# ChainLink

## A Decentralized Oracle Network

Steve Ellis, Ari Juelsy, and Sergey Nazarov

4 September 2017 (v1.0)

**Abstract**

스마트 컨트랙트는 전통적인 법적 합의와 자동화된 법적 합의를 대체함으로서 다양한 산업들을 변혁시키는 목적으로 만들어졌다. 성능 검증과 실행은 그 법적관계의 관련된 당사자가 수동이나 자동으로 정보를 수시로 업데이트를 해야한다. 하지만 기본적인 합의 프로토콜 때문에 스마트 컨트랙트를 가능하게 할 수 있는 블록체인은 타 시스템들과의 native communication이 불가능하다.

현재 이 문제를 다루기위해 새롭게 소개된 솔루션은 다른 시스템들과 연결을 가능하게 하는 오라클이다. 현 존재하는 오라클들은 중앙화된 서비스들이다. 이 서비스를 사용하는 스마트 컨트랙트는 전통적인 중앙화된 디지털 합의와 다름없이 단일 장애점(single point of failure)이 있기 때문에 보안성이 더 강하지도 않다.

이 리포트에서 소개할 기술은 ChainLink라는 탈중앙화된 오라클 네트워크이다. ChainLink가 컨트랙트가 외부 연결이 가능하게 할 수 있도록 제공하는 요소들과 네트워크의 노드들을 실행시키는 소프트웨어들을 설명한다. On-chain 컨트랙트 데이터 집합 시스템(contract data aggregation system)과 더 효율적인 off-chain 합의 메커니즘을 소개한다. 다음으로는 유저들이 정보를 확보한 상태에서 제공사를 선택할수있는 능력과 불리한 조건에서도 효과적인 서비스를 얻을수 있는 ChainLink의 현재 존재하는 명성과 보안 모니터링 서비스를 설명한다. 마지막으로 우리의 보안 전략의 도움을 위한 이상적인 오라클의 속성들을 나타내고, 기능이 풍부한 oracle programming, data-source infrastructure 수정 및 기밀  스마트 컨트랙트 등등을 포함한 미래에 가능할수도있는 개선들을 계획한다.

# Contents

# 1. Introduction

스마트 컨트랙트는 블록체인과같은 탈중앙화된 인프라이다. 그것은 조작이 되지않는데, 즉, 그것을 만든 사람까지 포함해서 어떠한 집단도 코드를 변경하거나 실행을 방해 할 수 없다는 것이다. 역사적으로 보면, 코드로 된 컨트랙트는 중앙화된 방식으로 사용되서 다른 집단의 의한 변경, 종료, 심지어 삭제도 가능했다. 반대로 스마트 컨트랙트의 실행은 모든 집단을 서면 동의서에 묶기때문에, 새롭고 강력한 신뢰의 관계를 생성함으로서 어느 한 조직의 신뢰의 의존하지 않는 관계를 보장한다. 스마트 컨트랙트는 자기검증, 자기실행이라는 기능들이 있기에 디지털 합의를 성립과 관리하는데 우월한 수단을 제공한다.

하지만 이 강력한 신뢰모델은 연결성이라는 새로운 기술적인 문제를 의미한다. 스마트 컨트랙트의 실제 적용도[1]는 블록체인 외부의 data feeds나 API들에서 가져오는 데이터에 의존한다. 블록체인의 기반이 되는 합의 메커니즘의 메커니즘 때문에 블록체인이 그 대단히 중요한 데이터를 가져오는것은 불가능하다.

우리가 이 문제의 답으로 제안하는것이 바로 보안된 오라클 네트워크인 ChainLink다. ChainLink가 다른 오라클 솔루션들과의 다른점은 완전히 탈중앙화된 네트워크다. 이 탈중앙화된 접근은 어느 한 집단의 신뢰의 의존을 제한함으로써 스마트 컨트랙트와 API의 end-to-end operation까지 조작을 불가능하게 한다. 현재 사용되고있는 디지털 합의들을 대체하려면 스마트 컨트랙트를 API같은 data feed들과 연동을 시키는 기능이 필수적이다.

현재 대부분의 디지털로 자동화된 전통적인 계약상의 동의들은 외부 데이터를 사용해서 계약상의 성능을 검증하고 data output을 외부 시스템들에게 보내야한다. 스마트 컨트랙트가 현재 존재하는 계약상의 메커니즘을 대체하면 같은 데이터 input과 output들의 신뢰도가 더 높은 version들을 필요로 할 것이다. 다음세대의 가능할수도있는 스마트 컨트랙트와 그것이 필요로 할 데이터들의 예를 들자면:

- 채권이나 금리 파생 상품같은 유가증권 스마트 컨트랙트가 생긴다면 금융시장의 정보들을 가져오는 API를 접속 할 수 있는 권한이 필요 할 것이다.
- 보험 스마트 컨트랙트는 그 보험의 관련된 이벤트의 관한 IoT 데이터를 제공하는 data feed가 필요 할 것이다. 예를 들면 warehouse가 도난당했던 시점에 문이 잠겼었는지, 회사의 방화벽이 작동중이였는지, 아니면 비행기가 제시각에 도착을 했는지.
- 무역 금융 스마트 컨트랙트는 계약적인 의무를 만족시키려면 수송품의 GPS 데이터, 공급 사슬 ERP 시스템, 그리고 수송품의 세관정보들을 필요로 할 것이다.

이 예시들에서 흔히 일어나는 또 다른 문제는 스마트 컨트랙트가 데이터를 블록체인 외부의 시스템들에게 전달하지 못하는 점이다. 이 데이터를 전달하려면 유저들이 이미 계정이 있는 전통적인 중앙화된 인프라로 지불 메세지를 보내야 한다 (e.g. 계좌 송금, PayPal). 스마트

---

[1] [1]The main use of smart contracts in Ethereum today is management of tokens, which are a common functionality in most smart contract networks. We believe that the current focus on tokens to the exclusion of many other possible applications is due to a lack of adequate oracle services, a situation ChainLink specifically aims to remedy.

컨트랙트를 대신해서 데이터를 안전하게 API와 다양한 legacy 시스템으로 전달 할 수 있는 ChainLink의 기능은 외부와 연동되고 조작이 불가능한 계약의 생성을 가능하게 할 수 있다.

## Whitepaper Roadmap

이 백서에서는 ChainLink의 구조를 관찰한다 (Section 2). 다음에는 오라클들의 보안을 우리가 어떻게 정의했는지를 설명한다 (Section 3). ChainLink가 탈중앙화 및 오라클과 data source들의 분산을 어떻게 접근을 했는지에 대해서 적은 후 (Section 4), ChainLink가 제공하는 4개의 보안 서비스들과 LINK token의 역할을 설명한다 (Section 5). 그 다음에는 향상된 기밀유지, 신뢰할수있는 하드웨어의 사용, 인프라의 변화들, 그리고 전체적 오라클 프로그래밍 가능성을 포함한 장기적인 개발 공략을 제안한다 (Section 6). 간단히 대체 가능한 오라클 디자인을 살펴본 뒤 (Section 7), ChainLink 개발을 인도하는 디자인의 원칙과 철학에 대한 짧은 논의를 가져본다 (Section 8).

## 2 Architectural Overview

ChainLink의 핵심 기술적 목적은 on-chain 과 off-chain, 이 두 환경을 연결시키는것이다. 밑에서 ChainLink의 구성 요소들의 구조를 하나씩 설명한다. ChainLink는 일단 Ethereum[16], [35]에 설계 될 것 이지만, 현재 가장 많이 쓰이는 스마트 컨트랙트 네트워크들 모두를 지원 할 수 있기를 의도한다. ChainLink는 on-chain과 off-chain version 둘 다에서 모듈성을 염두하고 디자인 됐다. 미래에 기능성이 더 높은 기술과 경쟁력 있는 구현성이 생기는것을 대비해서 ChainLink 시스템의 모든 구성 요소들은 업그레이드가 가능하다.

## 2.1 On-Chain Architecture

ChainLink는 오라클 서비스로서, ChainLink 노드들은 유저 계약들이 보낸 데이터 요청이나 query에 답변을 한다. 우리는 이 계약 요청을 USER-SC라고 표현한다. ChainLink의 계약 요청을 하는 on-chain interface 자체도 on-chain 계약이고, 이 interface를 CHAINLINK-SC라고 표현한다.

CHAINLINK-SC 안에는 주 계약 3개로 만들어진 on-chain 요소가 있다. 이 계약들은 명성 계약, 순서 매칭 계약, 그리고 집계 계약이다. 명성 계약은 오라클 서비스 제공자의 performance 단위 (metrics)를 기록한다. 순서 매칭 계약은 서비스 표준 합의를 받고, SLA 한도들을 기록하고, 오라클 제공사들에게서 경쟁 입찰(bid)을 모은다. 다음으로는 명성 계약을 사용해서 oracle SLA를 완성시킨다. 집계 계약은 오라클 제공사들의 응답들을 모은 후 ChainLink의 마지막 집단적인 결과를 계산하고 oracle 제공사의 performance 단위를 명성 계약에 기록을 한다. ChainLink 계약들은 모듈성 있게 디자인 됐음으로, 유저들이 필요한대로 구성이나 교체를 할 수 있다. On-chain 작업 흐름은 3단계가 있다: 1. oracle 선택, 2. 데이터 보고, 3. 응답 집계.

**Oracle Selection**

오라클 서비스 구매자는 구매 당시 서비스 표준 합의 (SLA)를 이루는 구체적인 요건들을 제시한다. SLA 제안서는 query의 한도와 구매자가 필요한 오라클의 개수 같은 정보를 포함한다. 추가로, 구매자는 합의에 쓰일 명성 계약과 집계 계약을 요구한다.

구매자들은 on-chain에 유지되고있는 명성과 과거 계약들에서 모은 더 정확한 데이터 집합을 사용해서 off-chain listing service들을 통해 수동으로 oracle들을 분류하고 필터하고 선택할 수 있다. 우리의 의도는 ChainLink가 이 listing service중 딱 하나만 유지하는것이다. 이것을 유지함으로써 ChainLink에 관련된 기록들을 모으고 등록된 oracle 계약들의 binaries를 검증을 할 수 있다. Section 5에서 listing service와 명성 시스템들에 대해 더 구체적으로 설명을 한다. Listing들을 생성하는데 사용되는 데이터는 블록체인에서 가져온다. 이로써 대체가 가능한 oracle-listing 서비스들을 설계할 수 있을것이다. 구매자들은 SLA 제안서들을 off-chain oracle들에게 제출할것이며, on-chain으로 SLA를 완성하기 전에 합의에 이를것이다.

수동 matching은 모든 상황에 가능하지 않다. 예를 들자면, 계약이 oracle 서비스를 데이터 분량에 따라 수시로 필요로 할 수도 있다. 자동화된 솔루션들은 이 문제를 해결하고 실용성을 향상시킨다. 이런 이유들 때문으로 ChainLink는 order-matching 계약들을 통해서 자동화된 oracle matching도 제안하고 있다.

구매자가 SLA 제안서를 지정한후에는 oracle들을 직접 접하지 않고 SLA를 order-matching 계약에게 제출할 것이다. 이 제안서의 제출은 oracle 제공자들이 감시하고 기술력과 서비스 목적에 따라 필터를 할 수 있는 기록을 유발한다. 다음엔 ChainLink 노드들이 그 제안서에 경쟁 입찰 (bid)을 할지 말지를 결정한다. Oracle 서비스 제공자가 계약의 경쟁 입찰을 하면 그 계약을 지켜야 하고, 안 지켰을 경우에는 SLA에 적혀있는데로 패널티 벌금의 액수를 꼭 첨부를 해야한다.

경쟁 입찰은 bidding window라는 정해진 기간동안 가능하다. SLA가 자격있는 입찰들을 충분히 받고 bidding window가 끝나면 구매자가 부탁한 개수의 oracle이 입찰들중에서 선택된다. 선택되지 않은 oracle들에게는 첨부했던 패널티 벌금이 다시 돌려지고, 완성된 SLA 기록이 생성된다. 완성된 SLA가 기록되면, 선택을 당한 oracle들에게 통보를 보낸다. 그리고 그 oracle들은 SLA에 적힌 지시를 실행한다.

**Data Reporting**

새로운 oracle이 생성된 후, off-chain oracle들이 지시를 실행하고 on-chain에 보고를 한다. 더 자세한 설명은 Section 2.2와 4을 참고하자.

**Result Aggregation**

Oracle들이 oracle의뢰인에게 결과를 보고한 후, 그 결과들은 집계 계약에 기록이 된다. 집계 계약은 집단적인 결과들을 기록을 하고 평균 값을 낸다. 다음, 모든 oracle의 응답들은 명성 계약에 보고가 된다. 마지막으로는 계산된 평균 값이 USER-SC에 있는 지정된 계약 기능에게 반환된다.

부정확하거나 틀린 답을 발견하는것은 data feed와 적용된 방식에 따라 다를수 있다. 예를 들면, 평균 값을 계산하기전에 오답을 발견하는것은 숫자 데이터를 다룰때 필요할 수

있지만, boolean데이터를 다룰때는 아니다. 이 이유 때문에 특정된 집계 계약은 없을것이지만, 구매자가 지정한 구성이 가능한 계약 주소가 있을것이다. ChainLink는 집계 계약들의 기본 세트는 포함할것이지만, 기본 계산 interface에 따르기만 하면 맞춤형 계약도 가능할 것이다.

## 2.2 Off-Chain Architecture

Off-chain에서 ChainLink는 일단 이더리움 네트워크에 연결된 oracle노드들로 만들어진 네트워크로 이루어졌다. 우리는 이것으로 모든 스마트 컨트랙트 네트워크들을 이용할 때 사용할 생각이다. 이 노드들은 독립적으로 off-chain으로 의뢰의 대한 답변들을 모은다. 이 답변들은 사용 가능한 합의 메커니즘 여러개중 하나로 단 하나의 글로벌 답변으로 집계된다. 그 글로벌 답변은 의뢰인 계약인 USER-SC한테 보내진다. ChainLink 노드들은 기본 블록체인 소통, 예정, 그리고 평범한 외부 자원들과의 연결을 다루는 기본 open source core implementation으로 운영된다. 노드 관리자들은 외부 어댑터라는 추가 소프트웨어 기능을 추가할 수 있는 선택권이 있다. 관리자들은 외부 어댑터를 사용해서 특수화된 off-chain 서비스들을 추가할 수 있다. ChainLink 노드들은 벌써 기업들의 public 블록체인들과 private 네트워크들에 투입되고있다. 노드들을 탈중앙화된 방식으로 운영하는게 ChainLink network의 목적이다.

**ChainLink Core**

주요 노드 소프트웨어는 블록체인과 소통, 예정, 그리고 다양한 외부 서비스들 사이에 주어진 일을 분산시키는 일을 한다. 이 일은 assignment로 구성된다. 각 assignment는 더 작은 subtask들로 이루어 졌으며, 이 subtask들은 pipeline형식으로 구성되어있다. 각 subtask는 특정된 기능이 있고, 그 기능을 거쳐서 나온 결과가 다음 subtask으로 넘어간다. 이렇게 계속 이어지면 결국에는 결정적인 마지막 결과에 도달한다. ChainLink의 노드 소프트웨어는 HTTP request, JSON parsing, 그리고 다양한 블록체인 서식들로 변환하는 subtask들을 기본적으로 포함하고 있다.

**External Adapters**

맞춤화된 subtask들은 adapter들을 생성해 요청 할 수 있다. Adapter들은 minimal REST API가 포함된 외부 서비스들이다. 어댑터를 서비스 지향 방식으로 모델링하면 모든 프로그래밍 언어의 프로그램을 프로그램 앞에 작은 중간 API를 추가하여 간단히 구현할 수 있다. 마찬가지로 복잡한 다중 단계 API와의 연동은 한도가 정해진 개별 subtask들로 단순화 할 수 있다.

**Subtask Schemas**

우리는 많은 adapter들이 오픈 소스가되어 다양한 커뮤니티 회원들이 서비스를 감시하고 실행할 수있을 것으로 예상한다. 많은 개발자들이 다양한 종류들의 adapter들을 개발하므로 adapter들간의 호환성을 보장하는 것이 중요하다. ChainLink는 현재 JSON Schema [36] 를 기반으로하는 schema 시스템으로 작동하여 각 adapter에 필요한 입력 및 형식 방법을 지정한다. 마찬가지로 adapter는 각 subtask의 출력 형식을 설명하는 output schema를 지정한다.

## 3. Oracle Security

ChainLink의 보안 아키텍처를 설명하려면, 우리는 먼저 보안이 중요한 이유와 이것이 의미하는 바를 설명해야한다.

**Why must oracles be secure?**
1장에서 간단한 예제로 돌아가보면, 스마트 컨트랙트 보안이 잘못된 data feed를 얻으면 잘못된 계약자(party)에게 지급 될 수 있으며, 스마트 컨트랙트 보험 data feeds가 피보험자에 의해 변조 될 수 있는 경우 보험 사기가 발생 할 수 있고, GPS 무역 금융 계약에 제공된 데이터는 데이터 제공 업체를 떠난 후 수정 될 수 있으며 도착하지 않은 상품에 대해서는 지불이 이루어 질 수 있다.

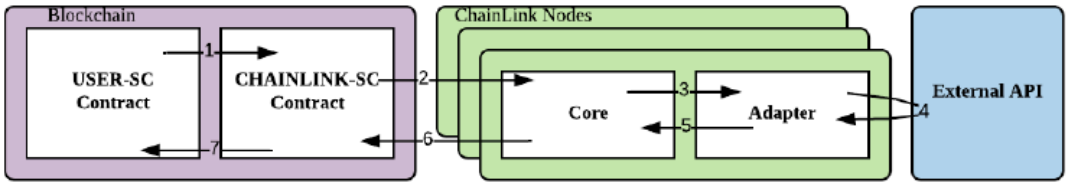


**Figure 1:** ChainLink workflow: **1)** USER-SC makes an on-chain request; **2)** CHAINLINK-SC logs an event for the oracles; **3)** ChainLink core picks up the event and routes the assignment to an adapter; **4)** ChainLink adapter performs a request to an external API; **5)** ChainLink adapter processes the response and passes it back to the core; **6)** ChainLink core reports the data to CHAINLINK-SC; **7)** CHAINLINK-SC aggregates responses and passes them back as a single response to USER-SC.

일반적으로, 그것의 장부(ledger) 또는 게시판 추상화(bulletin-board abstraction)와 함께 잘 동작하는 블록체인은 매우 강력한 보안 속성을 제공한다. 사용자들은 트랜잭션을 올바르게 검증하고 데이터 변경을 방지하는 기능으로 블록체인을 사용한다. 그들은 그것을 사실상 신뢰할 수 있는 제 3자(아래에서 논의하는 개념)처럼 다룬다. 오라클 서비스 지원은 자신이 지원하는 블록체인에 상응하는 수준의 보안을 제공해야 한다. 따라서 오라클은 사용자에게 신뢰할 수 있는 제 3자로 제공돼야하며 정확하고 시기 적절한 응답을 매우 높은 확률로 제공해야한다. 모든 시스템의 보안은 가장 약한 링크만큼만 강력하므로(가장 약한 링크의 보안이 뚫리면 위험하니) 잘 설계된 블록체인의 신뢰성을 유지하려면 신뢰할 수 있는 오라클이 필요하다.

**Defining oracle security: An ideal view.**
오라클 보안에 대해 설명하기 위해 먼저 이를 정의해야한다. 오라클 보안에 대해 추론할 수 있는 유익하고 원칙적인 방법은 다음과 같은 사고 실험에서 비롯된다. 모든 지시를 충실하게 수행 할 수 있는 신뢰할 수 있는 제 3자(trusted third party, TTP)가 오라클을 실행하라는 작업을 받았다고 생각해보자. 우리는 이 오라클을 ORACLE(일반적으로 모든 대문자를 사용하여 사용자가 완전히 신뢰하는 엔티티를 나타냄)로 나타내고 TTP가 완벽하게 신뢰할 수 있는 데이터 소스인 Src로부터 데이터를 얻었다고 가정한다. 마법의 서비스 오라클을 고려해볼 때, 우리가 어떤 지시를 내릴까?
신뢰성 속성이라고 불리는 무결성[24] 속성을 이루기 위해 ORACLE이 다음 단계를 수행하도록 요청하면 된다.

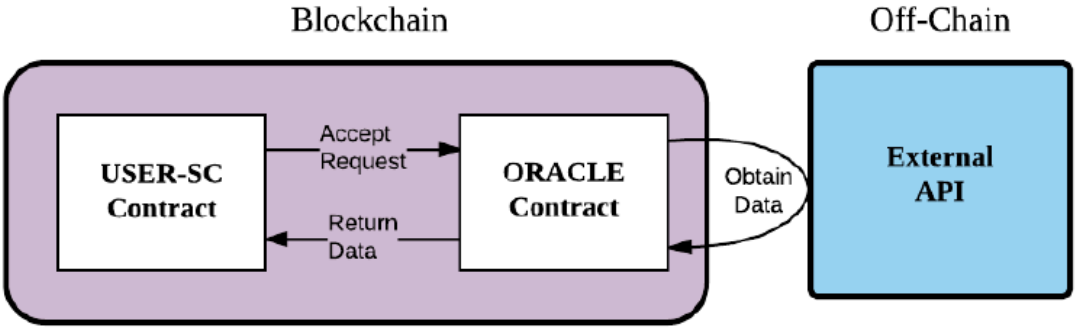


**Figure 2:** Behavior of an ideal oracle ORACLE is defined by steps: **1)** Accept request; **2)** Obtain data; **3)** Return data. Additionally, to protect the confidentiality of a request, upon decrypting it, ORACLE never uses or reveals the data it contains, except to query Src.

1. 요청 수용(Accept request) : 스마트 컨트랙트 USER-SC에서 데이터 소스 Src, 시간 또는 범위 τ 및 쿼리 q를 지정하는 Req = (Src, τ, q) 요청을 수집한다.
2. 데이터 획득(Obtain data) : 쿼리 q를 시간 τ에 Src로 보낸다.
3. 데이터 반환(Return data) : 대답 a를 받으면 스마트 컨트랙트로 a를 반환한다.

이 지시들이 올바르게 수행 된다면 강력하고 의미 있으면서도 단순한 보안 개념을 정의한다. 직관적으로 보면 이 지시들은 ORACLE이 Src와 USER-SC 사이의 믿음직한 다리 역할을 하도록 명령한다.[2] 예를 들어, Src가 https://www.FountOfKnowledge.com고, 오후 4시이고 , q = "ticker INTC 가격"인 경우에는, 오라클이 오후 4시에 query한 INTC의 가격을 https://www.FountOfKnowledge.com으로부터 USER-SC에게 정확히 제공 할 것을 보장한다. 기밀성은 oracle들에게 있는 또 하나의 바람직한 속성이다. USER-SC가 블록체인에서 Req를 오라클로 보냄으로써 Req가 공개된다. Req가 민감한 정보를 가지고 있고 공개가 되면 안되는 경우가 많이 있다. 예를 들어, USER-SC가 항공 보험 계약이고 특정 사용자의 항공편 (q = "Ether Air Flight 338")에 대해 ORACLE에게 query Req를 전송한다면, 그 사용자의 비행 계획이 전 세계에 공개된다. USER-SC가 금융 거래 계약 인 경우 Req는 사용자의 거래 및 포트폴리오에 대한 정보를 유출 할 수 있다. 물론 많은 다른 예도 있다.

Req의 기밀성을 보호하기 위해서는 Req의 데이터를 ORACLE에 속한 공개 키로 암호화 되도록 요구 할 수 있다. 오라클의 TTP 특성을 계속 활용하면 단순히 ORACLE에게 이 정보 흐름 제약 조건을 부여 할 수 있다:

Req를 해독 할 때는 Src를 쿼리하는 것을 제외하고 절대로 Req의 데이터를 공개하거나 사용하지 마라.

전통적인 CIA (Confidentiality-Integrity-Availability) 트라이어드(triad)의 가용성같은 다른 중요한 오라클 속성들이 있다. 진정으로 이상적인 ORACLE 서비스는 절대로 폐쇄 되지

[2] 2Of course, many details are omitted here. ORACLE should communicate with both USER-SC and source Src over secure, i.e., tamperproof, channels. (If Src is a web server, TLS is required. To communicate with USER-SC, ORACLE must be sure to scrape the right blockchain and digitally sign A appropriately.)

않을 것이다. 가용성은 검열 저항과 같은 미묘한 특성들을 포함한다. 정직한 ORACLE은 특정 스마트 계약을 단정하지 않고 요청을 거부하지 않는다. TTP의 개념은 특정 모델에서 암호화 프로토콜의 보안을 증명하는데 사용되는 이상적인 기능 [7]의 개념과 유사하다. 이상적인 게시판을 유지하는 TTP와 유사한 조건들로 블록 체인을 모델링 할 수도 있다. 거래를 수락하고, 검증하고, 직렬화하고, 추가만 가능한 데이터 구조인 게시판에 거래를 영구적으로 유지하는 것이 이 블록체인의 책임들이다.

**Why the ideal oracle (ORACLE) is hard to achieve**
(왜 이상적인 oracle을 만들기가 어려운가)

물론 완벽하게 신뢰 할 수 있는 data source Src는 없다. 오류가 있는 website, 부정 행위 서비스 제공자 또는 정직한 실수로 인해 데이터가 실수로 또는 악의적으로 손상 되었을 수도 있다. Src를 신뢰할 수 없다면 ORACLE이 위에서 설명한대로 TTP와 똑같이 작동하더라도 여전히 client가 원하는 보안을 완전히 만족시키지 못한다. 오류가 있는 source Src가 주어지면 위에 정의된 정직한 특성은 더 이상 오라클의 대답이 옳다는 것을 의미하지 않다. 예를 들어 Intel 주식의 실제 가격이 $40이고 https://www.FountOfKnowledge.com이 $50로 잘못 보고하면 ORACLE은 잘못된 값인 a = $50를 USER-SC로 보낸다. 이 문제는 single source Src를 사용할 때 불가피하다. ORACLE은 Src가 query에 제공하는 답변이 정확한지 알 수 있는 방법이 없다.

물론 더 큰 문제는 오라클에 대한 TTP가 추상적인 개념이라는 사실이다. 무조건 신뢰할 만한 서비스 제공자는 없다. 최선을 다하는 제공자 조차도 버그가 있거나 해킹을 당할 수 있다. 따라서 사용자 또는 스마트 컨트랙트가 서비스 ORACLE이 지시를 충실하게 이행한다는 절대적인 확신을 가질 수 있는 방법은 없다. ChainLink는 이 ORACLE이라는 이상적인 기능성을 고려해서 보안 프로토콜을 설계했다. ChainLink의 목표는 현실적인 가정하에 이상적인 ORACLE의 속성들에 최대한 근접한 실제 시스템을 달성하는 것이다. 이제 이걸 어떻게 달성할지 설명할것이다. 다음 내용을 단순화하기 위해 ChainLink 계약의 전체 집합, 즉 on-chain의 전체 기능을 (계약 청구가 가능한 인터페이스뿐만 아니라) CHAINLINK-SC로 나타낼것이다. 따라서 시스템 구조에서 실제로 사용되는 여러 개의 개별 계약들을 한 계약으로 묶어서 표현할 것이다.

## 4. ChainLink Decentralization Approach

우리는 결함이 생긴  노드에 대해 세가지 보완적인 접근 방법을 제시한다.
(1) data sources; (2) Distribution of oracles; and (3) Use of trusted hardware.
우리는 이 장에서 분산화를 포함하는 처음 두 접근법을 이야기하려한다.
6장에서 우리는 차별화되고 상호 보완적인 접근을 하는 하드웨어에 대한 장기적 전략에 대해 이야기하고자 한다.

### 4.1 Distributing sources

결함이 있는 단일 소스 Src를 처리하는 간단한 방법은 여러 소스로부터 데이터를 얻는 것이다. 즉, 데이터 소스를 분산화 시키는 것이다.  신뢰할 수 있는 ORACLE는 소스 모음

$\mathrm{Src}_1, \mathrm{Src}_2, \ldots, \mathrm{Src}_k$ 을 조회(Query)할 수 있고 $a_1, a_2, \ldots, a_k$ 의 응답을 얻을 수 있고 그것들을 하나의 답 $A = \mathrm{agg}(a_1, a_2, \ldots, a_k)$ 으로 종합 할 수 있다.
예를 들면 다수결 투표가 있다. 만약 소스의 다수가(> k / 2) 동일한 값 a를 반환하면 함수 agg는 a를 반환한다. 그렇지 않으면 agg는 오류를 반환한다. 이 경우에 다수의(> k / 2) 소스가 올바르게 작동하고 있다면 ORACLE은 항상 올바른 값 A를 반환한다.

다양한 대체 가능한 함수들인 agg는 잘못된 데이터에 대해 견고성을 보장하거나 시간 경과에 따른 데이터 값의 변동(예 : 주가)을 처리 할 수 있다. 예를들어 agg는 특이치(특이치 : 비정상으로 크거나 작은 값)를 삭제하고(예 : 가장 크고 가장 작은 값 Ai) 그리고 남은 값들의 평균을 출력 할 수 있다.

물론 오류가 집계에 의해 제공되는 보장을 악화시키는 방식으로 데이터 소스간에 상관관계가 있을 수 있다. 만약 사이트 $\mathrm{Src}_1 = \mathrm{EchoEcho.com}$ 이 $\mathrm{Src}_2 = \mathrm{TheHorsesMouth.com}$ 에서 데이터를 가져오는 경우 Src2의 오류는 항상 Src1의 오류를 의미한다.
데이터 소스 간의 더 많은 미묘한 상관 관계도 발생 할 수 있다. Chainlink는 또한 오라클과 사용자가 원하지 않는 상관 관계를 피할 수 있도록 데이터 소스의 독립성을 쉽게 이해 할 수 있는 방법으로 매핑하고 보고하는 연구를 추진할 것을 제안한다.

## 4.2 Distributing oracles

소스가 분산될 수 있는 것처럼, 우리의 이상적인 서비스 ORACLE 자체의 이상적인 서비스는 분산 시스템으로 근사 될 수 있다. 다시 말하자면 단일 Oracle 노드 O 대신 우리는 n개의 다른 Oracle 노드 $\{O_1, O_2, \ldots, O_n\}$ 집합을 가질 수 있다.
각 Oracle Oi는 다른 Oracle들의 데이터 소스와 겹칠수도 있고 아닐 수도 있는 고유한 데이터 소스 집합에 접근한다.

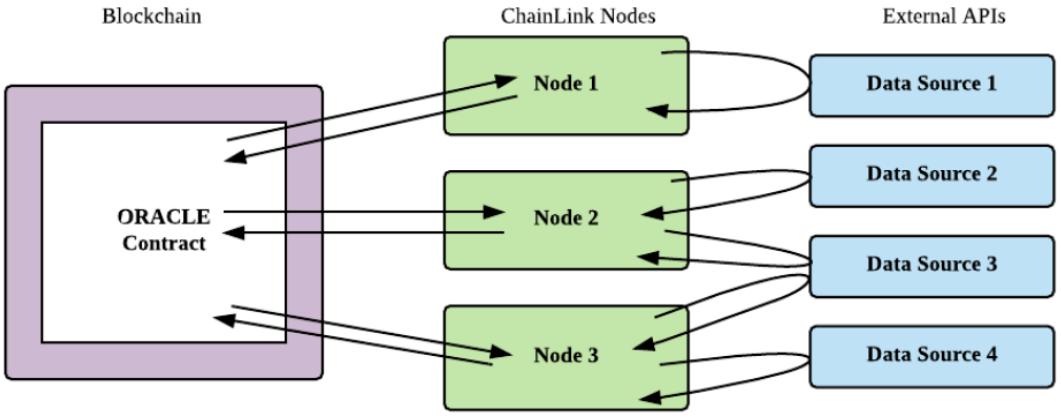


**Figure 3:** Requests are distributed across both oracles and data sources. This figure shows an example of such two-level distribution.

Oi는 데이터 소스들의 응답들을 모으고 자신의 고유한 답변인 Ai를 쿼리 Req로 출력한다.
이 Oracle들중 몇몇은 결함이 있을 수 있다. 그래서 오라클의 정답의 집합들은
$A_1, A_2, \ldots, A_n$ 은 단일하고 대표적인 값인 A로 명백히 집계되어야 한다.
그러나 결함있는 오라클이 주어질 가능성이 있다면, ChainLink에서 집합이 어디서 어떻게 발생할까?

**Initial solution: In-contract aggregation.**
**(**첫번째 해결책 **:** 컨트랙트 내부 집합**)**
ChainLink에서 처음 제안된 솔루션은 컨트랙트 집합이라 불린다.
CHAINLINK-SC는 또 다시 강조하자면 체인 링크의 on-chain 부분을 나타내며 자체 오라클 응답을 집계한다. (대체로 CHAINLINK-SC는 또 다른 집합 계약을 체결할 수 있지만 개념상 단순화 하기 위해 두 요소가 단일 계약을 체결한다고 가정한다.)
다른말로, CHAINLINK-SC는 몇가지 함수 Agg(위에서 설명한 agg와 유사한)
$A = \mathrm{Agg}(A_1, A_2, \ldots, A_n)$ 를 계산하고 결과 A를 USER-SC로 보낸다.

이러한 접근법은 작은 N에 대해 실용적이며 몇가지 분명한 이점이 있다.

- 개념적 단순성 : 오라클이 배포된다는 사실에도 불구하고 단일 엔티티 CHAINLINK-SC는 Agg를 실행하여 집계를 수행한다.
- 신뢰성 : CHAINLINK-SC의 코드를 공개적으로 검사 할 수 있으므로 올바른 동작 확인이 가능하다.(CHAINLINK-SC는 상대적으로 작고 간단한 코드일 것이다.) 또한 CHAINLINK-SC의 실행은 체인상에서 완벽히 볼 수 있다. 따라서 사용자, 즉 USER-SC의 작성자는 CHAINLINK-SC에서 높은 수준의 신뢰를 달성할 수 있다.
- 유연성 : CHAINLINK-SC는 대부분의 원하는 집계 함수 Agg(대부분의 함수, 평균 등)를 구현할 수 있다.

간단하지만, 이 접근법은 새롭고 흥미로운 기술적 과제, 즉 freeloding의 문제를 제시한다.
결함이있는(cheating) 오라클 Oz는 또다른 Oi의 Ai 응답을 관찰 할 수 있고 복사할 수 있다.
이러한 방식으로 오라클 Oz는 데이터 소스를 쿼리하는 비용을 피하여 쿼리 당 비용을 부과할 수 있다.
Freeloading은 데이터 소스 쿼리의 다양성을 손상시킴으로써 보안을 약화시키고 또한 오라클이 빠른 반응을 불필요하게 한다. 즉, 오라클에게는 느린 응답속도와 freeloading은 더 절약적인 전략이다.

우리는 이 문제에 대한 잘 알려진 해결책, 즉 commit / reveal 계획을 제안한다.
첫번째 라운드에서 오라클은 CHAINLINK-SC 암호화 약속을 그들의 응답에 보낸다.
그후 CHAINLINK-SC가 정족수(합의에 필요한 최소 인원의 수)의 응답을 받은 후에, 오라클이 응답을 나타내는 두번째 라운드를 시작한다.

Algorithm 1은 주어진 3f + 1 노드들의 가용성을 보장하는 간단한 순차 프로토콜을 보여준다.
그것은 freeloading을 막기위해 commit / reveal 계획을 사용한다. 오라클의 응답은 해체되어(decommited) 모든 약속이 이루어진 후에만 잠재적인 freeloader에 노출되므로 freeloader가 다른 오라클의 응답을 복사하는 것을 방지(excluding)한다.

On-chain 프로토콜은 블록 시간을 활용하여 동기식 프로토콜 설계를 지원 할 수 있다. 그러나 ChainLink의 경우, 오라클 노드들은 매우 다양한 응답시간을 가질 수 있는 소스에서 데이터를 가져오고 예를들어 Ethereum에서 상이한 가스 가격을 사용함으로써 노드에 의한 해체 시간이 달라질 수 있다. 그러므로 가능한 한 가장 빠른 프로토콜 응답을 보장하기 위해 Alg. 1은 비동기 프로토콜로 설계된다.

여기서 $\mathsf{Commit}_r(A)$ 는 값 A와 증인 r을 나타내는 반면, SID는 유효한 세션 id 집합을 나타낸다. 이 프로토콜은 모든 플레이어 중에서 인증된 채널을 사용한다고 가정한다.

Alg. 1이 성공적으로 종료된다는 것은 쉽게 알 수 있다. 총 3f + 1개의 노드가 주어지면 최대 f개 노드에는 결함이 있으므로, 4단계에서 최소 2f + 1개의 노드가 약속을 보낸다.

또한 A가 Alg.1에서 정확하다는 것을 쉽게 알 수 있다. 단일 값 A에 대한 f + 1 해체(decommitments)중 적어도 하나는 정직한 노드에서 와야한다.

Alg.1을 통한 계약 내 통합은 단기적으로 ChainLink가 지원하는 주된 접근법이 될 것이다. 제안된 초기 구현에서는 보다 정교하고 동시적인 알고리즘의 변형이 포함될 것이다. 우리의 장기적 제안은 부록 A에 있는 Algorithm 2와 3에 명시된 다소 복잡한 OCA(Off-Chain Aggregation) 프로토콜에 반영되어있다. OCA는 체인의 트랜잭션 비용을 최소화하는 오프체인 통합 프로토콜이다. 또한 이 프로토콜에서 오라클 노드에 대한 지급이 포함되어 있으며 freeloader들에 대한 지급을 보장한다.

**Algorithm 1** $\mathsf{InChainAgg}(\{O_i\}_{i=1}^n)$ (code for CHAINLINK-SC)

1: Wait until Req is received from USER-SC.
2: $\mathsf{sid} \xleftarrow{\$} \mathsf{SID}$
3: Broadcast $(\texttt{request}, \mathsf{sid})$.
4: Wait until set $C$ of $2f+1$ messages $(\texttt{commit}, c_i = \mathsf{Commit}_{r_i}(A_i), \mathsf{sid})$ from distinct $O_i$ are received.
5: Broadcast $(\texttt{committed}, \mathsf{sid})$.
6: Wait until set $D$ of $f+1$ distinct valid decommitments $(\texttt{decommit}, (r_i, A_i), \mathsf{sid})$ are received where, for some $A$, all $A_i = A$.
7: Send $(\texttt{Answer}, A, \mathsf{sid})$ to USER-SC.

References

[1] Parity. The Multi-sig hack: A postmortem. https://blog.ethcore.io/the-multisig-hack-a-postmortem/. 20 July 2017.
[2] Gun Sirer. Cross-Chain Replay Attacks. Hacking, Distributed blog. 17 July 2016.
[3] Adi Shamir. “How to share a secret”. In: Communications of the ACM 22.11 (1979), pp. 612–613.
[4] Claus-Peter Schnorr. “Efficient signature generation by smart cards”. In: Journal of cryptology 4.3 (1991), pp. 161–174.
[5] Rosario Gennaro, Stanislaw Jarecki, Hugo Krawczyk, et al. “Secure distributed key generation for discrete-log based cryptosystems”. In: Eurocrypt. Vol. 99. Springer. 1999, pp. 295–310.
[6] R. Canetti. “Universally Composable Security: A New Paradigm for Cryptographic Protocols”. In: FOCS. 2001.
[7] Ran Canetti. “Universally composable security: A new paradigm for cryptographic protocols”. In: Foundations of Computer Science, 2001. Proceedings. 42nd IEEE Symposium on. IEEE. 2001, pp. 136–145.
[8] Douglas R Stinson and Reto Strobl. “Provably secure distributed Schnorr signatures and a (t, n) threshold scheme for implicit certificates”. In: ACISP. Vol. 1. Springer. 2001, pp. 417–434.
[9] John R Douceur. “The sybil attack”. In: International Workshop on Peer-to-Peer Systems. Springer. 2002, pp. 251–260.
[10] Aniket Kate and Ian Goldberg. “Distributed key generation for the internet”. In: Distributed Computing Systems, 2009. ICDCS’09. 29th IEEE International Conference on. IEEE. 2009, pp. 119–128.
[11] Claudio Orlandi. “Is multiparty computation any good in practice?” In: Acoustics, Speech and Signal Processing (ICASSP), 2011 IEEE International Conference on. IEEE. 2011, pp. 5848–5851.
[12] Ittai Anati, Shay Gueron, Simon Johnson, et al. “Innovative technology for CPU based attestation and sealing”. In: Proceedings of the 2nd International Workshop on Hardware and Architectural Support for Security and Privacy. Vol. 13. 2013. url: https : / / software . intel . com / en - us / articles / innovative-technology-for-cpu-based-attestation-and-sealing (visited on 05/23/2016).
[13] Matthew Hoekstra, Reshma Lal, Pradeep Pappachan, et al. “Using Innovative Instructions to Create Trustworthy Software Solutions”. In: Proceedings of the 2Nd International Workshop on Hardware and Architectural Support for Security and Privacy. HASP ’13. Tel-Aviv, Israel: ACM, 2013, 11:1–11:1. isbn: 978-1-4503-2118-1. doi: 10.1145/2487726.2488370. url: http://doi.acm.org/10.1145/2487726.2488370.
[14] Frank McKeen, Ilya Alexandrovich, Alex Berenzon, et al. “Innovative instructions and software model for isolated execution.” In: Proceedings of the 2nd International Workshop on Hardware and Architectural Support for Security and Privacy. 2013, p. 10. url: http://css.csail.mit.edu/6.858/2015/readings/intel-sgx.pdf (visited on 05/23/2016).
[15] Intel. Intel Software Guard Extensions Programming Reference. 2014. (Visited

on 05/23/2016).
[16] GavinWood. “Ethereum: A secure decentralised generalised transaction ledger”. In: Ethereum Project Yellow Paper (2014).
[17] Jack Peterson and Joseph Krug. “Augur: a decentralized, open-source platform for prediction markets”. In: arXiv preprint arXiv:1501.01042 (2015).
[18] Victor Costan and Srinivas Devadas. “Intel SGX Explained”. In: Cryptology ePrint Archive (2016). url: https : / / eprint . iacr . org / 2016 / 086 . pdf (visited on 05/24/2016).
[19] Victor Costan, Ilia A Lebedev, and Srinivas Devadas. “Sanctum: Minimal Hardware Extensions for Strong Software Isolation.” In: USENIX Security Symposium. 2016, pp. 857–874.
[20] Kevin Delmolino, Mitchell Arnett, Ahmed Kosba, et al. “Step by step towards creating a safe smart contract: Lessons and insights from a cryptocurrency lab”. In: International Conference on Financial Cryptography and Data Security. Springer. 2016, pp. 79–94.
[21] Ahmed Kosba, Andrew Miller, Elaine Shi, et al. “Hawk: The blockchain model of cryptography and privacy-preserving smart contracts”. In: S&P’16. IEEE. 2016.
[22] Loi Luu, Duc-Hiep Chu, Hrishi Olickel, et al. “Making smart contracts smarter”. In: Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security. ACM. 2016, pp. 254–269.
[23] Bill Marino and Ari Juels. “Setting standards for altering and undoing smart contracts”. In: International Symposium on Rules and Rule Markup Languages for the Semantic Web. Springer. 2016, pp. 151–166.
[24] Fan Zhang, Ethan Cecchetti, Kyle Croman, et al. “Town Crier: An authenticated data feed for smart contracts”. In: Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security. ACM. 2016, pp. 270–282.
[25] Augur project page. https://augur.net. 2017.
[26] CSA Cloud Controls Matrix. URL: https://cloudsecurityalliance.org/group/cloudcontrols-matrix. 2017.
[27] Mark Flood and Oliver Goodenough. Contract as Automaton: The Computational Representation of Financial Agreements. https://www.financialresearch.gov/working- papers/files/OFRwp- 2015- 04_Contract- as- Automaton-The-Computational-Representation-of-Financial-Agreements.pdf. Office of Financial Research, 2017.
[28] Gnosis project page. https://gnosis.pm. 2017.
[29] Hyperledger Sawtooth. https://intelledger.github.io/introduction.html. 2017.
[30] Abhiram Kothapalli, Andrew Miller, and Nikita Borisov. “SmartCast: An Incentive Compatible Consensus Protocol Using Smart Contracts”. In: Financial Cryptography and Data Security (FC). 2017.
[31] Oraclize project page. http://www.oraclize.it. 2017.
[32] Rafael Pass, Elaine Shi, and Florian Tramer. “Formal abstractions for attested execution secure processors”. In: Eurocrypt. Springer. 2017, pp. 260–289.
[33] Town Crier Ethereum service. http://www.town-crier.org/. 2017.
[34] Florian Tramer, Fan Zhang, Huang Lin, et al. “Sealed-glass proofs: Using transparent

enclaves to prove and sell knowledge”. In: Security and Privacy (EuroS&P), 2017 IEEE European Symposium on. IEEE. 2017, pp. 19–34.
[35] Vitalik Buterin et al. Ethereum white paper. https://github.com/ethereum/wiki/wiki/White-Paper.
[36] JSON Schema. http://json-schema.org/.
[37] Hubert Ritzdorf, Karl Wüst, Arthur Gervais, et al. TLS-N: Non-repudiation over TLS Enabling Ubiquitous Content Signing for Disintermediation. IACR ePrint report 2017/578. URL: https://eprint.iacr.org/2017/578.