ChatGPT

수탁형 지갑의 출금(Withdrawal) 라이프사이클 연구 보고서

서론

기업이나 서비스가 고객 자산을 관리하는 **수탁형 지갑**(custodial wallet)을 설계할 때 가장 위험한 부분은 출금 처리입니다. 사용자의 출금 요청을 하나의 트랜잭션(hash)로 단순하게 생각하면 오류가 생겼을 때 문제를 추적하고 복구하는 과정이 복잡해지고 내부 관리를 자동화하기 어렵습니다. 이 보고서는 강의 자료에서 제공된 **Withdrawal**과 **TxAttempt** 상태기계를 바탕으로, 출금 라이프사이클과 관련된 배경 지식을 정리합니다. Ethereum과 유사한 계정 기반 체인을 기준으로 설명하되 다른 블록체인에도 적용할 수 있는 일반적인 원칙을 포함합니다.

1. Withdrawal(업무 단위)과 TxAttempt(체인 단위)의 분리

1.1 용어 정의

- **Withdrawal**: 사용자 또는 운영자가 요청한 "한 번의 출금 업무 단위"입니다. 사용자가 1ETH를 보내달라고 요청하면 Withdrawal 1건으로 취급합니다. 규정 준수 확인, 정책 체크, 승인, 서명, 전송 후 최종 회계 처리까지의 **업무 흐름**을 관리합니다.
- **TxAttempt**: 특정 출금을 처리하기 위해 블록체인에 보내는 개별 트랜잭션 시도입니다. 네트워크 상황이나 수수료 변경 때문에 같은 출금을 여러 번 시도할 수 있으며, 각 시도는 서로 다른 트랜잭션 해시(txhash)를 갖습니다. 체인에서 **nonce**와 가스 가격을 조정하면서 여러 개의 시도를 생성할 수 있습니다.
- **nonce**: 계정별 트랜잭션 순서를 나타내는 값으로, 현재까지 블록에 포함된 트랜잭션 수입니다. Ethereum에서는 각 트랜잭션이 이전 트랜잭션보다 더 큰 nonce를 가져야 하며, 중복 nonce나 누락된 nonce("nonce gap")가 있으면 후속 트랜잭션이 처리되지 않습니다 [1].

1.2 분리를 해야 하는 이유

블록체인의 트랜잭션은 네트워크의 상황에 따라 대기 시간, 가스 가격, 밈풀(mempool) 상태 등이 변화하며 하나의 출금이 여러 번 시도될 수 있습니다. "Withdrawal = TxHash 1개"라는 사고에 갇히면 다음과 같은 문제점이 생깁니다:

- 가스 가격을 올려 재전송할 때 이전 txhash가 Drop & Replace 될 수 있습니다 [2]. txhash만 추적하면 새로운 시도를 놓칠 수 있습니다.
- 트랜잭션이 mempool에서 사라지거나(RPC 별로 mempool 구성이 달라서) 다른 nonce의 트랜잭션에 의해 대체(replaced)될 수 있습니다 [3].
- 병렬 출금이 많은 환경에서는 nonce를 관리하는 단일 테이블과 `from` 주소별 single-writer 제약이 필요합니다 [4].

따라서 **Withdrawal**을 업무 흐름의 단위, **TxAttempt**를 체인 시도 단위로 분리하고, Withdrawal이 여러 TxAttempt를 참조하도록 설계해야 합니다.

2. Withdrawal 상태기계

Withdrawal 상태기계는 업무 관점에서 출금 요청이 어떤 단계에 있는지를 나타냅니다. 다음은 강의안의 표준 흐름(10단계)이며, 실제 시스템에서는 필요에 따라 세분화할 수 있습니다.

1. **W0 REQUESTED**: 사용자가 출금을 요청합니다. 요청 정보(받는 주소, 금액, 자금 출처 등)를 저장하고 초기 유효성 검사를 수행합니다.
2. **W1 POLICY_CHECKED**: AML/KYC 규정, 일일 한도, 수수료 정책 등을 자동 검사합니다. 이상 징후가 없으면 다음 단계로 이동합니다.
3. **W2 APPROVAL_PENDING**: 관리자가 승인해야 하는 단계입니다. 설정된 임계치(예: 일정 금액 이상)에서는 다중 승인(Multi-signer approval) 절차가 진행됩니다.
4. **W3 APPROVED**: 출금 요청이 승인되었습니다. 이제 서명 프로세스를 준비합니다.
5. **W4 SIGNING**: 지정된 키 관리 시스템(KMS/HSM)을 이용해 트랜잭션 서명을 수행합니다. 여러 서명 시도나 지갑 특성에 따라 시간이 걸릴 수 있습니다.
6. **W5 SIGNED**: 서명이 완료되어 TxAttempt를 생성할 수 있습니다.
7. **W6 BROADCASTED**: 첫 번째 TxAttempt가 블록체인 RPC에 전송되었습니다. mempool에서 관측되기를 기다립니다.
8. **W7 INCLUDED**: 하나 이상의 TxAttempt가 블록에 포함되어 `blockNumber`가 할당된 상태입니다. 아직 블록이 재조직될 수 있으므로 확정(finalized) 상태는 아닙니다.
9. **W8 SAFE/FINALIZED**: 블록체인에서 충분한 확정성을 얻어 되돌릴 수 없다고 판단되는 상태입니다(예: n개의 확인, PoS 체인의 Finality 체크). Chainlink 문서는 'Latest', 'Safe', 'Finalized' 세 가지 신뢰 수준을 언급합니다 [5]. 사용자는 보안 요구사항에 따라 적절한 확정성 수준을 선택해야 합니다.
10. **W9 LEDGER_POSTED**: 내부 장부에 최종 반영되고 회계 처리까지 마쳤습니다.
11. **W10 COMPLETED**: 모든 절차가 끝난 상태입니다.

Withdrawal 상태는 **업무 로직**을 담당하며, 각 단계에서 자동화/승인 정책을 적용할 수 있습니다. 예를 들어 W1에서 정책 검사 실패시 출금이 거절되거나 HOLD 상태에 둘 수 있습니다.

3. TxAttempt 상태기계

각 TxAttempt는 체인에 전송한 실제 트랜잭션을 추적합니다. 동일한 Withdrawal에 대해 여러 TxAttempt가 생성될 수 있으므로 시도 단위의 상태 추적이 필요합니다.

1. **A0 CREATED**: nonce를 예약하고 현재 가스 가격을 기반으로 tx 파라미터(가스 한도, fee cap 등)를 계산합니다.
2. **A1 SIGNED**: KMS/HSM을 이용해 서명된 raw transaction이 만들어졌습니다.
3. **A2 SENT_TO_RPC**: RPC 노드에 전송이 완료된 상태입니다. 여기서 오류가 발생할 경우 FAILED 예외가 발생할 수 있습니다.
4. **A3 SEEN_IN_MEMPOOL**: 하나 이상의 노드의 mempool에서 해당 트랜잭션을 관측했습니다. mempool은 각 노드가 유지하는 대기 트랜잭션 풀이며, 전파되지 않거나 가스 가격이 너무 낮으면 다른 노드의 mempool에는 나타나지 않을 수 있습니다 [3]. 드롭(dropped) 감지는 이 단계 이후에 수행합니다.
5. **A4 INCLUDED**: 블록에 포함되어 `blockNumber`가 할당되었습니다. 아직 확정되지는 않았습니다.
6. **A5 CONFIRMED**: 설정된 확인 수(`n confirmations`)를 만족했습니다. 이때도 체인 재조직(리오그) 가능성이 완전히 사라진 것은 아닙니다.
7. **A6 FINALIZED**: 체인의 합의 규칙에 따라 변경 불가능한 상태에 도달했습니다. Proof-of-Stake 체인의 경우 Justified → Finalized 블록을 의미합니다 [5].

Withdrawal과 TxAttempt 상태를 분리하면 `Dropped`나 `Replaced`가 발생했을 때 새 Attempt를 만들어도 Withdrawal 상태는 BROADCASTED 단계에 머물 수 있습니다. 또한 Attempt의 `from`와 `nonce`를 기반으로 시도 그룹을 추적하여 canonical txhash(실제로 체인에 포함된 시도)를 선정할 수 있습니다.

4. 예외 6종과 대응 전략

출금 시스템은 실패를 없애는 것이 아니라 **실패를 분류하고 자동/수동 경로로 라우팅하는 시스템**입니다. 아래 예외들은 실무에서 자주 발생하는 상황으로, 각 예외에 대한 감지 방법과 대응 전략을 제시합니다.

표 1. Withdrawal/TxAttempt 예외 유형과 대응 전략

예외 유형	의미/발생 조건	감지 신호	자동 재시도	사람 介入	권장 대응	출처
1) FAILED (시스템 오류)	서명 또는 브로드캐스트 단계에서 내부 오류. KMS/HSM 오류, RPC 오류, 내부 타임아웃 등.	서명 API 오류, RPC에서 `ErrNonceTooLow/TooHigh` 등 오류 코드.	☑ 가능	☐ 필요 없음 (오류 해결 시)	동일 nonce를 재사용할지 정책을 따르고 새 Attempt 생성.	[Blocknative `ErrNonceTooHigh` 오 nonce-gap [6].
2) EXPIRED	일정 시간 내 블록 포함 또는 확정에 실패(내부 SLA 초과). 정해진 만료시간은 시스템 정책에 따라 다르며, 블록체인 자체에는 시간 기반 만료가 없다 [7].	SLA 타이머 만료, 오랜 시간 `SEEN_IN_MEMPOOL` 상태로 유지.	☑ 가능	⚠ 경우에 따라 필요	가스 가격 재산정 후 재시도하거나 사용자가 취소할지 결정.	(노출되는 정 없으므로 내부 기반.)
3) DROPPED	트랜잭션이 mempool에서 사라지고 블록에도 포함되지 않음. 가스 가격이 너무 낮거나 노드의 mempool 정책에 의해 제거됨 [8].	일정 시간 `SEEN_IN_MEMPOOL` → 사라짐 (`eth_getTransactionByHash`가 null), 다른 노드에도 없음 [3].	☑ 가능	☐ 필요 없음	같은 nonce와 더 높은 가스 요금으로 재전송. 여러 RPC 엔드포인트를 사용해 전파 경로 확장.	[Etherscan] 트랜잭션은 수료가 반환되 운 트랜잭션을 할 수 있음 [2] mempool은 다름 [3].

예외 유형	의미/발생 조건	감지 신호	자동 재시도	사람 介入	권장 대응	출처
4) REPLACED	동일 nonce의 다른 트랜잭션이 우선 채택되어 기존 txhash가 무효화됨(보통 수수료를 올려 재전송하는 경우).	(from, nonce) 키에서 서로 다른 txhash가 블록에 포함됨. 기존 txhash의 receipt를 조회하면 `null` 또는 상태가 없고 새로운 txhash가 포함됨.	☑ 가능 (정리 필요)	⚠ 판단 필요	기존 Attempt를 `REPLACED`로 종료하고, 새로 채택된 txhash를 canonical attempt로 채택. 필요시 새 Withdrawal state로 업데이트.	[Alchemy] D & Replaced 설명: 낮은 가 된 트랜잭션이 스의 동일 no 잭션에 의해 [9].
5) REVERTED	트랜잭션이 블록에 포함됐지만 실행이 실패(EVM revert). 스마트컨트랙트 조건 불충족, require 실패 등.	`eth_getTransactionReceipt`의 `status` 필드가 `0`이고 `revertReason`가 포함될 수 있음 [10].	☐ 자동 재시도 불가 (원인 제거 전)	☑ 필요	계약 상태, 데이터, 인가 문제를 분석 후 수정. 재승인이 필요하다면 Withdrawal을 다시 생성.	[Besu] recei `status=0` 실패를 의미하 `revertRea` 드를 통해 오 를 얻을 수 있
6) RPC_INCONSISTENT	여러 RPC 노드 간에 트랜잭션 관측 상태가 달라서 결정적 판단을 내릴 수 없음. 공용 RPC는 과부하, 지연, 재구성 등으로 신뢰도가 떨어질 수 있음 [12].	A 노드에는 트랜잭션 있음, B 노드에는 없음; receipt 정보가 불일치.	☑ 다른 경로로 재전송 가능	⚠ 필요	다중 RPC 쿼럼을 구성하여 다수결에 따라 결정을 내리거나, 자체 노드 및 백업 노드를 운영하여 “소스 오브 트루스”를 확보.	[Medium] R 포인트가 불안 있으므로 여러 를 사용하고 해야 함 [13].

5. Nonce 관리와 병렬 출금 처리

5.1 Nonce gap 문제

블록체인은 계정별로 트랜잭션을 순서대로 처리합니다. nonce 값이 중복되거나 건너뛰면 이후 트랜잭션이 처리되지 않고 `ErrNonceTooLow` 또는 `ErrNonceTooHigh` 오류가 발생합니다 [1]. 여러 출금 요청을 병렬로 처리하는 환경에서는 같은 주소에 대해 한 프로세스에서 nonce를 할당해야 합니다. Circle의 지갑 설계 문서는 이를 "주소 단위 nonce는 **공유 자원**"이라 표현하며, 동시 출금이 많은 경우 중앙 관리 테이블에서 nonce를 예약해야 한다고 권고합니다 [4].

5.2 Nonce reservation 정책

1. **Nonce reservation table**: 각 `from` 주소별 다음 사용 가능한 nonce를 기록하고, 하나의 프로세스만이 이를 업데이트하도록 락(lock) 또는 DB 트랜잭션으로 보호합니다.
2. **Single writer per address**: 한 번에 하나의 프로세스만 nonce를 할당하여 데이터 경쟁을 방지합니다. 다른 스레드는 대기하거나 큐에 넣어 순차적으로 처리합니다.
3. **Nonce gap detection**: 현재 계정의 confirmed nonce(블록에 포함된 트랜잭션 수)를 모니터링하고, pending한 TxAttempt들의 nonce 목록과 비교해 누락된 nonce가 있는지 확인합니다. Blocknative는 누락된 nonce를 찾은 후 그 nonce로 다시 전송해야 higher nonce 트랜잭션들이 처리될 수 있다고 설명합니다 [6].
4. **Nonce gap recovery**: 누락된 nonce가 발견되면 가스 가격을 재설정해 동일 nonce 트랜잭션을 재전송합니다 [14].

5.3 Replacement, Cancel, Speed-up

- **Replacement / Speed-up**: 기존 트랜잭션의 가스 가격이 너무 낮아 채굴되지 않을 경우, 같은 nonce로 가스 가격을 **최소 10% 이상 높여** 새 트랜잭션을 보내면 채굴자가 더 높은 수수료를 선택하면서 기존 트랜잭션이 대체됩니다 [15]. 이때 새로운 txhash가 생성되며, 이전 txhash는 `DROPPED` 또는 `REPLACED` 상태로 처리합니다.
- **Cancel**: 사용자가 보낸 트랜잭션을 취소하려면 같은 nonce와 더 높은 가스 가격을 사용하되, `to` 주소를 자신 주소로 설정하고 `value=0`인 트랜잭션을 전송합니다. 성공적으로 채굴되면 기존 트랜잭션은 무력화됩니다 [15].

시스템 설계 시 이러한 동작을 상태머신에 반영해야 합니다. 동일 nonce 그룹을 `attempt_group_key=(from, nonce)`로 묶고, 그 그룹의 canonical attempt를 최종 블록에 포함된 txhash로 결정합니다.

6. Finality, Confirmation, and Mempool 고려사항

6.1 Finality vs Confirmation

- **Confirmation**: 블록체인에서 트랜잭션이 포함된 이후 추가 블록이 몇 개 쌓였는지를 의미합니다. 예를 들어, `n confirmations`는 현재 블록 이후 n개의 블록이 더 생성된 것을 뜻하며, 일반적으로 n이 클수록 되돌릴 가능성이 감소합니다.
- **Finality**: 블록이 합의 규칙에 따라 더 이상 되돌릴 수 없는 상태입니다. Chainlink는 신뢰 수준을 `Latest`(실시간, finality 보장 없음), `Safe`(블록이 리오그될 가능성이 낮음), `Finalized`(되돌릴 수 없음)로 분류합니다 [5]. Proof-of-Work 체인에서는 6~12 confirmations 후 사실상의 finality로 간주하는 반면, Proof-of-Stake 체인(예: Ethereum 2.0)에서는 chain finalized 체크를 통해 확정 블록을 제공합니다. Withdrawal 상태 W8('SAFE/FINALIZED')는 이 finality를 만족한 시점을 의미하며, 이후 회계 처리(W9)로 이동할 수 있습니다.

6.2 Mempool과 노드 간의 불일치

- **Mempool 개념**: 각 이더리움 노드는 블록에 포함되기 전 트랜잭션을 mempool에 저장합니다. Alchemy 등 자료에 따르면 mempool은 각 노드의 프라이빗한 큐이며, 노드 간에 거래를 브로드캐스트하여 공유하지만 모든 노드가 동일한 mempool을 유지하는 것은 아닙니다 [16]. 일부 연구는 검증자들이 약 60%의 트랜잭션을 공유하지만 40%는 특정 노드의 mempool에서 보이지 않을 수 있다고 설명합니다 [3].
- **Dropped transactions**: 트랜잭션이 mempool에서 제거되는 이유는 여러 가지입니다. Alchemy와 Etherscan은 낮은 가스 가격, 잘못된 nonce, 트랜잭션 풀 제한에 도달, 불충분한 잔고 등을 원인으로 들고 있습니다 [8]. dropped된 트랜잭션은 블록에 포함되지 않았기 때문에 자산은 반환되며 새로 전송하면 됩니다 [2].
- **RPC inconsistency**: 공용 RPC는 서버 과부하, 지리적 지연, 버전 차이, 클라이언트별 구현 차이 등으로 동일한 결과를 반환하지 않을 수 있습니다 [12]. 따라서 출금 시스템은 **여러 RPC 엔드포인트**를 사용해 트랜잭션의 상태를 크로스체크하고, quorum(다수결) 모델로 진실을 결정해야 합니다. 자체 노드와 신뢰할 수 있는 백업 노드를 운영하면 이 문제를 완화할 수 있습니다 [13].

7. 운영 로직: 프로처럼 보이는 포인트

1. **Nonce 관리**: 주소별로 하나의 프로세스만 nonce를 배정해야 하며, reservation 테이블을 통해 충돌을 방지합니다 [4]. `nonce gap`을 감지하고 새 Attempt를 재전송해 갭을 메워야 higher nonce 트랜잭션이 처리됩니다 [6].
2. **Replacement/Cancel/Speed-up 정책**: 동일 nonce 그룹을 추적하면서 fee bump(재전송), cancel, speed-up을 지원합니다. 이전 txhash를 실패로 처리하는 것이 아니라 `REPLACED`로 구분합니다 [9].
3. **TxTracker의 키**: 첫 번째 키는 **tx_hash**로 쉽게 조회할 수 있지만, 대체/삭제 사례에서는 불완전합니다. 두 번째 키로 `(from, nonce)`를 유지하여 replacement나 dropped 상황에서도 트랜잭션 상태를 정확히 파악합니다.
4. **다중 RPC와 모니터링**: 노드마다 mempool과 상태가 다르므로, 여러 RPC를 병행하여 모니터링하고 inconsistency를 감지합니다. 모니터링 시스템은 `SEEN_IN_MEMPOOL`을 관찰하다가 일정 시간 후 사라지면 drop 감지 로직을 수행합니다.
5. **사람개입 시점 정의**: 자동화는 중요하지만 오라클이나 스마트컨트랙트 오류, 사용자 키 관리 오류 등 인간의 판단이 필요한 경우를 정의해야 합니다. 예를 들어 `REVERTED`나 `RPC_INCONSISTENT` 상황에서 재승인 혹은 노드 교체를 결정할 때입니다.
6. **출금 SLA와 만료 정책**: 블록체인 트랜잭션은 일정 기간이 지나도 자동으로 사라지지 않으며, 만료는 내부 SLA에 따라 정의됩니다 [7]. SLA를 명시하고 시간이 초과되면 수수료 재산정이나 사용자의 재요청을 유도해야 합니다.
7. **재무/회계 처리**: W9(LEDGER_POSTED) 단계에서 내부 회계 시스템에 트랜잭션을 반영하고 잔고를 갱신합니다. 출금 시스템은 외부 트랜잭션과 내부 회계 간의 일관성을 유지해야 합니다.

8. 결론

수탁형 지갑에서 안전하고 신뢰성 있는 출금 처리를 위해서는 **업무 흐름(Withdrawal)**과 **체인 시도(TxAttempt)**를 분리하고, **예외를 분류하고 라우팅**하는 상태머신을 구현해야 합니다. mempool과 nonce 관리, finality 개념, RPC 불일치 등 외부 환경의 복잡성을 이해하고 이에 맞추어 **자동화 정책과 수동개입 시점**을 설계해야 합니다. 이 보고서에서 정리한 자료와 출처는 강의 준비에 참고가 될 것이며, 실제 시스템 설계 시에는 체인별 특성과 내부 정책을 반영하여 세부 구현을 조정해야 합니다.

---

[1] Managing nonces - Nethereum Documentation
https://docs.nethereum.com/en/latest/nethereum-managing-nonces/

[2] [8] Etherscan Information Center | Transaction Dropped & Replaced
https://info.etherscan.com/transaction-dropped-replaced/

3 Transactions do not appear in Ethereum mempool - Ethereum Stack Exchange
https://ethereum.stackexchange.com/questions/159502/transactions-do-not-appear-in-ethereum-mempool

4 Wallet Nonce Management - Circle Docs
https://developers.circle.com/cpn/concepts/transactions/wallet-nonce-management

5 Finality and Confidence Levels | Chainlink Documentation
https://docs.chain.link/cre/concepts/finality-ts

6 Ethereum Failed Transactions: Top 10 Errors to Avoid
https://www.blocknative.com/blog/ethereum-transaction-errors

7 gas - How long before transaction is removed from the Mempool - Ethereum Stack Exchange
https://ethereum.stackexchange.com/questions/129611/how-long-before-transaction-is-removed-from-the-mempool

9 16 Ethereum Transactions - Pending, Mined, Dropped & Replaced | Alchemy Docs
https://www.alchemy.com/docs/ethereum-transactions-pending-mined-dropped-replaced

10 11 Include revert reason | Besu documentation
https://besu.hyperledger.org/private-networks/how-to/send-transactions/revert-reason

12 13 Understanding and Resolving RPC Issues in Blockchain | by lanre | Medium
https://medium.com/@akintayolanre2019/understanding-and-resolving-rpc-issues-in-blockchain-858822941ed7

14 Blockchain Transactions in Ethereum Mempool - Coding Edition
https://chainstack.com/a-developers-guide-to-the-transactions-in-mempool-code-edition/

15 Ethereum Speed Up Transactions: How to Accelerate Your TX
https://www.blocknative.com/blog/speed-up-transactions