ChatGPT

멀티체인 어댑터 설계 배경지식

1. 멀티체인 추상화는 차이를 숨기는 기술이 아니다

주요 메세지는 “멀티체인 추상화는 차이를 숨기는 것이 아니라, **차이를 안전하게 노출하는 계약(Contract)**”이다. 서로 다른 체인은 nonce/수수료/확정성(finality) 등 근본적인 실행 모델이 다르기 때문에 하나의 정적인 인터페이스로 감싸면 사고가 발생한다. 따라서 개발자는 공통 인터페이스를 제공하되, 체인별 리스크와 정책의 차이를 **정책 파라미터로 노출**해야 한다.

본 세션에서는 백엔드 개발자가 수탁형 지갑(MPC 지갑 등)의 멀티체인 어댑터를 설계할 때 필요한 배경지식을 정리한다. 각 체인의 실행 모델, 수수료 구조, 컨펌/확정성 차이, RPC 불확실성, 멀티체인 어댑터 설계 원칙을 이해하여 언제든지 질문에 답변할 수 있는 수준까지 준비한다.

2. 체인별 실행 모델 차이

2.1 Nonce·Sequence·UTXO 모델

체인/모델	특징	중요 키워드
EVM 계정 기반 (Account Model)	모든 EVM 체인(Ethereum, Polygon 등)은 계정(address)과 **nonce**를 이용해 거래 순서와 충돌을 결정한다. `nonce`는 계정에서 발행한 트랜잭션 개수로, 거래를 병렬로 보낼 경우 nonce 충돌 문제가 발생하며 이를 관리하기 위한 전략이 필요하다. Stack Exchange에서는 계정의 nonce가 replay attack을 막고 트랜잭션을 순차화하지만, 다수의 트랜잭션을 병렬로 보내는 경우 애플리케이션이 자체적으로 nonce 관리를 해야 한다고 설명한다 [1].	account_nonce, sequentialisation, RBF
UTXO 기반 (Bitcoin 등)	Bitcoin 등 UTXO 체인은 “Unspent Transaction Output”를 이용해 잔액을 관리한다. 각 트랜잭션은 이전 UTXO를 입력으로 가져오고 새로운 UTXO를 만들어 잔액을 갱신한다. 별도의 nonce 필드가 없으며, 어떤 UTXO를 소비하는지와 수수료/입력 조합이 트랜잭션의 충돌 여부를 결정한다 [2]. 따라서 병렬 처리 가능성이 있지만 입력 UTXO를 잘 선택해야 한다.	utxo_selection, input_combination
Sequence/ Account Number 모델 (Cosmos 등)	Cosmos SDK를 사용하는 체인은 **account number**와 **sequence** 값을 가진다. account number는 계정을 고유하게 식별하며 최초 수신 시 부여된 후 변하지 않는다 [3]. sequence는 계정에서 발송한 트랜잭션 수이며 트랜잭션마다 포함하고 1씩 증가해야 한다 [4]. 이를 통해 트랜잭션 순서를 강제하고 재전송 공격을 방지한다.	account_number, sequence, replay-protection

체인/모델	특징	중요 키워드
Durable Nonce(솔라나)	Solana는 최근 블록해시를 nonce로 사용하지만, 장기 보관 혹은 오프라인 서명을 위해 **Durable Nonce**를 제공한다. Durable nonce를 사용하면 트랜잭션을 미리 서명해 두었다가 나중에 제출할 수 있어 스케줄링·멀티시그 등에 유용하다. Nonce 계정에 32바이트 값을 저장하고 `nonceAdvance` 명령으로 값을 갱신하여 트랜잭션을 고유하게 만든다 [5] [6] .	durable_nonce, offline_signing

정리: EVM 계정 모델은 `nonce` 충돌에 따른 직렬화가 필수이며, UTXO 모델은 입력 UTXO 선택이 충돌을 결정한다. Cosmos-기반 체인은 account number/sequence를 통해 트랜잭션 순서를 관리하고, Solana처럼 별도의 durable nonce 기능을 가진 체인도 있다. 어댑터는 이러한 모델 차이를 **정책 파라미터**로 노출해야 한다.

2.2 수수료(Fee) 모델

2.2.1 EIP-1559 (Dynamic Fee)

이더리움의 EIP-1559 업그레이드는 가스 수수료 시장을 **Base Fee + Priority Fee + Max Fee** 구조로 변경했다. Blocknative의 설명에 따르면:

- `Base Fee`는 네트워크 혼잡도를 기반으로 블록마다 자동 조정된다 [7] . 이전 블록이 50% 차 있으면 그대로 유지하고, 100% 찼다면 다음 블록의 base fee가 12.5% 증가한다 [8] . 이 값은 소각되어 사용자에게 반환되지 않는다.
- `Max Priority Fee`는 채굴자/검증자에게 지급되는 **팁**이다. 기본적으로 2 gwei 정도의 팁을 권장하지만, 네트워크 혼잡 시 더 높은 팁을 설정해 우선 순위를 높일 수 있다 [9] .
- `Max Fee per Gas`는 사용자가 지불할 **최대 가스비**로, base fee 변동으로 트랜잭션이 묶이지 않도록 (2×base fee + tip) 같은 방식으로 설정한다 [10] .

이 구조는 거래가 즉시 실행되기보다 **견적 기반**이며, 멀티체인 어댑터는 `estimateFee()` 메서드에서 사실을 반환하고 정책 엔진에서 `feePolicyId`에 따라 maxFee/tip 증액 규칙을 결정해야 한다.

2.2.2 고정 수수료 및 단순 모델

일부 체인은 고정 가스 가격을 사용하거나 `단가 × 가스`와 같이 단순한 수수료 체계를 사용한다. UTXO 기반의 비트코인은 바이트 크기와 네트워크 상황을 기반으로 한 수수료 추정이 필요하고, Cosmos 체계의 L1/L2도 각자의 수수료 정책을 가진다. MEV나 네트워크 혼잡 등을 고려해야 하므로 수수료는 **계산이 아닌 정책**으로 관리해야 한다.

2.3 완료 선언(Confirmations & Finality)

2.3.1 이더리움 PoS의 'Latest/Safe/Finalized'

이더리움의 Beacon Chain은 블록을 **latest(현재 헤드)**, **safe(정당화된 블록)**, **finalized(최종화된 블록)**로 구분한다. Alchemy의 설명에 따르면 최신 블록은 가장 최근에 생성된 블록이지만 재조직(re-org)의 위험이 있다 [11] . 안전한(safe) 블록은 **2/3 이상의 검증자**가 attest하여 "정당화된" 블록으로, 리오그 가능성이 매우 낮다 [12] . finalized 블록은 safe 블록보다 **한 에폭(32 슬롯)** 이전의 블록으로 사실상 변경 불가능하다 [13] . 개발자는 latest 블록만 신뢰할 수 없으며, 정책에 따라 safe 혹은 finalized까지 기다려야 한다.

2.3.2 확률적 최종성(Probabilistic Finality)

Trail of Bits의 'Engineer's Guide to Blockchain Finality'는 비트코인과 같은 PoW 체인이 "실제로는 **완전히 확정되지 않는**" 확률적(finality) 모델을 가진다고 지적한다. 블록이 더 많은 블록 위에 쌓일수록 re-org될 가능성이 감소하

며, 충분한 시간이 지나면 재조직 가능성이 거의 0에 수렴한다 [14]. 각 체인의 합의 알고리즘과 포크 선택 규칙이 다르므로, 거래를 "완료"로 선언할 시점은 체인별로 정책적으로 결정해야 한다.

2.3.3 즉각적 또는 변형된 확정성

일부 L1/L2 체인(예: Cosmos의 Tendermint, Sei, Celestia 등)은 블록이 생성되면 즉시 최종성이 보장된다. Solana도 optimistic finality와 "durable nonce" 등 별도의 개념을 제공한다. 반면 Proof-of-Authority(POA)나 PBFT 기반 체인은 블록이 빠르게 확정되지만 탈중앙화와 속도 사이의 trade-off가 있다.

3. RPC의 불완전성과 멀티 RPC 전략

블록체인 노드와 통신하는 **RPC(Remote Procedure Call) 엔드포인트는 '진실'이 아니라 관측'**이라는 점을 인식해야 한다. 노드 A와 B가 서로 다른 블록 헤드를 보고 있거나, 트랜잭션/로그 정보가 누락·지연되는 경우가 흔하다.

CompareNodes는 하나의 RPC 노드 제공자만 사용하는 경우 **중앙집중화 위험**이 있으며, 여러 공급자를 사용하면 **중요한 중복성 및 신뢰성**을 얻을 수 있다고 강조한다. 여러 공급자를 이용해 부하를 분산하면 지연(bottleneck)과 장애를 최소화할 수 있고, 지리적으로 다른 위치의 노드를 사용하면 지역적 장애에 대응할 수 있다 [15]. 단일 공급자에 의존하면 해당 공급자가 중단될 경우 전체 서비스가 마비될 수 있으므로 **멀티 RPC + 쿼럼 판정**을 기본값으로 설계해야 한다. 또한 RPC 실패율, head mismatch rate 등을 모니터링하여 degrade 모드를 도입하면 혼잡 시 수동介入으로 전환할 수 있다.

4. 멀티체인 어댑터 설계 원칙

세션에서 제시한 7가지 원칙을 보다 깊이 있게 해석한다.

1. **관측 가능한 실행 모델만 제공** – 어댑터는 체인의 모든 기능을 추상화하려고 하지 않고, 지갑 백엔드에 필요한 핵심 기능(트랜잭션 준비, 브로드캐스트, 상태 추적, 실패 분류, 헤드 정보)만 제공한다. 정책 결정은 상위 모듈이 담당한다.

2. **최소 공통 인터페이스 + 명시적 확장 포인트** – 공통 인터페이스가 너무 크면 새로운 체인을 추가하기 어렵고, 너무 작으면 예외 처리가 상위로 새어 나온다. 따라서 core API 4개(`prepareSend`, `broadcast`, `getTxStatus`, `getHeads`)를 고정하고, 체인별 특수 기능은 확장(extension)으로 분리한다.

3. **어댑터는 정책을 결정하지 않는다** – 어댑터는 네트워크 상태·estimateFee·헤드 정보를 **사실로만 보고**하고, "최종성을 언제 선언할지", "수수료 상한을 어떻게 설정할지" 같은 정책은 오케스트레이터/Policy Engine이 처리한다. 이것은 이더리움의 safe/finalized 개념처럼 정책 결정이 체인마다 다르기 때문이다 [16].

4. **실패는 에러 문자열이 아닌 표준 코드로 분류** – 재시도 가능(retryable), 수동介入(manual), 교체됨(replaced), 드롭됨(dropped) 등으로 분류하여 상위 시스템이 자동으로 대응할 수 있도록 한다. RPC 노드마다 에러 메시지가 다르므로 코드 기반 분류가 필수이다.

5. **체인별 시간·확률을 정책 파라미터로 노출** – "10컨펌이면 끝" 같은 하드코딩을 피하고, `FinalityPolicy(chain=ETH, tier=HIGH)`처럼 체인과 리스크 수준을 정책 테이블로 관리한다. 이는 비트코인처럼 확률적 finality인 체인과 이더리움 PoS처럼 안전 블록/최종 블록이 존재하는 체인의 차이를 반영한다 [14].

6. **멱등성 키 관리** – HTTP 레이어의 idempotency key와 체인 레벨의 nonce/sequence는 서로 다른 개념이다. 오케스트레이터가 전역적으로 idempotency key를 관리하되, 어댑터도 내부적으로 nonce reservation ID

등 멱등성 정보를 받아야 한다. 병렬 Nonce 예약, RBF(replace-by-fee) 등 고급 전략을 도입하려면 별도의 NonceManager를 둬야 한다.

7. **인터페이스 버전 관리** – 멀티체인 어댑터 API는 변화를 피할 수 없으므로 v1, v2 등 버전을 명확히 관리하고, 상위 시스템에서 여러 버전을 처리할 수 있도록 한다.

5. 공통 인터페이스와 데이터 모델 설계

5.1 핵심 메서드

1. `prepareSend(request) → PreparedTx`
   체인별 nonce 예약/fee 견적/서명 데이터 등을 처리하여 준비된 트랜잭션을 반환한다.

2. `broadcast(preparedTx) → BroadcastResult`
   준비된 트랜잭션을 네트워크에 브로드캐스트하고 결과(트랜잭션 해시, 실패 코드 등)를 반환한다.

3. `getTxStatus(query) → TxStatusSnapshot`
   지정된 트랜잭션의 현재 상태를 질의한다. 상태는 `UNKNOWN`, `PENDING`, `INCLUDED`, `SAFE`, `FINALIZED`, `FAILED`, `DROPPED`, `REPLACED` 중 하나로 표준화한다.

4. `getHeads() → HeadsSnapshot`
   최신 헤드·안전 헤드·최종 헤드(또는 지원되는 헤드 수준)를 반환한다. 일부 체인이 safe/finalized 개념을 지원하지 않는 경우 `null`로 반환하고 capability를 명시한다.

5.2 데이터 모델

- **SendRequest** – 체인 식별자(`chainId`), 자산(`asset`), 수신 주소(`to`), 금액(`amount`), 발신 지갑 ID(`fromWalletId`), 수수료 정책(`feePolicyId`), nonce 정책(`noncePolicy`), idempotency key 등을 포함한다.
- **TxStatusSnapshot** – 트랜잭션의 상태(`state`), 블록 번호/해시, confirmations(선택사항), finality hint(safe/finalized 여부) 및 정규화된 오류 코드 등을 포함한다. `confirmations` 필드는 일부 체인에 의미가 없으므로 **옵셔널**이어야 한다.
- **HeadsSnapshot** – 체인별 latest/safe/finalized 헤드 정보를 제공한다.

6. Nonce·Fee·Finality 차이를 흡수하는 전략

6.1 Nonce 전략

- **Serial Strict** – 계정별로 동시에 하나의 in-flight 트랜잭션만 허용한다. 가장 안전하고 단순하지만 처리량이 낮다. 멀티체인 관리에서 처음 도입하기 쉬운 전략이다.
- **Pipelined** – 일정 범위의 nonce를 선점하여 여러 트랜잭션을 병렬로 준비한다. 성능은 올라가지만 nonce 관리 복잡성이 증가하며, 실패/교체 시 효과적으로 롤백할 로직이 필요하다.
- **Pinned** – 특정 nonce를 사용해 교체(replace-by-fee) 트랜잭션을 보내거나 수동介入을 가능하게 한다. RBF와 유사하다.

EVM 체인은 nonce와 가스비에 따라 트랜잭션 교체가 가능하다. PoS/UTXO 체인은 nonce가 없거나 다른 형태의 sequence를 사용하므로 이러한 전략을 체인별로 분리해야 한다.

6.2 Fee 전략

어댑터는 수수료를 계산하지 않고 **estimateFee()**를 통해 base fee/priority fee 같은 객관적 데이터를 제공한다. Policy Engine은 `feePolicyId`를 통해 **NORMAL**, **FAST**, **SURGE**와 같은 정책을 선택하고, 네트워크 혼잡을 감지하여 `maxFee`를 주기적으로 인상하거나 교체 트랜잭션을 발송한다. 예를 들어 Blocknative는 `maxFee = 2 × baseFee + tip`을 추천한다 [17].

6.3 Confirm/Finality 전략

- 어댑터는 가능한 경우 latest/safe/finalized 헤드를 반환하여 상위 모듈이 정책에 따라 완료를 선언할 수 있게 한다.
- 체인별 finality policy는 다음과 같이 구성할 수 있다:
- **PoW/확률적 finality** – `CONFIRMED >= N` 블록과 `reorg buffer`를 고려하여 settle.
- **PoS (Ethereum)** – `SAFE` 혹은 `FINALIZED` 헤드 도달 시 settle [16].
- **즉각적 finality 체인** – 체인이 자체적으로 finality를 제공하는 경우 `included` 이후 즉시 settle.

7. RPC 신뢰/일관성 문제

멀티체인 지갑은 서로 다른 RPC 엔드포인트를 신뢰해야 한다. 하지만 노드 간 **split-view** 현상이나 노드 장애, rate limit 등으로 인해 RPC 결과가 항상 일관적이지 않다. Trail of Bits는 한 체인에서 동일 블록에 대한 두 개의 유효한 블록이 동시에 제안되는 경우를 보여 주며, 이로 인해 re-org가 발생하고 트랜잭션이 제거될 수 있다고 설명한다 [18]. 따라서 다음과 같은 설계가 필요하다.

- **멀티 RPC + 쿼럼** – 최소 두 개 이상의 RPC 공급자를 사용하고, 중요한 상태(e.g., finalized 헤드)는 하나의 RPC만으로 신뢰하지 않는다 [15].
- **관측 스냅샷 저장** – 각 RPC의 관측 결과를 스냅샷으로 저장하여 추후 감사·사고 분석 시 비교할 수 있도록 한다.
- **불일치 감지 지표** – head mismatch rate, receipt null rate 등을 모니터링하고 기준치를 초과하면 degrade mode로 전환하여 출금 속도를 제한하거나 수동介入으로 전환한다.

8. 인터페이스 불변성 유지 전략

체인을 추가할수록 "공통 인터페이스를 바꾸고 싶다"는 유혹이 커지나, 이를 최소화해야 상위 시스템의 복잡도가 폭발하지 않는다. 전략은 다음과 같다.

1. **Core + Extension 패턴** – 코어 API 4개 메서드는 웬만하면 변경하지 않는다. 체인별 특수 기능은 `supportsFinalizedHead()`, `supportsReplaceTx()` 같은 확장 메서드로 분리한다.

2. **옵셔널 필드 + capability 선언** – `confirmations`나 `finalityHint`처럼 체인별 의미가 다른 정보는 옵션 필드로 두되, 어댑터가 어떤 capability를 지원하는지 명시한다.

3. **버전 관리** – API v1은 최소 스펙, v2에서 서명 형식(EIP-712 등) 확장. 상위 모듈은 버전별 fallback을 지원해야 한다.

9. 예상 질문과 답변 (Q&A)

Q1. EVM의 nonce와 솔라나의 durable nonce는 무엇이 다른가?

- **EVM** – 계정(address) 단위로 단순하게 증가하는 `nonce`를 사용하여 트랜잭션 순서를 보장한다. 하나의 계정에서 병렬 트랜잭션을 보낼 경우 nonce 충돌 문제가 발생하며, 어댑터는 `NonceManager`를 통해 nonce를 예약하고 교체 트랜잭션(RBF)을 처리해야 한다 [1].
- **Solana** – 기본적으로 최근 블록해시가 nonce 역할을 하지만, 오프라인 서명 및 예약 실행을 위해 **Durable Nonce** 기능을 제공한다. durable nonce는 별도의 계정에 저장된 32바이트 값이며 트랜잭션을 미리 서명하고 나중에 제출할 수 있게 한다 [5]. 이를 사용하려면 첫 번째 명령으로 `nonceAdvance`를 호출하여 nonce를 갱신해야 하고, nonce 계정에 최소 잔액을 유지해야 한다 [6].

Q2. 왜 '확인 수(confirmations)' 대신 'finality policy'를 사용해야 하나?

확률적 finality 체인에서는 블록이 늘어나면 리오그 확률이 기하급수적으로 줄어들지만 완전한 확정성은 없다 [14]. 이더리움의 PoS 환경에서는 safe/finalized 블록 개념이 도입되어 두 번째 에폭까지 기다리면 거의 변경되지 않는다 [16]. 각 체인의 확정성 모델이 다르므로 "6 컨펌" 같은 하드코딩은 잘못된 안전 가정이다. 대신 체인·리스크·액수에 따라 `FinalityPolicy`를 정의하고, 정책 엔진이 조건을 충족할 때 완료를 선언해야 한다.

Q3. 멀티 RPC 설계 시 가장 중요한 점은?

- **중복성과 분산** – 여러 RPC 공급자를 사용하여 한 노드의 오류가 전체 시스템에 영향을 주지 않도록 한다. CompareNodes는 여러 공급자를 이용하면 **중요한 중복성**과 **지리적 다양성**을 확보할 수 있다고 설명한다 [15].
- **일관성 판정** – 두 RPC의 헤드 정보가 불일치하면 쿼럼을 통해 신뢰할 헤드를 결정하고, 불일치 지표가 높아지면 출금 속도를 줄이거나 수동介入 모드로 전환한다.

Q4. Cosmos 계정의 account number와 sequence는 무엇인가?

Medium 기사에 따르면 account number는 계정이 처음 자금을 받을 때 할당되는 고유 ID로 변경되지 않는다 [3]. sequence는 계정이 발송한 트랜잭션의 개수이며 트랜잭션마다 포함되어야 하고, 1씩 증가하여 **순차성을 강제하고 replay 공격을 방지**한다 [4].

Q5. EIP-1559 수수료 모델의 설계 이유는?

기존의 1차 가격 경매 방식은 수수료의 변동성이 높고 예측이 어려웠다. EIP-1559는 네트워크가 결정하는 **base fee**와 사용자가 지정하는 **priority fee**를 분리했다. base fee는 블록의 사용률에 따라 자동으로 증가 또는 감소하며 [8], priority fee는 채굴자에게 팁을 제공한다 [9]. 사용자는 `maxFee`를 (2×base fee + tip) 같은 방식으로 충분히 높게 설정해 base fee 상승에도 거래가 묶이지 않도록 해야 한다 [17]. 멀티체인 어댑터에서는 이를 단순히 `estimateFee()`로 제공하고, 정책 엔진이 max fee와 증액 규칙을 결정해야 한다.

10. 시각 자료

아래 인포그래픽은 체인별 실행 모델 차이를 직관적으로 정리한 것이다. **Nonce**, **수수료 모델**, **완료 선언**이라는 세 가지 축에서 EVM, UTXO, PoS/Sequence 체인의 차이를 요약한다.

11. 결론

멀티체인 수탁형 지갑에서의 추상화는 단순한 인터페이스 통일이 아니다. 각 체인의 실행 모델 차이(Nonce/UTXO/Sequence), 수수료 체계(EIP-1559 vs. 고정 수수료), 확정성 모델(확률적 finality vs. safe/finalized), RPC의 불완전

성 등을 이해하고, **정책 파라미터로 노출**하는 것이 핵심이다. 어댑터는 사실만 보고하고, 정책 결정은 상위 오케스트레이터가 수행해야 하며, 기본 인터페이스는 고정하고 확장과 버전 관리를 통해 새로운 체인을 유연하게 추가해야 한다. 이러한 원칙과 배경지식을 토대로, 수강생은 멀티체인 어댑터 설계 시 안전성·유연성·확장성을 동시에 달성할 수 있을 것이다.

---

1 Concurrency patterns for account nonce - Ethereum Stack Exchange
https://ethereum.stackexchange.com/questions/39790/concurrency-patterns-for-account-nonce

2 UTXO vs. Account Models | Alchemy Docs
https://www.alchemy.com/docs/utxo-vs-account-models

3 4 How do I get the Cosmos account number and sequence? | by Felix | Medium
https://ctrl-felix.medium.com/how-do-i-get-the-cosmos-account-number-and-sequence-3f1643af285a

5 6 Durable & Offline Transaction Signing using Nonces | Solana
https://solana.com/developers/guides/advanced/introduction-to-durable-nonces

7 8 9 10 17 EIP-1559 Gas Fees: Base Fee, Priority Fee, & Max Fee
https://www.blocknative.com/blog/eip-1559-fees

11 12 13 16 What are Ethereum commitment levels?
https://www.alchemy.com/overviews/ethereum-commitment-levels

14 18 The Engineer’s Guide to Blockchain Finality - The Trail of Bits Blog
https://blog.trailofbits.com/2023/08/23/the-engineers-guide-to-blockchain-finality/

15 Should You Work With More Than One RPC Node Provider? - comparenodes
https://www.comparenodes.com/blog/should-you-work-with-multiple-rpc-node-providers/