 실행은 트랜잭션 효과를 생성하는데, 트랜잭션 효과는 트랜잭션 실행이 성공했을 때 (*SuccessEffects*, 표 6 참조)와 실패했을 때(*AbortEffects*, 표 6 참조)가 다르다.

트랜잭션이 성공적으로 실행되면, 트랜잭션 효과에는 수이 전역 객체 풀에 가해진 변경 사항(기존 객체의 갱신 및 새로 생성된 객체 포함)과 트랜잭션 실행 중 발생한 이벤트(events)에 대한 정보가 포함된다. 또한 성공적인 실행의 또 다른 효과로는 전역 풀에서의 객체 제거(삭제) 또는 한 객체를 다른 객체 안에 래핑하는 (wrapping) 연산이 있다. 래핑된 객체는 전역 풀에서 사라지고, 이를 래핑한 객체의 일부로만 존재하게 되므로 삭제와 유사한 효과를 갖는다. 삭제되거나 래핑된 객체는 더 이상 전역 풀에서 접근할 수 없기 때문에, 이러한 효과는 해당 객체의 ID와 버전으로 표현된다.

이벤트는 전역 객체 풀의 업데이트 이외에 발생하는 성공적인 트랜잭션 실행의 부수 효과(side effects)를 인코딩한다. 구조적으로 이벤트는 무브 구조체와 그 타입으로 구성된다. 이벤트는 블록체인 외부 주체가 소비하도록 설계되었으며, 무브 프로그램에서는 이를 읽을 수 없다.

무브 트랜잭션은 '*all-or-nothing*' 의미론을 따른다. 즉, 트랜잭션이 실행 도중 어떤 시점에서 중단되면 (예: 예기치 않은 실패 발생 시), 그 이전에 객체가 변경되었거나 이벤트가 발생했더라도, 이러한 효과는 트랜잭션 중단 시 모두 무효화된다. 대신, 중단된 트랜잭션의 효과에는 숫자형 중단 코드(abort code)와 중단이 발생한 모듈의 이름이 포함된다. 중단된 트랜잭션에 대해서도 가스 수수료는 여전히 부과된다.

4. 수이 시스템(The Sui System)

이 장에서는 비잔틴 장애(Byzantine failures)가 존재하더라도 권한 주체 간의 안전성과 생존성을 보장하기 위한 수이의 시스템적 관점을 설명한다.

또한 시스템의 전체 상태를 검증하지 않고도 상태에 대한 일정 수준의 신뢰를 필요로 하는 라이트 클라이언트 등 클라이언트의 동작 방식도 함께 설명한다.

간략한 배경(Brief Background). 시스템 수준에서 수이는 FastPay [3]의 저지연(low-latency) 결제 시스템을 발전시킨 형태이다. 좀 더 구체적으로 사용자 정의 스마트 계약을 통해 임의의 객체에 대해 동작하도록 확장되었고, 비허가적인 위임 지분 증명 기반 위원회 구성 방식을 채택하고 있다 [2]. 객체 소유자에 의한 기본적인 자산 관리는 Byzantine consistent broadcast [6]의 변형된 형태를 기반으로 하며, 이는 전통적인 비잔틴 합의 [8, 11, 12] 구현에 비해 더 적은 지연과 더 높은 확장성을 제공한다. 완전한 합의가 필요한 경우, 고성능 DAG 기반 합의 [9]를 사용하며(예: 잠금 관리), 서로 다른 공유 객체들에 대한 실행은 병렬화된다.

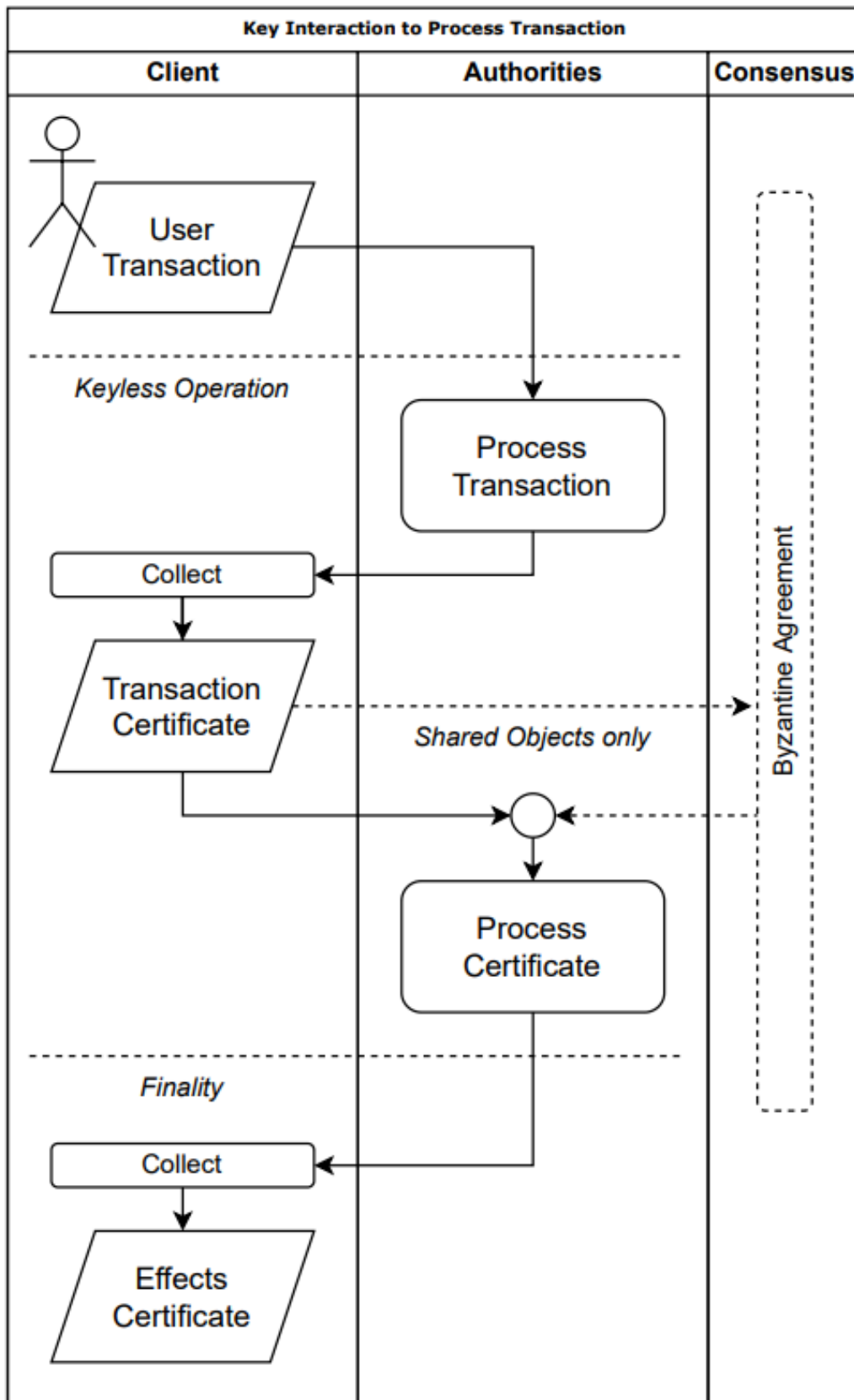


Figure 1: Outline of interactions to commit a transaction.

프로토콜 개요(Protocol Outline). 그림 1은 트랜잭션을 커밋(commit)하기 위해 클라이언트와 수이 권한 주체 간에 발생하는 상호작용의 개요를 보여준다. 아래에서는 그 절차를 간략히 설명한다.

- 사용자는 개인 서명 키를 사용하여 자신이 소유하거나 공유된 객체를 변경하기 위한 *사용자 트랜잭션 (user transaction)* 을 생성하고 서명한다. 이후 단계에서는 사용자 서명 키가 더 이상 필요하지 않으며, 나머지 과정은 사용자 클라이언트가 직접 수행하거나, 사용자 대신 게이트웨이(gateway)가 수행할 수 있다(그림에서는 '*keyless operation*' 으로 표시됨).
- 사용자 트랜잭션은 수이 권한 주체들에게 전송되며, 각 권한 주체는 이를 검증하고, 검증이 성공하면 트랜잭션에 서명한 후 *서명된 트랜잭션* 을 클라이언트에게 반환한다. 클라이언트는 정족수 (quorum)를 충족하는 권한 주체의 응답을 수집하여 *트랜잭션 인증서(transaction certificate)* 를 구성한다.
- 트랜잭션 인증서는 다시 모든 권한 주체들에게 전송된다. 트랜잭션이 공유 객체를 포함하는 경우, 인증서는 또한 수이 권한 주체가 주도하는 *비잔틴 합의 프로토콜 운영자(Byzantine agreement protocol operator)* 에게 전송된다. 권한 주체들은 인증서를 확인하고, 공유 객체가 관련된 경우 합의 프로토콜을 통해 해당 트랜잭션이 다른 공유 객체 트랜잭션과의 순서를 결정할 때까지 대기한 후 트랜잭션을 실행하고, 그 결과를 요약한 *서명된 효과 응답(signed effects response)* 을 생성한다.
- 권한 주체 정족수가 인증서를 실행하고 나면, 그 트랜잭션의 효과는 최종적(final)으로 확정된다(그림에서 *finally* 로 표시됨). 클라이언트는 정족수를 충족하는 권한 주체의 응답을 수집하여 *효과 인증서(effects certificate)* 를 생성하고, 이를 트랜잭션 효과의 최종성(finality)을 입증하는 증거로 사용할 수 있다.

이 장에서는 위 과정의 각 단계별 동작을 상세히 설명하고, 권한 주체 간의 상태 재구성(reconfiguration) 및 상태 관리(state management) 메커니즘에 대해서도 다룬다.

4.1 시스템 모델(System Model)

수이는 $\$e \in \{0, \dots\}$ 로 표시되는 일련의 에포크 단위로 동작한다.

각 에포크는 위원회 $\$C_e = (V_e, S_e(\cdot))$ 에 의해 관리된다.

여기서 $\$V_e$ 는 공개 검증 키와 네트워크 엔드포인트를 가진 권한 주체의 집합이며, 함수 $\$S_e(v)$ 는 각 권한 주체 $\$v \in V_e$ 를 해당 권한 주체에게 위임된 지분 단위(stake units) 수로 매핑한다.

각 에포크의 위원회 $\$C_e$ 는 이전 에포크 $\$e - 1$ 의 권한 주체 지분 정족수에 의해 서명된 것으로 가정한다(위원회 구성 및 관리에 대한 내용은 4.7절에서 논의한다).

하나의 에포크 내에서 일부 권한 주체는 올바르게 프로토콜을 충실히 수행하며 활성(live) 상태를 유지하는 정상(authority) 이고, 나머지는 프로토콜에서 임의로 이탈할 수 있는 *비잔틴(Byzantine)* 권한 주체일 수 있다.

보안 가정은 다음과 같다.

즉, 정직한 권한 주체 집합 $\$H_e \subset V_e$ 가 에포크 내에서 전체 지분의 정족수를 차지하며, 다음이 성립한다:

$$\sum_{h \in H_e} S_e(h) > \frac{2}{3} \sum_{v \in V_e} S_e(v)$$

즉, 전체 지분의 3분의 2를 초과하는 지분을 보유한 권한 주체 집합을 정족수(quorum)로 정의한다.

각 인증서(certificate)에 대해 *릴레이(relay)*로서 기능할 수 있는 최소 한 명의 활성화된 동시에 올바른 주체가 존재한다(자세한 내용은 4.3절 참조).

이는 생존성을 보장하며, 'Byzantine broadcast'의 최종 전달 속성(eventual delivery property, [6]의 신뢰 가능한 전파의 전체성(totality))을 제공한다.

각 권한 주체는 개별적으로 또는 집합적 전파 프로토콜(collective dissemination protocol)을 통해 이러한 릴레이를 운영한다.

수이 라이트 클라이언트, 레플리카(replicas), 외부 서비스 등 외부 주체도 이 역할을 수행할 수 있다.

이때 수동적 권한 주체 코어(*passive authority core*)와, 보다 신뢰성이 낮거나 신뢰할 수 없는 내부 또는 외부 활성화 릴레이 구성 요소(*active relay component*)를 구분함으로써, 수이의 안전성과 생존성이 의존하는 신뢰 컴퓨팅 기반(Trusted Computing Base, TCB) [15]을 명확히 분리하고 최소화한다.

4.2 권한 주체 및 레플리카 데이터 구조(Authority & Replica Data Structures)

수이의 권한 주체는 시스템 상태를 표현하기 위해 여러 데이터 구조에 의존한다. 이러한 구조들은 지원하는 연산에 따라 정의되며, 모두 결정적(deterministic) 바이트 표현을 가진다.

*객체(Object, Obj)*는 수이 내에서 사용자 스마트 컨트랙트와 데이터를 저장한다. 이는 2장에서 소개된 무브 객체의 수이 시스템 수준 인코딩이다. 객체는 다음과 같은 연산들을 지원한다.

- `ref(Obj)`는 객체의 참조(ObjRef)를 반환한다. 즉, (ObjID, Version, ObjDigest)의 삼중쌍(triplet)을 반환하며, ObjID는 새로 생성된 모든 객체에서 사실상 전역적으로 고유하고, Version은 객체가 변경될 때마다 증가하는 양의 정수이다.
- `owner(Obj)`는 객체의 소유자(owner)에 대한 인증자(Auth)를 반환한다. 가장 단순한 경우 Auth는 공개키를 나타내는 주소이며, 이 키를 가진 사용자가 해당 객체를 사용할 수 있다. 보다 복잡한 형태의 인증자도 존재한다(자세한 내용은 4.4절 참조).
- `read-only(Obj)`는 객체가 읽기 전용(read-only)인 경우 true를 반환한다. 읽기 전용 객체는 수정, 래핑, 삭제될 수 없으며, 소유자뿐 아니라 누구나 사용할 수 있다.
- `parent(Obj)`는 해당 객체를 마지막으로 변경하거나 생성한 트랜잭션의 다이제스트(TxDigest)를 반환한다.
- `contents(Obj)`는 객체의 타입(Type)과 데이터(Data)를 반환하며, 이는 트랜잭션의 유효성을 검증하고 객체의 애플리케이션 특화 정보를 포함하는 데 사용된다.

객체 참조(ObjRef)는 객체를 인덱싱하는 데 사용되며, 동시에 객체의 전체 내용에 대한 커밋으로서 인증에도 사용된다. ObjDigest는 객체의 전체 내용을 해시하여 생성된 값이다.

*트랜잭션(Transaction, Tx)*은 하나 이상의 객체에 대한 상태 전이를 나타내는 구조체이다. 트랜잭션은 다음과 같은 연산들을 지원한다.

- `digest(Tx)`는 트랜잭션의 암호학적 커밋먼트(cryptographic commitment)를 반환하며, 이는 트랜잭션의 다이제스트(TxDigest)로 표현된다.
- `epoch(Tx)`는 해당 트랜잭션이 실행될 수 있는 에포크 식별자(EpochID)를 반환한다.
- `inputs(Tx)`는 트랜잭션 실행에 필요한 객체 참조들의 시퀀스([ObjRef])를 반환한다.
- `payment(Tx)`는 가스 지불에 사용될 ObjRef 참조를 반환하며, 여기에는 최대 가스 한도와 가스

단위와 가스 결제 객체 내 가치 단위 간의 변환 비율이 포함된다.

- `valid(Tx, [Obj])`는 지정된 입력 객체가 제공된 경우, 트랜잭션이 유효한지 여부를 반환한다. 유효성은 4.4절에서 자세히 다루며, 이는 트랜잭션이 해당 입력 객체에 대해 수행할 권한이 있는지 여부와, 실행 비용을 충당하기 위한 충분한 가스가 존재하는지 여부와 관련된다.
- `exec(Tx, [Obj])`는 트랜잭션을 실행하고, 그 효과를 나타내는 *효과 구조체(Effects)*를 반환한다. 유효한 트랜잭션의 실행은 실패하지 않으며(*infallible*), 그 출력은 결정적(*deterministic*)이다.

트랜잭션은 그 `TxDigest`로 인덱싱되며, 이 다이제스트는 트랜잭션의 전체 내용을 인증하는 데에도 사용된다. 모든 유효한 트랜잭션(하드코딩된 제네시스 트랜잭션을 제외하고)은 최소 하나의 소유된 입력(*owned input*)을 가지며, 이는 가스 지불에 사용되는 객체이다.

트랜잭션 효과(Transaction Effects, Effects) 구조체는 트랜잭션 실행의 결과를 요약한 것이다. 이 구조체는 다음의 연산을 지원한다.

- `digest(Effects)`는 효과 구조체에 대한 커밋먼트(*EffDigest*)를 반환한다. 이는 인덱싱 또는 인증을 위한 식별자로 사용된다.
- `transaction(Effects)`는 해당 효과를 생성한 실행된 트랜잭션의 다이제스트(`TxDigest`)를 반환한다.
- `dependencies(Effects)`는 해당 트랜잭션이 실행되기 전에 반드시 실행되어야 하는 의존성 트랜잭션들의 다이제스트 시퀀스(`[TxDigest]`)를 반환한다.
- `contents(Effects)`는 실행 요약(*summary of execution*)을 반환한다. 여기에는 스마트 컨트랙트 실행의 상태, 생성/변경/래핑/언래핑/삭제된 객체 참조 목록, 그리고 실행 중 발생한 이벤트 목록이 포함된다.

*트랜잭션 인증서(Transaction Certificate, TxCert)*는 트랜잭션 내용 자체와, 정족수(*quorum*)에 해당하는 권한 주체들의 식별자 및 서명(*signature*)을 포함한다. 단, 인증서는 반드시 유일(*unique*)하지 않을 수 있다. 동일한 논리적 인증서가 서로 다른 권한 주체 집합에 의해 형성된 정족수로 표현될 수 있기 때문이다. 또한, 인증서는 정확히 3분의 2 정족수에 의해 서명되지 않을 수도 있으며, 응답성이 높은 권한 주체가 더 많은 경우 추가 서명이 포함될 수도 있다. 그러나 동일한 트랜잭션에 대한 두 개의 서로 다른 유효 인증서는 의미적으로 동일한 인증서로 간주해야 한다. *부분 인증서(Partial Certificate, TxSign)*는 동일한 정보를 포함하지만, 정족수에 미치지 못하는(일반적으로 단일 권한 주체에 해당하는) 서명을 포함한다. 인증서에는 서명자들의 식별자와 서명이 포함되며(즉, 책임 가능한 서명(*accountable signatures*) [?]), 이를 통해 권한 주체를 식별할 수 있다. 인증서를 처리할 준비가 된 권한 주체이거나, 인증서 처리를 위해 과거 정보를 다운로드해야 하는 경우(4.8절 참조)에도 동일한 구조가 사용된다.

마찬가지로, *효과 인증서(Effects Certificate, EffCert)*는 효과 구조체(*effects structure*) 자체와, 해당 에포크에서 트랜잭션이 유효함을 나타내는 권한 주체들의 서명을 포함한다. (참고로, 서명 알고리즘이 허용하는 경우 권한 주체 서명은 압축될 수 있으나, 가스 보상 분배 및 네트워크 상태 측정을 위해 추적 가능해야 하므로, 항상 책임 가능한 서명 집계(*accountable signature aggregation*)를 사용해야 한다.) 비고 사항으로는, 효과 인증서 역시 트랜잭션 인증서와 마찬가지로 비고유성(*non-uniqueness*) 및 동일성(*identity*)에 대한 동일한 주의가 적용된다.

단일 권한 주체의 서명과 효과 구조체만을 포함하는 부분 효과 인증서(*partial effects certificate*)는 `EffSign`으로 표시된다.

영구 저장소(Persistent Stores). 각 권한 주체 및 레플리카는 하나 이상의 영구 저장소를 유지한다. 이 저장소는 영구적 맵(persistent map) 의미론을 구현하며, 키-값 쌍(map[key] → value)의 집합으로 표현될 수 있다. 각 키에 대해 하나의 쌍만 존재한다. 새로운 키-값 쌍이 삽입되기 전에는 contains(key) 호출이 false를 반환하고, get(key) 호출은 오류를 반환한다. 삽입 후에는 contains(key)가 true를 반환하며, get(key)는 해당 값을 반환한다. 각 권한 주체는 다음의 영구 저장소들을 유지한다:

- **오더 락 맵(Order Lock Map)** \$Lock_v[ObjRef] \to TxSignOption\$ 은 권한 주체가 소유 객체 버전(ObjRef)에 대해 처음으로 관찰하고 서명한 유효한 트랜잭션(Tx)을 기록한다. 만약 해당 객체 버전이 존재하지만 이를 입력으로 사용하는 유효한 트랜잭션이 아직 관찰되지 않았다면, 값은 None으로 설정된다. 또한 이 맵은 해당 객체를 입력으로 사용하는 첫 번째 인증서(certificate) 역시 기록할 수 있다. 이 테이블과 그 업데이트 규칙은 수이 권한 주체 간의 객체에 대한 분산 락(distributed lock)의 상태를 나타내며, 트랜잭션이 동시에 처리될 때 안전성을 보장한다.
- **인증서 맵(Certificate Map)** \$Ct_v[TxDigest] \to (TxCert, EffSign)\$ 은 권한 주체가 유효 에포크(validity epoch) 내에서 처리한 모든 완전한 인증서(TxCert)를 기록한다. 각 엔트리는 트랜잭션(Tx) 자체뿐 아니라, 그에 대한 서명된 효과(EffSign)도 함께 포함하며, 트랜잭션 다이제스트(TxDigest)를 기준으로 인덱싱된다.
- **객체 맵(Object Map)** \$Obj_v[ObjRef] \to Obj\$ 은 \$Ct_v\$ 내의 트랜잭션들에 의해 생성된 모든 객체(Obj)를 기록하며, 객체 참조(ObjRef)를 기준으로 인덱싱된다. 이 저장소는 \$Ct_v\$ 내의 모든 인증서를 재실행함으로써 완전히 복원할 수 있다. 또한, Obj ID를 해당 ID를 가진 최신 객체에 매핑하는 보조 인덱스(secondary index)를 유지한다. 이 인덱스는 새로운 트랜잭션을 처리하기 위해 필요한 유일한 정보이며, 이전 버전은 읽기 및 감사를 용이하게 하기 위해서만 유지된다.
- **동기화 맵(Synchronization Map)** \$Sync_v[ObjRef] \to TxDigest\$ 는 \$Ct_v\$ 내의 모든 인증서를 인덱싱하며, 각 인증서가 생성, 변경, 또는 삭제한 객체들을 ObjRef 튜플 형태로 기록한다. 이 구조는 \$Ct_v\$ 의 모든 인증서를 처리함으로써 완전히 재구성할 수 있으며, 클라이언트가 관심 있는 객체에 영향을 미치는 트랜잭션들을 동기화(synchronize)하는 데 사용된다.

권한 주체는 위의 네 가지 구조를 모두 유지하며, 자신의 인증서 맵의 로컬 체크포인트(local checkpoint)에 접근할 수 있도록 제공한다. 이를 통해 다른 권한 주체나 레플리카가 완전한 인증서 집합을 다운로드하여 업데이트할 수 있다. 레플리카는 트랜잭션을 직접 처리하지 않으며, 인증서만 처리하여 다른 테이블들을 업데이트하기 위해 재실행한다. 또한 비이중성(non-equivocation) 감사를 위해 오더 락 맵을 유지한다.

권한 주체는 읽기 및 동기화를 용이하게 하기 위해, 네 가지 저장소(및 체크포인트)를 모두 유지하는 완전 레플리카로 설계될 수 있다. 이 경우 새로운 트랜잭션 및 인증서를 처리하는 데 필요한 최신 객체 버전과 객체 락만 유지하는 최소 권한 주체 코어(minimal authority core)와 결합할 수 있다. 이 설계는 안전성을 위해 의존하는 신뢰 컴퓨팅 기반을 최소화한다.

오더 락 맵만이 강력한 키 자체 일관성(key self-consistency)을 요구한다. 즉, 존재하는 키에 대해 읽기는 항상 해당 키에 값이 존재하는지(value) 또는 존재하지 않는지(None)를 정확히 반환해야 하며, 락을 None이 아닌 값으로 설정하는 업데이트와 이러한 확인은 원자적으로 수행되어야 한다. 이는 키 간 강한 일관성보다 약한 제약이지만, 저장소의 샤딩(sharding)을 효율적으로 수행하여 확장성을 향상시킬 수 있게 한다. 나머지 저장소들은 최종적 일관성(eventual consistency)을 가져도 안전성에 영향을 미치지 않는다.

4.3 권한 주체의 기본 연산(Authority Base Operation)

트랜잭션 처리(Process Transaction). 권한 주체는 트랜잭션 Tx 를 수신하면 다음의 일련의 검증 절차를 수행한다:

1. $\$epoch(Tx)$ 가 현재 에포크(current epoch)인지 확인한다.
2. $\$inputs(Tx)$ 내의 모든 객체 참조와, $\$payment(Tx)$ 내의 가스 객체 참조가 $\$Obj_v$ 내에 존재하는지 확인하고, 이를 $\$[Obj]$ 로 로드한다. 소유 객체의 경우 정확한 객체 참조가 존재해야 하며, 읽기 전용 또는 공유객체의 경우 객체 ID만 존재하면 된다.
3. 트랜잭션 실행 비용을 충당할 만큼 충분한 가스가 가스 객체 내에 존재하는지 확인한다.
4. $\$valid(Tx, [Obj])$ 가 true인지 검사한다. 이 단계는 트랜잭션 내의 인증 정보가 소유 객체에 대한 접근 권한을 적절히 부여하는지를 검증한다.
5. 모든 소유 객체 입력(owned $\$inputs(Tx)$ objects)에 대해, $\$Lock_v[ObjRef]$ 가 존재하며 그 값이 None이거나 동일한 $\$Tx$ 로 설정되어 있는지를 확인하고, 이를 원자적으로(atomic) $\$TxSign$ 으로 설정한다.(이 과정을 “락 검증(locks checks)”이라 부른다.)

만약 위의 어느 한 검증이라도 실패하면, 트랜잭션 처리를 중단하고 오류를 반환한다. 그러나 $\$Lock_v$ 에 대한 일부 부분 업데이트가 지속되는 것은 안전하다. (현재의 구현은 부분 업데이트 대신, 모든 락을 원자적으로 갱신하는 방식을 사용한다.)

모든 검증이 성공적으로 완료되면, 권한 주체는 해당 트랜잭션에 대한 서명을 반환한다. 즉, 부분 인증서인 $\$TxSign$ 을 반환한다. 트랜잭션 처리는 성공 시 멍등성(idempotent)을 가지며, 결과로 부분 인증서($\$TxSign$) 또는 이미 완전 인증서가 존재하는 경우 전체 인증서($\$TxCert$)를 반환한다.

모든 참여자는 특정 에포크 $\$e$ 에서 정족수를 형성하는 권한 주체 집합으로부터 수집된 트랜잭션과 서명들($\$TxSign$)을 조합하여 트랜잭션 인증서($\$TxCert$)를 생성할 수 있다.

인증서 처리(Process Certificate). 권한 주체는 인증서를 수신하면, 트랜잭션의 모든 유효성 조건을 검사하되, 락 검증(locks checks)에 해당하는 부분은 제외한다. 대신 다음과 같은 검증 절차를 수행한다:

$inputs(Tx)$ 내의 각 소유 입력 객체(owned input object)에 대해 락이 존재하는지 확인하고, 그 값이 None, 임의의 $TxSign$, 또는 현재 인증서와 동일한 트랜잭션에 대한 인증서로 설정되어 있는지를 검사한다. 이 수정된 락 검증이 실패할 경우, 권한 주체는 복구 불가능한 비잔틴 장애를 감지한 것이므로, 정상 운영을 중단하고 재해 복구(disaster recovery) 절차를 시작한다. 공유 객체(4.4절 참조)의 경우, 권한 주체는 락이 합의를 통해 시퀀싱된 인증서를 통해 설정되었는지 확인한다. 이를 통해 어떤 버전의 공유 객체를 사용할지를 결정한다. 만약 시퀀싱이 완료된 경우 트랜잭션을 즉시 실행할 수 있으며, 그렇지 않다면 시퀀싱이 완료될 때까지 대기해야 한다.

검증이 성공하면, 권한 주체는 인증서를 자신의 인증서 맵에 추가하고, 그 실행 결과로 생성된 효과(effects)를 함께 저장한다. 즉, 다음과 같이 기록한다: $\$Ct_v[TxDigest] \rightarrow (TxCert, EffSign)$ 또한 락 맵을 갱신하여, 아직 인증서로 설정되지 않은 모든 소유 객체 입력에 대해 해당 락을 인증서로 기록한다: $\$Lock_v[ObjRef] \rightarrow TxCert$ 그리고 $\$Input(Tx)$ 내의 모든 객체가 $\$Obj_v$ 에 삽입되면, $\$EffSign$ 내의 모든 효과도 객체 참조($ObjRef$)와 함께 그 내용을 $\$Obj_v$ 에 추가하여 구체화(materialize)된다. 마지막으로, $\$EffSign$ 내에서 생성되거나 변경(mutated)된 모든 객체에 대해, 동기화 맵($\$Sync_v$)을 갱신하여 이를 트랜잭션 $\$Tx$ 에 매핑한다.

비고 (Remarks). 트랜잭션 및 인증서를 처리하는 로직은 다음과 같은 핵심 속성들을 보장한다.

- **인과성 및 병렬성(Causality & Parallelism).** 트랜잭션 및 인증서의 처리 조건은 인과적 실행 (causal execution)을 보장한다. 권한 주체는 트랜잭션이 의존하는 모든 객체(소유, 공유, 읽기 전용)에 대한 인증서 처리를 완료한 후에만 트랜잭션에 서명(즉, 투표)한다. 마찬가지로, 권한 주체는 트랜잭션 인증서를 처리하기 전에 그 인증서가 의존하는 모든 입력 객체가 자신의 로컬 객체 맵에 존재해야만 이를 처리한다. 이러한 설계는 인과적 실행 순서를 부과하지만, 서로 인과적으로 독립적인 트랜잭션은 서로 다른 코어 또는 머신에서 병렬 실행이 가능하도록 한다.
- **단일 서명과 안전성(Sign Once, and Safety).** $\$Lock_v[\cdot]\$$ 내의 모든 소유 입력 객체의 락은 해당 객체를 사용하는 첫 번째 유효 트랜잭션 $\$Tx\$$ 와, 이후 해당 객체를 입력으로 사용하는 첫 번째 인증서로만 설정된다. 이를 *객체를 트랜잭션에 락킹(locking the object to this transaction)* 이라고 하며, 하나의 에포크 내에서는 락이 해제되지 않는다. 따라서 권한 주체는 각 락에 대해 단 하나의 트랜잭션에만 서명하며, 이는 consistent broadcast [6]의 핵심 요소이자 수이의 안전성을 보장하는 핵심 원리이다.
- **재해 복구(Disaster Recovery).** 동일한 락에 대해 서로 상충되는 두 개의 인증서를 감지한 권한 주체는, 복구 불가능한 비잔틴 행위(Byzantine behaviour)가 발생했음을 입증할 수 있다. 이는 정직한 권한 주체 정족수(quorum honest authority) 가정이 깨졌음을 의미하며, 두 상충 인증서가 바로 *사기 증명(fraud proof)* [1] 이 된다. 이러한 사기 증명은 다른 모든 권한 주체와 레플리카에게 공유되어 재해 복구 절차를 자동으로 발동할 수 있다. 또한 다른 형태의 비잔틴 행위 증거도 존재할 수 있다. 예를 들어, 전체 지분의 3분의 1을 초과하는 수의 서명($> 1/3$ signatures)을 가진 잘못된 효과($\$EffSign\$$), 혹은 이전 인증서의 출력과 일치하지 않는 입력 객체를 사용하는 인증서 등이다. 이러한 경우에도 사기 증명으로 포장되어 네트워크 전체에 전파될 수 있다. 이는 지분 기준으로 3분의 1 이하의 권한 주체 혹은 일부 객체 소유자가 비잔틴 상태이거나 이중 서명(equivocating)하는 상황과는 구분된다. 이러한 소수의 결함은 서비스 중단 없이 허용 가능한 수준으로 처리될 수 있다.
- **최종성(Finality).** 권한 주체는 $\$Lock_v\$, \$Ct_v\$, \$Obj_v\$, \$Sync_v\$$ 내의 인덱스에 대한 모든 읽기 요청에 대해 인증서($\$TxCert\$$)와 서명된 효과($\$EffSign\$$)를 반환한다. 트랜잭션은 정족수에 해당하는 권한 주체가 $\$Tx\$$ 를 자신의 $\$Ct_v\$$ 저장소에 포함시켰다고 보고할 때 *최종적(final)* 으로 간주된다. 따라서 효과 인증서($\$EffCert\$$)는 최종성에 대한 전달 가능한 증명 (transferable proof of finality) 역할을 한다. 또한, 어떤 객체를 사용하는 인증서가 존재한다면, 그 인증서의 인과 경로(causal path)에 있는 모든 종속 인증서들도 이미 최종 확정되었음을 의미한다. 어떤 주체에게 인증서를 제공하여 이를 정족수를 초과하는(supermajority) 권한 주체에게 제출하면, 해당 인증서의 효과 역시 최종성을 확보할 수 있다. 수이에서의 최종성은 재구성 (reconfiguration) 시 안전성을 보장하기 위해 FastPay [3]보다 다소 늦게 확정된다. 그러나 권한 주체는 커밋(commit)을 기다리지 않고, 인증서를 수신하는 즉시 트랜잭션의 효과를 적용할 수 있다.

4.4 소유자, 권한 부여, 그리고 공유 객체(Owners, Authorization, and Shared Objects)

트랜잭션 유효성 검증(4.3절 참조)은 트랜잭션이 지정된 모든 입력 객체를 포함할 권한이 있는지를 보장한다. 이 검증은 객체의 성격과 owner 필드의 속성에 따라 달라진다.

읽기 전용 객체(Read-only Objects)는 변경되거나 삭제될 수 없으며, 모든 사용자가 동시에 트랜잭션 내에서 사용할 수 있다. 예를 들어 무브 모듈이 이에 해당한다. 이러한 객체는 스마트 컨트랙트의 일부로 사용될 수 있는 `owner` 필드를 가질 수 있지만, 그 소유자는 객체 사용에 대한 권한 부여에 아무런 영향을 미치지 않는다. 즉, 읽기 전용 객체는 어떤 트랜잭션에도 자유롭게 포함될 수 있다.

소유 객체 (Owned Objects)는 `owner` 필드를 가진다. 이 필드는 공개키를 나타내는 주소로 설정될 수 있다. 이 경우, 트랜잭션이 해당 주소의 서명으로 서명되어 있다면, 그 트랜잭션은 객체를 사용할 뿐 아니라 변경할 권한도 부여된다. 하나의 트랜잭션은 단일 주소로 서명되며, 따라서 해당 주소가 소유한 하나 이상의 객체를 동시에 사용할 수 있다. 그러나 하나의 트랜잭션이 여러 주소가 소유한 객체를 동시에 사용할 수는 없다. 또한 객체의 소유자는 다른 객체의 `Obj ID`로 설정될 수도 있다. 이때 자식 객체의 소유자는 부모 객체가 된다. 즉, 자식 객체는 부모 객체가 트랜잭션에 함께 포함된 경우에만 사용할 수 있으며, 이 경우 트랜잭션은 해당 객체에 대한 사용 권한이 부여된 것으로 간주된다. 이 기능은 스마트 컨트랙트가 효율적인 컬렉션(collection)이나 복합 데이터 구조(complex data structure)를 구현하는 데 유용하게 활용될 수 있다.

공유 객체 (Shared Objects)는 변경 가능한 객체이지만 특정한 소유자를 가지지 않는다. 따라서 서로 다른 주체(parties)가 트랜잭션에서 이를 포함할 수 있으며, 별도의 권한 부여 절차가 필요하지 않다. 대신, 공유 객체는 자체적인 권한 검증 로직(authorization logic)을 수행한다. 이러한 객체는 다수의 작성자(writers)를 동시에 지원하면서도 안전성과 생존성을 보장해야 하므로, 안전한 사용을 위해 완전한 합의 프로토콜(full agreement protocol)이 필요하다. 따라서 실행 전에 추가적인 논리적 절차가 요구된다. 권한 주체들은 소유 객체 및 읽기 전용 객체의 락을 관리하기 위해 4.3절에서 설명된 절차에 따라 트랜잭션을 처리한다. 하지만 공유 객체의 락은 consistent broadcast에 의존하지 않는다. 대신, 공유 객체를 포함하는 트랜잭션의 생성자는 해당 트랜잭션의 인증서(certificate)를 고성능 합의 시스템(high-throughput consensus system, 예: [9])에 삽입한다. 모든 권한 주체들은 이러한 인증서의 일관된 순서를 관찰하고, 각 트랜잭션에서 사용된 공유 객체의 버전을 이 순서(sequence)에 따라 할당한다. 그 후 실행이 진행되며, 모든 권한 주체 간에 일관성이 보장된다. 또한 권한 주체들은 트랜잭션 실행 시 사용된 공유 객체의 버전을 *효과 인증서(Effects Certificate)* 내에 포함한다.

위 규칙에 따라, 읽기 전용 및 소유 객체를 포함하는 트랜잭션의 실행에는 consistent broadcast와 단일 인증서만으로 충분하며, 비잔틴 합의는 공유 객체가 포함된 트랜잭션에 대해서만 필요하다. 따라서 스마트 컨트랙트 개발자는 단일 사용자 객체의 전송 및 연산을 빠르게 처리하면서, 필요할 때 공유 객체를 활용할 수 있도록 타입과 연산을 설계함으로써 효율성과 유연성을 모두 달성할 수 있다.

4.5 클라이언트(Clients)

풀 클라이언트 및 레플리카(Full Clients & Replicas). 레플리카는 *풀 클라이언트(full clients)* 라고도 불리며, 새로운 트랜잭션을 검증하지는 않는다. 대신 시스템의 유효한 상태를 일관되게 유지하여, 감사를 수행하거나 트랜잭션을 재실행하고, 라이트 클라이언트 쿼리를 위한 읽기 인프라(read infrastructure) 또는 서비스 운영을 지원하는 역할을 한다.

라이트 클라이언트 (Light Clients). 객체 참조와 트랜잭션 모두 해당 객체나 트랜잭션의 생성 또는 실행에 이르기까지의 전체 인과적 연쇄(causal chain)를 인증할 수 있는 정보를 포함한다. 구체적으로, 객체 참조(ObjRef)에는 객체의 전체 상태를 인증하는 ObjDigest가 포함되어 있으며, 이에는 객체를 생성한 트랜잭션 다이제스트(TxDigest)를 나타내는 parent(Obj) 필드도 포함된다. 마찬가지로, 트랜잭션 다이제스트(TxDigest)는 트랜잭션 자체를 인증하며, 이를 통해 inputs(Tx)를 추출하여 입력 객체들의 참조를 확인할 수 있다. 따라서 객체들과 인증서의 집합은 **자가 인증적(self-authenticating)** 양분 그래프(bipartite graph)를 형성한다. 또한, 효과 구조체는 서명되어 있으며, 이들은 트랜잭션 실행 결과를 직접 인증하는 효과 인증서로 결합될 수 있다.

이러한 기능 덕분에, 풀 노드를 유지하지 않고도 수이의 상태를 신뢰성 높게(high-integrity) 읽을 수 있는 *라이트 클라이언트(light client)*를 지원할 수 있다. 예를 들어, 권한 주체(authority) 또는 풀 노드는 트랜잭션 Tx에 대한 인증서(TxCert)와, inputs(Tx)에 해당하는 입력 객체들([Obj])을 포함하는 간결한 증거 묶음(bundle of evidence)을 제공하여, 라이트 클라이언트가 해당 상태 전이가 수이 내에서 실제로 발생할 수 있음을 검증할 수 있도록 할 수 있다. 라이트 클라이언트는 이 인증서를 직접 제출하거나, 정족수 또는 일부 표본(authority sample)에 상응하는 권한 주체들이 이를 확인했는지를 검사하여 최종성(finality)을 보장받을 수 있다. 또는 이 결과 객체들을 사용하여 새로운 트랜잭션을 생성하고, 그 트랜잭션이 성공적으로 실행되는지를 관찰할 수도 있다.

보다 직접적으로, 서비스는 클라이언트에게 효과 인증서를 제공함으로써 수이 내에서 발생한 상태 전이의 존재와 최종성을 추가 상호작용 없이 입증할 수 있다. 만약 에포크 경계 또는 기타 시점에서 최종 확정된 인증서가 존재한다면, 입력 객체와 인증서, 그리고 해당 인증서가 체크포인트에 포함되어 있음을 증명하는 포함 증명(proof of inclusion)을 함께 제공하는 것도 최종성의 증거(proof of finality)가 된다.

권한 주체들은 주기적 체크포인팅 메커니즘을 사용하여 최종 확정된 트랜잭션들의 집합 및 수이의 상태를 시간에 따라 집단적으로 기록할 수 있다. 체크포인트에 대해 정족수를 충족하는 지분의 서명을 포함한 인증서는 라이트 클라이언트가 최근 객체 상태나 발생한 이벤트를 효율적으로 검증하는 데 사용할 수 있다. 체크포인팅 메커니즘은 에포크 간 위원회 재구성을 위해 필수적이며, 보다 빈번한 체크포인트는 라이트 클라이언트에 유용할 뿐만 아니라, 권한 주체가 내부 데이터 구조를 압축하고 다른 권한 주체들과의 상태를 보다 효율적으로 동기화하는 데에도 활용될 수 있다.

4.6 브리지(Bridges)

비잔틴 합의에 의해 관리되는 공유 객체와 라이트 클라이언트에 대한 네이티브 지원을 통해, 수이는 다른 블록체인의 양방향 브리지(two-way bridge)를 구현할 수 있다 [13]. 이러한 브리지의 신뢰 가정(trust assumption)은 수이와 상대 블록체인의 신뢰 가정에 각각 반영되며, 상대 블록체인 또한 라이트 클라이언트를 지원한다면 신뢰할 수 있는 오라클(trusted oracles)이나 하드웨어 신뢰(hardware trust)에 의존할 필요가 없다 [7].

브리지는 다른 블록체인에서 발행된 자산을 끌어와, 이를 수이 시스템 내에서 래핑된 자산으로 표현하고 사용할 수 있도록 한다. 이러한 래핑된 자산은 원래의 블록체인에서 잠금 해제(unlock)되어 사용자의 지갑으로 다시 전송될 수 있다. 반대로, 수이에서 발행된 자산도 잠금 조치해 다른 블록체인 상에서 래핑된 자산으로 사용될 수 있다. 이 경우, 다른 시스템에서 해당 래핑 객체가 소멸되면, 수이 내의 원래 객체는 상태(state)나

소유권의 변경을 반영하도록 업데이트되고, 다시 잠금 해제된다.

브리징된 자산의 의미론(semantics)은 래핑 자산의 유용성을 보장하기 위해 매우 중요하다. 대체 가능 자산(fungible assets)은 블록체인 간 브리징 시, 래핑된 상태에서도 분할 및 전송이 가능하도록 더 풍부한 표현을 제공할 수 있다. 대체 불가능 자산(non-fungible assets)은 분할될 수 없으며, 단지 전송만 가능하다. 또한 래핑된 상태에서 제한된 방식으로 상태를 변경할 수 있는 다른 연산을 지원할 수도 있으며, 이는 언래핑 시 사용자 정의 스마트 컨트랙트 로직의 실행을 요구할 수 있다. 수이는 유연한 설계를 가지고 있어, 개발자가 이러한 사용자 경험을 정의할 수 있도록 한다. 브리지는 수이의 고유 기능이 아니라, 무브로 구현된 스마트 컨트랙트일 뿐이며, 무브 언어가 제공하는 조합성(composability)과 안전성 보장을 그대로 활용할 수 있다.

4.7 위원회 재구성(Committee Reconfiguration)

재구성은 한 에포크(epoch) e 의 위원회 C_e 가 다음 에포크 $e' = e + 1$ 의 위원회 $C_{e'}$ 로 교체될 때 발생한다. 재구성의 **안전성**은, 에포크 e 또는 그 이전에 커밋된 트랜잭션 T_x 가 이후 에포크에서 상충되는 트랜잭션에 의해 커밋되지 않도록 보장한다. 또한 **생존성**은, 에포크 e 또는 그 이전에 커밋된 트랜잭션 T_x 가 이후 에포크에서도 커밋되어야 함을 보장한다.

수이는 스마트 컨트랙트 시스템을 활용하여, 재구성 과정에서 필요한 대부분의 작업을 자동화한다. 시스템 수준의 스마트 컨트랙트가 존재하여 사용자가 후보 권한 주체(candidate authority)에게 지분을 위임하거나 잠그는 기능을 제공한다. 에포크 동안 코인 보유자는 토큰을 잠가 위임하거나, 잠금을 해제하여 위임을 철회하거나, 다른 하나 이상의 권한 주체로 위임 대상을 변경할 수 있다.

에포크 종료 시점에서, 해당 에포크의 정족수에 상응하는 지분이 에포크 종료에 동의하면, 권한 주체들은 체크포인트를 커밋하고, 다음 위원회를 결정하고, 에포크를 변경하기 위한 정보를 교환한다. 먼저, 권한 주체들은 합의 프로토콜 [9]의 도움을 받아 체크포인팅 프로토콜을 실행하여 에포크 e 의 종료 시점을 위한 인증된 체크포인트(certified checkpoint)에 합의한다. 이 체크포인트에는 정족수 이상의 권한 주체들이 처리한 모든 트랜잭션과, 그로 인해 생성된 객체들의 집합이 포함된다. 따라서, 어떤 트랜잭션이 정직한 권한 주체 정족수에 의해 처리되었다면, 그 중 하나의 정직한 권한 주체가 해당 트랜잭션을 에포크 종료 체크포인트에 포함시켜, 그 트랜잭션과 그 효과가 에포크 간에도 지속(durable)됨을 보장한다. 또한 인증된 체크포인트는 에포크 e 의 모든 트랜잭션이 정직한 권한 주체들에게 접근 가능함을 보장한다.

에포크 종료 체크포인트에 기록된 지분 위임(stake delegation) 정보는 다음 에포크 $e + 1$ 의 새로운 권한 주체 집합을 결정하는 데 사용된다. 이때 이전 에포크의 권한 주체 정족수와 새로운 권한 주체 정족수 모두가 새로운 위원회 $C_{e'}$ 와 새로운 에포크의 체크포인트에 서명한다. 두 집합의 서명이 모두 확보되면, 새로운 권한 주체들이 새 에포크의 트랜잭션 처리를 시작하고, 이전 권한 주체들은 자신의 에포크 서명 키(epoch signing keys)를 폐기할 수 있다.

복구 (Recovery). 클라이언트의 오류나 이중 서명으로 인해,

특정 소유 객체(owned object)가 하나의 에포크 내에서 “잠금” 상태에 빠져 해당 객체와 관련된 트랜잭션이 인증서로 확정되지 못할 수 있다. 예를 들어, 동일한 객체 버전을 사용하는 두 개의 트랜잭션에 대해 각각 절반의 권한 주체가 서명한 경우, 어느 한쪽도 정족수 이상의 서명을 모아 인증서를 생성할 수 없게 된다.

복구 절차는 에포크 변경 시 이러한 객체들이 다시 트랜잭션에서 사용 가능한 상태로 복원되도록 보장한다. 즉, 인증서를 생성할 수 없는 경우 원본 객체는 다음 에포크 시작 시점에 다시 활성화되어 조작이 가능하다. 또한 트랜잭션에는 에포크 번호가 포함되어 있으므로, 이전 에포크에서 발생한 이중 서명 트랜잭션은 새 에포크에서 객체를 다시 잠그지 않으며, 객체 소유자는 다시 이를 사용할 수 있다.

보상 및 암호경제학 (Rewards & Cryptoeconomics). 수이는 고정 공급량을 가진 네이티브 토큰 SUI를 보유한다. SUI는 가스 비용 지불에 사용되며, 또한 에포크 내에서 권한 주체에게 위임된 지분으로 사용된다.

에포크 내 권한 주체의 투표권(voting power)은 해당 위임 지분의 크기에 비례한다. 에포크가 종료되면, 처리된 모든 트랜잭션에서 수집된 수수료(fees)는 수이의 운영에 기여한 정도에 따라 권한 주체에게 분배되며, 권한 주체들은 그 중 일부를 자신에게 지분을 위임한 주소에게 보상으로 분배한다. 수이의 전체 토큰 경제(token economics)에 대한 세부 사항은 별도의 페이퍼에서 다루어진다.

4.8 권한 주체 및 레플리카 업데이트(Authority & Replica Updating)

클라이언트 주도 (Client-driven). 클라이언트의 오류나 비잔틴이 아닌 권한 주체의 장애로 인해 일부 권한 주체는 모든 인증서를 처리하지 못할 수 있다. 이로 인해 해당 인증서에서 생성된 객체에 의존하는 인과적으로 연결된 트랜잭션은 거부될 수 있다. 그러나 클라이언트는 언제든지 정직한 권한 주체를 업데이트하여 올바른 트랜잭션을 처리할 수 있도록 만들 수 있다. 이를 위해 클라이언트는 자신의 과거 인증서 저장소를 사용하거나, 하나 이상의 다른 정직한 권한 주체를 과거 인증서의 소스로 활용할 수 있다.

인증서 $\$c\$$ 와, $\$c\$$ 및 그 인과적 이력을 포함하는 저장소 $\$Ct_v\$$ 가 주어졌을 때, 클라이언트는 정직한 권한 주체 $\$v'\$$ 를 업데이트하여 $\$c\$$ 가 적용될 수 있는 상태로 만들 수 있다. 이 과정은 $\$v'\$$ 에 존재하지 않는 인증서 중 최소한의 집합을 찾아 적용함으로써, 업데이트된 권한 주체가 $\$c\$$ 의 모든 입력 객체를 포함하도록 하는 것이다. 뒤처진 권한 주체 $\$B\$$ 를, 인증서 $\$TxCert\$$ 를 포함하는 저장소 $\$Ct_v\$$ 를 사용해 업데이트하는 과정은 다음과 같다:

- 클라이언트는 동기화할 인증서 목록을 유지하며, 처음에는 $\$TxCert\$$ 하나만 포함한다.
- 클라이언트는 동기화가 필요한 마지막 인증서를 선택하고, 그 안의 트랜잭션을 추출한 뒤 모든 입력 객체를 파생시킨다.
- 각 입력 객체에 대해, 해당 객체를 마지막으로 생성하거나 변경한 트랜잭션을 $\$Ct_v\$$ 의 동기화 인덱스를 사용해 확인한다. 그 트랜잭션의 인증서가 $\$B\$$ 에 존재하지 않으면, $\$Ct_v\$$ 에서 해당 인증서를 읽어와 동기화 목록에 추가한다.
- 더 이상 추가할 인증서가 없으면(즉, $\$B\$$ 에서 누락된 입력이 더 이상 없을 경우), 인증서 목록을 인과 순서로 정렬한 뒤 $\$B\$$ 에 제출한다.

위 알고리즘은 특정 객체를 새로운 트랜잭션 실행을 위해 특정 버전으로 갱신할 때에도 동일하게 적용된다. 이 경우, 해당 객체 버전을 생성한 트랜잭션의 인증서를 뒤처진 권한 주체에 제출한다. $\$B\$$ 에서 이 인증서가 실행되면, 해당 버전의 객체가 올바르게 복원되어 다시 사용할 수 있게 된다.

이러한 작업을 수행하는 클라이언트를 *릴레이어*라고 한다. 릴레이어는 여러 개가 독립적이며 동시에 작동할 수 있다. 릴레이어는 무결성 측면에서 신뢰가 필요하지 않으며, 키를 필요로 하지 않는 무상태 방식으로 동작한다. 클라이언트뿐만 아니라 권한 주체들도 릴레이어 로직을 실행하여 서로를 갱신할 수 있으며, 서비스를 운영하는 레플리카 역시 뒤처진 권한 주체를 갱신하기 위해 릴레이어로 작동할 수 있다.

대량 업데이트(Bulk). 권한 주체는 인증서를 처리할 때 팔로워가 최신 상태를 자동으로 수신할 수 있도록 하는 기능을 제공한다. 이를 통해 레플리카는 권한 주체의 최신 상태를 항상 동기화된 상태로 유지할 수 있다. 또한 권한 주체들은 단기적으로 **푸시-풀 기반 가십 네트워크**를 사용하여 최근 처리된 트랜잭션 정보를 서로 교환함으로써, 릴레이어가 수행해야 할 동기화 작업의 필요성을 줄일 수 있다. 장기적으로는 뒤처진 권한 주체들이 주기적인 상태 커밋을 수행하여 에포크 경계나 그보다 더 짧은 주기로 특정 체크포인트까지의 전체 인증서 집합을 처리했음을 보장할 수 있다. 이러한 메커니즘은 시스템의 일관성과 권한 주체 간 동기화를 효율적으로 유지하는 핵심 구성 요소이다.

5 확장성과 지연 시간(Scaling and Latency)

수이 시스템은 권한 주체가 더 많은 리소스(CPU, 메모리, 네트워크, 저장소 등)를 단일 머신 또는 여러 머신에 걸쳐 트랜잭션 처리에 투입함으로써 확장성을 달성한다. 리소스가 증가하면 트랜잭션 처리 능력이 향상되어 더 많은 수익을 창출할 수 있으며, 이는 다시 리소스 확충으로 이어진다. 또한 리소스가 충분할수록 제한된 자원을 기다리지 않고 작업을 수행할 수 있으므로, 전반적인 지연 시간도 줄어든다.

처리량(Throughput). 리소스 확충이 거의 선형적으로(capacity quasi-linearly) 처리 능력 증가로 이어지도록 하기 위해, 수이의 설계는 권한 주체 내부의 전역 락(global lock)을 필요로 하는 병목 지점과 동기화 지점을 선택적으로 제거했다. 트랜잭션 처리는 두 단계로 명확히 분리된다: (1) 트랜잭션이 특정 버전의 소유 객체나 공유 객체에 대한 배타적 접근 권한을 확보하는 단계, (2) 트랜잭션을 실행하고 그 결과를 커밋하는 단계이다.

1단계에서는 트랜잭션이 객체 단위(granularity)의 분산 락(distributed lock)을 획득해야 한다. 소유 객체의 경우, 이는 'reliable broadcast primitive'를 통해 수행되며, 권한 주체 내부에서 전역 동기화가 필요하지 않다. 따라서 객체 식별자(Obj ID)를 기준으로 락 관리 기능을 여러 머신으로 샤딩하여 확장할 수 있다. 반면, 공유 객체를 포함하는 트랜잭션은 합의 프로토콜을 통한 시퀀싱이 필요하다. 이는 트랜잭션에 전역 순서를 부여하므로 잠재적인 병목 지점이 될 수 있다. 그러나 최근의 고성능 합의 프로토콜 연구 [9]에 따르면, 상태 머신 복제(state machine replication)에서의 병목은 시퀀싱 자체가 아니라 순차 실행에 있다고 알려져 있다. 수이에서는 시퀀싱은 단순히 입력 공유 객체의 버전을 결정하는 용도로만 사용된다. 즉, 객체 버전 번호를 증가시키고 해당 버전을 트랜잭션 다이제스트와 연결하는 데 그친다.

2단계에서는 모든 입력 객체의 버전이 권한 주체 간에 합의되어 알려진 상태에서, 무브 트랜잭션을 실행하고 그 결과를 커밋한다. 입력 객체 버전이 확정된 후에는 실행이 완전히 병렬적으로 진행될 수 있다. 여러 코어 또는 물리적 머신의 무브 가상머신들이 버전이 지정된 입력 객체를 읽고, 실행 결과를 저장소에 기록한다. 저장소의 일관성 요구 사항(특히 객체 및 트랜잭션 저장소)은 느슨하며, 오더 락 맵을 제외하면 권한 주체 내부에서 확장 가능한 분산 키-값 저장소를 사용할 수 있다. 실행은 멍등성을 가지므로, 실행 중 충돌이나 하드웨어 장애가 발생해도 복구가 용이하다.

그 결과, 서로 인과적으로 독립적인 트랜잭션은 병렬로 실행될 수 있다. 스마트 컨트랙트 설계자는 이러한 병렬성을 최대한 활용할 수 있도록 객체의 데이터 모델과 연산을 설계할 수 있다.

체크포인팅과 상태 커밋(state commitment)은 핵심 트랜잭션 처리 경로 밖에서 수행되어 새로운 트랜잭션 처리를 차단하지 않는다. 이는 커밋된 데이터를 읽는 연산만 포함하며, 트랜잭션이 최종성에 도달하기 전에 추가적인 계산이나 합의를 요구하지 않는다. 따라서 새로운 트랜잭션의 지연 시간이나 처리량에는 영향을 주지 않으며, 이들 작업 또한 가용 리소스에 분산될 수 있다.

읽기 연산은 매우 적극적인 캐싱(caching)을 통해 성능 향상을 얻을 수 있다. 권한 주체들은 라이트 클라이언트가 요청하는 모든 데이터를 서명하여 제공하며, 이 데이터는 분산 저장소를 통해 정적 데이터로 제공될 수 있다.

인증서(certificates)는 그들의 인과적 연쇄(causal history) 전체에 대한 신뢰의 루트(root of trust) 역할을 한다. 상태 커밋은 시스템 전체가 일정 주기(최소한 매 에포크마다 또는 더 자주)마다 모든 상태와 트랜잭션에 대한 전역적 신뢰 루트를 가지도록 보장한다.

지연 시간(Latency). 스마트 컨트랙트 설계자는 트랜잭션에 포함되는 객체 유형(소유 또는 공유)에 따라 실행 전략을 유연하게 구성할 수 있다. 소유 객체를 포함하는 연산은 신뢰성 있는 브로드캐스트를 기반으로 실행 및 커밋되며, 최종성에 도달하기 위해 권한 주체 정족수와 두 번의 왕복이 필요하다. 반면 공유 객체를 포함하는 연산은 consistency broadcast를 통해 인증서를 생성하고, 이후 합의 프로토콜 내에서 처리되어야 하므로 지연 시간이 증가한다. 이는 [9]에서 언급된 바와 같이 권한 주체 정족수까지 4~8회의 왕복이 필요할 수 있다.

참고 문헌(References)

- [1] Mustafa Al-Bassam, Alberto Sonnino, Vitalik Buterin, and Ismail Khoffi. 2021. *Fraud and Data Availability Proofs: Detecting Invalid Blocks in Light Clients*. In *Financial Cryptography and Data Security – 25th International Conference, FC 2021, Virtual Event, March 1–5, 2021, Revised Selected Papers, Part II (Lecture Notes in Computer Science, Vol. 12675)*. Nikita Borisov and Claudia Diaz (Eds.). Springer, 279–298.
- [2] Eshar Ben, Alberto Sonnino, Mustafa Al-Bassam, Sarah Zohar, Patrick McCorry, Sarah Meiklejohn, and George Danezis. 2019. *SoK: Consensus in the Age of Blockchains*. In *Proceedings of the 1st ACM Conference on Advances in Financial Technologies, AFT 2019, Zurich, Switzerland, October 21–23, 2019*. ACM, 183–198.
- [3] Mathieu Baudet, George Danezis, and Alberto Sonnino. 2020. *FastPay: High-Performance Byzantine Fault Tolerant Settlement*. In *AFT '20: 2nd ACM Conference on Advances in Financial Technologies, New York, NY, USA, October 21–23, 2020*. ACM, 165–177.
- [4] Sam Blackshear, Evan Cheng, David D. Lill, Victor Gao, Ben Maurer, Todd Nowacki, Alistair Pott, Sha Qadeer, Ra In, Dario Russo, Stephane Sezer, Tim Zakian, and Runtian Zhou. 2019. *Move: A Language With Programmable Resources*. <https://developers.libra.org/docs/move-paper>

- [5] Sam Blackshear, David L. Dill, Shaz Qadeer, Clark W. Barrett, John C. Mitchell, Oded Padon, and Yoni Zohar. 2020. *Resources: A Safe Language Abstraction for Money*. CoRR abs/2004.05106 (2020). arXiv:2004.05106 <https://arxiv.org/abs/2004.05106>
- [6] Christina Garman, Rachid Guerraoui, and Luís Rodrigues. 2011. *Introduction to Reliable and Secure Distributed Programming*. Springer Science & Business Media.
- [7] Panagiotis Chatzigiannis, Rachid Guerraoui, and Konstantinos Chalkias. 2021. *SoK: Blockchain Light Clients*. IACR Cryptol. ePrint Arch. (2021), 1657.
- [8] Daniel Collins, Rachid Guerraoui, Jovan Komatovic, Petr Kuznetsov, Matteo Monti, Matej Pavlovic, Yvonne-Anne Pignolet, Dragos-Adrian Seredinschi, Alexander Trokhymchuk, and Athanasios Xyggkis. 2020. *Online Payments by Merely Broadcasting Messages*. In *49th Annual IEEE/IFIP International Conference on Dependable Systems and Networks, DSN 2020, Valencia, Spain, June 29 – July 2, 2020*. IEEE, 26–38.
- [9] George Danezis, Alberto Sonnino, and Alexander Spiegelman. 2021. *Narwhal and Tusk: A DAG-based Mempool and Efficient BFT Consensus*. CoRR abs/2105.11827 (2021). <https://arxiv.org/abs/2105.11827>
- [10] David L. Dill, Wolfgang Grieskamp, Julian Park, Shaz Qadeer, Meng Xu, and Jingyi Emma Zhong. 2021. *Fast and Reliable Formal Verification of Smart Contracts with the Move Prover*. CoRR abs/2110.08362 (2021). arXiv:2110.08362 <https://arxiv.org/abs/2110.08362>
- [11] Rachid Guerraoui, Petr Kuznetsov, Matteo Monti, Matej Pavlovic, and Dragos-Adrian Seredinschi. 2018. *Asynchronous Trustworthy Transfers*. CoRR abs/1812.08144 (2018).
- [12] Rachid Guerraoui, Petr Kuznetsov, Matteo Monti, Matej Pavlovic, and Dragos-Adrian Seredinschi. 2019. *The Consensus Number of a Cryptocurrency*. In *Proceedings of the 2019 ACM Symposium on Principles of Distributed Computing, PODC 2019, Toronto, Canada, July 29 – August 2 2019*, Peter Robinson and Faith Ellen (Eds.). ACM, 307–316.
- [13] Patrick McCorry, Chris Buckland, Bennet Yee, and Dawn Song. 2021. *Validating Bridges as a Scaling Solution for Blockchains*. CoRR ePrint Arch. (2021), 1894.
- [14] Mauro Patrignani and Sam Blackshear. 2021. *Robust Safety for Move*. CoRR abs/2110.04580 (2021). arXiv:2110.04580 <https://arxiv.org/abs/2110.04580>
- [15] Jerome H Saltzer and Michael D. Schroeder. 1975. *The protection of information in computer systems*. Proc. IEEE 63, 9 (1975), 1278–1308.

[16] Jingyi Emma Zhong, Kevin Cheng, David L. Dill, Victor Gao, Wolfgang Grieskamp, Sam Blackshear, Punith Kar, Yoni Zohar, Clark W. Barrett, and David L. Dill. 2020. *The Move Prover*. In *Computer Aided Verification – 32nd International Conference, CAV 2020, Los Angeles, CA, USA, July 21–24, 2020, Proceedings, Part I* (Lecture Notes in Computer Science, Vol. 12224). Shuvendu K. Lahiri and Chao Wang (Eds.). Springer, 137–150. https://doi.org/10.1007/978-3-030-53288-8_7