스마트 컨트랙트 zk 전자 투표 시스템을 위한 정형 명세 및 검증

<제목 차례>

<그림 차례>

I. 목적

본 문서는 스마트 컨트랙트 기반의 전자투표 시스템에 대한 정형 명세는 신뢰성과 보안성을 확보하기 위해 작성되었다 이 명세의 주요 목적은 투표 과정에서 발생할 수 있는 부정 투표, 중복 투표, 투표 내용 노출 등의 여러 문제를 사전에 방지하고, 시스템이 의도된 규칙을 항상 준수함을 수학적으로 검증하는 데 목적을 둔다. 특히 본 명세는 유권자 프라이버시 보호와 무결성 보장에 초점을 맞추고, 이를 위해 모든 투표는 암호화되어 처리되며 투표가 종료될 때까지 그 내용은 누구도 알 수 없도록 설계되었다. 이러한 암호화 기반 설계는 매표 행위나 투표 강요를 어렵게 하여 투표의 공정성을 높임과 동시에 스마트 컨트랙트의 논리적 결함이나 보안 약점 등을 정형 기법으로 검출 및 방지함으로써, 배포 전에 잠재적인 오류를 제거하고 안전한 시스템 동작을 보장한다.

본 명세는 SmartContractVoting 모듈의 동작과 제약을 엄밀하게 정의하며, 다음의 시스템 구성요소 및 동작을 포함한다:

온체인 투표 프로토콜: 스마트 컨트랙트 상에서 실행되는 유권자 등록, 투표 제출, 투표 검증 등 투표 절차 전반에 대해 다룬다. 컨트랙트 내부 상태와 상태전이 규칙을 TLA+ (Temporal Logic of Actions)로 기술하여, 투표 과정의 핵심 로직을 모두 모델링한다.

유권자 등록 절차: 투표에 참여할 수 있는 유권자를 사전에 등록하는 기능을 포함한다. 이때 등록된 유권자 집합은 머클 트리 구조로 관리되며, 정형 명세에서 해당 집합과 머클 루트 간의 관계를 표현한다.

투표 제출 및 검증: 유권자가 자신의 투표를 암호화하여 제출하고, 스마트 컨트랙트가 이를 검증하는 과정을 포함합니다. 투표는 영지식 증명을 통해 투표자가 등록된 유권자임을 증명하고 투표 내용의 유효성을 검증하도록 설계된다. 명세에서는 이러한 검증 절차를 추상화된 연산으로 표현하며, 올바른 증명이 제출된 경우에만

투표가 유효하게 반영한다.

검증 속성: 시스템 동작 중 항상 유지되어야 하는 불변 조건들과, 올바른 투표 진행을 위한 Safety 속성 및 Liveness 속성을 포함한다. 이는 한 유권자가 두 번 투표하지 못한다거나, 등록되지 않은 사용자는 투표가 불가능하다는 등의 규칙을 의미한다.

Ⅱ. 용어 정의 및 약어

1. 유권자(Voter): 해당 투표에 참여할 권한이 있는 개인 또는 주체를 말합니다. 보통 사전에 등록된 사용자 계정으로 표현되며, 한 번의 투표 권한을 가짐

2. 머클 트리(Merkle Tree): 다수의 데이터를 효율적으로 해시로 묶어 하나의 루트 해시로 대표하게 하는 트리 구조이다. 본 투표 시스템에서는 등록된 유권자들의 식별자 목록을 머클 트리의 leaf 노드로 삼아 루트 해시를 생성한다. 이렇게 얻은 머클 루트는 해당 유권자 집합에 대한 암호학적 커밋 역할을 하며, 이후 유권자는 Merkle proof를 통해 자신의 식별자가 이 머클 트리에 포함되어 있음을 증명

3. 영지식 증명(ZK Proof, Zero-Knowledge Proof): 자신이 어떤 비밀 정보를 알고 있거나 어떤 조건을 만족함을 해당 비밀을 밝히지 않고도 증명할 수 있는 암호학적 기법으로, 투표 시스템에서는 유권자가 본인이 등록된 유권자이며, 유효한 투표를 행사했다는 것을 블록체인 상에서 증명할 때 활용

4. 투표지(Ballot): 유권자의 실제 투표 행위를 나타내는 데이터 조각으로, 일반 선거에서 투표용지에 기표하듯, 전자투표에서는 후보 선택이나 찬반 등의 투표 선택지를 담은 전자 기록

5. 상태(State): 시스템이 일정 시점에 가지고 있는 정보들의 집합을 말합니다. 본 명세에서 상태는 주요 변수들의 값으로 표현되고, 투표 시스템의 상태는 시간에 따라 변하며, 정형 명세에서는 상태 간 변화 규칙을 행동으로 정의

6. 안전 속성(Safety Property): 시스템 실행 중 어떠한 잘못된 일이 발생하지 않음을 보장하는 성질

7. 활성 속성(Liveness Property): 시스템 실행에서 원하는 일이 결국 발생함을 보장하는 성질

Ⅲ. 스마트컨트랙트에서의 TLA+ 정형 명세 유용성

TLA+ 는 분산 시스템이나 동시성 시스템의 동작을 수학적으로 기술하고 검증하기 위해 개발된 정형 명세 언어이다. TLA+의 기반은 집합론과 논리, 그리고 시스템 상태의 변화를 기술하는 액션에 대한 시간 논리이다. 즉, 시스템이 취할 수 있는 모든 상태들과 상태 간 전이 규칙을 명시함으로써, 설계자가 의도한 대로 시스템이 동작하는지를 철저히 검사할 수 있도록 한다.

스마트 컨트랙트 분야에서 정형 명세와 TLA+는 특히 보안성 강화와 신뢰성 제고 측면에서 크게 유용하다. 스마트 컨트랙트는 한 번 배포되면 변경이 어렵고, 금전적 가치와 직결된 경우가 많으므로 작은 논리 오류도 큰 피해로 이어질 수 있다. 예를 들어 2016년 The DAO 해킹 사건의 경우, 컨트랙트의 재진입 버그로 인해 약 5천만 달러 상당의 이더리움이 탈취되었고, 이후에도 유사한 버그로 다수의 피해 사례가 발생했다.

이러한 치명적인 버그들은 코드 여러 부분에 흩어져 있는 상호작용이나 예외 상황의 조합에서 나타나기 때문에, 사람이 모두 예측하기 어렵고 기존 테스트로도 발견되지 않는 경우가 많다. TLA+는 이러한 문제에 대응하기 위해 개발되었으며, 모델 검사 기법을 통해 모든 가능한 상태 전이 경우를 자동으로 탐색하여 논리적 버그를 찾아낸다. TLA+로 작성된 시스템 모델은 도달 가능한 상태공간을 전부 탐색함으로써, 설계자가 미처 인지하지 못한 시나리오에서도 불변 조건 위배나 모순이 발생하지 않는지를 검증할 수 있다.

블록체인 투표 시스템에서는 동시에 여러 사용자가 행동할 수 있고, 절차의 순서나 조건에 따라 미묘한 문제가 생길 수 있다. 정형 명세를 활용하면 다음과 같은 이점을 얻을 수 있다.

1. 동시성 및 상호작용 검증

스마트 컨트랙트 투표에서는 유권자 등록, 투표 등의 단계가 있으며, 복수의 투표가 거의 동시에 이루어질 수도 있다. 정형 명세는 이러한 동시 이벤트들의 모든 순서를 다룰 수 있어, 경합 조건이나 논리적 충돌을 조기에 발견할 수 있다.

2. 안전성 증명

TLA+로 정의한 불변 조건이 처음부터 끝까지 깨지지 않음을 수학적으로 증명할 수 있다. 이는 곧 스마트 컨트랙트 코드에 해당 불변성이 구현되었을 때 어떤

입력이나 실행 순서가 오더라도 잘못된 상태로 가지 않음을 뜻한다.

3. 명세 기반 개발 및 수정 용이성

정형 명세를 수립함으로써 개발자는 명확한 기준을 얻는다. 이는 구현 이전에 시스템의 동작을 합의하고 이해하는 데 큰 도움이 되며, 코드 구현 중에도 명세와 대조하여 논리 일치를 유지할 수 있다. 또한 명세가 수학적으로 검증되었으므로, 구현이 명세를 준수하는 한도 내에서 코드의 신뢰성도 담보할 수 있다. 만약 요구사항 변경이나 기능 추가가 있을 경우, 기존 명세에서 추가 및 수정 과정을 거쳐 다시금 명세 및 검증을 진행 할 수 있다.

4. 보안성 강화

스마트 컨트랙트 보안에서는 재진입 공격, 정수 오버플로우, 접근 제어 실수 등 다양한 취약점이 문제가 된다. 이러한 취약점들 다수는 시스템 상태의 특정 조합에서만 발생하며, 정형 명세 검증 도구인 TLC Model Checker가 그러한 상황들을 철저히 검사하여 발견해낼 수 있다. 실제로 아마존 AWS에서는 TLA+를 자사 분산시스템에 적용해 사람이 발견하지 못한 미묘한 버그 세 건을 찾아낸 바 있다. 스마트 컨트랙트에 동일한 접근을 취하면, 사람의 분석에 의존할 때 놓칠 수 있는 논리적 허점을 자동으로 찾아 보완할 수 있다.

Ⅳ. TLA+를 활용한 zk 투표 시스템 명세

가. 명세 아키텍쳐

SmartContractVoting 시스템은 명확히 정의된 상태 전이와 행동 규칙에 따라 운영된다. 시스템은 INIT 상태에서 시작하여 REGISTRATION, VOTING, VERIFICATION, TALLYING, 그리고 최종적으로 PUBLISHED 상태에 이르기까지 일련의 엄격한 단계를 순차적으로 거친다. 각 단계는 명세된 사전 조건과 불변 조건을 만족해야 하며, 시스템은 이러한 제약 하에서만 다음 단계로 안전하게 진행할 수 있다.

시스템은 INIT 상태에서 모든 변수를 초기화한 후, StartRegistration을 통해 REGISTRATION 상태로 전이한다. 이때 유권자들은 RegisterVoter를 통해 등록되고, GenerateVoterKey를 통해 개인 키를 생성한다. 키 생성을 완료한 유권자가 존재하는 경우, 시스템은 StartVoting을 통해 VOTING 단계로 진입한다.

VOTING 단계에서는 유권자들이 CastBallot을 통해 투표를 제출하고, 제출된 투표는 EncryptAndVerifyBallot 과정을 통해 암호화된다. 일정 수 이상의 투표가 암호화되면, 시스템은 StartVerification을 통해 VERIFICATION 단계로 진입하며, 각 투표는 VerifyVote를 통해 유효성을 검증받는다. 검증이 완료된 투표들은 UpdateBlockchain을 통해 블록체인 데이터에 기록된다.

모든 검증이 완료되면 StartTallying을 호출하여 시스템은 TALLYING 상태로 진입하고, CountVotes를 통해 VERIFIED된 투표들을 집계하여 TALLIED 상태로 전환한다. 마지막으로 PublishResults를 호출하여 시스템은 PUBLISHED 상태로 진입하고, 전체 투표 결과를 공식 발표하게 된다.

이 일련의 흐름은 각 단계가 정상적으로 완료되어야만 다음 단계로 진행할 수 있도록 되어 있으며, 도중에 조건이 충족되지 않거나 오류가 발생할 경우, 시스템은 다음 상태로의 전이를 허용하지 않는다. 이러한 설계는 시스템의 안정성과 무결성을 보장하고, 투표 프로세스 전반에 걸쳐 안전성과 활성성 속성을 충족시키는 데 필수적이다.

나. 검증 시나리오

SmartContractVoting 시스템의 정형 명세는 다양한 검증 시나리오를 고려하여 설계되었다. 주요 검증 시나리오는 다음과 같다.

첫째, 유권자 등록 시나리오에서는, 등록되지 않은 유권자가 키를 생성하거나 투표를 시도하는 경우를 방지하고, 등록된 유권자만이 키를 생성하고 투표할 수 있음을 검증한다. 이를 통해 OnlyRegisteredVotersWithKeysCanVote, RegisterOnce 등의 Safety 속성이 검증된다.

둘째, 투표 제출 및 암호화 시나리오에서는, 투표 제출 시 유권자가 유효한 키를 보유하고 있으며, 제출된 투표가 적절히 암호화되고, 머클 트리에 올바르게 추가되는지를 검증한다. 이 과정에서 VotePrivacy, MerkleTreeConsistency 속성들이 검증된다.

셋째, 검증 및 집계 시나리오에서는, 모든 ENCRYPTED 상태의 투표가 검증을 통해 VERIFIED 상태로 전환되며, VERIFIED 투표가 집계되어 TALLIED로 전환되는 과정을 검증한다. 이는 EventuallyVerified 속성을 충족하는지를 확인하는 과정이다.

넷째, 최종 결과 발표 시나리오에서는, 시스템이 ENCRYPTED 또는 VERIFIED 상태의 투표를 남기지 않고 PUBLISHED 상태에 도달하는지, 결과 발표 시점에서 모든 투표가 집계되었는지를 검증한다. 이를 통해 AllPublishedVotesVerified 및 EventuallyComplete 속성이 충족되는지를 확인할 수 있다.

모델 검증 과정에서는 유권자가 중복 투표를 시도하는 경우, 미등록 유권자가 투표를 시도하는 경우, 검증되지 않은 투표가 블록체인에 기록되려고 하는 경우 등 다양한 오류 시나리오를 강제적으로 주입하여 시스템이 이들 상황에서도 불변 조건을 위반하지 않는지를 철저히 검증한다.

이러한 다양한 검증 시나리오를 통해 시스템은 모든 합법적 경로뿐 아니라 잠재적 오류 경로에 대해서도 철저히 점검되며, 명세된 모든 안전성과 활성성 속성이 실제로 시스템 실행 경로 전반에 걸쳐 유지되는지를 수학적으로 검증할 수 있다.

다. TLA+ 명세

INIT

초기화 단계는 SmartContractVoting 시스템의 출발점을 정의하는 가장 핵심적인 과정이다. 이 단계에서는 모든 변수와 시스템 상태가 일관성 있고 명확하게 설정되며, 이후 모든 시스템 동작이 올바르게 진행될 수 있도록 기반을 마련한다. 시스템의 전역 상태 변수인 systemState는 INIT으로 설정되어, 시스템이 아직 어떠한 사용자 상호작용도 허용하지 않는 초기 준비 상태임을 명시한다.

유권자 개인키와 공개키를 저장하는 voterKeys 변수는 Voters 집합에 속하는 모든 유권자에 대해 초기화되며, 각각의 key와 publicKey 필드는 빈 문자열로 설정된다. 이는 아직 어떤 유권자도 인증 수단을 갖추지 않았음을 의미한다. voterDB는 공집합으로 초기화되어 등록된 유권자가 없음을 명확히 하며, 등록되지 않은 유권자는 이후 키 생성이나 투표 등의 어떠한 행위도 수행할 수 없게 된다.

투표 데이터 검증을 위한 핵심 구조인 merkleTree는 초기 루트가 빈 문자열로 설정되고, nodes는 빈 집합으로 설정된다. 이는 아직 등록된 데이터가 없으며, 이후 데이터 추가 및 머클 루트 재계산이 필요함을 의미한다. ballots는 비어 있는 시퀀스로 설정되어, 시스템이 어떠한 투표 기록도 가지지 않는 초기 상태임을 명시한다.

블록체인 기록을 담당하는 blockchainData는 세 가지 구성 요소, 즉 transactions, merkleRoots, zkProofs가 모두 빈 집합으로 초기화된다. 이는 아직 온체인상에 기록된 투표 데이터나 검증 정보가 없음을 나타낸다. 마지막으로, voterStatus는 모든 유권자에 대해 NONE으로 설정된다. 이는 유권자가 아직 등록되지 않았고, 키를 생성하지 않았으며, 투표를 하지 않은 초기 상태임을 의미한다.

Init 단계는 시스템 전반의 불변성과 안전성을 보장하기 위한 필수적인 준비 작업이다. 모든 변수를 명세된 초기값으로 설정함으로써, 시스템은 이후의 모든 상태 변화가 명세에 부합하는지를 일관되게 검증할 수 있게 된다. 초기화가 명확히 정의되어 있지 않으면, 시스템은 예측할 수 없는 상태로 진입하거나 잠재적 오류를 초래할 수 있기 때문에, 이 단계는 전체 시스템 신뢰성 확보의 기초가 된다.

$$
\begin{aligned}
Init \triangleq\ & \\
& \wedge systemState = \text{“INIT”} \\
& \wedge voterKeys = [v \in Voters \mapsto [key \mapsto \text{“”}, publicKey \mapsto \text{“”}]] \\
& \wedge voterDB = \{\} \\
& \wedge merkleTree = [root \mapsto \text{“”}, nodes \mapsto \{\}] \\
& \wedge ballots = \langle\rangle \\
& \wedge blockchainData = [transactions \mapsto \{\}, merkleRoots \mapsto \{\}, zkProofs \mapsto \{\}] \\
& \wedge voterStatus = [v \in Voters \mapsto \text{“NONE”}]
\end{aligned}
$$

그림 1 INIT 명세

StartRegistration

StartRegistration 행위는 SmartContractVoting 시스템이 초기화된 이후, 유권자 등록 절차를 시작하는 데 사용된다. 이 행위는 시스템 상태가 INIT인 경우에만 실행 가능하며, 이를 통해 명확한 순차적 상태 전이를 강제한다. StartRegistration이 성공적으로 실행되면 systemState는 REGISTRATION으로 변경된다. 이를 통해 시스템은 이제 유권자 등록을 위한 상호작용을 허용하게 된다.

이 과정에서는 voterKeys, voterDB, merkleTree, ballots, blockchainData, voterStatus 등 다른 주요 변수들은 변경되지 않고 유지된다. 이는 시스템이 등록 단계로만 진입할 뿐, 구체적인 데이터 변경은 등록 행위를 통해 이루어져야 한다는 의미이다. 즉, StartRegistration은 상태 전이만 수행하고, 실제 데이터 삽입이나 갱신은 RegisterVoter 등의 후속 행동에서 이루어진다.

StartRegistration을 통해 시스템은 유권자 등록이라는 첫 번째 사용자 참여 단계로 진입하게 되며, 이로써 시스템이 외부 입력을 수용하기 시작하는 명시적 전환점이 된다. 또한 이 단계는 이후 모든 투표 절차의 전제 조건을 구성하기 때문에, StartRegistration의 정확한 수행은 시스템 전체 흐름의 정상성과 안정성 확보에 필수적이다.

$$
\begin{aligned}
&StartRegistration \triangleq \\
&\quad \land\ systemState = \text{“INIT”} \\
&\quad \land\ SetSystemState(\text{“REGISTRATION”}) \\
&\quad \land\ \text{UNCHANGED}\ \langle voterKeys,\ voterDB,\ merkleTree,\ ballots,\ blockchainData,\ voterStatus \rangle
\end{aligned}
$$

그림 2 StartRegistration 명세

RegisterVoter

RegisterVoter(voter) 행위는 특정 유권자를 시스템에 등록하는 작업을 수행한다. 이 행위는 systemState가 REGISTRATION 상태일 때만 허용되며, 등록하려는 유권자는 전체 Voters 집합에 포함되어야 하고, 아직 voterDB에 추가되지 않은 상태여야 한다. 또한 등록 대상 유권자의 현재 voterStatus는 NONE이어야 한다.

행위가 성공적으로 수행되면, voterDB에 해당 유권자가 추가되고, 해당 유권자의 voterStatus는 REGISTERED로 변경된다. 이 과정을 통해 시스템은 유권자가 등록된 적법한 상태로 전이되었음을 명시한다. 동시에 voterKeys, merkleTree, ballots, blockchainData, systemState 등 다른 변수들은 변경되지 않으며, 이는 등록 행위가 유권자 정보만을 갱신하는 국소적인 변동임을 보장한다.

RegisterVoter는 시스템의 무결성과 접근 통제를 보장하는 중요한 역할을 한다. voterDB에 포함되지 않은 유권자는 이후 어떠한 키 생성, 투표, 검증 행위에도 참여할 수 없기 때문이다. 이를 통해 등록되지 않은 사용자로부터의 부정 행위를 구조적으로 차단할 수 있으며, 투표 프로세스 전체의 신뢰성을 높일 수 있다. 또한 voterStatus를 통해 유권자 개개인의 상태를 명시적으로 관리함으로써, 후속 프로세스에서 상태 일관성을 쉽게 검증할 수 있다.

$$
\begin{aligned}
&RegisterVoter(voter) \triangleq \\
&\quad \land\ systemState = \text{“REGISTRATION”} \\
&\quad \land\ voter \in Voters \\
&\quad \land\ voter \notin voterDB \\
&\quad \land\ voterStatus[voter] = \text{“NONE”} \\
&\quad \land\ voterDB' = voterDB \cup \{voter\} \\
&\quad \land\ voterStatus' = [voterStatus\ \text{EXCEPT}\ ![voter] = \text{“REGISTERED”}] \\
&\quad \land\ \text{UNCHANGED}\ \langle voterKeys,\ merkleTree,\ ballots,\ blockchainData,\ systemState \rangle
\end{aligned}
$$

그림 3 RegisterVoter 명세

GenerateVoterKey

GenerateVoterKey(voter) 행위는 등록된 유권자가 개인키와 공개키를 생성하는 과정을 수행한다. 이 행위는 systemState가 REGISTRATION 또는 VOTING 상태일 때만 허용된다. 대상 유권자는 voterDB에 반드시 등록되어 있어야 하며, 현재 voterStatus가 REGISTERED여야 하고, voterKeys[voter].key가 빈 문자열이어야 한다. 이 조건들은 키 생성이 정확히 한 번만 발생하도록 강제하며, 중복 키 생성이나 무자격 키 생성을 방지한다.

행위가 성공적으로 수행되면, 해당 유권자의 voterKeys 필드가 새로운 개인키와 공개키 쌍으로 갱신된다. 키 생성은 유권자의 ID를 기반으로 단순한 규칙에 따라 이루어지며, 생성된 키는 이후 투표 암호화 및 서명에 사용된다. 또한 voterStatus는 KEY_GENERATED로 갱신되어, 유권자가 투표할 준비가 완료되었음을 시스템이 인식할 수 있게 한다.

GenerateVoterKey는 시스템 보안성의 기초를 형성하는 단계이다. 키가 없는 유권자는 투표를 제출할 수 없으며, 키를 통한 암호화 및 영지식 증명 생성이 가능해야만 투표가 유효하게 처리될 수 있다. 키 생성 절차를 명시적으로 정의하고, 조건부로 강제함으로써, 시스템은 유권자 인증과 무결성 검증을 체계적으로 수행할 수 있다.

$$
\begin{aligned}
&GenerateVoterKey(voter) \triangleq \\
&\quad \land\ systemState \in \{\text{“REGISTRATION”}, \text{“VOTING”}\} \\
&\quad \land\ voter \in voterDB \\
&\quad \land\ voterStatus[voter] = \text{“REGISTERED”} \\
&\quad \land\ voterKeys[voter].key = \text{“”} \\
&\quad \land\ voterKeys' = [voterKeys \ \text{EXCEPT}\ ![voter] = GenerateKey(voter)] \\
&\quad \land\ voterStatus' = [voterStatus \ \text{EXCEPT}\ ![voter] = \text{“KEY\_GENERATED”}] \\
&\quad \land\ \text{UNCHANGED}\ \langle voterDB, merkleTree, ballots, blockchainData, systemState \rangle
\end{aligned}
$$

그림 4 GenerateVoterKey 명세

StartVoting

StartVoting 행위는 SmartContractVoting 시스템이 유권자 등록 절차를 완료하고 본격적인 투표 절차로 진입할 준비가 되었음을 선언하는 전이 행동이다. 이 행위는 systemState가 REGISTRATION 상태여야만 실행 가능하며, 동시에 등록된 유권자 중 적어도 한 명 이상이 개인키를 성공적으로 생성하여 voterStatus가 KEY_GENERATED 상태에 도달해 있어야 한다. 이는 시스템이 충분한 투표 준비 상태를 갖추었음을 검증하는 사전 조건으로, 등록은 완료되었으나 키 생성이 이루어지지 않은 상태에서 투표를 시작하는 오류를 방지하는 역할을 한다.

행위가 성공적으로 수행되면 systemState는 VOTING으로 변경된다. 이 변경은 시스템이 유권자들로부터 실제 투표 입력을 받기 시작할 준비가 되었음을 의미하며, 이후 투표 제출(CastBallot) 등의 행동이 가능해진다. StartVoting이 수행되더라도 voterKeys, voterDB, merkleTree, ballots, blockchainData, voterStatus 등의 다른 주요 변수들은 변경되지 않는다. 이는 이 행위가 오로지 시스템의 운영 모드를 변경하는 것에 국한되며, 데이터 변경을 수반하지 않는다는 점을 명확히 한다.

StartVoting은 전체 시스템 흐름에서 매우 중요한 전이 지점을 구성한다. REGISTRATION 상태는 유권자 등록이라는 사전 준비 작업을 의미하는 반면, VOTING 상태는 실제 투표 행위를 허용하는 활성 운영 단계이다. 따라서 이 전이는 투표 프로세스가 정상적이고 준비된 상태에서만 진행될 수 있도록 보장하며, 동시에 상태 전이 오류나 부정한 투표 시도를 구조적으로 방지하는 역할을 한다.

$$
\begin{aligned}
&StartVoting \triangleq \\
&\quad \land\ systemState = \text{“REGISTRATION”} \\
&\quad \land\ \exists\, v \in voterDB : voterStatus[v] = \text{“KEY\_GENERATED”} \\
&\quad \land\ SetSystemState(\text{“VOTING”}) \\
&\quad \land\ \text{UNCHANGED}\ \langle voterKeys,\ voterDB,\ merkleTree,\ ballots,\ blockchainData,\ voterStatus \rangle
\end{aligned}
$$

그림 5 StartVoting 명세

CastBallot

CastBallot(voter, candidate) 행위는 등록된 유권자가 특정 후보자에게 투표를 제출하는 작업을 수행한다. 이 행위는 systemState가 VOTING 상태여야만 실행 가능하다. 투표를 시도하는 유권자는 반드시 voterDB에 포함되어 있어야 하며, voterStatus가 KEY_GENERATED 상태여야 한다. 또한 해당 유권자의 voterKeys[voter].key는 빈 문자열이 아니어야 한다. 이 조건들은 유권자가 등록 및 키 생성 과정을 적법하게 완료했는지를 확인하여, 미등록자나 키 미생성자의 부정 투표를 원천적으로 차단한다.

추가적으로, 현재 투표지 목록인 ballots 시퀀스의 길이는 MaxBallots를 초과할 수 없다. 이 제한은 시스템 리소스 보호 및 예측 가능한 상태 공간 관리를 위해 필수적이다. 행위가 성공적으로 수행되면, 새로운 투표지가 생성되어 ballots 시퀀스의 끝에 추가된다. 새로 생성된 투표지는 유권자(voter), 선택된 후보자(candidate), 투표 제출 시점의 타임스탬프(시퀀스 길이에 기반), 초기 상태 CAST, 암호화 여부 FALSE, 빈 zkProof로 구성된다. 동시에 투표를 제출한 유권자의 voterStatus는 VOTE_CAST로 갱신된다.

CastBallot은 투표 과정의 핵심 입력 지점을 담당하며, 투표 데이터가 시스템에 기록되기 시작하는 첫 번째 지점이다. 이 과정은 이후의 암호화, 검증, 집계 과정을 거쳐 최종 결과에 반영되므로, 초기 입력 단계에서의 정확성과 무결성 확보가 무엇보다 중요하다. 또한 투표 제출 후 voterStatus를 VOTE_CAST로 변경함으로써, 이후 동일 유권자가 중복 투표를 시도할 경우 이를 감지하고 차단할 수 있는 기반을 마련한다.

$$
\begin{aligned}
&CastBallot(voter,\ candidate) \triangleq \\
&\quad \land\ systemState = \text{“VOTING”} \\
&\quad \land\ voter \in voterDB \\
&\quad \land\ candidate \in Candidates \\
&\quad \land\ voterStatus[voter] = \text{“KEY\_GENERATED”} \\
&\quad \land\ voterKeys[voter].key \neq \text{“”} \\
&\quad \land\ Len(ballots) < MaxBallots \\
&\quad \land\ \text{LET} \\
&\qquad newBallot \triangleq [\\
&\qquad\quad voter \mapsto voter, \\
&\qquad\quad candidate \mapsto candidate, \\
&\qquad\quad timestamp \mapsto Len(ballots) + 1, \\
&\qquad\quad status \mapsto \text{“CAST”}, \\
&\qquad\quad encrypted \mapsto \text{FALSE}, \\
&\qquad\quad zkProof \mapsto \text{“”} \\
&\qquad] \\
&\quad\ \text{IN} \\
&\qquad \land\ ballots' = Append(ballots,\ newBallot) \\
&\qquad \land\ voterStatus' = [voterStatus\ \text{EXCEPT}\ ![voter] = \text{“VOTE\_CAST”}] \\
&\land\ \text{UNCHANGED}\ \langle voterKeys,\ voterDB,\ merkleTree,\ blockchainData,\ systemState \rangle
\end{aligned}
$$

그림 6 CastBallot 명세

EncryptAndVerifyBallot

EncryptAndVerifyBallot(ballotIndex) 행위는 제출된 투표를 암호화하고, 그에 대한 영지식 증명(zkProof)을 생성하는 과정을 수행한다. 이 행위는 systemState가 VOTING 상태여야 하며, 대상이 되는 투표는 ballots 시퀀스 내에서 상태가 CAST인 항목이어야 한다. 즉, 아직 암호화되지 않은 생 투표지만을 대상으로 한다.

행위가 성공적으로 수행되면, 해당 투표의 encrypted 필드는 TRUE로 설정되고, status는 ENCRYPTED로 변경된다. 동시에, 고유한 zkProof 문자열이 생성되어 투표의 zkProof 필드에 저장된다. 생성된 zkProof는 시스템이 투표자가 등록된 유권자이며, 유효한 선택을 했음을 검증할 수 있도록 돕는다. zkProof를 이용하면 투표자가 자신이 올바른 등록자임을 증명할 수 있으면서도, 투표 내용은 비공개로 유지할 수 있다.

암호화 및 zkProof 생성이 완료된 후에는, 새롭게 생성된 머클 노드가 만들어

져 merkleTree 구조에 추가된다. 이 머클 노드는 투표 인덱스와 zkProof를 조합하여 생성된 데이터 블록을 포함하며, 트리의 노드 수를 기반으로 인덱싱된다. 최종적으로, 생성된 zkProof는 blockchainData.zkProofs 집합에 추가되어, 이후 검증과정에서 참조될 수 있도록 한다.

EncryptAndVerifyBallot은 투표 데이터의 기밀성(confidentiality)과 무결성(integrity)을 동시에 확보하는 중간 단계로, 이후 검증(VerifyVote)과 집계(Tallying) 과정의 기반이 된다. 특히 암호화가 완료되지 않은 투표는 이후 단계로 진행할 수 없으므로, 이 행위는 시스템 진행성 확보를 위해 반드시 수행되어야 하며, 실패하거나 누락될 경우 시스템은 진행 정지(stuck) 상태에 빠질 수 있다.

$$
\begin{aligned}
&EncryptAndVerifyBallot(ballotIndex) \triangleq \\
&\quad \land\ systemState = \text{“VOTING”} \\
&\quad \land\ ballotIndex \in \text{DOMAIN } ballots \\
&\quad \land\ ballots[ballotIndex].status = \text{“CAST”} \\
&\quad \land\ \text{LET} \\
&\qquad ballot \triangleq ballots[ballotIndex] \\
&\qquad voter \triangleq ballot.voter \\
&\qquad zkProof \triangleq \text{“zk\_proof\_”} \circ ToString(ballot.voter) \circ \text{“\_”} \circ ToString(ballotIndex) \\
&\qquad encryptedBallot \triangleq [\\
&\qquad\quad ballot \text{ EXCEPT} \\
&\qquad\qquad !.encrypted = \text{TRUE}, \\
&\qquad\qquad !.status = \text{“ENCRYPTED”}, \\
&\qquad\qquad !.zkProof = zkProof \\
&\qquad] \\
&\qquad merkleNode \triangleq CreateMerkleNode(\\
&\qquad\quad \text{“ballot\_”} \circ ToString(ballotIndex) \circ \text{“\_proof\_”} \circ zkProof, \\
&\qquad\quad Cardinality(merkleTree.nodes) + 1 \\
&\qquad) \\
&\quad \text{IN} \\
&\qquad \land\ ballots' = [ballots \text{ EXCEPT } ![ballotIndex] = encryptedBallot] \\
&\qquad \land\ merkleTree' = UpdateMerkleTree(merkleTree, merkleNode) \\
&\qquad \land\ blockchainData' = [blockchainData \text{ EXCEPT } !.zkProofs = @ \cup \{zkProof\}] \\
&\quad \land\ \text{UNCHANGED } \langle voterKeys, voterDB, voterStatus, systemState \rangle
\end{aligned}
$$

그림 7 EncryptAndVerifyBallot 명세

StartVerification

StartVerification 행위는 투표 암호화가 완료된 후, 투표 검증 프로세스로 시스템을 전이시키는 역할을 한다. 이 행위는 systemState가 VOTING 상태여야 하며, ballots 시퀀스 내에 상태가 ENCRYPTED인 투표가 적어도 하나 이상 존재해야만 수행할 수 있다. 이는 시스템이 검증할 투표 데이터를 충분히 갖추었음을 보장하기 위한 전제 조건이다.

행위가 성공적으로 수행되면 systemState는 VERIFICATION으로 변경된다. 이후에는 암호화된 투표에 대해 유효성 검증을 수행할 수 있으며, 투표자는 자신이 정당하게 등록된 유권자임을 증명하는 zkProof를 이용하여 검증받을 수 있다. 이 단계에서는 voterKeys, voterDB, merkleTree, ballots, blockchainData, voterStatus 등의 주요 데이터는 변경되지 않고 유지된다. 단순히 시스템 상태만 변경되는 것이다.

StartVerification은 시스템 전체 흐름에서 중대한 의미를 갖는다. 이 단계에 진입함으로써 시스템은 제출된 투표의 유효성을 본격적으로 평가하기 시작하며, 투표 데이터의 정확성과 무결성을 보장하는 후속 과정으로 자연스럽게 이어지게 된다. 또한, 암호화되지 않은 투표를 검증 대상으로 삼지 않도록 시스템 전환 조건을 엄격히 설정함으로써, 검증 오류나 위조 투표 수용 가능성을 구조적으로 차단한다.

$$
\begin{aligned}
&StartVerification \triangleq \\
&\quad \land\ systemState = \text{“VOTING”} \\
&\quad \land\ \exists i \in \text{DOMAIN } ballots : ballots[i].status = \text{“ENCRYPTED”} \\
&\quad \land\ SetSystemState(\text{“VERIFICATION”}) \\
&\quad \land\ \text{UNCHANGED } \langle voterKeys, voterDB, merkleTree, ballots, blockchainData, voterStatus \rangle
\end{aligned}
$$

그림 8 StartVerification 명세

VerifyVote

VerifyVote(ballotIndex) 행위는 암호화된 투표지에 대해 영지식 증명(zkProof)을 이용한 검증을 수행하는 단계이다. 이 행위는 systemState가 VERIFICATION 상태여야 하며, 검증 대상이 되는 투표는 ballots 시퀀스 내에서 상태가 ENCRYPTED이어야 한다. 즉, 검증은 오직 암호화가 완료된 투표에 대해서만 수행할 수 있으며, 암호화되지 않은 투표는 검증 대상이 될 수 없다.

검증 과정은 두 가지 핵심 조건을 기반으로 한다. 첫째, 해당 투표의 zkProof 필드가 비어있지 않아야 하며, 둘째, 그 zkProof가 blockchainData.zkProofs 집합에 존재해야 한다. 이는 영지식 증명이 이미 시스템에 등록된 유효한 증명이어야 함을 의미하며, 외부에서 위조된 zkProof를 제출하거나, 아직 시스템에 기록되지 않은 zkProof를 사용하는 것을 방지한다.

검증이 성공적으로 완료되면, 해당 투표의 status는 VERIFIED로 변경된다. 동시에, 투표를 제출한 유권자의 voterStatus도 VERIFIED로 갱신된다. 만약 zkProof 검증에 실패하면, 해당 투표의 상태나 유권자의 상태는 변경되지 않고 그대로 유지된다. 이로써 시스템은 검증 실패를 감지하고, 부정확하거나 조작된 투표를 시스템에 반영하지 않는 강력한 방어 메커니즘을 갖추게 된다.

VerifyVote는 시스템 신뢰성 확보를 위한 핵심 과정을 담당한다. 이 단계를 통해 각 투표가 실제로 유효한 등록 유권자로부터 제출되었으며, 데이터가 변조되지 않았음을 체계적으로 확인할 수 있다. 검증 절차를 명확히 정의하고, zkProof에 기반한 철저한 검증 메커니즘을 마련함으로써, 시스템은 신뢰할 수 없는 투표 데이터가 최종 결과에 반영되는 것을 구조적으로 방지할 수 있다.

$$
\begin{aligned}
&VerifyVote(ballotIndex) \triangleq \\
&\quad \land\ systemState = \text{“VERIFICATION”} \\
&\quad \land\ ballotIndex \in \text{DOMAIN } ballots \\
&\quad \land\ ballots[ballotIndex].status = \text{“ENCRYPTED”} \\
&\quad \land\ \text{LET} \\
&\qquad ballot \triangleq ballots[ballotIndex] \\
&\qquad voter \triangleq ballot.voter \\
&\qquad zkProofValid \triangleq \\
&\qquad\quad \land\ ballot.zkProof \neq \text{“”} \\
&\qquad\quad \land\ ballot.zkProof \in blockchainData.zkProofs \\
&\qquad verifiedBallot \triangleq \\
&\qquad\quad \text{IF } zkProofValid \text{ THEN} \\
&\qquad\qquad [ballot \text{ EXCEPT } !.status = \text{“VERIFIED”}] \\
&\qquad\quad \text{ELSE} \\
&\qquad\qquad ballot \\
&\quad \text{IN} \\
&\qquad \land\ ballots' = [ballots \text{ EXCEPT } ![ballotIndex] = verifiedBallot] \\
&\qquad \land\ \text{IF } zkProofValid \text{ THEN} \\
&\qquad\qquad voterStatus' = [voterStatus \text{ EXCEPT } ![voter] = \text{“VERIFIED”}] \\
&\qquad\quad \text{ELSE} \\
&\qquad\qquad \text{UNCHANGED } voterStatus \\
&\land\ \text{UNCHANGED } \langle voterKeys, voterDB, merkleTree, blockchainData, systemState \rangle
\end{aligned}
$$

그림 9 VerfyVote 명세

UpdateBlockchain

UpdateBlockchain 행위는 검증이 완료된 투표들을 블록체인 데이터에 공식적으로 반영하는 작업을 수행한다. 이 행위는 systemState가 VERIFICATION 상태여야 하며, ballots 시퀀스 내에 상태가 VERIFIED인 투표가 적어도 하나 이상 존재해야만 수행할 수 있다. 이는 검증이 완료되지 않은 투표가 블록체인에 기록되는 오류를 방지하기 위한 안전 장치이다.

행위가 성공적으로 수행되면, VERIFIED 상태인 모든 투표에 대해 새로운 트랜잭션이 생성되어 blockchainData.transactions 집합에 추가된다. 각 트랜잭션은 투표자의 ID, 선택한 후보자, 투표 제출 시점의 타임스탬프, 그리고 해당 투표의 zkProof를 포함한다. 이 정보를 기반으로 블록체인 상에 투표 데이터가 불변적으로 기록된다. 또한, 현재 merkleTree.root 값이 blockchainData.merkleRoots에 추가

되어, 머클 트리 기반의 데이터 무결성이 블록체인에 반영된다. blockchainData.zkProofs는 기존 값이 그대로 유지된다.

UpdateBlockchain은 시스템 내 투표 데이터의 불변성과 무결성을 확보하는 데 핵심적인 역할을 한다. 블록체인에 기록된 데이터는 외부에서 변조하거나 삭제할 수 없으므로, 이후 시스템 감사(audit)나 외부 검증 시 신뢰할 수 있는 근거로 활용된다. 또한 트랜잭션과 머클 루트를 분리하여 기록함으로써, 데이터 검증 과정의 효율성과 안정성을 동시에 확보할 수 있다.

$$
\begin{aligned}
&\mathit{UpdateBlockchain} \triangleq \\
&\quad \land\ \mathit{systemState} = \text{“VERIFICATION”} \\
&\quad \land\ \exists\, i \in \text{DOMAIN } \mathit{ballots} : \mathit{ballots}[i].\mathit{status} = \text{“VERIFIED”} \\
&\quad \land\ \text{LET} \\
&\qquad \mathit{newTransactions} \triangleq \\
&\qquad\quad \{[\\
&\qquad\qquad \mathit{voter} \mapsto \mathit{ballots}[i].\mathit{voter}, \\
&\qquad\qquad \mathit{candidate} \mapsto \mathit{ballots}[i].\mathit{candidate}, \\
&\qquad\qquad \mathit{timestamp} \mapsto \mathit{ballots}[i].\mathit{timestamp}, \\
&\qquad\qquad \mathit{zkProof} \mapsto \mathit{ballots}[i].\mathit{zkProof} \\
&\qquad\quad] : i \in \{j \in \text{DOMAIN } \mathit{ballots} : \mathit{ballots}[j].\mathit{status} = \text{“VERIFIED”}\}\} \\
&\qquad \mathit{updatedBlockchainData} \triangleq [\\
&\qquad\quad \mathit{transactions} \mapsto \mathit{blockchainData}.\mathit{transactions} \cup \mathit{newTransactions}, \\
&\qquad\quad \mathit{merkleRoots} \mapsto \mathit{blockchainData}.\mathit{merkleRoots} \cup \{\mathit{merkleTree}.\mathit{root}\}, \\
&\qquad\quad \mathit{zkProofs} \mapsto \mathit{blockchainData}.\mathit{zkProofs} \\
&\qquad] \\
&\quad \text{IN} \\
&\qquad \mathit{blockchainData}' = \mathit{updatedBlockchainData} \\
&\quad \land\ \text{UNCHANGED } \langle \mathit{voterKeys}, \mathit{voterDB}, \mathit{merkleTree}, \mathit{ballots}, \mathit{voterStatus}, \mathit{systemState} \rangle
\end{aligned}
$$

그림 10 UpdateBlockchain 명세

StartTallying

StartTallying 행위는 투표 검증이 완료된 후, 시스템이 최종 집계 단계로 진입하는 과정을 정의한다. 이 행위는 systemState가 VERIFICATION 상태여야 하며, ballots 시퀀스 내에 상태가 여전히 ENCRYPTED인 투표가 없어야만 수행할 수 있다. 이 조건은 모든 암호화 투표가 검증되었거나 무효 처리되어야 함을 의미하며, 미검증 상태의 투표가 집계 과정에 포함되는 오류를 방지한다.

StartTallying이 수행되면 systemState는 TALLYING으로 변경된다. 이후 시스템은 VERIFIED 상태의 투표들을 집계하여 최종 결과를 산출하는 과정으로 진입하게 된다. 다른 주요 변수들은 변경되지 않고 유지된다. 이로써 시스템은 데이터 변동 없이 상태만 전이하여 안정적인 단계적 흐름을 유지할 수 있다.

StartTallying은 투표 절차의 전환점 중 하나로, 유권자의 참여와 투표 데이터의 무결성 검증이 모두 완료되었음을 나타낸다. 이 행위를 통해 시스템은 더 이상 입력을 받지 않고, 제출된 데이터를 기반으로 결과를 산출하는 최종 처리 단계로 자연스럽게 이행할 수 있게 된다. 이 단계로의 정확한 전이는 시스템 신뢰성과 결과의 정당성을 확보하는 데 필수적이다.

$$
\begin{aligned}
&StartTallying \triangleq \\
&\quad \land\ systemState = \text{“VERIFICATION”} \\
&\quad \land\ \forall\, i \in \text{DOMAIN } ballots : \\
&\qquad ballots[i].status \neq \text{“ENCRYPTED”} \\
&\quad \land\ SetSystemState(\text{“TALLYING”}) \\
&\quad \land\ \text{UNCHANGED } \langle voterKeys,\ voterDB,\ merkleTree,\ ballots,\ blockchainData,\ voterStatus \rangle
\end{aligned}
$$

그림 11 StartTallying 명세

CountVotes

CountVotes 행위는 최종 집계 단계에서 VERIFIED 상태의 투표를 TALLIED 상태로 전환하는 작업을 수행한다. 이 행위는 systemState가 TALLYING 상태여야만 실행할 수 있으며, ballots 시퀀스 내 VERIFIED 상태의 모든 투표를 대상으로 한다.

집계 과정은 매우 체계적으로 이루어진다. VERIFIED 상태인 투표는 각각 TALLIED 상태로 변경되며, VERIFIED 상태가 아닌 투표는 기존 상태를 유지한다. 이를 통해 시스템은 검증이 완료된 투표만을 집계 대상으로 삼으며, 검증되지 않은 투표나 오류가 발생한 투표는 결과 산출에서 자연스럽게 제외할 수 있다.

CountVotes는 최종 결과 산출을 위한 전제 조건을 형성한다. VERIFIED 상태를 통해 유효성을 보장받은 투표들만을 집계에 포함시키는 엄격한 규칙을 마련함으로써, 시스템은 결과의 신뢰성과 투명성을 강화할 수 있다. 또한 이 과정을 통해 모든 유효한 투표가 시스템 내에서 명시적으로 집계되었음을 보장할 수 있다. 이로써 투표 결과를 외부 감사나 검증을 통해 언제든지 재현 가능하게 만들 수 있다.

$$
\begin{aligned}
&CountVotes \triangleq \\
&\quad \land\ systemState = \text{“TALLYING”} \\
&\quad \land\ \text{LET } verifiedBallots \triangleq \{i \in \text{DOMAIN } ballots : ballots[i].status = \text{“VERIFIED”}\} \\
&\qquad updateBallot(b, i) \triangleq \text{IF } i \in verifiedBallots \text{ THEN } [b[i] \text{ EXCEPT } !.status = \text{“TALLIED”}] \text{ ELSE } b[i] \\
&\qquad talliedBallots \triangleq [i \in \text{DOMAIN } ballots \mapsto updateBallot(ballots, i)] \\
&\quad\ \text{IN} \\
&\qquad ballots' = talliedBallots \\
&\quad \land\ \text{UNCHANGED } \langle voterKeys, voterDB, merkleTree, blockchainData, voterStatus, systemState \rangle
\end{aligned}
$$

그림 12 CountVotes 명세

PublishResults

PublishResults 행위는 시스템이 최종적으로 투표 결과를 공식 발표하는 단계이다. 이 행위는 systemState가 TALLYING 상태여야 하며, ballots 시퀀스 내 모든 투표는 상태가 TALLIED, CAST, 또는 NONE이어야 한다. 이는 검증되지 않은 상태(예를 들어 ENCRYPTED 상태)가 남아있는 경우 결과를 발표하지 않도록 강제하는 안전 장치이다.

행위가 수행되면 systemState는 PUBLISHED로 변경된다. 이로써 시스템은 더 이상 어떠한 상태 전이나 데이터 변경을 수행하지 않고, 결과를 공식적으로 외부에 노출하는 정지 상태에 들어가게 된다. 다른 주요 변수들은 변경되지 않고 유지되며, 이미 집계된 데이터는 그대로 유지된다.

PublishResults는 전체 투표 프로세스의 공식적인 종료를 나타낸다. 이 단계가 완료되면 시스템은 신뢰성 있는 최종 결과를 산출했다고 간주할 수 있으며, 이후 이 결과는 외부 공개, 규제 제출, 감사 등 다양한 목적에 활용될 수 있다. 이 과정을 명확히 정의하고 엄격한 전이 조건을 부과함으로써, 시스템은 모든 투표가 정상적으로 처리되고 결과가 무결하게 발표되었음을 수학적으로 보장할 수 있다.

$$
\begin{aligned}
&PublishResults \triangleq \\
&\quad \land\ systemState = \text{“TALLYING”} \\
&\quad \land\ \forall\, i \in \text{DOMAIN } ballots : ballots[i].status \in \{\text{“TALLIED”}, \text{“CAST”}, \text{“NONE”}\} \\
&\quad \land\ SetSystemState(\text{“PUBLISHED”}) \\
&\quad \land\ \text{UNCHANGED } \langle voterKeys, voterDB, merkleTree, ballots, blockchainData, voterStatus \rangle
\end{aligned}
$$

그림 13 PublishResults 명세

Safety 속성

OnlyRegisteredVotersWithKeysCanVote

OnlyRegisteredVotersWithKeysCanVote 속성은 투표 시스템 내 모든 투표가 반드시 등록된 유권자에 의해, 그리고 키가 생성된 유권자에 의해서만 제출되었음을 강제한다. 구체적으로, ballots 시퀀스 내 존재하는 모든 투표 항목에 대해, 해당 투표를 제출한 voter는 반드시 voterDB 집합에 포함되어 있어야 하며, 동시에 voterKeys[voter].key가 빈 문자열이 아니어야 한다.

이 속성은 투표의 신뢰성과 무결성을 확보하는 데 있어 가장 핵심적인 역할을 한다. 등록 절차를 거치지 않은 사용자가 투표를 제출하는 경우, 투표 결과의 정당성은 심각하게 훼손될 수 있다. 또한 키 생성이 완료되지 않은 상태에서 투표가 제출되면, 후속 암호화 과정이나 검증 과정에서 시스템이 일관성을 유지할 수 없게 된다. 이러한 오류는 시스템 불안정성으로 이어질 수 있으며, 결과적으로 전체 투표 프로세스의 신뢰도를 저하시킬 수 있다.

따라서 OnlyRegisteredVotersWithKeysCanVote 속성은 시스템이 투표권을 가진 적법한 유권자만을 수용하고, 이들이 사전에 준비된 인증 수단(개인키)을 통해 투표를 제출하도록 강제함으로써, 부정행위 가능성을 구조적으로 제거한다. 이는 전자 투표 시스템의 기본 요건인 투표권 제한과 동일성을 수학적으로 강력히 보장하는 안전 장치로 작용한다.

RegisterOnce

RegisterOnce 속성은 모든 유권자가 시스템 내에서 단 한 번만 등록 절차를 밟을 수 있도록 강제한다. 구체적으로, 각 voter는 voterStatus가 NONE이거나, voterStatus가 NONE이 아니면 반드시 voterDB에 포함되어 있어야 한다. 이 규칙은 유권자가 시스템에 등록될 때 중복 등록을 방지하고, 등록 이력이 없는 유권자가 부당하게 키를 생성하거나 투표를 시도하는 것을 원천 차단한다.

이 속성은 시스템 상태 일관성과 유권자 데이터베이스 무결성 유지에 필수적이다. 만약 동일한 유권자가 여러 번 등록된다면, 중복된 키 생성, 중복 투표, 검증 절차 혼란 등 다양한 문제가 발생할 수 있다. 이는 단순한 데이터 무결성 문제를 넘어, 최종 투표 결과의 신뢰성과 공정성에 중대한 영향을 미칠 수 있다.

RegisterOnce는 시스템이 유권자 관리에 있어서 일관된 참조 기반을 유지하도록 하며, 모든 투표 절차가 명확히 단일 등록 기록에 연결되도록 강제한다. 이를 통해 시스템은 내부적으로 항상 완전하고 일관된 유권자 집합을 유지할 수 있으며, 외부 공격자나 실수로 인한 중복 등록을 효과적으로 방지할 수 있다.

MerkleTreeConsistency

MerkleTreeConsistency 속성은 암호화되거나 검증되거나 집계된 모든 투표가 반드시 merkleTree 구조 내에 포함되어야 함을 규정한다. 구체적으로, 각 ballots[i]가 ENCRYPTED, VERIFIED, 또는 TALLIED 상태라면, 해당 투표에 대한 데이터(투표 인덱스 및 zkProof를 조합한 문자열)가 merkleTree.nodes 중 하나에 반드시 존재해야 한다.

이 속성은 투표 데이터의 무결성과 변조 방지를 위한 필수 조건을 제공한다. Merkle Tree는 데이터 집합의 무결성을 소형화된 형태로 증명할 수 있도록 해주며, 특정 데이터 블록이 집합에 포함되었음을 O(log N) 시간 복잡도로 검증할 수 있는 강력한 구조를 제공한다. 만약 암호화된 투표가 머클 트리에 반영되지 않는다면, 해당 투표가 나중에 부정하게 변경되거나 삭제되어도 시스템은 이를 감지할 수 없게 된다.

따라서 MerkleTreeConsistency는 모든 중요한 투표 데이터가 cryptographic commitment 하에 안전하게 보호되도록 강제하며, 나아가 외부 감사나 분쟁 발생 시에도 투표 무결성을 효과적으로 입증할 수 있도록 시스템을 구조화한다. 이는 투표 데이터에 대한 완전성 증명의 핵심 기반을 형성한다.

OneVotePerVoter

OneVotePerVoter 속성은 모든 유권자가 시스템 전체 수명 주기 동안 최대 한 번만 투표를 제출할 수 있도록 강제한다. 구체적으로, ballots 시퀀스 내에서 동일한 voter를 가진 투표 항목의 개수는 최대 1을 초과할 수 없다.

이 속성은 투표 공정성(fairness) 확보에 있어 가장 기본적이면서도 중요한 규칙이다. 한 유권자가 여러 번 투표할 수 있게 되면, 특정 유권자가 시스템 결과에 과도한 영향을 미치거나 투표 결과를 조작하는 것이 가능해질 수 있다. 이는 전통

적 선거 시스템뿐 아니라 블록체인 기반 투표 시스템에서도 심각한 신뢰 훼손을 초래한다.

OneVotePerVoter는 시스템이 투표권을 철저히 1인 1표로 제한하도록 보장한다. 등록과 키 생성 이후, 투표 제출 및 검증에 이르는 전 과정에서 이 불변조건을 유지함으로써, 시스템은 항상 투표 권리 행사에 대한 형평성을 보장하고 결과의 정당성을 확보할 수 있다. 이는 공정성 원칙의 수학적 강제 표현이라 할 수 있다.

AllPublishedVotesVerified

AllPublishedVotesVerified 속성은 시스템이 최종적으로 PUBLISHED 상태에 도달했을 때, ballots 시퀀스 내 존재하는 모든 투표는 TALLIED, CAST, 또는 NONE 상태 중 하나에 반드시 포함되어야 함을 규정한다. 이는 시스템이 미완료 상태의 투표(예: ENCRYPTED, VERIFIED 등)를 남긴 채 결과를 발표하는 일을 방지한다.

이 속성은 최종 결과의 완전성과 정확성을 보장하는 데 필수적이다. 미검증 상태의 투표가 결과에 영향을 미치거나, 미집계 투표가 존재할 경우, 결과의 신뢰성은 심각하게 저하된다. 특히 블록체인 기반 시스템에서는 결과가 한번 발표되면 변경이 어렵기 때문에, 발표 이전에 모든 투표 데이터가 적절히 정리되고 검증되어야 한다.

AllPublishedVotesVerified는 투표 진행 과정이 완결성(completeness)과 일관성(consistency)을 만족하도록 강제하며, 결과 발표 시점에 시스템이 전적으로 안정된 상태에 있음을 보장한다. 이 속성은 최종 상태 도달의 품질과 신뢰성을 결정짓는 핵심 기준이 된다.

VotePrivacy

VotePrivacy 속성은 시스템이 VOTING 이후 모든 투표에 대해 기밀성을 철저히 보장하도록 강제한다. 구체적으로, ballots 시퀀스 내 존재하는 모든 ENCRYPTED, VERIFIED, 또는 TALLIED 상태의 투표는 반드시 encrypted 플래그가 TRUE여야 하며, zkProof 필드가 빈 문자열이 아니어야 한다.

이 속성은 투표 프라이버시 보호를 위한 핵심 기제이다. 암호화되지 않은 투

표는 외부 관찰자가 투표 내용을 쉽게 파악할 수 있는 위험을 내포하고 있으며, 이는 유권자 프라이버시 침해뿐 아니라 투표 강요, 매표 등의 부정 행위로 이어질 수 있다.

VotePrivacy는 시스템이 모든 유효한 투표를 암호화 상태로 관리하도록 강제하며, 동시에 zkProof를 통한 유효성 증명을 통해 투표 무결성도 확보하도록 한다. 이로써 유권자는 자신의 투표 내용이 안전하게 보호되고 있으며, 자신이 유효한 권한을 행사했음을 외부에 증명할 수 있다. 이는 민주적 절차로서의 투표의 본질적 가치를 기술적으로 보장하는 강력한 수단이다.

$$
\begin{aligned}
&\mathit{OnlyRegisteredVotersWithKeysCanVote} \triangleq \\
&\quad \forall\, i \in \text{DOMAIN } \mathit{ballots} : \\
&\qquad \land\, \mathit{ballots}[i].\mathit{voter} \in \mathit{voterDB} \\
&\qquad \land\, \mathit{voterKeys}[\mathit{ballots}[i].\mathit{voter}].\mathit{key} \neq \text{“”} \\
&\mathit{RegisterOnce} \triangleq \\
&\quad \forall\, v \in \mathit{Voters} : ((\mathit{voterStatus}[v] = \text{“NONE”}) \\
&\qquad\qquad \lor (\mathit{voterStatus}[v] \neq \text{“NONE”} \Rightarrow (v \in \mathit{voterDB}))) \\
&\mathit{MerkleTreeConsistency} \triangleq \\
&\quad \forall\, i \in \text{DOMAIN } \mathit{ballots} : \\
&\qquad (\mathit{ballots}[i].\mathit{status} \in \{\text{“ENCRYPTED”}, \text{“VERIFIED”}, \text{“TALLIED”}\}) \Rightarrow \\
&\qquad\quad \exists\, n \in \mathit{merkleTree.nodes} : \\
&\qquad\qquad n.\mathit{data} = (\text{“ballot\_”} \circ \mathit{ToString}(i) \circ \text{“\_proof\_”} \circ \mathit{ballots}[i].\mathit{zkProof}) \\
&\mathit{OneVotePerVoter} \triangleq \\
&\quad \forall\, v \in \mathit{Voters} : (\mathit{Cardinality}(\{i \in \text{DOMAIN } \mathit{ballots} : \mathit{ballots}[i].\mathit{voter} = v\}) \leq 1) \\
&\mathit{AllPublishedVotesVerified} \triangleq \\
&\quad \mathit{systemState} = \text{“PUBLISHED”} \Rightarrow \\
&\qquad \forall\, i \in \text{DOMAIN } \mathit{ballots} : \mathit{ballots}[i].\mathit{status} \in \{\text{“TALLIED”}, \text{“CAST”}, \text{“NONE”}\} \\
&\mathit{VotePrivacy} \triangleq \\
&\quad \mathit{systemState} \neq \text{“INIT”} \Rightarrow \\
&\qquad \forall\, i \in \text{DOMAIN } \mathit{ballots} : \\
&\qquad\quad (\mathit{ballots}[i].\mathit{status} \in \{\text{“ENCRYPTED”}, \text{“VERIFIED”}, \text{“TALLIED”}\} \Rightarrow \\
&\qquad\qquad \land\, \mathit{ballots}[i].\mathit{encrypted} = \text{TRUE} \\
&\qquad\qquad \land\, \mathit{ballots}[i].\mathit{zkProof} \neq \text{“”})
\end{aligned}
$$

그림 14 Safety 속성 명세

Liveness 속성

EventuallyGetKeys

EventuallyGetKeys 속성은 시스템이 등록된 모든 유권자에 대해 결국 개인 키 생성 과정을 완료할 수 있음을 보장한다. 구체적으로, voterDB에 등록된 모든 유권자에 대해 언젠가는 voterKeys[v].key가 빈 문자열이 아닌 값으로 설정될 것임을 요구한다. 즉, 등록된 유권자는 키 생성을 무한정 미루지 않고, 언젠가 키를 성공적으로 발급받아 투표 준비를 완료하게 된다.

이 속성은 시스템 활성성 확보의 근간을 형성한다. 등록은 완료했지만 키를 생성하지 않은 유권자가 계속 존재할 경우, 해당 유권자는 투표할 수 없는 상태에 머무르게 되며, 전체 투표 프로세스의 진행을 저해할 수 있다. 특히 일정 수의 유권자가 필수적으로 투표에 참여해야 하는 시스템에서는, 키 생성 지연이 전체 프로세스의 교착상태(deadlock)를 초래할 수 있다.

EventuallyGetKeys는 시스템이 장기 실행 경로에서 모든 적법한 유권자에게 투표 참여 기회를 보장하고, 프로세스의 정상적 진척을 막는 요인을 구조적으로 제거함을 의미한다. 이를 통해 시스템은 키 생성을 완료하지 못한 유권자로 인해 무한 대기 상태에 빠지는 위험을 방지하며, 전 유권자가 실질적으로 투표 프로세스에 참여할 수 있는 가능성을 열어놓게 된다.

EventuallyVerified

EventuallyVerified 속성은 시스템이 진행 중인 모든 암호화된 투표에 대해 검증을 완료하고, 이후 검증된 투표들을 최종 집계할 수 있음을 보장한다. 이 속성은 두 부분으로 구성된다.

첫 번째 부분은 ballots 시퀀스 내 모든 ENCRYPTED 상태의 투표에 대해, 언젠가는 그 상태가 VERIFIED, CAST, 또는 NONE 중 하나로 변경될 것임을 요구한다. 즉, 암호화만 완료되고 검증이 이루어지지 않은 투표가 무기한 남아 있는 상황을 방지한다.

두 번째 부분은 VERIFIED 상태의 투표에 대해, 언젠가는 그 상태가 TALLIED로 변경될 것임을 요구한다. 이는 검증이 완료된 투표가 언젠가는 집계되어 최종

결과에 반영되리라는 것을 보장한다.

EventuallyVerified 속성은 시스템이 유권자의 입력을 받은 이후 이를 무한히 지연시키지 않고, 일정 시점에 투표 프로세스를 완료하는 진행성을 확보함을 의미한다. 특히 분산 환경이나 블록체인 기반 시스템에서는, 개별 투표가 인증되지 않은 채 무기한 정체되거나, 검증된 투표가 집계되지 않는 문제가 발생할 수 있다. 이는 결과 산출 지연 뿐 아니라, 투표 결과 조작 가능성, 시스템 신뢰성 훼손 등을 초래할 수 있다.

따라서 EventuallyVerified는 투표 제출 이후 검증, 집계 과정이 체계적으로 진행되며, 모든 유효 투표가 시스템 최종 결과에 반영될 수 있도록 보장하는 핵심 활성성 속성이다. 이 속성은 시스템 진행성 확보와 공정성 유지를 동시에 뒷받침한다.

EventuallyComplete

EventuallyComplete 속성은 시스템이 모든 절차를 완료하고, 최종적으로 PUBLISHED 상태에 도달할 것임을 요구한다. 이 속성은 시스템이 무한정 등록, 투표, 검증 단계 중 하나에 머무르지 않고, 일정 시점에 투표 결과를 공식적으로 발표하는 단계에 도달함을 보장한다.

EventuallyComplete는 투표 프로세스의 완결성(completeness)과 종료 가능성(termination)을 보장하는 가장 상위 수준의 활성성 조건이다. 투표 시스템이 설계 의도대로 작동하려면, 각 유권자가 등록하고, 키를 생성하고, 투표를 제출하고, 검증하고, 집계하는 모든 단계를 거쳐야 하며, 이 모든 단계를 완료한 후에는 반드시 결과를 공식 발표해야 한다.

만약 EventuallyComplete 속성이 성립하지 않는다면, 시스템은 특정 단계(예를 들어 검증)에서 무한 루프에 빠지거나, 투표 결과를 발표하지 못한 채 무기한 대기하는 상황에 빠질 수 있다. 이는 투표 결과에 대한 신뢰성을 심각하게 저하시킬 뿐만 아니라, 실질적인 시스템 가용성을 크게 약화시킨다.

EventuallyComplete는 시스템이 설계된 기능을 완성도 있게 수행하고, 사용자와 외부 세계에 대해 명확하고 최종적인 결과를 제공할 수 있도록 하는 필수적인 활성성 속성이다. 이 속성이 충족될 경우, 투표 시스템은 신뢰성과 기능성을 모두

갖춘 완전한 분산 애플리케이션으로 작동할 수 있다.

$$
\begin{array}{l}
\mathit{EventuallyGetKeys} \triangleq \\
\quad \forall\, v \in \mathit{Voters} : (v \in \mathit{voterDB} \Rightarrow \Diamond(\mathit{voterKeys}[v].\mathit{key} \neq \text{“”})) \\
\\
\mathit{EventuallyVerified} \triangleq \\
\quad \Diamond \\
\qquad (\forall\, i \in \text{DOMAIN } \mathit{ballots} : \\
\qquad\quad (\mathit{ballots}[i].\mathit{status} = \text{“ENCRYPTED”}) \\
\qquad\quad \Rightarrow (\mathit{ballots}[i].\mathit{status} \in \{\text{“VERIFIED”}, \text{“CAST”}, \text{“NONE”}\}) \\
\qquad) \\
\quad \wedge \\
\quad \Diamond \\
\qquad (\forall\, i \in \text{DOMAIN } \mathit{ballots} : \\
\qquad\quad (\mathit{ballots}[i].\mathit{status} = \text{“VERIFIED”}) \Rightarrow \\
\qquad\quad (\mathit{ballots}[i].\mathit{status} = \text{“TALLIED”}) \\
\qquad) \\
\\
\mathit{EventuallyComplete} \triangleq \\
\quad \Diamond(\mathit{systemState} = \text{“PUBLISHED”})
\end{array}
$$

그림 15 Liveness 속성 명세

V. 결론 및 기대효과

본 시스템의 정형 명세는, 스마트 컨트랙트 기반 전자 투표 시스템에 요구되는 높은 수준의 신뢰성과 보안성을 수학적으로 보장하는 데 중점을 두었다. 명세는 투표 프로세스의 각 단계를 엄격히 정의하고, 모든 상태 전이에 대해 불변 조건과 진행 조건을 명시함으로써, 시스템이 의도된 동작을 항상 유지할 수 있도록 설계되었다.

Safety 속성들은 시스템이 어떤 실행 경로를 따라가더라도 잘못된 일이 발생하지 않음을 보장한다. 예를 들어, 등록되지 않은 사용자가 투표하거나, 유권자가 중복 투표하거나, 암호화되지 않은 투표가 시스템에 저장되는 일은 결코 발생하지 않는다. 이는 시스템의 기본 무결성과 공정성을 수학적으로 입증하는 데 결정적인 역할을 한다.

Liveness 속성들은 시스템이 무한정 대기하거나 교착 상태에 빠지지 않고, 모든 유권자가 키를 생성하고, 모든 투표가 검증되고, 최종 결과가 발표되는 프로세스가 반드시 완료됨을 보장한다. 이를 통해 시스템은 실질적인 기능성을 확보하고, 사용자에게 신뢰할 수 있는 결과를 제공할 수 있다.

정형 명세를 기반으로 한 시스템 검증은 단순한 테스트 기반 검증이 갖는 한계를 극복할 수 있으며, 모든 가능한 상태 전이와 경로를 포괄적으로 탐색함으로써 논리적 오류를 사전에 제거할 수 있다. 이는 블록체인 기반 시스템, 특히 변경이 불가능하고 금전적 가치와 직결되는 스마트 컨트랙트 시스템에 있어서 치명적인 오류를 예방하는 가장 강력한 방법론이다.

SmartContractVoting 명세는 이러한 정형 검증을 통해 안전성과 활성성을 모두 충족함을 수학적으로 입증하며, 실제 스마트 컨트랙트 구현 시에도 이러한 속성이 유지될 수 있도록 강력한 설계 가이드라인을 제공한다. 이를 통해 최종 시스템은 높은 신뢰성, 투명성, 공정성을 갖춘 전자 투표 플랫폼으로 자리매김할 수 있을 것으로 기대된다.

참고문헌

[1] Leslie Lamport, TLA+ Home Page - "TLA+ is a formal specification language for concurrent systems

[2] Ognjen Marić, Eliminating Smart Contract Bugs with TLA+

[3] Palina Tolmach et al., A Survey of Smart Contract Formal Specification and Verification

[4] Chinthaka Weerakkody, Privote: ZK Proof-Based Private Voting Dapp

[5] Seongho Yoon & Jin-Young Choi, Formal Specification and Verification of Smart Contract-Based Loan Management System Using TLA+

[6] Becker, Georg. "Merkle signature schemes, merkle trees and their cryptanalysis."

