**Quorum Wiki**



# 

# Quorum wiki에 오신 것을 환영합니다!

본 사이트는 다음 분들을 위해 제작되었습니다.   
\* Quorum에 기여하고자 하는 블록체인 개발자   
\* Quorum 상에 어플리케이션을 구축하고자 하는 스마트 컨트랙트 개발자   
\* Quorum에 대해 배우고자 하는 누구나

J.P. Morgan의 분산원장기술 개발에 대한 더욱 많은 정보를 얻기 위해서는 저희 [웹사이트](https://www.jpmorgan.com/quorum)를 방문해주세요.

**NOTE**  
Quorum은 go-ethereum 클라이언트(geth)를 포크하여 최소한의 수정만 수행하였습니다 . 이더리움을 처음 접하신다면 [이더리움 위키](https://github.com/ethereum/wiki/wiki)를 먼저 읽고 [go-ethereum 클라이언트](https://github.com/ethereum/go-ethereum) 개념에 익숙해지시길 권장합니다. 본 위키는 계정, 트랜잭션, 블록, 스마트 컨트랙트 등 핵심 이더리움 개념에 대해 기본적인 이해가 있다는 전제 하에 작성하였습니다.

## Wiki 사용하기

Quorum 아키텍처 및 Quorum이 어떻게 작동하는지 자세히 알고 싶다면 [Quorum 알아보기] 섹션을 확인해보세요. Quorum을 사용할 준비가 되었다면 (혹은 준비가 되지 않아도 바로 시작해보고 싶다면) [셋업하기]와 [Quorum 사용하기] 섹션에서 여러분만의 허가형 네트워크를 셋팅해보고 프라이버시가 보장된 상태에서 거래를 진행해보세요. 또한 [Quorum 예제](https://github.com/jpmorganchase/quorum-examples)를 통하여 Quorum 네트워크 상에 빠르게 적용할 수 있는 스크립트와 Quorum의 핵심 기능을 확인해보세요. 더 많은 정보는 [FAQ]를 읽어주시길 바라며 [제품 로드맵]에서 Quorum에 추가될 다양한 핵심 기능들을 확인해 보세요.

본 Quorum 문서들이 최대한 도움이 되길 바라며 질문이나 피드백이 있다면 주저하지 말고 연락주십시오.

## Wiki 목차

**Part 1 Quorum 알아보기**

* + Quorum 개요
    - Quorum이란?
    - 아키텍처
    - 구성요소
  + 트랜잭션 처리 및 프라이버시
    - 퍼블릭 트랜잭션
    - 프라이빗 트랜잭션
    - 트랜잭션 처리
  + Quorum 합의 알고리즘
    - Raft 합의 알고리즘
    - 이스탄불 BFT 합의 알고리즘
  + 보안 및 네트워크 권한 승인
    - 네트워크 권한
    - 엔클레이브(Enclave) 암호화 기술
    - 프라이빗키 저장 알고리즘

**Part 2 셋업하기**

* + 셋업 및 퀵 스타트
  + 소스를 통한 Quorum 노드 빌드
  + 테스트 실행
  + Constellation 설치
  + Tessera 설치
  + 고급 셋업

**Part 3 Quorum 사용하기**

* + 스마트 컨트랙트 개발
  + 허가형 네트워크 셋업
  + Quorum API
  + ZSL 사용
  + ZSL

**Part 4 FAQ**

**Part 5 제품 로드맵**

## Part 1 Quorum 알아보기

**[Quorum 개요]**

## Quorum이란?

Quorum은 이더리움 기반의 분산원장 프로토콜으로, 이더리움에 프라이빗한 특성을 적용하여 금융서비스 산업에서 트랜잭션과 컨트랙트의 프라이버시를 보장받을 수 있도록 하기 위해 개발되었습니다.

Quorum은 geth라고 흔히 불리는 go-ethereum 클라이언트를 포크하여 최소한의 수정만 하였기 때문에, 이더리움 개발자 커뮤니티에서 구현된 모듈들의 활용이 가능합니다.

퍼블릭 이더리움에서 추가된 Quorum의 대표적 기능은 다음과 같습니다.

* 트랜잭션과 컨트랙트 프라이버시
* 다수의 투표 기반 합의 메커니즘
* 네트워크/개인의 허가 관리
* 향상된 속도

Quorum은 현재 다음과 같은 구성을 가지고 있습니다:

* Quorum 노드 (Geth 클라이언트에서 수정된 버전)
* Constellation/Tessera - 트랜잭션 매니저
* Constellation/Tessera - 엔클레이브(Enclave)

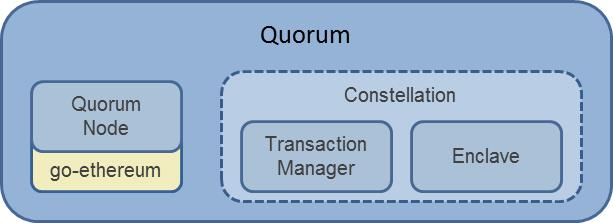
Quorum은 금융 서비스에 적용하는 것을 염두에 두고 설계되었지만 금융 서비스에만 국한되지 않고 상기 명시한 기본 기능을 필요로 하는 이더리움 활용에 관심이 있는 다른 산업들에도 적용 가능합니다.

### 참고 문서

Quorum의 설계와 배경에 대한 자세한 내용은 [**Quorum 백서**](https://github.com/jpmorganchase/quorum-docs/blob/master/Quorum%20Whitepaper%20v0.1.pdf)를 읽거나 [Hyperledger deck](https://drive.google.com/open?id=0B8rVouOzG7cOeHo0M2ZBejZTdGs) 혹은 2016년 9월 22일 하이퍼레저 프로젝트 기술 운영위원회 회의에 제공된 [프레젠테이션](https://drive.google.com/open?id=0B8rVouOzG7cOcDg4UkxqdTBacm8)을 참고하십시오.

## 

## 논리적 아키텍처 다이어그램



## 구성요소

### Quorum 노드

Quorum 노드는 지속적으로 성장하고 있는 이더리움 커뮤니티의 연구 및 개발 결과물과 호환성을 이룰 수 있도록 의도적으로 geth에 최소한의 수정만을 하는 형태로 설계되었습니다. 이러한 설계를 바탕으로 Quorum은 geth 릴리스에 맞추어 함께 업데이트할 것입니다.

Quorum 노드는 geth를 다음과 같이 수정하였습니다.

1. 합의는 작업증명방식(PoW) 대신 Raft 또는 이스탄불 BFT(Istanbul BFT) 합의 알고리즘을 사용합니다.   
2. 허가된 노드와의 연결만 허용하도록 P2P 레이어를 수정하였습니다.   
3. 블록 생성 로직에서 ‘global state root’를 ‘global public state root’로 대체하였습니다.   
4. 블록 검증 로직에서 블록 헤더의 ‘global state root’를 ‘global public state root’로 대체하였습니다.  
5. 상태전이 패트리샤 트리 (State Patricia trie)를 퍼블릭 상태 트리와 프라이빗 상태 트리로 분리하였습니다.   
6. 블록 검증 로직에서 ‘프라이빗 트랜잭션’을 처리하도록 수정하였습니다.   
7. 데이터 프라이버시 보호를 위하여 트랜잭션 데이터를 암호화된 해시로 대체하도록 트랜잭션 생성 로직을 수정하였습니다.   
8. 가스의 개념은 남아있지만 가스 가격(gas price)은 제거하였습니다.

### Constellation 및 Tessera

[Constellation](https://github.com/jpmorganchase/constellation)과 [Tessera](https://github.com/jpmorganchase/tessera)는 안전한 방식으로 정보를 전달하기 위한 범용 시스템으로 Haskell 및 Java 로 구현하였습니다. 이 시스템들은 메시지를 PGP(Pretty Good Privacy)로 암호화하는 MTA(Message Transfer Agents, 메시지 전송 에이전트) 네트워크와 비교할 수 있습니다. 해당 시스템들은 블록체인에 국한되는 것은 아니며, 다양한 이해관계자가 존재하는 네트워크에서 개별적으로 암호화된 메시지 교환을 원하는 다양한 애플리케이션에 적용할 수 있습니다. Constellation과 Tessera 모듈은 다음 두 개의 하위 모듈로 구성되어 있습니다.   
\*노드(Quorum의 PrivateTransactionManager에서 사용)  
 \*엔클레이브(enclave)

#### 트랜잭션 매니저

Quorum의 트랜잭션 매니저는 트랜잭션의 프라이버시를 담당합니다. 트랜잭션 매니저는 암호화된 트랜잭션 데이터의 저장 및 접근 제어, 다른 트랜잭션 매니저와 암호화된 페이로드 교환을 수행하지만 트랜잭션을 암/복호화하는 프라이빗 키에는 접근할 수 없습니다. 또한 트랜잭션 매니저는 암호화 기능을 위해 엔클레이브를 사용합니다. (엔클레이브가 선택적으로 트랜잭션 매니저 자체에서 호스팅되는 것도 가능합니다.)

트랜잭션 매니저는 restful/stateless 서비스로 쉽게 부하 분산이 가능합니다.

트랜잭션 매니저가 엔클레이브와 어떻게 상호작용하는지에 대한 더욱 자세한 내용은 [트랜잭션 처리 및 프라이버시]를 참조십시오.

#### 엔클레이브 (Enclave)

분산원장 프로토콜은 일반적으로 트랜잭션 유효성 검증, 참여자 인증, 과거 데이터 보존 (즉, 암호화된 해시 데이터 체인 사용)을 위해 암호화 기술을 활용합니다. 이러한 암호화 기술을 적용함에 있어서 성능향상 및 모듈간 명확한 역할 분담을 위하여 암호화 기능들의 병렬 처리 및 대칭 키 생성, 데이터 암/복호화와 같은 대부분의 암호화 작업은 엔클레이브에서 처리합니다.

엔클레이브는 트랜잭션 매니저와 상호 통신하며 트랜잭션 데이터를 처리합니다. 이때 트랜잭션 매니저와는 분리된 방식으로 암/복호화를 관리함으로써 트랜잭션의 프라이버시를 강화합니다. 엔클레이브는 다른 구성 요소들과 격리된 “가상 HSM”로 프라이빗 키를 보관하고 있습니다.

엔클레이브에 대한 더욱 상세한 내용은 [트랜잭션 처리 및 프라이버시]를 참조하십시오.

**[트랜잭션 처리 및 프라이버시]**

Quorum의 주요 기능 중 하나는 트랜잭션 프라이버시입니다. 이를 위해 ‘퍼블릭 트랜잭션’ 및 ‘프라이빗 트랜잭션’ 개념을 설명드리도록 하겠습니다. 이것은 개념적인 콘셉트일 뿐이며 Quorumㅇ새로운 트랜잭션 유형을 도입하지 않지만 이더리움 트랜잭션 모델에서 선택적인 privateFor 파라미터를 포함하도록 확장되었으며(이로 인해 Quorum에서 트랜잭션이 프라이빗한 것으로 처리됨) 트랜잭션 타입에는 이러한 트랜잭션을 식별하는 새로운 IsPrivate 요소가 추가되었습니다.

[Constellation](https://github.com/jpmorganchase/constellation) / [Tessera](https://github.com/jpmorganchase/tessera)는 암호화 및 그 관련 작업을 수행하며 프라이빗 페이로드(payloads)를 의도된 수신자에게 전송하기 위한 목적으로 Quorum에서 사용됩니다.

## 퍼블릭 트랜잭션 (Public Transactions)

소위 말하는 ‘퍼블릭 트랜잭션’은 같은 Quorum 네트워크의 모든 참가자가 페이로드를 볼 수 있는 트랜잭션입니다. 이들은 [일반적인 방식의 표준 이더리움 트랜잭션으로 생성됩니다.](https://github.com/ethereum/wiki/wiki/JavaScript-API#web3ethsendtransaction)

퍼블릭 트랜잭션의 예로는 일부 서비스 제공업체의 마켓 데이터 업데이트 또는 채권 보안 정의에 대한 수정과 같은 일부 참고 데이터 업데이트가 있습니다.

노트: ‘퍼블릭’ 트랜잭션은 퍼블릭 이더리움 네트워크의 거래가 아닙니다. ‘공동’ 혹은 ‘글로벌’ 트랜잭션이 더욱 적절한 표현일 수 있지만, ‘프라이빗’ 트랜잭션과 대조하기 위해 ‘퍼블릭’이라는 용어를 사용합니다.

## 프라이빗 트랜잭션 (Private Transactions)

소위 말하는 ‘프라이빗 트랜잭션’은 프라이빗 키가 트랜잭션의 privateFor 파라미터에 지정된 네트워크 참여자에게만 페이로드가 보여지는 트랜잭션입니다. privateFor은 쉼표로 구분된 목록에서 여러 개의 주소를 추출할 수 있습니다. ([Quorum 사용하기] 섹션에서 ‘프라이빗 트랜잭션 생성’ 참조)

Quorum 노드가 null이 아닌 privateFor 값을 가진 트랜잭션과 만나면 트랜잭션 서명의 V 값을 37 또는 38로 설정합니다. (이더리움 옐로우 페이퍼에 명기된 바와 같이 표준 이더리움 상에서 트랜잭션이 ‘퍼블릭’임을 나타내기 위한 값이 27 또는 28인 것과는 대조적)

## 트랜잭션 처리

### 퍼블릭 vs 프라이빗 트랜잭션 핸들링

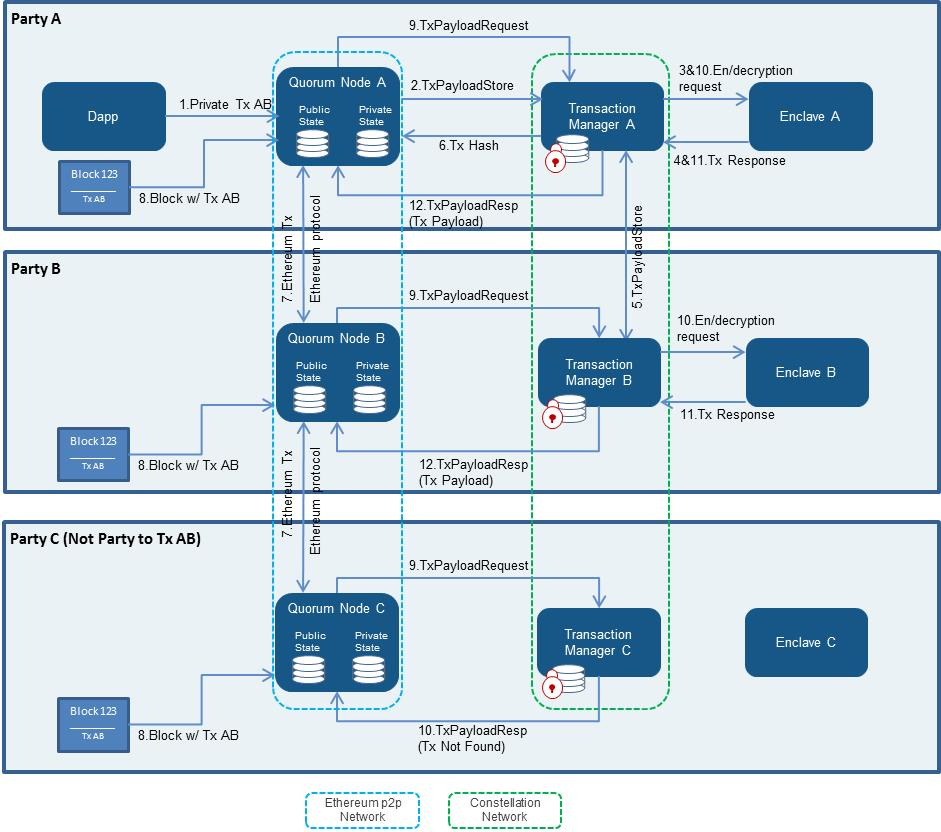
퍼블릭 트랜잭션은 표준 이더리움 방식으로 실행되므로 퍼블릭 트랜잭션이 컨트랙트 코드를 보유한 계정으로 전송되면 각 참여자는 동일한 코드를 실행하고 이에 따라 StateDB가 업데이트 됩니다.

하지만 프라이빗 트랜잭션은 표준 이더리움에 따라 실행되지 않습니다. 발신자의 Quorum 노드가 트랜잭션을 나머지 네트워크로 전달하기 전에 원래 트랜잭션 페이로드를 Constellation/Tessera에서 받은 암호화된 페이로드의 해시로 바꿉니다. 거래 당사자인 참여자들은 Constellation/Tessera 인스턴스를 통해 실제 페이로드로 해시를 대체할 수 있으나 당사자가 아닌 참여자들은 해시만 볼 수 있습니다.

그 결과, 프라이빗 트랜잭션이 컨트랙트 코드를 보유한 계정으로 전송되면 거래 당사자가 아닌 참여자들은 거래를 생략하고 결국 컨트랙트 코드를 실행하지 않게 됩니다. 그러나 거래 당사자인 참여자들은 실행을 위해 EVM을 호출하기 전에 해시를 원래 페이로드로 대체하고 그에 따라 StateDB가 업데이트됩니다. geth 클라이언트에 상응하는 변경 사항이 없을 경우, 이 두 세트의 참여자는 서로 다른 StateDB를 가진 채로 끝나게 되며 합의에 도달하지 못하게 됩니다. 따라서 컨트랙트 상태의 이러한 분기점을 해결하기 위해 Quorum은 퍼블릭 컨트랙트 상태를 전역적으로 동기화된 프라이빗 스테이트 트리(Trie)에 저장하고 프라이빗 컨트랙트 상태를 전역적으로 동기화되지 않은 프라이빗 스테이트 트리(Trie)에 저장합니다. 이와 관련하여 컨센서스가 달성되는 방법에 대한 자세한 내용은 [Quorum 합의 알고리즘] 섹션을 참조하십시오.

### 프라이빗 트랜잭션 프로세스 플로우

다음은 프라이빗 트랜잭션이 Quorum에서 처리되는 방법에 대한 설명입니다.



아래 예시에서 참여자 A와 B는 트랜잭션AB의 당사자이고 참여자 C는 당사자가 아닙니다.

1. 참여자 A는 트랜잭션 페이로드를 지정하고 privateFor를 참여자 A와 B의 퍼블릭 키로 설정하여 Quorum 노드에 트랜잭션을 전송합니다.
2. 참여자 A의 Quorum 노드는 트랜잭션을 페어링된 트랜잭션 관리자에게 전달하여 트랜잭션 페이로드를 저장하도록 요청합니다.
3. 참여자 A의 트랜잭션 매니저는 발신자의 유효성을 검사하고 페이로드를 암호화하기 위해 엔클레이브를 호출합니다.
4. 참여자 A의 엔클레이브는 참여자 A의 프라이빗 키를 확인하고 유효함이 확인되면 트랜잭션 변환을 수행합니다. 이는 다음을 수반합니다:
   1. 대칭키 및 랜덤 논스(Nonce) 생성
   2. 1번의 대칭키로 트랜잭션 페이로드 및 논스 암호화
   3. 2번에서 암호화 된 페이로드의 SHA3-512 해시를 계산
   4. 트랜잭션 수신자 목록 (이 경우 참여자 A와 B)을 반복하고 수신자의 퍼블릭키(PGP 암호화)를 이용하여 1번에서의 대칭키를 암호화
   5. 단계 2에서 암호화 된 페이로드를 및 단계 3에서의 해시, 그리고 4단계에서 암호화된 키 (각 수신자 별)를 트랜잭션 관리자에게 반환
5. 참여자 A의 트랜잭션 매니저는 암호화된 페이로드 (대칭키로 암호화됨)와 해시를 인덱스로 사용하여 암호화된 대칭키를 저장한 다음, 해시, 암호화된 페이로드 및 참여자 B의 퍼블릭키를 통해 암호화된 대칭키를 안전하게 (HTTPS를 통해) 참여자 B의 트랜잭션 매니저에게 전송합니다. 참여자 B의 트랜잭션 매니저는 Ack/Nack response로 응답합니다. 참여자 A가 응답을 받지 못하거나 참여자 B로부터 Nack을 수신하면 트랜잭션이 네트워크에 전파되지 않습니다. 수신자가 전달된 페이로드를 저장하는 것은 전제 조건입니다.
6. 참여자 B의 트랜잭션 관리자에게의 데이터 전송이 성공하면 참여자 A의 트랜잭션 매니저는 해시를 Quorum 노드에 반환하고 Quorum 노드는 트랜잭션의 원래 페이로드를 해당 해시로 바꾼 다음, 트랜잭션의 V값을 37 또는 38로 변경합니다. 이는 다른 노드들에게 이 해시는 무의미한 바이트 코드가 있는 퍼블릭트랜잭션과는 달리 연관된 암호화 된 페이로드가 있는 프라이빗 트랜잭션을 나타냄을 보여줍니다.
7. 트랜잭션은 표준 이더리움 P2P 프로토콜을 사용하여 나머지 네트워크로 전파됩니다.
8. 트랜잭션 AB를 포함하는 블록이 생성되어 네트워크의 각 참여자에게 배포됩니다.
9. 블록을 처리하는 과정에서 모든 참여자들은 트랜잭션 처리를 시도합니다. 각 Quorum 노드는 37 또는 38의 V값을 인식하여 페이로드가 해독을 필요로 하는 것으로 트랜잭션을 식별하고 로컬 트랜잭션 관리자를 호출하여 트랜잭션을 보류할지 확인합니다. (조회 시, 해시를 인덱스로 사용)
10. 참여자 C는 트랜잭션을 보유하지 않으므로 NotARecipient 메시지를 받고 트랜잭션을 생략하고 프라이빗 StateDB는 업데이트되지 않습니다. 참여자 A와 B는 로컬 트랜잭션 관리자에서 해시를 조회하고 트랜잭션을 보유하고 있음을 확인합니다. 그런 다음 각각 엔클레이브로 호출을 걸어 암호화 된 페이로드, 암호화 된 대칭키 및 서명을 전달합니다.
11. 엔클레이브는 서명의 유효성을 검사한 다음 참여자의 개인키를 활용하여 대칭키를 해독하고 현재 공개된 대칭키를 사용하여 트랜잭션 페이로드를 해독한 후, 트랜잭션 매니저에게 암호화된 페이로드를 반환합니다.
12. 참여자 A와 B에 대한 트랜잭션 매니저는 컨트랙트 코드 실행을 위해 해독된 페이로드를 EVM으로 보냅니다. 이 실행은 Quorum 노드의 프라이빗 StateDB에서만 상태를 업데이트합니다. 참고: 일단 코드가 실행되면 폐기되므로 위의 프로세스를 거치지 않고 읽을 수 없습니다.

**[Quorum 합의 알고리즘]**

허가형 네트워크에서는 불필요하게 해시 파워가 낭비되는 작업 증명(PoW, Proof-of-Work)나 지분 증명(PoS, Proof-of-Stake) 방식을 사용할 필요가 없습니다. 그렇기 때문에 Quorum은 컨소시엄 체인(허가형 네트워크)에 보다 적합한 다음과 같은 합의 메커니즘을 제공합니다:

\***Raft 합의 알고리즘**: 빠른 블록 생성시간(blocktimes)과 거래 완결성(transaction finality)을 가지며, 요청이 들어오는 경우에만 블록을 생성하는 합의 모델입니다. 자세한 내용은 [이더리움/Quorum을 위한 Raft 알고리즘](https://github.com/jpmorganchase/quorum/blob/master/docs/Consensus/raft.md)에서 확인하세요.   
\***이스탄불 BFT (Byzantine Fault Tolerance) 합의 알고리즘**: PBFT(Practical Byzantine Fault Tolerance)에서 영감을 받아 AMIS에서 구현한 거래 완결성을 지닌 합의 알고리즘입니다. 자세한 내용은 이스탄불 BFT 합의 알고리즘, [RPC API](https://github.com/getamis/go-ethereum/wiki/RPC-API), [본 기술 문서](https://medium.com/getamis/istanbul-bft-ibft-c2758b7fe6ff)에서 확인하세요.

\* **클릭 POA**(**Clique Proof-of-Authority) 합의 알고리즘**: Go-ethereum(Geth)에서 기본적으로 제공하는 권위 증명(PoA) 합의 알고리즘입니다. 자세한 내용은 [클릭 PoA 합의 알고리즘](https://github.com/ethereum/EIPs/issues/225), [puppeth](https://blog.ethereum.org/2017/04/14/geth-1-6-puppeth-master/), [클릭(clique) json 셋업 가이드](https://hackernoon.com/hands-on-creating-your-own-local-private-geth-node-beginner-friendly-3d45902cc612/)를 확인해보세요.

**[보안 및 네트워크 권한]**

## 네트워크 권한

네트워크 권한이란 어떤 노드가 특정 노드에 연결하거나 데이터를 전송하는 것을 제어하는 기능입니다. 노드를 기동할 때 --permissioned 플래그를 통해 네트워크 권한 적용이 가능하며, 권한은 각 노드 별로 관리됩니다.

--permissioned 플래그가 설정되면 노드는 *<data-dir>/permissioned-nodes.json* 파일을 찾습니다. 이 파일은 해당 노드가 연결하거나 연결을 허용하는 노드들의 enode 목록입니다. 즉, 네트워크 권한 기능을 사용하면 *permissioned-nodes.json* 파일에 나열된 노드만 네트워크에 참여 가능합니다. --permissioned 플래그를 설정하였지만 *permissioned-nodes.json* 파일에 아무 내용도 기술하지 않는 경우, 이 노드는 다른 어떤 노드와도 연결할 수 없습니다.

*permissioned-nodes.json* 파일은아래와 같이 static-nodes.json 파일(노드가 항상 연결하는 정적(static) 노드 목록)과 비슷한 패턴을 따릅니다.

[   
"enode://remoteky1@ip1:port1",  
"enode://remoteky1@ip2:port2",  
"enode://remoteky1@ip3:port3",   
]

샘플 파일: (뒷부분 노드 ID 일부 생략)

[   
"enode://6598638ac5b15ee386210156a43f565fa8c485924894e2f3a967207c047470@127.0.0.1:30300",  
]

참고: 현재 버전에서는 모든 노드에 자체의 *permissioned-nodes.json* 파일 사본이 있습니다. 이 경우, 여러 노드가 서로 다른 리모트 키(remote key) 목록을 가지면서 서로 다른 허가된 노드 목록을 가지게 되면(네트워크를 구성하려는 노드들의*permissioned-nodes.json* 파일 내용이 서로 다른 경우) 동작 상에 예상치 못한 문제가 발생할 수 있습니다. 향후에는 허가된 노드 목록을 *permissioned-nodes.json* 파일이 아니라 스마트 컨트랙트로 관리할 것입니다. 이를 통해 네트워크 연결을 확인하려는 모든 노드들은 하나의 글로벌 온 체인(on-chain) 목록을 사용하도록 할 것입니다. 추가로 예정된 개선 사항은 [제품 로드맵]을 참조하십시오.

## 엔클레이브(Enclave) 암호화 기술

엔클레이브는 xsalsa20poly1305(페이로드 컨테이너에서 사용하는 암호화 모듈) 와 curve25519xsalsa20poly1305(수신자 박스에서 사용하는 비대칭 암호화 모듈)를 사용하는 트랜잭션 매니저를 통해 송신된 페이로드를 암호화합니다. 페이로드 암호화는 각 페이로드별로 암호화를 처리하기 위한 페이로드 컨테이너와 N개의 수신자 박스를 생성하는데, 여기서 N은 트랜잭션의 privateFor 파라미터에 지정된 수신자의 수입니다.   
\*페이로드 컨테이너에는 대칭 키와 임의의 논스(nonce)로 암호화된 페이로드가 들어 있습니다.   
\*수신자 박스는 임의의 논스를 사용하는 수신자의 공개키를 위해 암호화된 페이로드 컨테이너의 마스터 키입니다. (수신자 박스는 기본적으로 PGP(Pretty Good Privacy)가 동작하는 방식과 동일하게 동작하지만 [NaCl](https://nacl.cr.yp.to/) 암호화 라이브러리를 사용한다는 점에 유의하십시오.)

현재 시스템은 퍼블릭 키 목록을 화이트리스트에 수동으로 정의하여야 하고, 키 교체가 자동화 되어있지 않지만 네트워크를 구성하는 노드들에서 여러 키를 한 번에 공고할 수 있도록 하여 기본적인 키 교체 기능을 제공하고 있습니다. 이를 원활하고 자동화할 수 있도록 돕는 툴은 [제품 로드맵]에서 확인하십시오. 또한, Quorum은 현재 PKI 시스템을 가지고 있지 않지만, 화이트리스트에 수동으로 추가될 키를 임의로 생성할 수 있도록 기능을 제공하고 있습니다. (예시: 블록체인에서 허가된 다른 노드의 레지스트리) 이 프로세스는 현재 운영자가 키 쌍을 생성한 다음 수동으로 화이트리스트에 추가하는 것입니다.

## 프라이빗 키 저장 알고리즘

프라이빗 키를 관리하는 방법은 아래와 같습니다.   
1. 패스워드 P 부여   
2. 임의의 Argon2i 논스 생성   
3. 임의의 NaCl 시크릿박스 논스 생성   
4. 32바이트 마스터키(MK)에 Argon2i (및 Argon2i nonce)를 사용하여 P 늘림   
5. 시크릿박스 논스 및 Argon2로 늘린 마스터키를 이용하여 시크릿박스의 프라이빗 키 암호화

**Part 2 셋업하기**

## [셋업 및 퀵스타트]

Quorum을 사용하기 위해서는 Quorum 노드와 Constellation/Tessera 노드를 설치하고 환경에 맞게 설정한 후 실행되어 있어야 합니다 (두 노드 모두 하단의 구축/설치 가이드를 참고하십시오). 키 생성, 제네시스 블록 및 Constellation/Tessera 구성 등 직접 Quorum을 셋업하기 위한 단계별 가이드를 곧 제공할 예정입니다. 현재로써 Quorum을 시작하기 위한 가장 좋은 방법은 Quorum Readme의 [Quickstart](https://github.com/jpmorganchase/quorum/#quickstart) 섹션에서도 설명했다시피 [Quorum Examples](https://github.com/jpmorganchase/quorum-examples)를 실행하기 위해 만들어진 Vagrant 환경을 활용하는 것입니다. Vagrant 환경은 테스트 Quorum 네트워크를 자동적으로 셋팅하여 단 몇 분 만에 개발 가능한 상태로 구성하기 때문에 Quorum을 처음 사용하고자 할 때 권장드립니다. Vagrant를 통하여 Quorum Examples 을 사용하는 대신 직접 Quorum을 셋업하고 싶다면 아래 내용을 참고해주세요. (참고: 본 문서화 작업은 진행중입니다.)

## [소스를 통한 Quorum 노드 빌드]

Github에서 소스를 복사하여 빌드해보세요:

git clone <https://github.com/jpmorganchase/quorum.git>  
cd quorum  
make all

바이너리 파일은 $REPO\_ROOT/build/bin에 생성됩니다. geth와 bootnode를 쉽게 사용할 수 있도록 해당 폴더를 개인적으로 설정한 PATH로 이동하여, 바이너리 파일을 기존 PATH(e.g. /usr/local/bin)에 복사하십시오.

~/.bashrc 혹은 ~/.bash\_aliases파일에 PATH=$PATH:/path/to/repository/build/bin와 같이 생성한 바이너리 파일의 경로를 설정하면 보다 편리하게 사용할 수 있습니다.

테스트를 실행해보세요:

make test

## [Constellation 설치하기]

[Constellation releases 페이지](https://github.com/jpmorganchase/constellation/releases)에서 여러분의 플랫폼에 맞는 패키지를 다운로드하신 후, 압축을 해제하십시오. 압축을 해제하여 나온 바이너리 파일을 설정한 PATH(e.g. /usr/local/bin)로 이동시키세요.

## [Tessera 설치하기]

[Tessera 프로젝트 페이지](https://github.com/jpmorganchase/tessera)에 나와 있는 설치 가이드를 참고해주세요.

## [고급셋업]

## Raft 합의 알고리즘의 쿼럼

1. [셋업하기] 섹션에서 설명하는대로 Quorum을 빌딩하십시오. PATH에 geth와 bootnode가 있는지 확인하십시오.
2. 새 노드의 기반이 될 작업 디렉토리를 만들고 변경하십시오.
3. 이 노드를 위해 geth를 사용하여 한 개 이상의 계정을 만들고 (--datadir new-node-1 account new) 계정 주소를 제거하십시오. 하고자 하는 것에 따라 자금이 있는 계좌가 필요할 수 있습니다.
4. genesis.json 파일을 만드십시오. [여기](https://github.com/jpmorganchase/quorum-examples/blob/master/examples/7nodes/genesis.json)에 예제가 있습니다. 이전 단계에서 생성한 계정으로 alloc 필드를 미리 입력해야 합니다.
5. 노드 키 bootnode 생성 (--genkey=nodekey)하고 datadir에 복사
6. bootnode --nodekey=new-node-1/nodekey --writeaddress 를 실행하고 표시된 아웃풋을 기록하십시오. 이것은 새로운 노드의 enode 아이디입니다.
7. static-nodes.json라는 파일을 만들고 [예제](https://github.com/jpmorganchase/quorum-examples/blob/master/examples/7nodes/permissioned-nodes.json)와 일치하도록 편집하십시오. 파일에는 enode id오devp2p 및 raft에 사용할 포트가 있는 노드에 대한 라인이 하나 있어야 합니다. 이 파일이 노드 데이터 디렉토리에 있는지 확인하십시오.
8. geth로 새로운 노드를 초기화하십시오 --datadir new-node-1 init genesis.json
9. 노드를 시작하고 다음을 통해 백그라운드로 보내십시오. PRIVATE\_CONFIG=ignore nohup geth --datadir new-node-1 --nodiscover --verbosity 5 --networkid 31337 --raft --raftport 50000 --rpc --rpcaddr 0.0.0.0 --rpcport 22000 --rpcapi admin,db,eth,debug,miner,net,shh,txpool,personal,web3,quorum,raft --emitcheckpoints --port 21000 2>>node.log &

당신의 노드는 이제 작동 가능하며 geth attach new-node-1/geth.ipc를 연결할 수 있습니다. 이 구성은 프리픽스 PRIVATE\_CONFIG=ignore에서 확인할 수 있는 것처럼 프라이버시 서포트 없이 Quorum을 시작합니다. [프라이버시 트랜잭션 매니저로 프라이버시 지원 방법] 섹션을 참조하십시오.

### 노드 추가하기

1. 앞의 가이드의 1, 2, 5, 6단계를 완료하십시오.
2. 현재 체인인 genesis.json 및 static-nodes.json.json를 회수하십시오. static-nodes.json 은 노드 data dir에 배치되어야 합니다.
3. geth로 새로운 노드를 초기화 하십시오 --datadir new-node-2 init genesis.json
4. static-nodes.json을 편집하고 구성중인 새 노드에 대한 새로운 항목을 추가하십시오 (마지막에 있어야 함)
5. 이미 실행중인 체인 노드에 연결하고 실행합니다.
6. raft.addPeer('enode://new-nodes-enode-address-from-step-6-of-the-above@127.0.0.1:21001?discport=0&raftport=50001')
7. 노드를 시작하고 다음을 통해 백그라운드로 보내십시오. PRIVATE\_CONFIG=ignore nohup geth --datadir new-node-2 --nodiscover --verbosity 5 --networkid 31337 --raft --raftport 50001 --raftjoinexisting RAFT\_ID --rpc --rpcaddr 0.0.0.0 --rpcport 22001 --rpcapi admin,db,eth,debug,miner,net,shh,txpool,personal,web3,quorum,raft --emitcheckpoints --port 21001 2>>node.log &. 여기서 RAFT\_ID는 5단계의 raft.addPeer() 커맨드에 대한 응답입니다.
8. 옵션: 다른 모든 체인 참여자들과 새로운 static-nodes.json를 공유합니다.

추가 노드는 이제 작동가능하며 이전에 설정된 노드와 동일한 체인의 일부입니다.

### 노드 제거

체인의 이미 실행중인 노드에 연결하고 raft.cluster를 실행하여 제거해야 하는 노드에 해당하는 RAFT\_ID를 가져옵니다.

raft.removePeer(RAFT\_ID)를 실행하십시오.

제거된 노드에 해당하는 geth 프로세스를 중지하십시오.

## 이스탄불 BFT 합의 알고리즘의 Quorum

[셋업하기]에서 설명한대로 Quorum을 빌딩하십시오. PATH에 geth와 bootnode가 있는지 확인하십시오.

[istanbul-tools](https://github.com/jpmorganchase/istanbul-tools) 를 설치하고 PAT에 이스탄불 바이너리를 넣으세요.

X개의 초기 유효성 검사기 노드 각각에 대한 작업 디렉토리를 만듭니다.

노드의 작업 디렉토리 중 선두에 있는(선두라고 생각하는 것) 노드로 변경하고 istanbul setup을 실행하여 X 초기 유효성 검사 노드를 위한 셋업 파일들을 생성합니다. setup --num X --nodes --quorum --save --verbose **이 명령은 X번이 아니라 한 번만 실행합니다.** 이 커맨드는 0에서 X-1까지의 번호가 지정된 디렉토리에 있는 모든 초기 유효성 검사기 노드에 대해 tatic-nodes.json, genesis.json, 및 nodekeys와 같은 몇 가지 항목을 생성합니다.

모든 초기 유효성 검사 노드의 의도된 IP 및 포트 번호를 포함하도록 static-nodes.json을 갱신하십시오. static-nodes.json에서는 각 노드마다 다른 행이 표시됩니다. 설치 가이드의 나머지 부분에서 Y행은 Y노드를 말하고 행 1은 리드 노드에 해당한다고 가정합니다.

각 노드의 작업 디렉토리에서 data라는 데이터 디렉토리를 만들고 data 내부에는 geth 디렉토리를 생성합니다.

이제 필요한 노드의 작업 디렉토리에서 --datadir data account를 실행하여 모든 노드에 대한 초기 계정을 생성합니다. 결과적으로 터미널에 나타난 퍼블릭 계정 주소를 기록해야 합니다. 필요한 만큼 이를 반복하십시오. 하고자 하는 바에 따라 자금이 있는 계좌가 필요할 수 있습니다.

초기 블록에 계정을 추가하려면 리드 노드의 작업 디렉토리에서 genesis.json파일을 편집하고 이전 단계에서 생성된 계정으로 alloc 필드를 업데이트 하십시오.

다음으로 현재 리드 노드의 작업 디렉토리에 있는 파트 4에서 생성된 파일을 다른 모든 노드에 배포해야 합니다. 이렇게 하려면 genesis.json을 모든 노드의 작업 디렉토리에 두고 각 노드의 데이터 폴더에 static-nodes.json을 놓고 X/nodekey 를 (X-1)의 data/geth 디렉토리에 위치시킵니다.

리드노드의 작업 디렉토리로 전환하고 geth --datadir data init genesis.json으로 초기화하십시오. 3단계에서 생성된 모든 작업 디렉토리 X에 대해 반복하십시오. geth init*을 실행하여 얻은 해시는 모든 노드와 일치해야 합니다.*

모든 노드를 시작하고 다음을 통해 백그라운드로 보냅니다: PRIVATE\_CONFIG=ignore nohup geth --datadir data --permissioned --nodiscover --istanbul.blockperiod 5 --syncmode full --mine --minerthreads 1 --verbosity 5 --networkid 10 --rpc --rpcaddr 0.0.0.0 --rpcport YOUR\_NODES\_RPC\_PORT\_NUMBER --rpcapi admin,db,eth,debug,miner,net,shh,txpool,personal,web3,quorum,istanbul --emitcheckpoints --port YOUR\_NODES\_PORT\_NUMBER 2>>node.log

그리고 YOUR\_NODES\_RPC\_PORT\_NUMBER 및 YOUR\_NODES\_PORT\_NUMBER을 노드의 지정된 포트 번호로 바꾸십시오. YOUR\_NODES\_PORT\_NUMBER은 파트 5에서 결정된 이 노드의 포트 번호와 일치해야 합니다.

이제 노드가 작동하며 geth attach data/geth.ipc를 연결할 있습니다. 이 구성은 프리픽스 PRIVATE\_CONFIG=ignore에서 확인할 수 있는 것처럼 프라이버시 지원이 없는 쿼럼을 시작합니다. 하단의 [프라이버시 트랜잭션 매니저를 이용한 프라이버시 지원 방법|기초부터 시작하기#프라이버시-트랜잭션-매니저 추가하기] 섹션을 참조하십시오.

이스탄불 툴은 X개의 노드를 생성하는데 사용될 수 있으며 자세한 정보는 [docs](https://github.com/jpmorganchase/istanbul-tools)에서 확인할 수 있습니다.

### 검증자(Validator) 추가하기

추가해야 할 새로운 노드의 작업 디렉토리를 만듭니다.

새 노드의 작업 디렉토리로 변경하고 istanbul setup --num 1 --verbose --quorum --save를 실행합니다. 그러면 Address, NodeInfo 및 genesis.json을 포함하여 검증자의 세부 정보가 생성됩니다. 검증자의 주소를 복사하고 현재 검증자의 절반 이상에서 istanbul.propose(<address>, true)를 실행합니다.

istanbul.getValidators()를 실행하여 새로운 검증자가 검증자 목록에 추가되었는지 확인합니다.

[셋업하기] 섹션에서 설명한대로 Quorum을 빌딩하십시오. PATH에 geth가 있는지 확인하십시오. 기존 체인에서 static-nodes.json 및 genesis.json을 복사하십시오. static-nodes.json은 새로운 노드 data dir에 배치되어야 합니다.

static-nodes.json을 편집하고 새로운 검증자 노드 정보를 파일의 끝에 추가하십시오. 새로운 검증자 노드 정보는 2단계에서 실행된 istanbul setup --num 1 --verbose --quorum --save 커맨드의 결과물에서 얻을 수 있습니다. 노드 정보의 IP주소와 포트를 사용하고자 하는 검증자와 포트의 IP 주소와 일치하도록 업데이트하십시오.

istanbul setup 커맨드로 생성된 노드키를 작업 디렉토리의 geth 디렉토리에 복사하십시오.

geth --datadir new-node-1 account new를 사용하여 이 노드에 대해 하나 이상의 계정을 생성하고 계정 주소를 제거하십시오.

geth --datadir new-node-1 init genesis.json를 사용하여 새로운 노드를 초기화하십시오.

노드를 시작하고 PRIVATE\_CONFIG=ignore nohup geth --datadir data --permissioned --nodiscover --istanbul.blockperiod 5 --syncmode full --mine --minerthreads 1 --verbosity 5 --networkid 10 --rpc --rpcaddr 0.0.0.0 --rpcport YOUR\_NODES\_RPC\_PORT\_NUMBER --rpcapi admin,db,eth,debug,miner,net,shh,txpool,personal,web3,quorum,istanbul --emitcheckpoints --port YOUR\_NODES\_PORT\_NUMBER 2>>node.log를 이용하여 백그라운드로 보내십시오. 그리고 YOUR\_NODES\_RPC\_PORT\_NUMBER 및 YOUR\_NODES\_PORT\_NUMBER를 지정된 포트 번호로 바꾸십시오. YOUR\_NODES\_PORT\_NUMBER은 파트 7에서 결정된 이 노드의 포트 번호와 일치해야 합니다.

### 검증자 제거하기

실행중인 검증자에 연결하고 istanbul.getValidators()를 실행하고 제거해야하는 검증자의 주소를 확인하십시오.

현 검증자의 절반 이상에서 제거해야 하는 검증자의 주소를 전달하여 istanbul.propose(<address>, false)를 실행하십시오.

istanbul.getValidators()를 실행하여 검증자가 제거되었는지 확인하십시오.

제거된 검증자에 해당하는 geth 프로세스를 중지하십시오.

### 비검증자 노드 추가하기

추가해야 할 새로운 노드의 작업 디렉토리를 만드십시오.

새로운 노드의 작업 디렉토리로 변경하고 istanbul setup --num 1 --verbose --quorum --save를 실행하십시오. 그러면 Address, NodeInfo and genesis.json을 포함한 노드의 세부정보가 생성됩니다.

[셋업하기] 섹션에서 설명한대로 Quorum을 빌딩하십시오. PATH에 geth가 있는지 확인하십시오.

기존 체인에서 static-nodes.json 및 genesis.json을 복사하십시오. static-nodes.json은 새로운 노드 data dir에 배치되어야 합니다.

static-nodes.json을 편집하고 새로운 노드 정보를 파일의 끝에 추가하십시오. 새로운 노드 정보는 2단계에서 실행된 istanbul setup --num 1 --verbose --quorum --save 커맨드의 결과물에서 얻을 수 있습니다. 노드 정보의 IP주소와 포트를 사용하고자 하는 검증자와 포트의 IP 주소와 일치하도록 업데이트하십시오.

istanbul setup 커맨드로 생성된 노드키를 작업 디렉토리의 geth 디렉토리에 복사하십시오.

geth --datadir new-node-1 account new를 사용하여 이 노드에 대해 하나 이상의 계정을 생성하고 계정 주소를 제거하십시오.

geth --datadir new-node-1 init genesis.json를 사용하여 새로운 노드를 초기화하십시오.

노드를 시작하고 PRIVATE\_CONFIG=ignore nohup geth --datadir data --permissioned --nodiscover --istanbul.blockperiod 5 --syncmode full --verbosity 5 --networkid 10 --rpc --rpcaddr 0.0.0.0 --rpcport YOUR\_NODES\_RPC\_PORT\_NUMBER --rpcapi admin,db,eth,debug,net,shh,txpool,personal,web3,quorum,istanbul --emitcheckpoints --port YOUR\_NODES\_PORT\_NUMBER 2>>node.log를 이용하여 백그라운드로 보내십시오. 그리고 YOUR\_NODES\_RPC\_PORT\_NUMBER 및 YOUR\_NODES\_PORT\_NUMBER를 지정된 포트 번호로 바꾸십시오. YOUR\_NODES\_PORT\_NUMBER은 5단계에서 결정된 이 노드의 포트 번호와 일치해야 합니다.

### 비검증자 노드 제거하기

제거해야 할 노드에 해당하는 geth 프로세스를 중지하십시오.

## Clique 합의 알고리즘의 Quorum

# 프라이버시 트랜잭션 매니저 추가하기

## Tessera

Quorum을 빌딩하고 [셋업하기] 섹션에서 설명하는 대로 [Tessera](https://github.com/jpmorganchase/tessera/releases)를 설치하십시오. PATH에 geth와 bootnode가 포함되어 있는지 확인하십시오. tessera.jar 릴리스 파일의 위치를 알고 있어야 합니다.

java -jar /path-to-tessera/tessera.jar -keygen -filename new-node-1를 사용하여 새 키들을 생성하십시오.

새롭게 생성된 키가 참조된 [샘플1](https://github.com/jpmorganchase/tessera/wiki/Sample-configuration)과 [샘플2](https://github.com/jpmorganchase/quorum-examples/blob/master/examples/7nodes/tessera-init.sh)를 참조하여 새로운 구성 파일을 만드십시오. 파일 이름을 기록하거나 이 예제에서와 같이 파일 이름을 config.json로 하십시오.

tessera 노드를 시작하고 java -jar /path-to-tessera/tessera.jar -configfile config.json >> tessera.log 2>&1 &를 통해 백그라운드로 보내십시오.

노드를 시작하고 PRIVATE\_CONFIG=tm.ipc nohup geth --datadir new-node-1 --nodiscover --verbosity 5 --networkid 31337 --raft --raftport 50000 --rpc --rpcaddr 0.0.0.0 --rpcport 22000 --rpcapi admin,db,eth,debug,miner,net,shh,txpool,personal,web3,quorum,raft --emitcheckpoints --port 21000 2>>node.log &를 통해 백그라운드로 보내십시오.

당신의 노드는 이제 작동 가능하며 attach new-node-1/geth.ipc를 통해 연결할 수 있습니다. Tessera IPC 브릿지는 일반적으로 프리픽스 PRIVATE\_CONFIG=tm.ipc에 나와 있는 것처럼 tm.ipc라는 이름으로 config.json에 정의된 파일 이름 위에 있습니다. 이제 노드가 프라이빗 트랜잭션을 주고 받을 수 있고 퍼블릭 노드키는 new-node-1.pub 파일에 있게 됩니다. Tessera는 구성에 있어서 많은 유연성을 제공합니다. [Tessera wiki](https://github.com/jpmorganchase/tessera/wiki)에서 완벽한 최신 구성 옵션을 확인하십시오.

## Constellation

Quorum을 빌딩하고 [셋업하기] 섹션에서 설명한 바와 같이 [Constellation](https://github.com/jpmorganchase/constellation/releases)을 설치하십시오. PATH에 geth, bootnode 및 constellation 노드 바이너리가 있는지 확인하십시오.

constellation-node --generatekeys=new-node-1로 새로운 키들을 생성하십시오.

constellation 노드를 시작하고 다음을 통해 백그라운드로 보내십시오: constellation-node --url=https://127.0.0.1:9001/ --port=9001 --workdir=. --socket=tm.ipc --publickeys=new-node-1.pub --privatekeys=new-node-1.key --othernodes=https://127.0.0.1:9001/ >> constellation.log 2>&1 &

Start your node and send it into background with PRIVATE\_CONFIG=tm.ipc nohup geth --datadir new-node-1 --nodiscover --verbosity 5 --networkid 31337 --raft --raftport 50000 --rpc --rpcaddr 0.0.0.0 --rpcport 22000 --rpcapi admin,db,eth,debug,miner,net,shh,txpool,personal,web3,quorum,raft --emitcheckpoints --port 21000 2>>node.log &

당신의 노드는 이제 작동 가능하며 attach new-node-1/geth.ipc를 통해 연결할 수 있습니다. Constellation IPC bridge는 당신의 설정에 정의된 파일 이름 위에 있을 것입니다. 위의 3번 단계에서 --socket=file-name.ipc 옵션을 보십시오. 이제 노드가 프라이빗 트랜잭션을 주고 받을 수 있고 퍼블릭 노드키는 new-node-1.pub 파일에 있게 됩니다.

# 허가형 구성 사용하기

Quorum은 사용자 지정 화이트리스트를 기반으로 허가형 시스템을 함께 제공합니다. 자세한 설명은 [Network Permissioning](https://github.com/jpmorganchase/quorum/wiki/Security#network-permissioning)에서 확인할 수 있습니다.

**Part 3 Quorum 사용하기**

## [스마트 컨트랙트 개발]

Quorum은 스마트 컨트랙트 작성 시 표준 [Solidity](https://solidity.readthedocs.io/en/develop/)를 사용하고 있으며, 일반적으로 이더리움에서 스마트 컨트랙트를 개발할 때와 동일한 방법으로 개발하시면 됩니다. 스마트 컨트랙트는 퍼블릭(즉, Quorum 네트워크의 모든 참여자가 볼 수 있고 실행 가능한 형태) 혹은 지정한 참여자들에게만 공개하는 프라이빗한 형태로 사용할 수 있습니다. 유의하실 점은 Quorum은 새로운 컨트랙트 형태를 도입하지 않는다는 점입니다. 대신, [트랜잭션 처리 및 프라이버시] 섹션에서 설명한 것과 같이 퍼블릭과 프라이빗 컨트랙트는 개념적으로만 이루어집니다.

### 퍼블릭 트랜잭션/컨트랙트 생성

Quorum 네트워크 상에서 모든 참여자들이 트랜잭션/스마트 컨트랙트를 조회하거나 실행 가능하게 하기 위해서는 네트워크에 이더리움 트랜잭션을 전송하기만 하면 됩니다. (트랜잭션이 컨트랙트를 생성하길 원한다면 to 파라미터는 비워두세요)

표준 이더리움 API에서 수정된 sendTransaction API에 관한 자세한 내용은 [Quorum API](https://github.com/jpmorganchase/quorum/blob/master/docs/api.md) 페이지에서 확인할 수 있습니다.

참고: Quorum 컨트랙트 생성 시 주요 참고사항들은 하단의 ‘Quorum 컨트랙트 디자인 시 고려사항’ 섹션을 참고해주세요.

### 프라이빗 트랜잭션/컨트랙트 생성

Quorum 네트워크 상에서 일부 참여자들만 트랜잭션/스마트 컨트랙트를 실행할 수 있도록 하고 싶은 경우, 표준 이더리움 트랜잭션에 privateFor 파라미터만 추가하시면 됩니다. privateFor 파라미터에는 트랜잭션이나 컨트랙트를 실행할 수 있는 참여자의 퍼블릭 키 목록을 설정하시면 됩니다

JSON 메시지 예시:

'{"jsonrpc":"2.0","method":"eth\_sendTransaction","params":[{"from": $FROM\_AC, "to": $TO\_AC, "data": $CODEHASH, "privateFor": ["$PUBKEY1,PUBKEY2"]}],"id":$ID}'

표준 이더리움 API에서 수정된 sendTransaction API에 관한 자세한 내용은 [Quorum API](https://github.com/jpmorganchase/quorum/blob/master/docs/api.md) 페이지에서 확인할 수 있습니다.

참고: Quorum 컨트랙트 생성 시 주요 참고사항들은 하단의 ‘Quorum 컨트랙트 디자인 고려사항’ 섹션을 참고해주세요.

### Quorum 컨트랙트 디자인 시 고려사항

1. *프라이빗 컨트랙트는 퍼블릭 컨트랙트를 업데이트 할 수 없습니다.* 이는 모든 참여자들이 프라이빗 컨트랙트를 실행할 수는 없는 상황에서 프라이빗 컨트랙트가 퍼블릭 컨트랙트를 업데이트 할 수 있다면 각각의 참여자들이 제각각 다른 상태의 퍼블릭 컨트랙트를 가지게 되기 때문입니다.
2. *퍼블릭* *컨트랙트를 프라이빗 컨트랙트로 변경할 수 없습니다.* 퍼블릭 컨트랙트를 프라이빗으로 변경해야 한다면 블록체인에서 해당 컨트랙트를 삭제하고 새로운 프라이빗 컨트랙트를 생성해야 합니다.

## [허가형 네트워크 셋업]

네트워크 권한은 개별 노드 기동 시, --permissioned 플래그를 통해 설정 가능합니다. 해당 플래그가 추가되면 노드는 *<data-dir>* 폴더의 *permissioned-nodes.json* 파일을 조회합니다:

*permissioned-nodes.json* 파일은 해당 노드가 연결하거나 연결을 허용하는 노드 지정자 목록(enode://remotekey@ip:port)입니다.

--permissioned 플래그를 설정하였는데 *permissioned-nodes.json* 파일이 비어있거나 노드의 *<data-dir>* 폴더에 없다면, 노드가 기동은 하지만 다른 어떤 노드와도 연결할 수 없습니다. 연결을 시도하거나 요청받는 경우 발생하는 오류들은 로그 파일에 기록됩니다.

--permissioned 플래그는 geth 옵션의 기타 리스트에서 확인 가능합니다:

$ geth --help  
 ( truncated output )  
 MISCELLANEOUS OPTIONS:  
 --permissioned If enabled, the node will allow only a defined list of nodes to connect

참고: 네트워크 접속 허용 권한을 개별 노드에게 위임하는 것은 여러 문제들을 야기할 수 있습니다. 이를 방지하기 위해 권한 설정을 스마트 컨트랙트 기반 모델로 변경하는 것에 대하여 [제품 로드맵]에서 확인 가능합니다.

### 네트워크 권한 설정(Permissioning) 과정:

1. *<data-dir>* 폴더에 *permissioned-nodes.json* 파일을 생성하세요. JSON 포맷인지 꼭 확인하십시오.
2. 해당 파일에 셋업 과정에서 설정(static-nodes.json) 하거나 연결하고자 하는 Quorum 네트워크 상 노드들의 enode 아이디들을 기술하세요.
3. 노드를기동 시, --permissioned 커맨드라인 플래그를 설정하세요.

*permissioned-nodes.json* 파일의 형식은 다음과 같습니다:

[  
"enode://remoteky1@ip1:port1",  
"enode://remoteky1@ip2:port2",  
"enode://remoteky1@ip3:port3",   
]

예시 (가독성을 위해 노드 아이디의 끝부분 생략):

[  
"enode://8475a01f22a1f48116dc1f0d22ecaaaf77e[::]:30301", "enode://b5660501f496e60e59ded734a889c97b7da[::]:30302", "enode://54bd7ff4bd971fb80493cf4706455395917[::]:30303"  
]

위 예시는 해당 노드가 이 화이트리스트 상의 3개의 노드로부터 들어오거나 나가는 연결만 수용할 수 있음을 보여줍니다.

### 새로운 노드 추가하기:

*permissioned-nodes.json*에 더해지는 추가사항들은 노드로 들어오거나 나가는 요청이 생성될 때 서버가 즉각적으로 반영합니다. *permissioned-nodes.json* 파일의 변경을 반영하기 위해 노드를 다시 시작할 필요가 없습니다.

### 기존 노드 삭제하기:

*permissioned-nodes.json* 파일에서 기존에 연결되어 있던 노드를 제거해도 즉각적으로 반영되지는 않습니다. 하지만 어떤 이유로든 연결이 끊어지고, 해당 노드 아이디로부터 연결 요청이 들어오면 요청은 반려될 것입니다.

참고: 노드 제거의 즉각적 반영은 [제품 로드맵]에서 확인 가능합니다.

## [Quorum API]

자세한 내용은 [Quorum API](https://github.com/jpmorganchase/quorum/blob/master/docs/api.md) 페이지를 참고해주세요.

## [ZSL 사용하기]

J.P. Morgan과 Zcash 팀은 Quorum에 ZSL(Zcash Security Layer) 적용에 대한 개념 증명 (PoC, Proof of Concept)을 위해 파트너십을 맺었습니다. 이는 ZSL 기반의 퍼블릭 스마트 컨트랙트인 ‘z-contracts’를 이용하여 디지털 자산(z-tokens )을 발행할 수 있도록 합니다. ‘Z-tokens’는 퍼블릭 네트워크에는 노출시키지 않으면서 트랜잭션의 프라이버시를 보장할 수 있습니다. 보호한 트랜잭션이 처리되었는지는 프라이빗 컨트랙트를 통해 확인할 수 있습니다. 프라이빗 컨트랙트의 상태가 퍼블릭 ‘z-contracts’를 이용해 보호된 트랜잭션에 따라 변경되었다면 정상적으로 처리되었다는 것을 의미합니다.

이러한 Constellation과 Tessera의 프라이빗 컨트랙트와 ZSL의 ‘z-contracts’ 조합은 완전한 프라이버시와 비밀을 보장하는 동시에 프라이빗 컨트랙트의 역할을 ‘z-tokens’의 트랜잭션 보호 기능을 통해 해결할 수 있도록 합니다.

더 많은 정보는 wiki의 [[ZSL](https://github.com/jpmorganchase/quorum/wiki/ZSL)] 페이지에서 확인하세요.

## 네트워크와 체인 아이디(Chain ID)

초기 이더리움 네트워크는 네트워크 아이디(Network ID)를 사용하였으나 [EIP-155](https://github.com/ethereum/EIPs/blob/master/EIPS/eip-155.md)이후에는 체인 아이디(Chain ID)를 통해 운영하고 있습니다.

EIP-155 이전에 ‘네트워크 아이디’와 ‘체인 아이디’는 비슷한 의미로 혼용하여 사용하였으나, 이후에는 서로 다른 의미를 갖게 되었습니다.

네트워크 아이디는 피어(peer)가 운영하고 있는 체인이 아닌 피어의 속성입니다. 네트워크 아이디는 --networkid <id> 커맨드 라인을 통해 전달할 수 있습니다. 네트워크 아이디는 서로 다른 네트워크 아이디로 운영하고 있는 피어들을 분리하기 위해 사용됨으로, 서로 다른 네트워크 아이디를 가지는 노드 간의 동기화는 불가능합니다. 하지만, 네트워크 아이디 변경을 통한 피어간 분리는 Quorum의 --permissioned 플래그에 비해 보안성이 낮기 때문에 단순한 분리를 위해서만 사용됩니다.

체인 아이디는 노드에 의해 운영되는 체인의 속성으로 트랜잭션의 리플레이 방지(replay protection)를 위해 사용됩니다. 이는 EIP-155 이전, 트랜잭션이 서명되어 있었다면 특정 체인에서 다른 체인으로 트랜잭션을 복사하여 실행하는 리플레이 공격(replay attack)을 방지하기 위함입니다.

체인 아이디를 설정하는 것은 트랜잭션의 파라미터들 중 하나를 바꾸게 되고 이를 V 파라미터(V parameter)라고 합니다. EIP에서 설명하듯이, V 파라미터는 ‘2\*ChainID + 35/36’로 설정됩니다. 이더리움 재단의 메인넷은 체인 아이디 1을 가지고 있으며, 이는 모든 트랜잭션이 37 혹은 38의 값을 가지고 있다는 것을 의미합니다.

genesis 구성 파일의 config 섹션에 설정된 체인 아이디는 블록 번호가 eip155Block에 설정한 값보다 높을 때만 사용됩니다. 해당 예시는 [quorum-examples genesis files](https://github.com/jpmorganchase/quorum-examples/blob/master/examples/7nodes/genesis.json) 에서 확인할 수 있습니다. 체인의 블록 번호가 eip155Block 번호보다 낮은 상태이면 필요에 따라 얼마든지 변경이 가능하며, 변경 후 geth init 를 재실행하면 됩니다. 이 경우 현재 동기화 프로세스나 저장된 블록을 삭제하거나 수정하지 않습니다!

Quorum에서 트랜잭션의 V 파라미터가 37이나 38로 설정된 경우 프라이빗 트랜잭션으로 간주하기 때문에 체인 아이디 1을 가진 네트워크와 충돌합니다. 이러한 이유로 Quorum은 체인 아이디를 사용하지 않으며, 2.1.0 버전 이후부터는 체인 아이디를 설정하여 노드를 기동하는 경우 실행 즉각 중단될 것입니다. 2.1.0 이전 버전으로 실행하고 있다면 EIP-155 서명이 사용되지 않았기 때문에 체인 아이디 1설정이 가능합니다. 현재 업데이트된 버전을 이용하기 전에 위 사항을 고려하여 genesis 파일을 수정하시고 geth init 를 실행해야 할 것입니다.

## 구성 가능한 트랜잭션 크기:

Quorum은 제네시스 블록을 통해 블록 생성자(operator) 들이 블록에 수용하는 트랜잭션의 최대 사이즈를 늘릴 수 있도록 합니다. 현재 기본으로 설정된 Quorum의 트랜잭션 사이즈는 이더리움의 트랜잭션 사이즈인 32kb에서 64kb로 늘어난 상태이고, 옵션 설정을 통해서는 128kb까지 증가시킬 수 있습니다. 이러한 옵션 구성을 위해서는 genesis 파일의 config 섹션에 다음과 같이 txnSizeLimit을 추가하면 됩니다.   
 "config": {  
 "chainId": 10,  
 "isQuorum":true  
 ...  
 "txnSizeLimit": 128  
}

## [ZSL]

# ZSL 개념증명 (POC)

* [개요](https://docs.google.com/document/d/16veGjVbwgWoyqfsoGYfM844RmOwXbmZl/edit#bookmark=id.gjdgxs)
* [구현](https://docs.google.com/document/d/16veGjVbwgWoyqfsoGYfM844RmOwXbmZl/edit#bookmark=id.30j0zll)
* [주식 거래 적용 예시](https://docs.google.com/document/d/16veGjVbwgWoyqfsoGYfM844RmOwXbmZl/edit#bookmark=id.1fob9te)
* [프로토콜](https://docs.google.com/document/d/16veGjVbwgWoyqfsoGYfM844RmOwXbmZl/edit#bookmark=id.3znysh7)

## 개요

Quorum은 퍼블릭 컨트랙트 (표준 이더리움 방식으로 실행되며 분산 원장의 모든 참여자가 볼 수 있음)와 프라이빗 컨트랙트 (Tessera를 사용하는 프라이빗 컨트랙트의 당사자 간에 공유되지만 다른 사람이 읽을 수는 없음)를 모두 지원합니다. 이 접근법은 프라이빗 컨트랙트 당사자의 프라이버시를 보존하고 프라이빗 컨트랙트의 비즈니스 논리를 기밀로 유지해 줍니다. 그러나 중요한 제한 사항은 프라이빗 컨트랙트 내에서 교환되는 디지털 자산에 대한 이중 지출 방지를 지원하지 않는다는 것입니다.

ZSL (0지식 보안 레이어, zero-knowledge security layer)은 zk-SNARKS를 활용하여 발신자, 수신자 또는 자산 수량에 대한 정보를 공개하지 않고 분산원장에서 디지털 자산을 전송할 수 있도록 Zcash 팀이 설계한 프로토콜입니다.

J.P. Morgan과 Zcash팀은 ZSL이 도입된 퍼블릭 스마트 컨트랙트(z-contracts)을 사용하여 디지털 자산을 발행할 수 있도록 Quorum에 ZSL의 개념증명(POC)을 구현하기 위해 파트너십을 체결했습니다. 이렇나 디지털 자산을 ‘z-토큰’이라고 합니다. Z-토큰은 공개되지 않고 프라이빗하게 거래될 수 있습니다. 차폐 거래가 실행된 증거는 프라이빗 컨트랙트에 나타나며 퍼블릭 z-컨트랙트를 사용하여 실행된 차폐 거래에 따라 프라이빗 컨트랙트는 자신의 상태를 업데이트합니다.

Tessera의 프라이빗 컨트랙트와 ZSL의 z-컨트랙트의 결합은 완전한 개인정보 및 기밀을 유지하면서도 프라이빗 컨트랙트로부터 발생하는 의무를 해결할 수 있게 합니다.

더욱 많은 정보는 [POC Technical Design Document](https://github.com/jpmorganchase/zsl-q/blob/master/docs/ZSL-Quorum-POC_TDD_v1.3pub.pdf)를 참조해 주십시오.

## 구현

ZSL의 개념증명은 다음과 같이 구현됩니다. \* ZSL 특정 코드는 [zsl-q](https://github.com/jpmorganchase/zsl-q) 저장소에 있습니다. \* Quorum의 통합은 Quorum 저장소(the [zsl\_geth1.5](https://github.com/jpmorganchase/quorum/tree/zsl_geth1.5) branch)의 별도 브랜치로 구현되었습니다. \* Quorum 예시 저장소를 위한 ZSL 특정 브랜치 또한 존재하며 (the [zsl\_geth1.5](https://github.com/jpmorganchase/quorum-examples/tree/zsl_geth1.5) branch) \* [zsl-q-params](https://github.com/jpmorganchase/zsl-q-params) 저장소는 zk-SNARK 증명을 생성하고 검증하는 데 필요한 공유 매개 변수를 포함합니다.

ZSL이 도입된 Quorum을 설치하는 방법에 대한 상세한 가이드는 [zsl-q README](https://github.com/jpmorganchase/zsl-q/blob/master/README.md)를 참조하십시오.

이 개념증명(POC)은 ZSL이 Quorum을 보완할 수 있는 방법을 보여주고 다양한 사례를 실험하고 탐색할 수 있는 플랫폼을 제공하기 위한 것입니다. 신속한 프로토타이핑이 가능하도록 Zerocash 프로토콜의 간소화된 버전을 구현합니다. 프로토콜에 대한 공식적인 보안 증거는 없으며 증명 검증을 위한 예외 처리가 구현되지 않았고 소프트웨어는 엄격한 테스트는 거치지 않았기 때문에 **생산 준비 완료 단계라고 간주되어서는 안됩니다.**

대체로, Quorum ZSL은 가상 자금이 암호화되어 난독화된 ‘노트’에 묶여들어 갈 수 있도록 하는 컨트랙트를 제공합니다. 각 노트는 가치 저장소를 나타내며 비밀 지출 키로만 잠금 해제하거나 ‘상환’할 수 있습니다. 예를 들어, 프라이빗 전송을 수행하기 위해 Alice는 값을 노트에 묶어서 비공개 오프 체인 채널을 통해 노트의 비밀 키를 Bob에게 전송할 수 있습니다. 그런 다음 Bob은 이 노트를 온체인에서 사용하면서 그 과정에서 Alice와 자신 사이의 퍼블릭 링크를 공개하지 않을 수 있습니다. 이전 버전에서는 이더리움과 노트 서명을 연결하지 못한 것이 일종의 ‘전면 실행’ 공격을 가능하게 했습니다. 이 점은 PR [#587](<https://github.com/jpmorganchase/quorum/pull/587>)에 의해 수정되었습니다.

## 주식 거래에서의 활용 사례

다음 예시는 Alice가 Bob으로부터 ACME 주식을 구매하는 단순한 주식거래를 통해 ZSL이 도입된 Quorum의 특정 사용 사례를 보여줍니다. 개념증명(POC)에는 이 사례를 구현하는 데모가 포함되어 있습니다. 이를 실행하기 위한 가이드는 [여기](https://github.com/jpmorganchase/zsl-q/blob/master/README.md#example-2---private-contract-trade)를 참조하십시오. .

[Quorum 주식 거래 활용 사례 다이어그램]Quorum 주식 거래 활용 사례 다이어그램

### 사례의 전제:

* Z-contracts는 US달러(USD z-컨트랙트)와 ACME주식 거래를 위해 만들어졌습니다.
* Z-토큰은 관련 발급기관에 의해 두 컨트랙트 모두에 발행이 된 다음 차폐되어 Alice와 Bob에게 양도되었습니다.
* Alice는 USD z-토큰을 소유하고 있고 Bob은 ACME z-토큰을 소유하고 있습니다. 둘 모두의 소유 상태는 차폐되어 있습니다. (제 3자는 누가 어떤 것을 소유하고 있는지 알지 못합니다.)

### 사용자 스토리:

1. **Tessera를 사용하여 Alice와 Bob 간에 프라이빗 컨트랙트가 설정됩니다.** 
   1. 프라이빗 컨트랙트는 주식 거래 상에서의 두 특정 당사자인 Alice(ACME 주식을 매수하는 사람)와 Bob(ACME 주식을 매도하는 사람), 거래 시 ACME 주식의 특정 수량과 그 USD환산 가격을 지정합니다.
   2. 프라이빗 컨트랙트는 USD 및 ACME z-컨트랙트, 관련 퍼블릭키 및 당사자의 페이먼트 주소를 포함합니다.
   3. 일방이 계약을 초기화합니다. (이는 매수/매도와 동일한 개념입니다.) 어떤 당사자가 해당 작업을 수행하는지는 중요하지 않습니다. 본 예시에서는 Alice로 하겠습니다.
   4. 초기화 된 후 계약 상태는 “매수” (Bob이 초기화하였다면 “매도”) 입니다.
2. **상대방은 프라이빗 컨트랙트에 조건의 수락을 나타내는 트랜잭션을 보냅니다.**
3. 본 예시에서는 Bob이 Alice의 매수를 수락하는 것으로 하겠습니다.
4. 이 시점에서 거래가 “완료”됩니다. (즉, 조건이 합의되고 양 당사자가 거래에 참여하게 됩니다.) 이제 남은 것은 조정 단계입니다. 달러가 우선 지불되어야 한다고 가정해보겠습니다.
5. 컨트랙트 상태: 완료
6. **프라이빗 컨트랙트가 지불을 지시합니다.**
7. 컨트랙트 상대가 “완료”로 업데이트 되면 컨트랙트는 구매자(여기서 Alice)의 클라이언트에게 상응하는 금액의 USD를 판매자(Bob)에게 지불하도록 지시합니다.
8. Alice의 클라이언트는 해당 지시를 받아 대기상태로 놓고 차폐 지불을 지시합니다.
9. **구매자는 판매자에게 USD를 지불합니다.**
10. Alice는 필요한 zk-SNARK 증명을 생성하고 이를 USD z-컨트랙트로 내보내 Bob의 USD 지불주소로 거래에 상응하는 금액의 USD z-토큰을 지불합니다.
11. 차폐 거래가 발생하여 Bob만이 사용할 수 있는 z-컨트랙트 내역이 생성됩니다. (즉, Bob의 z-토큰 잔액이 증가합니다.)
12. 이에 따라 Alice의 USD z-토큰 잔액은 감소합니다.
13. **구매자는 프라이빗 컨트랙트에 지불증거를 제공합니다. ,**
14. Alice는 USD 지불의 결과에 대한 노트를 포함한 트랜잭션을 프라이빗 컨트랙트로 보냅니다.
15. 이는 Bob에게도 노트를 전송하여 Bob 또한 이를 사용할 수 있습니다.
16. **해당 프라이빗 컨트랙트는 지불을 확인해줍니다.**
17. 프라이빗 컨트랙트는 해당 지불이 유효한지 확인하기 위해 Alice가 제공한 노트를 이용하여 USD z-컨트랙트에 지속적으로 함수를 호출합니다.
18. z-컨트랙트는 바이너리 방식으로 응답하여 노트 커미트먼트가 z-컨트랙트의 노트 누적기에 있는지 여부(이 경우 차폐된 지불이 유효함)를 나타냅니다.
19. 유효한 경우, 컨트랙트의 상태는 “지불 완료”로 업데이트되고...
20. **프라이빗 컨트랙트는 “딜리버리”를 지시합니다.**
21. 프라이빗 컨트랙트는 판매자(여기서 Bob)의 클라이언트에게 구매자에게 상응하는 양의 ACME 주식을 구매자에게 양도도록 지시를 내립니다.
22. Bob의 클라이언트는 해당 지시를 받아 대기상태로 놓고 지불을 요청합니다.
23. **판매자는 ACME 주식을 구매자에게 양도합니다.**
24. Bob은 필요한 zk-SNARK 증명을 생성하고 이를 ACME z-컨트랙트로 전송하여 상응하는 ACME z-토큰을 Alice의 지불 주소로 보냅니다.
25. 차폐 거래가 이루어지고 Alice만 사용 가능한 노트 아웃풋을 생성합니다. (즉, Alice의 ACME z-토큰 잔액이 증가합니다.)
26. 이에 따라 Bob의 ACME z-토큰 잔액은 감소합니다.
27. **판매자는 프라이빗 컨트랙트에 해당 전송에 대한 증거를 제출합니다.**
28. Bob은 ACME 전송에 대한 아웃풋 노트를 포함한 트랜잭션을 프라이빗 컨트랙트에 보냅니다.
29. 이는 Alice에게도 노트를 전송하여 이를 “사용”할 수 있도록 합니다. (즉, 해당 토큰들을 다른 사람들에게 전송할 수 있게 됩니다.)
30. **프라이빗 컨트랙트가 딜리버리를 확인합니다.**

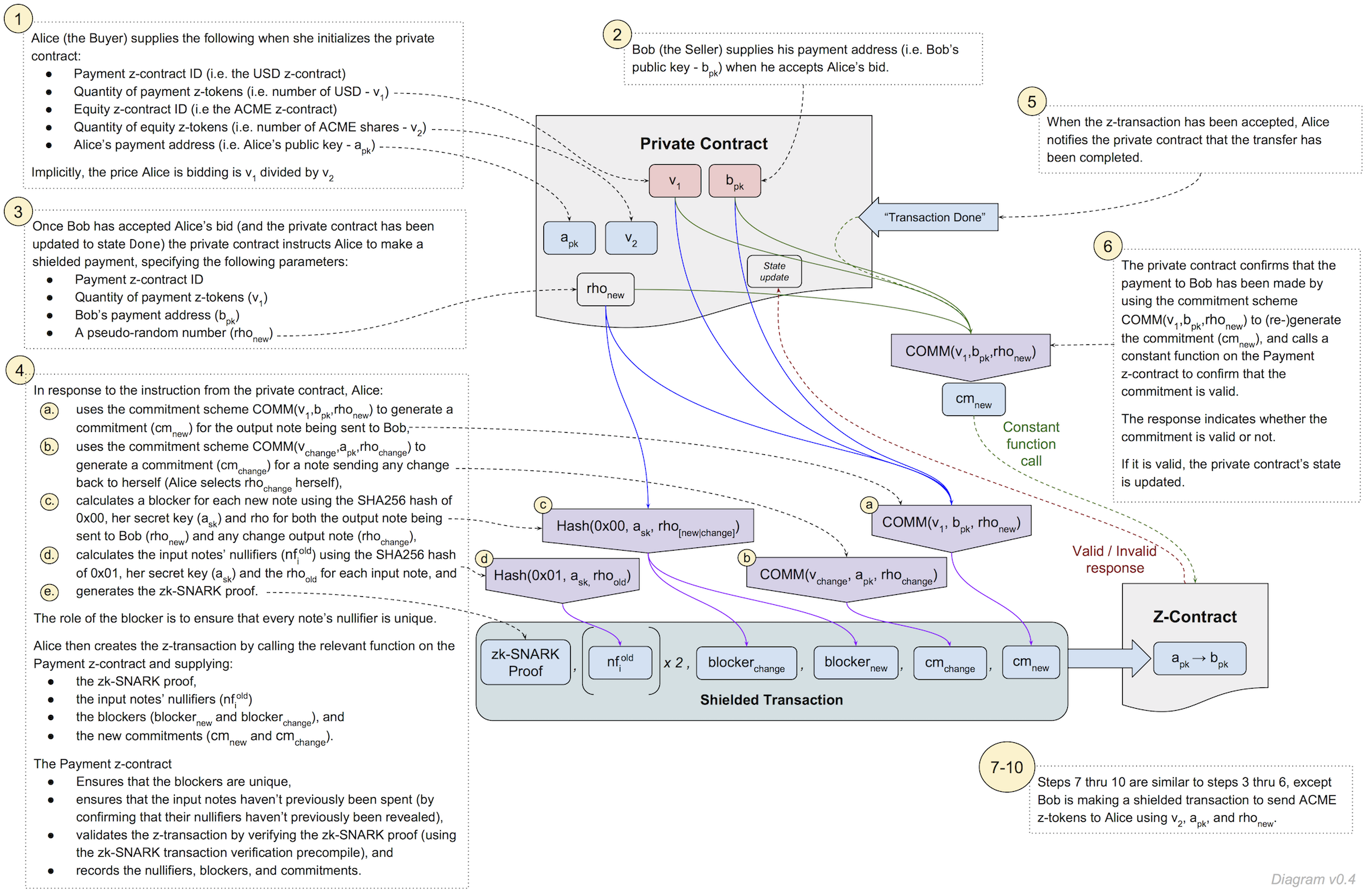
프라이빗 컨트랙트는 Bob이 제공한 노트를 이용하여 ACME z-컨트랙트를 호출(상수 함수를 사용)하며 전송이 유효한지 확인합니다.

유효한 경우, 컨트랙트 상태가 “정산됨(Settled)”으로 업데이트됩니다.

Alice가 USD z-토큰을 5번째 단계에서와 같이 Bob에게 전송을 완료하면 Bob은 해당 토큰들을 제 3자(에로, Carol)에게 전송할 수 있게 됩니다. \*Carol은 토큰의 출처는 알 수 없습니다. (Bob이 Alice에게 해당 토큰을 받았음을 알지 못합니다.) \*Alice는 Bob이 제 3자에게 언제 해당 토큰을 전송하는지, 누구에게 전송하는지 알 수 없습니다. Alice는 (트랜잭션이 Alice가 접근권한을 가지고 있는 메인 Quorum 체인 상의 z-컨트랙트에 작성되었기 때문에) 거래가 일어났다는 사실은 볼 수 있지만 보낸 사람, 받는 사람 및 전송된 토큰의 수량은 알 수 없습니다. \*앞의 내용은 Alice가 Bob으로부터 취득한 ACME z-토큰의 사례에도 적용됩니다.

### 프로토콜

아래의 다이어그램은 위의 예시에서 암호화 프로토콜이 1단계에서 6단계까지를 지원하는 방법을 보여줍니다.



**Part 4 FAQ**

* [Quorum을 사용하면서 문제가 생겼습니다. 어디서 도움을 받을 수 있나요?](#_heading=h.gjdgxs)
* [Quorum은 트랜잭션의 프라이버시를 어떻게 보장하나요?](http://hyperlink#30j0zll)
* [프라이빗 트랜잭션에서 Quorum은 어떻게 합의를 이루나요?](#_heading=h.30j0zll)
* [프라이빗 트랜잭션에서 트랜잭션 사이즈에 대한 제한 사항(트랜잭션이 암호화되어 있기 때문에)은 없나요?](#_heading=h.1fob9te)
* [프라이빗 트랜잭션에 트랜잭션을 생성한 노드도 포함시켜야 하나요?](#_heading=h.3znysh7)
* [트랜잭션 매니저 없이 Quorum 노드를 실행하는 것이 가능한가요?](#_heading=h.2et92p0)
* [Raft 합의 노드 구성의 흔한 오류](#_heading=h.gjdgxs)
* [Quorum/Constellation/Tessera의 공식적 도커(Docker) 이미지가 있나요?](#_heading=h.gjdgxs)
* [Quorum 노드를 다른 합의를 사용하는 노드들과 혼합하여 사용해도 되나요?](#_heading=h.gjdgxs)

### Quorum을 사용하면서 문제가 생겼습니다. 어디서 도움을 받을 수 있나요?

Quorum의 기술팀이 지속적으로 모니터링하는 곳이 두 군데 있습니다. 이 사이트, 관련 저장소들과 Quorum Slack입니다. Quorum Slack은 커뮤니티에 질문하고 즉각적인 해답을 얻을 수 있는 가장 효과적인 채널입니다. [이곳](https://clh7rniov2.execute-api.us-east-1.amazonaws.com/Express/)을 통해 바로 Quorum Slack 채널에 가입하실 수 있습니다.

### [Quorum은 트랜잭션의 프라이버시를 어떻게 보장하나요?](https://docs.google.com/document/d/1ITkzxwaaGyoHiKdPsPwp3G0J5tA5jAtU/edit#bookmark=id.3dy6vkm)

Quorum은 트랜잭션 프라이버시를 다음을 통해 달성합니다.   
 1. 트랜잭션을 송신하는 측에서 privateFor 파라미터를 이용하여 해당 트랜잭션을 공유할 노드들을 설정합니다.  
 2. 프라이빗 트랜잭션의 페이로드(payload)를 암호화된 페이로드의 해시(hash)값으로 대체하여 원래의 페이로드를 트랜잭션 공유를 허용하지 않은 노드들에게는 보이지 않도록 합니다.   
 3. 암호화된 프라이빗 데이터를 별도로 구성한 트랜잭션 매니저([Constellation](https://github.com/jpmorganchase/constellation) 혹은 [Tessera](https://github.com/jpmorganchase/tessera))에 저장합니다. 트랜잭션 매니저는 암호화된 데이터를 공유를 허용한 노드들에게 배포하고 복호화된 페이로드를 해당 노드들에게 제공합니다.

[트랜잭션 처리 및 프라이버시]에서 더 많은 정보를 확인할 수 있습니다.

### 프라이빗 트랜잭션에서 Quorum은 어떻게 합의를 이루나요?

기본 이더리움에서는 모든 노드들이 모든 거래를 처리하기 때문에 각각의 노드가 같은 state root를 가지고 있습니다. Quorum에서는 노드들이 모든 ‘퍼블릭’ 트랜잭션(예로, 참고자료나 시장 데이터 요약 등)은 처리 하지만 ‘프라이빗’ 트랜잭션은 허용된 노드들만 처리할 수 있습니다.

Quorum 노드들은 두 가지 패트리샤 머클 트리(Patricia Merkle Trie)를 관리합니다. 하나는 프라이빗 스테이트를 위한 것이고 또 하나는 퍼블릭 스테이트를 위한 것입니다. 이로 인해 신규 블록을 추가할 때, 블록 검증 단계에서 ‘퍼블릭 스테이트 루트 (state root)’의 스테이트 확인 과정을 거칩니다. 또한 블록 검증 단계에는 블록의 모든 퍼블릭/프라이빗 트랜잭션 해시를 확인하는 ‘글로벌 트랜잭션 해시(global Transaction hash)’ 확인 절차 또한 포함합니다. 이는 각각의 노드가 다른 노드들과 동일한 트랜잭션 세트를 보유함을 검증할 수 있다는 것을 말합니다. **EVM은 동기화된 퍼블릭 스테이트 루트를 통해 결정되고 프라이빗 트랜잭션은 노드들 간에 동기화(글로벌 트랜잭션 해시 확인 절차) 되기 때문에 모든 노드들에서 프라이빗 스테이트가 동기화된다는 것을 예상할 수 있습니다. 또한 Quorum은 특정 블록의 특정 트랜잭션에 있는 프라이빗 스테이트 해시를 조회하는 eth\_storageRoot API를 제공하고 있습니다. 애플리케이션 레이어에서 이 API를 사용하여 상대방의 오프 체인 상태 검증을 수행할 수 있습니다.**

[Quorum 합의 알고리즘]와 [트랜잭션 처리 및 프라이버시] 페이지에서 더 많은 정보를 확인할 수 있습니다.

### 프라이빗 트랜잭션에서 트랜잭션 사이즈에 대한 제한 사항 (트랜잭션이 암호화되어 있기 때문에)은 없나요?

트랜잭션의 유일한 제한 사항은 가스 한도 (gas limit)입니다. Constellation/Tessera에서는 (설정이 가능은 하겠지만) 트랜잭션 사이즈에 대한 제한이 없습니다. **사이즈가 큰 프라이빗 트랜잭션을 실행할 때, 오히려 속도가 향상되는데 이는 대부분의 네트워크가 해시 값 (hash digest)만을 보기 때문입니다. 이 특성으로 인하여** 큰 데이터를 지리적으로 분산되어 있는 노드들에게 전파하는 속도가 PGP가 파일을 암호화하여 http/https로 전달하는 속도와 유사하므로 트랜잭션의 전달이 매우 빠르다고 볼 수 있습니다. 순차적인 트랜잭션을 처리하는 경우라면 당연히 해당 트랜잭션들이 전달되기를 기다려야 하겠지만, 독립적이거나 동시다발적인 트랜잭션을 처리하는 경우라면 페이로드의 크기가 전송속도에 영향을 미치지 않으므로 네트워크 대역폭(network bandwidth)이 트랜잭션 전송의 제한 사항이 될 수 있습니다. Constellation/Tessera는 모든 트랜잭션들을 동시에 처리합니다.

### 프라이빗 트랜잭션에 트랜잭션을 생성한 노드도 포함시켜야 하나요?

아니요, 포함시키면 안됩니다. Quorum에서는 트랜잭션을 생성한 노드를 privateFor에 포함하면 오류가 발생합니다. 해당 트랜잭션을 생성한 노드에게만 보이는 프라이빗 컨트랙트를 만들고자 할 때는 다음의 형식만 이용하시길 바랍니다: privateFor: []  
<https://github.com/jpmorganchase/quorum/pull/165>

### 트랜잭션 매니저 없이 Quorum 노드를 실행하는 것이 가능한가요?

트랜잭션 매니저(Transaction Manager)가 없어도 노드를 실행하는 것이 가능합니다. 이를 위해서는 Quorum 노드의 소켓 구성 시, 대응되는 Tessera/Constellation 대신 PRIVATE\_CONFIG=ignore ...로 셋팅해야 합니다. Quorum 노드를 이렇게 구성하는 경우, 노드가 해당 프라이빗 키들을 브로드캐스트하지 않기 떄문에(실행 중인 트랜잭션 매니저가 없는지 확인하십시오) 프라이빗 트랜잭션을 사용할 수 없습니다.

### [Raft 합의 노드 구성의 흔한 오류](https://docs.google.com/document/d/1ITkzxwaaGyoHiKdPsPwp3G0J5tA5jAtU/edit#bookmark=id.1t3h5sf)

[https://github.com/jpmorganchase/quorum/issues/410](https://github.com/jpmorganchase/quorum/issues/410%EB%A5%BC) 를 참고해주세요

### [Quorum/Constellation/Tessera의 공식적 도커(Docker) 이미지가 있나요?](https://docs.google.com/document/d/1ITkzxwaaGyoHiKdPsPwp3G0J5tA5jAtU/edit#bookmark=id.4d34og8)

네! 있습니다. 공식적 [도커](https://hub.docker.com/u/quorumengineering/) 컨테이너:

quorumengineering/quorum:latest   
quorumengineering/constellation:latest   
quorumengineering/tessera:latest

### [Quorum 노드를 다른 합의를 사용하는 노드들과 혼합](https://docs.google.com/document/d/1ITkzxwaaGyoHiKdPsPwp3G0J5tA5jAtU/edit#bookmark=id.2s8eyo1)하여 사용해도 되나요?

아쉽지만, 안됩니다. Raft 합의 알고리즘을 사용하는 Quorum 노드들은 Raft 합의 알고리즘을 사용하는 다른 노드들과만 문제없이 상호작용할 수 있습니다. 이것은 지원되는 모든 합의 알고리즘에 적용됩니다.

**Part 5 Product Roadmap**

## 2018 / 2019년 전망

#### 프라이버시

* 조직 레벨의 프라이버시 지정
* 프라이버시가 보장된 자산 이동 (ZKP, Zero Knowledge Proof)
* 프라이빗 컨트랙트 확장성
* 프라이빗 스테이트 합의 (동기화 메소드 외에)

#### 권한

* 두 개의 롤 지원: 읽기 vs 쓰기 노드/계정
* 스마트 컨트랙트 기반의 네트워크 권한 관리
* 인증 및 보호된 RPC API 액세스
* 합의 노드 화이트리스팅(whitelisting)

#### 성능

* 성능 벤치마크 가이드라인
* 트랜잭션 병렬처리 R&D
* EVM 최적화
* eWASM 지원

#### 복구

* 트랜잭션 매니저 클러스터
* 손실의 경우를 대비한 트랜잭션 매니저 데이터 자동 복구

#### 확장성 및 상호운영성

* Raft 확장성 향상
* PoA 및 기타 합의 알고리즘
* Quorum 간 자산 이동
* 요율 제한 (가스 가격)

#### 도구

* 원클릭 네트워크 배포
* 향상된 데이터베이스 어댑터
* 모니터링/로깅(Logging) 툴
* 아이디 관리 툴

#### 기타 / 진행중

* EEA 규격 컴플라이언스 확인
* 고이더리움(geth) 업그레이드 사항 통합