

# zk-DEX: Private and Decentralized Exchange Protocol

Onther Inc.

Nov, 2019

## Abstract

zk-DEX는 zk-Dai[2]기반의 프라이버시(Privacy)를 보장하는 탈중앙화된 거래 프로토콜로, 두번에 걸친 조건부 전송(Conditional Transfer)의 올바른 실행을 영지식 증명[23]을 통해 검증하여 거래과정에서 거래 당사자의 어떠한 정보도 노출시키지 않는다. zk-DEX에서 사용자는 블록체인에서 동작하는 탈중앙 거래소(Decentralized Exchange; DEX)[6]에 자신의 거래 내역(어카운트 주소, 토큰 수량)이 노출되는 트랜잭션을 전송하는 대신, 자신이 올바르게 거래과정을 올바르게 수행했음을 영지식 증명을 통해 생성된 증거(Proof)만을 제출하여 거래한다. 이를 통해 거래의 양 당사자들은 제 3자에게 어떠한 정보도 노출시키지 않고 거래를 완료할 수 있다.

## 1 Introduction

비트코인의 탄생 이후로 수많은 암호화폐(Cryptocurrency)들이 만들어지고, 사라지고, 동시에 거래되고 있다. 현재 전체 암호화폐의 시가총액은 약 2000억 달러 규모[5]에 달하며, 대부분의 암호화폐는 블록체인 외부에서 운영되는 중앙화된 거래소(Centralized Crypto Exchange; CCE)에서 거래가 되고 있다[7]. 중앙화된 거래소는 사용자들에게 블록체인에서 직접 트랜잭션을 발생시키지 않고 암호화폐를 서로 거래할 수 있게 하는 편의성을 제공하고, 실물자산과 암호화폐의 교두보 역할을 수행한다. 하지만 중앙화 거래소는 사용자들의 암호화폐를 직접 관리하여 블록체인 외부에서 거래를 체결하기 때문에, 해당 거래소에 어떠한 이유로든 문제가 발생했을 경우 사용자들의 자산에 대한 안전성을 보장할 수 없다[8]는 문제가 끊임없이 제기되어 왔다.

대표적으로 해킹으로 인해 약 85만개의 비트코인을 도난당한 마운트곡스 사례[15]와, 갑작스러운 거래소 대표의 사망으로 고객의 자산이 보관된 지갑에 접근할 수 없게 되어 약 1.5억 달러 규모의 고객 자산이 동결되는 쿼드리가 CX[3]의 사례가 있다.

이러한 거래소 모델의 문제를 근본적으로 해결하기 위해 임의의 집단이 블록체인 외부에서 거래를 체결해주는 것이 아니라, 사용자가 직접 암호화폐에 대한 소유권을 가진 상태에서 블록체인 상에서 트랜잭션을 통해 직접 거래를 수행하는 탈중앙 거래소 모델[6]이 제안되었다. 이러한 특징은 탈중앙 거래소가 기존 거래소 모델의 근본적인 문제점이었던 암호화폐 관리 문제로부터 자유로울 수 있게 하였다. 또한 중앙화 거래소에서 암호화폐를 거래하는 당사자의 신원을 사전에 파악하여 거래 사전 혹은 사후에 이를 검열할 수 있는 반면, 블록체인에서 거래가 이루어지는 탈중앙 거래소의 경우 불특정 다수의 마이너(Miner) 혹은 검증자(Validator) 노드가 이를 처리하기 때문에 검열 저항성 측면에서 더 개선될 수 있었다[8].

하지만 이러한 장점을 갖는 탈중앙 거래소 또한 크게 두가지 문제가 지적되고 있다. 첫번째는 확장성 문제[13]로 블록체인 상에서 동작하는 탈중앙 거래소의 특성상 해당 블록체인의 확장성 문제에서 자유로울 수 없게 된다. 두번째는 프라이버시 문제[18]로 역시 이 또한 탈중앙 거래소가 동작하는 블록체인의 구조적 문제에서 비롯된다. 블록체인 프로토콜 자체에서 프라이버시를 보호하지 않는 이상 블록체인의 모든 트랜잭션과 상태들은 모두에게 공개될 수 밖에 없다. 이로 인해 탈중앙 거래소를 이용하는 모든 사용자들의 거래 기록이 노출되며 프라이버시 문제로 이어지게 된다.

### 1.1 This work

본 연구는 이러한 프라이버시 문제를 해결하는 새로운 탈중앙 거래소인 zk-DEX를 제시한다. 먼저 zk-DEX에서 사용자들은 토큰을 그대로 사용하는 대신, zk-Dai와 같이 토큰을 노트라는 암호화된 형태로 변환하게 된다. 노트는 비트코인[16]의 UTXO와 같이 소유권과 토큰의 수량 등에 대한 정보를 포함한다. 하지만 해당 정보를 그대로 기록하는 것이 아니라 이를 해싱(Hashing)하여 저장하기 때문에 제3자는 이에 대해 알 수 없게 된다. 사용자가 노트를 사용할 때는 영지식 증명(zk-SNARKs)[**zk-snarks**]을 통해 소유권을 증명한다.

사용자들은 이러한 노트를 바탕으로 거래를 진행하게 된다. 거래 과정은 총 세단계로, 거래를 생성(MakeOrder)하고, 거래에 응하고(TakeOrder), 마지막으로 거래를 청산(SettleOrder)하게 된다. 사용자는 각 거래 과정에서 거래 조건에 따라 올바른 노트를 사용 및 생성했음을 역시 영지식 증명을 통해 검증 받는다.

모든 과정에서 사용자들이 제출하는 영지식 증거는 블록체인 외부(off-chain)에서 생성되어 블록체인으로 제출된다. 제출된 증거는 블록체인의 스마트 컨트랙트에 의해 그 유효성이 검증된다. 거래 결과 모든 영지식 증거가 올바르게 구성되었음이 성공적으로 검증되면, 거래에 참여한 사용자에게 일련의 노트가 전송된다. 이를 통해 제3자가 특정 거래에 연관된 거래자의 식별자(e.g address)와, 처리된 거래량에 대해 어떠한 정보도 알 수 없도록 한다.

zk-DEX에서 거래는 탈중앙 거래소 스마트 컨트랙트에 등록된 일련의 거래목록인 오더북(Order book)만을 이용해 이루어진다. 이 때 오더북에는 거래량은 표시되지 않고 오직 가격에 대한 정보만 표시된다. 이를 통해 거래 과정에서 프라이버시를 보장함과 동시에, 시장에 존재하는 모든 거래들을 가격에 따라 확인할 수 있다.

또한 zk-DEX의 모든 거래 과정은 별도의 암호화된 통신채널을 필요로 하지 않는다. 별도의 통신채널을 이용할 경우, zk-DEX와 동일하게 거래 과정에서 프라이버시를 보장할 수 있지만, 이는 사용자에게 추가적인 관리 의무를 부여하게 된다. zk-DEX는 별도의 통신채널을 추가하는 대신, 거래 과정을 기존에 비해 한단계 추가하는 방향을 선택했다.

## 2 Related Work

본 글이 작성된 시점을 기준으로 이더리움[1]과 같이 프로토콜 수준에서 프라이버시가 보장되지 않는 블록체인에서 이를 보장하기 위한 단순 지불 프로토콜에 대한 연구는 다양하게 진행되고 있지만, 거래 프로토콜에 대한 연구는 많지 않은 상황이다. 따라서 프라이버시가 보장되는 단순 지불 프로토콜에 관한 연구를 살펴보고, ZEXE[**zexe**]에서 제시된 프라이버시가 보장되는 탈중앙 거래소인 Private DEX에 대해 살펴보도록 하겠다.

## 2.1 Private Payment

### 2.1.1 Hopper

Hopper[4]는 이더리움 기반의 매우 단순한 형태의 믹서로, 이더의 송금인과 수신인 사이의 관계를 추적 불가능하게 만드는 단순 지불 프로토콜이다. Hopper에서 토큰을 전송하기 위해서는 먼저 Deposit 과정을 수행해야 한다. Deposit을 통해 사용자는 특정 해시값을 등록하고 해당 해시값에 전송을 원하는 만큼의 토큰을 예치한다. 이 때 해시값  $hash(secret, address)$ 는 전송을 원하는 상대방의 주소와 자신만이 알고 있는 일련의 비밀값으로 구성된다. 토큰이 정상적으로 예치되면 해당 해시는 Hopper 컨트랙트의 머클트리에 등록된다. 이후 다시 해당 해시가 머클트리에 있고 해시값을 구성하는 값들이 정확히 무엇인지 알고 있다는 것을 영지식 증명을 통해 증명하면 해시값에 포함된 주소로 토큰을 전송하게 된다. 중요한 것은 이 과정에서 머클트리에 등록된 해시값 중 어떤 값이 사용되었는지 특정할 수 없다는 것이다. 대신 이중지불을 방지하기 위해 해당 해시값의 Nullifier =  $hash(secret)$ 을 스마트 컨트랙트에 등록한다. 하지만 사용자가 적을 경우 프라이버시를 보장하기 어려워지며, 임의의 액수가 아닌 특정 고정액만을 주고 받을 수 있다는 한계가 있다.

### 2.1.2 Heiswap

Heiswap[20]은 Hopper와 같은 단순한 형태의 믹서로, 연결가능한 링 서명(Linkable Ring Signature)[21]과 스텔스 주소(Stealth Address)[19]를 이용하여 이더의 송금인과 수신인 사이의 관계를 추적할 수 없도록 한다. Heiswap에서 이더를 송금하려면 Deposit을 수행해야 한다. Deposit을 통해 이더를 예치하고 원하는 수신 어카운트를 링풀(Ring pool)에 등록하게 되며, 그 결과 hei-token이 생성된다. 수신자는 Withdraw를 통해 자신 소유의 hei-token을 입력하고 자신이 해당 링 서명(Ring signature)[17]를 올바르게 구성할 수 있음을 증명하여 이더를 인출할 수 있게 된다. 하지만 Heiswap은 링풀에 참여하는 사용자의 수가 적을 경우 프라이버시를 높은 수준으로 보장할 수 없게 되며, 항상 동일한 고정액만을 주고 받을 수 있다는 한계가 있다.

### 2.1.3 zk-Dai

zk-Dai[2]는 이더리움 기반의 프라이버시를 보장하는 단순 지불 프로토콜로, 송금인과 수신인의 식별자와 관계, 송금액을 모두 추적 불가능하게 한다. zk-Dai의 핵심은 토큰을 그대로 사용하지 않고 노트의 형태로 변환하여 사용한다는 것이다. 노트는  $[pubKey, value]$ 로 구성되며 블록체인에는 오직 노트의 해시값과 암호화된 노트만 저장된다. 노트를 사용하기 위해서는 노트의 해시를 올바르게 구성할 수 있으며, 해당 노트의 공개키에 대응하는 비밀키를 소유하고 있으며, 올바르게 토큰의 양을 기록했다는 것을 영지식증명을 통해 증명해야 한다. zk-Dai의 노트의 구조는 비트코인의 UTXO와 매우 유사하며, 단지 사용권한에 대한 증명을 영지식 증명을 통해 노트의 구체적인 정보를 노출시키지 않는다는 점에서 zcash[zerocash]의 접근방식과 거의 동일하다. 단, 송금에 사용된 노트 해시가 어떤 것인지는 누구나 식별이 가능하므로, 이를 통해 일련의 노트 생성 기록은 누구나 추적할 수 있다는 한계가 존재한다.

### 2.1.4 Ethereum 9%

Ethereum 9%[11]은 이더리움에서 이더와 토큰들을 익명으로 송금할 수 있게 하는 프로토콜로, 믿블윔블(Mimblewimble)[9], Nullifier, 머클 마운틴 레인지(Merkle Mountain Range)[14]를 활용한다.

Ethereum 9<sup>3</sup>/<sub>4</sub>은 송금인과 수신인의 식별자와 관계, 송금액은 물론 거래에 사용된 암호화된 값(e.g zk-Dai의 note)이 무엇인지도 노출시키지 않는다. Ethereum 9<sup>3</sup>/<sub>4</sub>에서 맴블웜블 트랜잭션들은 머클 마운틴 레인저로 저장되며, 이를 사용하기 위해서는 머클 트리에 포함된 올바른 트랜잭션 인풋을 사용하였다는 것을 영지식 증명을 통해 검증받아야 한다. 또한 이중지불을 방지하기 위해 특정 트랜잭션 인풋이 사용되었다고 기록하는 것이 아닌, 해당 인풋을 대체하지만 정보는 노출시키지 않는 연결 불가능한 태그(Unlinkable tag)가 사용되었음을 기록한다.

### 2.1.5 Zether

Zether[[zether](#)]는 이더리움과 같은 스마트 컨트랙트 플랫폼에서 익명으로 송금이 가능하게 하는 프로토콜이다. 역시 영지식 증명의 일종인 Bulletproofs-based range proof와 ElGamal encryption을 통해 익명성을 보장한다. Zether의 특이사항은 다른 프로토콜과 달리 어카운트 기반의 모델이라는 것이다. 대부분의 프로토콜들은 토큰들을 암호화된 형태(Commitment)로 변환하여 이에 대한 검증을 영지식 증명을 통해 진행하는데, 이 경우 암호화된 토큰들을 별도로 따로 관리를 해야 한다는 부담이 있다. 이러한 비효율을 개선하기 위해 Zether는 어카운트 기반의 모델을 활용한다는 특징이 있다.

### 2.1.6 AZTEC protocol

AZTEC protocol[[22](#)]은 퍼블릭 블록체인에서 프라이버시가 보장된 송금을 가능하게 하는 프로토콜이다. 구체적으로 송금인과 수신인의 신원에 대한 정보, 송금액 등에 대한 정보를 노출시키지 않는다. 사용자들은 토큰들을 암호화된 형태인 아즈텍 노트(Aztec Note)로 변환하여 사용하게 된다. zk-Dai와 유사하게 인풋으로 사용되는 노트들과 아웃풋 노트들의 가치가 차이가 없음을 영지식 증명을 통해 검증하게 된다. 중요한 점은 zk-SNARKs와 같은 방법을 사용하는 대신 동형 암호 연산(Homomorphic arithmetic)과 범위 증명(Range proof)의 결합을 통해 영지식 증명을 자체적으로 구성했다는 것이다. 이와 신뢰하에 진행되는 초기 설정(Trusted Setup)을 통해 효율적으로 증명 및 검증작업을 수행할 수 있게 한다.

## 2.2 Private DEX

### 2.2.1 ZEXE

ZEXE[[zexe](#)]는 트랜잭션 연산에 대한 유효성을 영지식 증명을 통해 검증하여, 트랜잭션에 대한 모든 정보를 숨길 수 있는 프로토콜이다. ZEXE는 앞서 다루었던 프로토콜들과 달리 단순 지불에 대한 것이 아닌 임의의 연산에 대한 유효성과 프라이버시를 보장하는 것에 대한 프로젝트라는 점에서 차이가 있다.

ZEXE에서 상태와 그 상태 전이 함수는 records nano-kernel이라는 형태로 표현이 된다. records nano-kernel은 저장된 데이터인 record와 두개의 임의로 정의된 함수인 birth/death predicate로 구성된다. record를 사용하여 새로운 record'을 생성할 경우 기존 record의 deate predicate가 올바르게 실행되었는지 검증하고, record'의 birth predicate가 올바르게 실행되었는지 검증하는 것을 통해 유효성을 보장한다.

ZEXE에서 활용할 수 있는 예시로 private DEX를 제시하고 있는데, 비 오더북 기반의(Intent based) 거래소 프로토콜이 이에 해당한다. 해당 프로토콜에서는 사용자들이 사전에 별도의 통신을 통해 거래

조건과 필요한 정보를 교환하여 거래를 하게 된다. 따라서 기존 오더북 기반의 탈중앙 거래소에서는 해당 프로토콜을 활용하는데 어려움이 있다.

### 2.2.2 Voxelot zk-dex

Voxelot의 zk-dex[10]는 영지식증명을 활용하여 프라이버시가 보장된 탈중앙 거래소 프로토콜이다. zk-Dai와 매우 유사한 방식을 통해 노트를 이용하여 거래를 하게 된다. 하지만 역시 거래조건에 대한 정보를 공유하고 합의하기 위해 별도의 통신 채널을 두는 것이 필요하다. 따라서 이 역시 오더북 기반의 탈중앙 거래소 모델에 적용이 어렵다.

## 2.3 Comparison

Privacy Support Coverage and Efficiency							
Index	Hopper	Heiswap	zk-Dai	Eth 9¾	Zether	Aztec	zk-DEX
Link	✓	✓	✓	✓	✓	✓	✓
Amount	-	-	✓	✓	✓	✓	✓
Identity	-	-	✓	✓	✓	✓	✓
Input	-	-	-	✓	✓	-	-
Proving Cost	Medium	Low	Medium	High	High	Low	Medium

Table 1: Private Payment

Privacy Support Coverage and Efficiency			
Index	ZEXE	Voxelot	zk-DEX
Link	✓	✓	✓
Amount	✓	✓	✓
Identity	✓	✓	✓
Input	-	-	-
Order-book support	-	-	✓
Proving Cost	Medium	Medium	High

Table 2: Private DEX

**Remark.** 표 1과 2의 *Link*, *Amount*, *Identity*, *Input*은 해당 프로토콜이 어느 범위까지 프라이버시를 보장할 수 있는지에 대한 지표를 나타낸다. *Link*는 송금인과 수신인의 관계를 의미하며, *Amount*는 송금액, *Identity*는 송금인과 수신인의 정보, *Input*은 송금에 사용된 암호화된 값을 의미한다. *Zether*의 경우 별도의 인풋 아웃풋 구조가 없으므로 이에 대한 프라이버시가 보장되는 것으로 간주하였다.

증명 비용(*Proving Cost*)은 증명을 생성하는 비용을 의미한다. 이를 염밀하게 비교하기 위해서는 동일한 환경에서 성능을 측정해야 하지만, 구조적으로 한계가 있기 때문에 각 프로토콜에서 제시한 자료와 논리적 복잡도를 고려하여 산정하였다. *Low*, *Medium*, *High*의 세 가지 등급이 있으며 동일 등급 내에서도 얼마든지 성능에 구체적인 차이가 있을 수 있다. 중요한 점은 프라이버시를 강력하게 보장 할수록 이에 대한 비용도 상승한다는 점이다. 이러한 점을 고려할 때 *Aztec Protocol*은 프라이버시도 적정 수준 보장하면서 연산 비용도 낮은 점을 보였다.

통상적으로 영지식 증명에는 증명 비용뿐 아니라 검증 비용(*Verification Cost*) 또한 중요하게 여겨진다. 하지만 표 1 및 2에 포함하지 않은 이유는 대다수의 프로토콜이 *zk-SNARKs*를 사용하고 있으며, 이 경우 검증 비용이 일정하므로 비교의 의미가 없다. 또한 간결한(*Succinct*) 영지식 증명에서 검증 비용은 증명 비용보다 항상 낮으므로, 병목은 검증이 아닌 증명에서 발생한다고 볼 수 있다. 따라서 오직 증명 비용만을 비교하는 것으로 각 프로토콜의 효율성을 비교해보기에 충분하다고 판단하였다.

### 3 zk-DEX protocol

#### 3.1 Problem Statement

탈중앙 거래소에서 두명의 거래자 앤리스(Alice)와 밥(Bob)이 서로 거래하는 상황을 상정해보자. 앤리스는 자신이 보유한 이더(Ether)를 다이(Dai)와 거래하고자 한다. 이 때 이더의 가격은 200다이/1이더라고하자. 앤리스가 거래하고자 하는 이더의 양은 10이더이며, 밥이 거래하고자 하는 다이의 양은 1000다이이다.

이 경우 일반적인 탈중앙 거래소의 거래과정은 다음과 같이 진행된다. 먼저 앤리스가 탈중앙 거래소 스마트 컨트랙트에 이더를 다이와 거래하는 주문을 등록하며, 가격은 200다이로 기재한다. 동시에 컨트랙트에 10이더를 전송한다. 밥은 해당 주문을 확인한 뒤 1000다이를 컨트랙트에 전송하여 주문한다. 그 즉시 탈중앙 거래소는 10이더와 1000다이를 가격인 200다이/1이더에 맞게 교환 및 청산하는 과정을 수행하며, 앤리스에게는 1000다이를 밥에게는 5이더를 전송하고 잔액 5이더를 앤리스에게 전달한다.

문제는 탈중앙 거래소에서 앤리스가 주문을 등록하는 행위나, 밥이 주문하는 행위가 모두 트랜잭션으로 이루어진다는 것이다. 이러한 트랜잭션들은 재실행을 통한 검증을 위해 블록체인의 모든 참가자에게 공개될 수 밖에 없기 때문에 앤리스와 밥의 거래내역에 대한 정보는 모두 노출되게 된다. 구체적으로 앤리스와 밥의 1) 식별자가 노출되며, 2) 서로 얼마의 이더와 다이를 주고받았는지, 즉 거래량에 대한 정보가 노출되며, 3) 거래에 사용된 토큰의 타입(Token type)인 이더와 다이에 대한 정보가 노출된다.

##### 3.1.1 Objectives

여기서 우리는 별도의 추가적인 암호화된 통신 채널 없이 앤리스와 밥이 각자의 식별자를 노출시키지 않고, 거래량에 대한 정보도 노출시키지 않은채로 탈중앙 거래소를 이용해 거래를 할 수 있는 프로토콜을 설계하고자 한다. 단, 본 프로토콜은 토큰의 타입에 대한 프라이버시를 보장하지는 못한다.

#### 3.2 Notes

zk-DEX에서 모든 자산들은 노트(Note)라는 형태로 표현되며, 오직 노트를 암호화한 형태의 데이터만 블록체인에 저장된다.

**Definition 1.** Note 노트는 토큰의 소유권, 액면가액, 타입등을 나타내는 증권으로, 총 다섯 개의 데이터 필드인  $pk$ ,  $v$ ,  $type$ ,  $vk$ ,  $salt$ 로 구성된다.

$$note = [pk, v, type, vk, salt]$$

-  $pk$ 는 노트의 소유자의 공개키(Public Key) 값이며, 소유권 증명을 위해 사용된다.  $note.pk$ 로 표기한다.

-  $v$ 는 해당 노트의 액면가액을 나타낸다.  $note.v$ 로 표기한다.

-  $tokenType$ 은 노트의 토큰 타입이 무엇인지를 나타낸다.  $note.type$ 로 표기한다.

-  $vk$ 는 노트의 뷰잉키(Viewing Key) 값이며, 이는 거래과정에서 Taker만이 노트를 확인할 수 있도록 암호화하기 위해 사용된다.  $note.vk$ 로 표기한다.

- *salt*는 임의의 랜덤값으로 각 노트가 고유한 값을 갖게 하기 위해 사용된다. *note.salt*로 표기한다.

**Definition 2.** *Note hash* 노트해시는 노트의 각 필드값을 해싱한 값으로, 블록체인에 정보 노출 없이 노트를 저장하기 위해 사용된다.

$$\text{noteHash} = \text{hash}(\text{note.pk}, \text{note.v}, \text{note.type}, \text{note.vk}, \text{note.salt})$$

암호화된 노트는 다음과 같이 표현한다.

$$\text{encNote} = \text{encrypt}(\text{pk}, \text{note})$$

*encNote*는 노트에 포함되는 각 데이터 필드값들을 *pk*으로 암호화한 값으로 노트의 소유권을 이전한 이후 새로운 소유주만이 해당 노트의 데이터를 읽을 수 있도록 하는데 사용된다. *encNote*는 *noteHash*와 함께 스마트 컨트랙트에 기록된다.

### 3.2.1 State of Note

노트에는 *INVALID*, *VALID*, *SPENT*, *TRADING*의 총 4가지 상태가 존재한다. 각 상태의 의미는 다음과 같다.

- *INVALID*: 해당 노트가 생성되지 않았음을 의미한다. 생성되지 않았으므로 사용될 수도 없다.
- *VALID*: 노트가 올바르게 생성되었고, 사용 가능함을 의미한다.
- *SPENT*: 노트가 이미 사용되었음을 의미한다. 더 이상 사용될 수 없다.
- *TRADING*: 노트가 거래에 사용되고 있음을 의미한다. 거래가 완료되기 전까지 사용될 수 없다.

### 3.2.2 Create

노트를 생성하기 위해서는 *Create* 과정을 수행해야 한다. *Create*는 *Create.prove* 과정과 *Create.verify*로 구성되며, 전자는 오프체인에서 노트에 대한 영지식 증명 증거를 생성하는 과정이며, 후자는 해당 증거를 온체인에서 검증하는 과정이다. *Create* 뿐만 아니라 이후 서술할 노트를 다루는 모든 과정은 *prove*와 *verify*로 구성된다. *Create*는 Algorithm 1와 같이 구성된다.

*Create*를 통해 검증하는 것은 노트 생성시에 예치한 토큰의 양과 노트의 액면가가 일치하는지, 또한 예치한 토큰의 타입이 노트의 타입과 일치하는지이다. 검증이 성공하면 *nh*와 *en*가 기록되며 정상적으로 노트를 사용할 수 있게 된다. 또한 해당 노트의 상태는 *INVALID*에서 *VALID*로 변경된다. *Create.verify*를 통해 노트를 생성하는 과정에서 노트의 소유주에 대한 정보는 노출되지 않지만, 검증과정에서 해당 노트의 액면가와 타입이 노출되는 것을 알 수 있다.

### 3.2.3 Spend

*Spend*는 생성한 노트를 전송하는 과정이다. *Spend*는 자신 소유의 *oldNote*를 사용하여 두개의 새로운 노트인 *newNote1*, *newNote2*를 생성하게 된다. *newNote1*는 노트를 전송하기 위해 사용되며, *newNote2*는 주로 전송 후 남은 잔액을 돌려받는 용도로 사용될 수 있다. 물론 노트의 액면가를 모두 사용하여 두명의 사용자에게 전송하는 것 또한 가능하다. *Spend*는 Algorithm 2와 같이 구성된다.

---

**Algorithm 1:** Create

---

**Input:**

- private
- $n$ : note to be created
- public
- $nh$ : hash of  $n$
- $v$ : value of  $n$
- $t$ : toktype of  $n$

**Output:**

$p$ : zero-knowledge proof

**1 Function CreateProve:**

```
2    $nh = \text{hash}(n)$                                 // check note hash with note data
3    $n_v = v$ 
4    $n_{type} = t$                                     // check value and type
5   return  $p$ 
6
```

**Input:**

- $nh$ : hash of  $n$
- $en$ : encrypted  $n$
- $v$ : value of  $n$
- $t$ : type of  $n$

$p$ : zero-knowledge proof

**Output:**

$s$ : bool type value indicating result of verification

**7 Function CreateVerify:**

```
8    $s = \text{verifyProof}(nh, en, v, t, p)$ 
9   return  $s$ 
```

---

---

**Algorithm 2:** Spend

---

**Input:**

- private
  - $on$ : old note to be spent,  $oldNote$
  - $nn1$ : new note 1 to be created,  $newNote1$
  - $nn2$ : new note 2 to be created,  $newNote2$
  - $sk$ : secret key of  $oldNote$

- public

- $oh$ : hash of  $on$
- $nh1$ : hash of  $nn1$
- $nh2$ : hash of  $nn2$

**Output:**

$p$ : zero-knowledge proof

**1 Function** SpendProve:

---

```
2   |    $oh = \text{hash}(on)$ 
3   |    $nh1 = \text{hash}(nn1)$ 
4   |    $nh2 = \text{hash}(nn2)$ 
5   |    $on_{pk} = \text{proveOwnership}(sk)$                                 // check note hash with note data
6   |    $on_v = nn1_v + nn2_v$                                          // prove ownership of note
7   |    $on_{type} = nn1_{type} = nn2_{type}$                                 // check value and type
8   |   return  $p$ 
9
```

**Input:**

- $oh$ : hash of  $on$
- $nh1$ : hash of  $nn1$
- $nh2$ : hash of  $nn2$
- $enn1$ : encrypted  $nn1$
- $enn2$ : encrypted  $nn2$

**Output:**

$s$ : bool type value indicating result of verification

**10 Function** SpendVerify:

---

```
11  |    $s = \text{verifyProof}(oh, nh1, nh2, enn1, enn2)$ 
12  |   return  $s$ 
```

---

*Spend*를 통해 검증하는 것은 전송에 사용된 노트의 액면가와 전송으로 인해 새롭게 생성된 노트들의 액면가의 합과 일치하는지와 노트들의 타입이 일치하는지이다. 또한 해당 노트에 대한 소유권도 검증한다. *Spend*과정이 끝나면  $oh$ ,  $nh1$ ,  $nh2$ ,  $enn1$ ,  $enn2$ 이 기록된다. 또한 *on*의 상태는 *SPENT*로,  $nn1$ 과  $nn2$ 의 상태는 *VALID*로 기록된다.

*Spend*는 *Create*와 달리 과정에서 액면가나 타입에 대한 정보가 전혀 노출되지 않는다. 이를 통해 사용자는 노트에 대한 어떠한 정보도 노출시키지 않고 다른 사용자에게 전송할 수 있다.

### 3.2.4 Liquidate

*Liquidate*는 노트를 다시 토큰으로 변환하는 과정으로 Algorithm 3과 같이 구성된다.

---

#### Algorithm 3: Liquidate

---

##### Input:

- private
  - $n$ : note to be liquidated
  - $sk$ : secret key of  $oldNote$
- public
  - $nh$ : hash of  $n$
  - $v$ : value of  $n$
  - $t$ : token type of  $n$

##### Output:

$p$ : zero-knowledge proof

1 **Function** LiquidateProve:

```
2    $nh = \text{hash}(n)$                                      // check note hash with note data
3    $on_{pk} = \text{proveOwnership}(sk)$                    // prove ownership of note
4    $n_v = v$ 
5    $n_{type} = t$                                          // check value and type
6   return  $p$ 
7
```

##### Input:

- $nh$ : hash of note
- $v$ : value of note
- $t$ : type of note
- $p$ : zero-knowledge proof

##### Output:

$s$ : bool type value indicating result of verification

8 **Function** LiquidateVerify:

```
9    $s = \text{verifyProof}(nh, v, t, p)$ 
10  return  $s$ 
```

---

*Liquidate*에서 검증하는 내용들은 추가적으로 소유권에 대한 검증을 하는 것 외에는 *Create*에서 검증하는 내용 유사하다. *Liquidate* 결과  $n$ 의 상태는 *SPENT*로 기록된다. 또한 *Create*와 마찬가지로 *Liquidate*와 동시에 노트에 포함된 토큰의 양과 타입이 노출된다.

### 3.3 Smart Note

스마트노트는 특수한 형태의 노트로 소유권한을 다른 노트를 통해 증명할 수 있는 노트이다. 다음과 같이 기존 노트의 구조에서  $pk$  필드에 공개키가 아닌 다른 노트의 해시값이 포함된다. 스마트노트를 사용하기 위해서는 스마트노트에 포함된 다른 노트의 소유권을 증명하여야 한다.

**Definition 3.** *Smart Note* 스마트 노트는 소유권한을 다른 노트를 통해 증명할 수 있는 특수한 형태의 노트이다.

$$SmartNote = [noteHash, v, type, vk, salt]$$

스마트노트의 장점은 상대방의 공개키를 몰라도 오로지 노트의 해시값만을 이용해 소유권을 이전할 수 있다는 것이다. 이는 이후에 거래과정에서 상대방의 공개키를 모른채로 거래를 진행하는데 사용된다. 또한 스마트노트에 사용되는 다른 노트의 해시값은 해당 해시값을 알더라도 소유주에 대한 어떠한 정보도 알 수 없기 때문에 사실상 스마트노트의 소유주가 누구인지도 알 수 없게 된다. 따라서 스마트노트를 통해 제 3자가 알 수 있는 정보는 없다.

#### 3.3.1 Convert

*Convert*는 스마트 노트를 일반적인 노트의 형태로 변환하는 과정이다. *smartNote*의 소유권을 다른 노트인 *originNote*를 통해 증명하게 되며, 새로운 일반 노트인 *newNote*를 생성하게 된다. *Convert*는 4와 같이 구성된다.

*Convert*의 핵심은 소유권 증명을 *sn*의 공개키가 아니라 *on*의 공개키값을 이용해 진행한다는 것이다. *Convert*는 단순히 스마트 노트를 일반적인 노트로 변환하는 것이기 때문에 토큰의 양에 변동이 생기지 않는다. *Convert*과정이 끝나면 *nh*와 *enn*가 기록되며, *sn*의 상태는 *SPENT*로 *nn*의 상태는 *VALID*로 기록된다.

---

**Algorithm 4:** Convert

---

**Input:**

- private
  - $sn$ : note to be converted to new note, *smartNote*
  - $on$ : note used to prove ownership of  $sn$ , *originNote*
  - $nn$ : new note to be converted from  $sn$ , *newNote*
  - $sk$ : secret key of  $on$
- public
  - $sh$ : hash of  $sn$
  - $oh$ : hash of  $on$
  - $nh$ : hash of  $nn$

**Output:**

$p$ : zero-knowledge proof

**1 Function ConvertProve:**

```
2    $sh = \text{hash}(sn)$ 
3    $oh = \text{hash}(on)$ 
4    $nh = \text{hash}(nn)$                                      // check note hash with note data
5    $on_{pk} = \text{proveOwnership}(sk)$                    // prove ownership of note
6    $sn_v = nn_v$ 
7    $sn_{type} = sn_{type}$                                 // check value and type
8   return  $p$ 
9
```

**Input:**

- $sh$ : hash of  $sn$
- $oh$ : hash of  $on$
- $nh$ : hash of  $nn$
- $enn$ : encrypted  $nn$
- $p$ : zero-knowledge proof

**Output:**

$s$ : bool type value indicating result of verification

**10 Function ConvertVerify:**

```
11   $s = \text{verifyProof}(sh, oh, nh, enn, p)$ 
12  return  $s$ 
```

---

## 4 Trading

앞서 기본적인 노트의 형태와 구조, 토큰을 노트로 변환하고 노트를 토큰으로 변환하는 과정과 노트를 전송하는 방법에 대해서 다루었다. 이 장에서는 이를 바탕으로 하여 프라이버시가 보장된 거래 프로토콜을 구성한다.

### 4.1 Order

먼저 zk-DEX에서 거래는 다음과 같이 구성된다.

*Order = [makerNoteHash, makerV<sub>k</sub>, sourceToken, targetToken, price, takerNoteHash, parentNoteHash]*

각 항목들이 의미하는 바는 다음과 같다.

- makerNoteHash: maker가 소유한 노트인 makerNote의 해시값.
- makerV<sub>k</sub>: maker의 뷔잉키. takerNote의 암호화에 사용된다.
- sourceToken: makerNote의 토큰 타입으로 maker가 targetToken과 거래하고자 하는 토큰이다.
- targetToken: takerNoteToMaker의 토큰 타입 - targetToken: takerNoteToMaker의 토큰 타입
- price: 1 sourceToken과 교환을 원하는 targetToken의 수
- takerNoteHash: taker가 maker에게 전송한 노트의 해시
- parentNoteHash: takerNoteToMaker의 부모 노트

거래의 각 항목들은 이후 다룰 *MakeOrder*, *TakeOrder*과정을 통해 등록되며, 최종적으로 *SettleOrder*를 통해 거래는 청산되게 된다.

### 4.2 Make Order

먼저 거래를 진행하기 위해서는 거래의 생성자인 Maker가 *MakeOrder*를 통해 거래를 생성해야 한다. *MakeOrder*는 Maker가 소유한 노트인 *makerNote*를 거래에 등록하게 되며 구체적으로 Algorithm 5과 같이 구성된다.

*MakeOrder*가 완료되면 새로운 *order*가 생성되며, 해당 *order*에 *sourceToken* 타입의 노트임이 증명된 *mh*와 *sourceToken*, *targetToken*, *makerV<sub>k</sub>*, *price*가 등록된다. 또한 *mn*의 상태는 *TRADING*로 기록된다. *order*를 등록할 때 거래를 원하는 토큰쌍을 기록해야 하므로 토큰의 타입에 대한 프라이버시는 보장되지 않는다.

### 4.3 Take Order

*TakeOrder*는 Taker가 *MakeOrder*를 통해 생성된 거래에 응하는 과정이다. *TakeOrder*를 통해 Taker는 자신의 소유인 *parentNote*를 사용하여 *takerNote*를 Maker에게 조건부 전송을 하게 된다. *TakeOrder*는 Algorithm 6과 같이 구성된다.

---

**Algorithm 5:** MakeOrder

---

**Input:**

- private
  - $mn$ : maker's note to be listed on order,  $makerNote$
  - $sk$ : secret key of  $mn$
- public
  - $mh$ : hash of  $mn$
  - $t$ : type of  $mn$ ,  $sourceToken$

**Output:**

$p$ : zero-knowledge proof

**1 Function** MakeOrderProve:

---

```
2    $mh = \text{hash}(mn)$                                 // check note hash with note data
3    $mn_{pk} = \text{proveOwnership}(sk)$                 // prove ownership of note
4    $mn_{type} = t$                                      // check type
5   return  $p$ 
```

---

6

**Input:**

- $mh$ : hash of  $mn$   
 $t$ : type of  $mn$ ,  $sourceToken$   
 $mvk$ : maker's viewing key,  $makerVk$   
 $price$ : price of  $sourceToken$  in  $targetToken$   
 $p$ : zero-knowledge proof

**Output:**

$s$ : bool type value indicating result of verification

**7 Function** MakeOrderVerify:

---

```
8    $s = \text{verifyProof}(mh, t, mvk, price, p)$ 
9   return  $s$ 
```

---

---

**Algorithm 6:** TakeOrder

---

**Input:**

- private
  - $pn$ : taker's note used to create  $tn$ , *parentNote*
  - $tn$ : taker's note to be listed on order, *takerNote*
  - $sk$ : secret key of  $pn$
- public
  - $ph$ : hash of  $pn$
  - $th$ : hash of  $tn$
  - $etn$ : encrypted  $tn$
  - $order$ : order to be taken

**Output:**

$p$ : zero-knowledge proof

**1 Function** TakeOrderProve:

---

```
2    $ph = \text{hash}(pn)$ 
3    $th = \text{hash}(tn)$                                      // check note hash with note data
4    $pn_{pk} = \text{proveOwnership}(sk)$                   // prove ownership of note
5    $tn_{pk} = order_{makerNoteHash}$                       // check if it is sent to maker
6    $pn_{type} = tn_{type} = order_{targetToken}$            // check type
7    $etn = \text{encrypt}(order_{makerVk}, tn)$             // check it is encrypted with maker's viewing key
8   return  $p$ 
```

9

**Input:**

- $ph$ : hash of  $pn$
- $th$ : hash of  $tn$
- $etn$ : encrypted  $tn$
- $order$ : order to be taken
- $p$ : zero-knowledge proof

**Output:**

$s$ : bool type value indicating result of verification

**10 Function** TakeOrderVerify:

---

```
11   $s = \text{verifyProof}(ph, th, etn, order, p)$ 
12  return  $s$ 
```

*TakeOrder*가 완료되면 대상 *order*에 *order<sub>targetToken</sub>* 타입임이 증명된 *th*와 *ph*가 등록된다. 주목 해야 할 점은 이 과정에서 Taker가 Maker에게 스마트노트를 이용해 *tn*의 소유권을 이전한다는 점이다. *tn<sub>pk</sub>*는 *order<sub>makerNoteHash</sub>*로 해당 노트 해시의 소유주가 *tn*를 사용할 수 있게 한다. *TakeOrder*가 완료된 시점에서 *tn*의 상태는 *TRADING*으로 기록되기 때문에 Maker는 거래의 마지막 단계인 청산 과정없이 *tn*를 사용할 수 없다. 또한 *pn*의 상태는 *SPENT*로 기록된다.

## 4.4 Settle Order

*TakeOrder*가 완료되면 해당 *order*의 생성자인 Maker는 *SettleOrder*를 통해 거래를 청산하여야 한다. Maker는 *order*에 등록된 *makerNote*와 *takerNote*를 이용하여 *order<sub>price</sub>*에 따라 청산하게 되며, 그 결과 *rewardNote*, *paymentNote*, *changeNote*를 생성하게 된다. *rewardNote*는 Taker에게 주어지는 *order<sub>sourceToken</sub>* 타입의 노트이며, *paymentNote*는 Maker에게 지불된 *order<sub>targetToken</sub>* 타입의 노트이다. *changeNote*는 잔액을 반환하는 노트로 *makerNote*와 *takerNote*의 *v*에 따라 소유주가 결정된다. *SettleOrder*는 Algorithm 7과 같이 구성된다.

*SettleOrder*에 여러가지 검증하는 과정이 있지만 핵심은 Maker와 Taker가 등록한 토큰의 양을 가격을 이용하여 올바른 교환비를 계산하는 것이다. Maker가 등록한 거래량보다 많은 양을 Taker가 등록했다면 청산 후에 Taker에게 잔액이 발생하고 그렇지 않은 경우 Maker에게 잔액이 발생하게 된다. *SettleOrder*가 완료되면 거래는 청산되며 *rh*, *ph*, *ch*가 기록된다.

### 4.4.1 Necessity of Settlement

이후 section 5에서 다시 다루지만, zk-DEX의 추가적인 청산과정은 분명히 거래 당사자가 수동으로 거래를 청산하게 한다는 점에서 단점이 존재한다. 하지만 그럼에도 불구하고 거래 과정에서 프라이버시를 보장하기 위해서는 반드시 이 추가적인 청산 과정이 필수적이다. 그 이유는 어떠한 암호화 알고리즘을 사용하더라도 추가적인 청산과정 없이 Maker가 소유한 암호화된 토큰에 대한 정보를 Taker에게 전달하는 것이 불가능하기 때문이다.

그 이유를 구체적으로 살펴보자. Maker의 추가적인 청산 과정을 없애려면 Taker가 거래에 응하자마자 해당 거래가 자동으로 청산되어야 한다. 하지만 프라이버시를 보장하기 위해서는 거래소 시스템이 해당 거래의 거래량에 대해 알 수 없어야 하므로, 거래소의 자동 청산을 기대할 수는 없다. 따라서 이 경우 Taker가 스스로 거래를 청산할 수 있어야 한다.

청산은 거래 당사자가 전송한 토큰의 양과 토큰의 교환 비율을 나타내는 가격에 맞게 토큰을 올바르게 주고 받는 과정을 의미한다. 즉, 청산을 위해서는 Maker와 Taker가 거래에 등록한 토큰의 양이 정확히 얼마인지 알아야 한다는 것을 의미한다. 하지만 Maker가 거래를 생성하는 시점에서 Taker는 식별할 수 없기 때문에, Maker가 등록한 토큰의 양은 거래 생성 후에 어떠한 Taker도 확인이 불가능하다. 따라서 Taker가 직접 청산을 하는 것은 불가능하다.

물론 Maker가 등록한 토큰의 양이 정확히 얼마인지는 모르지만 올바르게 연산은 가능한 방법을 생각해 볼 수 있다. 다중키 동형암호[12]와 같은 방법을 사용하는 것이 대표적인 예이다. 다중키 동형암호를 이용하면 여러 사용자가 각자 암호화한 값들에 대해 연산을 한 결과가 원본 값에 대해 연산을 한 결과를 동일하게 유지할 수 있다. 동형암호화 함수를 *HH*라고 정의하고, Maker가 거래하고자 하는

---

**Algorithm 7:** SettleOrder

---

**Input:**

- private
  - $mn$ : maker's note listed on  $order$ ,  $makerNote$
  - $tn$ : taker's note listed on  $order$ ,  $takerNote$
  - $rn$ : note to be given to taker,  $rewardNote$
  - $pn$ : note to be given to maker,  $paymentNote$
  - $cn$ : change note,  $changeNote$
  - $sk$ : secret key of  $mn$
- public
  - $mh$ : hash of  $mn$ ,  $th$ : hash of  $tn$ ,  $rh$ : hash of  $rn$ ,  $ph$ : hash of  $pn$ ,  $ch$ : hash of  $cn$
  - $ern$ : encrypted  $rn$ ,  $ecn$ : encrypted  $cn$
  - $order$ : order to be settled

**Output:**

$p$ : zero-knowledge proof

**1 Function** SettleOrderProve**:**

```
2    $mh = \text{hash}(mn)$ ,  $th = \text{hash}(tn)$ ,  $rh = \text{hash}(rn)$ ,  $ph = \text{hash}(pn)$ ,  $ch = \text{hash}(cn)$ 
3    $mn_{pk} = \text{proveOwnership}(sk)$ 
4    $tn_{pk} = mh$ 
5    $rn_{pk} = order_{parentNoteHash}$ 
6   if  $mn_v >= tn_v/order_{price}$  then
7      $rn_v = tn_v/order_{price}$ 
8      $pn_v = tn_v$ 
9      $cn_v = mn_v - tn_v/order_{price}$ 
10     $cn_{pk} = mh$ 
11     $rn_{type} = mn_{type} = cn_{type}$ ,  $pn_{type} = tn_{type}$ 
12  else
13     $rn_v = mn_v$ 
14     $pn_v = mn_v * order_{price}$ 
15     $cn_v = tn_v - mn_v * order_{price}$ 
16     $cn_{pk} = order_{parentNoteHash}$ 
17     $rn_{type} = mn_{type}$ ,  $pn_{type} = tn_{type}$  =  $cn_{type}$ 
18    // check every note is settled correctly
19    if  $mn_v < tn_v/order_{price}$  then
20       $ecn = \text{encrypt}(tn_{vk}, rn)$ 
21      return  $p$ 
22    // check every note is encrypted correctly
```

**Input:**

$mh$ : hash of  $mn$ ,  $th$ : hash of  $tn$ ,  $rh$ : hash of  $rn$ ,  $ph$ : hash of  $pn$ ,  $ch$ : hash of  $cn$

$ern$ : encrypted  $rn$ ,  $ecn$ : encrypted  $cn$

$order$ : order to be settled

$p$ : zero-knowledge proof

**Output:**

$s$ : bool type value indicating result of verification

**23 Function** SettleOrderVerify**:**

```
24   $s = \text{verifyProof}(mh, th, rh, ph, ch, ern, ecn, order, p)$ 
25  return  $s$ 
```

양을  $mv$ , Taker가 거래하고자 하는 양을  $tv$ 라고 하자. Maker는 동형암호화된 값  $HH(mv)$ 로 등록하고, Taker 역시  $HH(tv)$ 를 등록하여 가격에 맞게  $HH(mv * tv)$ 를 연산할 수 있을 것이다.

하지만 문제는 연산의 가능 여부가 아니라 암호화된 값을 복호화하는데에 있다. Maker가 처음에 등록한 암호화된 값  $HH(mv)$ 는 오직 Maker만이 복호화할 수 있는 값이다. 이는 역시 Maker가 거래를 생성하는 시점에서 Taker를 식별할 수 없기 때문에 발생하는 문제인데, Taker가 동형암호를 이용해 청산 자체는 가능하지만 청산 이후에 발생한 암호화된 값을 복호화 할 수 없게 된다. 따라서 이 경우 또한 Taker가 직접 청산을 하는 것이 불가능하다.

#### 4.4.2 Incentive of settlement

zk-DEX에서는 Maker에게 일정한 수수료 수익을 보장한다. 전통적으로 시장에 유동성을 공급하는 Maker는 거래소 차원에서 우대하였다. zk-DEX는 Maker가 이러한 유동성 공급 역할에 대해 거래소가 수행해야 할 청산과정 또한 수행하기 때문에 더 큰 보상을 제공할 필요가 있다. 따라서 Maker는 거래 과정에서 수수료를 지불하는 것이 아닌 거래를 통해 수수료 수익을 얻을 수 있다.

### 4.5 Cancel Order

zk-DEX에서 거래를 중간에 취소할 수 있는 두가지 경우가 존재한다. 첫째는 Maker의 거래 생성 이후 해당 거래에 대한 Taker의 노트 등록 이전에 Maker가 거래를 취소하는 것이다. 둘째는 Taker의 노트 등록 이후에 Maker의 청산이 이루어지지 않을 경우 Taker가 해당 거래를 취소할 수 있는 것이다.

#### 4.5.1 Cancel Order Before Taken

Maker는 거래 등록 이후 Taker가 해당 거래에 노트를 등록하기 전까지 거래를 취소할 수 있다. 이 경우 거래에 등록되었던 노트인 *makerNote*의 상태는 *TRADING*에서 *VALID*로 변경되며, 다시 해당 노트를 정상적으로 사용할 수 있게 된다.

#### 4.5.2 Cancel Order After Taken

Taker는 거래에 노트를 등록한 이후에 Maker의 청산이 일정 시간 내에 이루어지지 않을 경우 역시 거래를 취소할 수 있다. 이 경우 거래에 등록되었던 *takerNote*의 상태는 *TRADING*에서 *SPENT*로, *parentNote*의 상태는 *SPENT*에서 *VALID*로 변경된다.

## 5 Limitations

zk-DEX는 오더북 기반의 탈중앙 거래소 구조에서 프라이버시가 보장되는 거래가 가능하게끔 하였지만 여러 한계점 또한 존재한다. 이 장에서는 zk-DEX의 한계점과 해당 한계들을 완화하거나 개선할 수 있는 가능한 해결책들에 대해 서술한다.

### 5.1 Additional Settlement

zk-DEX는 탈중앙 거래소의 일반적인 거래 과정에서 청산이라는 단계가 추가되었다. Taker가 Maker에게 노트를 전송하는 것으로 거래가 자동으로 청산되지 않기 때문에 Maker는 추가적으로 청산을 수행해야 하고, Taker또한 노트 전송 이후에 청산이 완료되기 전까지 기다려야 하는 문제가 발생한다. 이는 거래가 즉시 처리되기 위해서는 Maker가 기존의 탈중앙 거래소 시스템과 같이 상시 온라인인 상태로 거래를 청산하는 역할을 수행해야 함을 의미한다. 하지만 Maker또한 일반적인 사용자이기 때문에 이러한 역할을 기대하는 것에는 무리가 있다. 따라서 필연적으로 거래의 청산까지는 일정 기간의 지연시간이 발생할 수 밖에 없으며, 이는 zk-DEX의 사용성을 저해하는 결과로 이어지게 된다.

#### 5.1.1 Solution

zk-DEX에서는 청산의 주체인 Maker에게 수수료 수익을 제공하여 경제적 인센티브를 제공한다. 이러한 혜택을 제공하는 대신 Maker의 청산에 대한 역할을 보다 강력하게 강제할 수 있다. Maker는 주어진 시간 내에 청산을 수행할 경우 수수료 수익을 보장받을 수 있지만, 청산을 수행하지 못할 경우 소정의 범칙금을 부과받게 된다. 이를 통해 경제적으로 거래의 청산과정이 빠르게 이루어지도록 권장 및 강제할 수 있다. 하지만 이 역시 청산을 빠르게 하도록 유도하는 것이기 때문에, 악의적으로 청산을 지연시키는 경우나 혹은 네트워크 장애로 인해 청산이 지연되는 문제등을 모두 방지하는 것은 불가능하다는 한계가 있다.

### 5.2 Trivial Order

zk-DEX는 거래량에 대한 프라이버시를 보장하는 대신에 거래를 등록한 Maker이외에 그 누구도 거래량이 얼마인지 사전에 알 수 없다. 때문에 이러한 점을 악용하여 악의적인 Maker는 0에 가까운 낮은 양의 토큰이 포함된 노트를 이용하여 다수의 거래를 생성하여 정직한 사용자들의 이용을 방해할 수 있다. Taker는 거래가 청산되기 전까지 거래에 사용된 노트를 활용할 수 없고, 거래가 청산되더라도 실질적으로 처리된 거래량이 0에 가깝기 때문에 Taker는 해당 거래가 청산되기까지의 시간 동안 실질적인 거래 활동을 할 수 없게 된다.

또한 Taker또한 이러한 점을 악용할 수 있는데, 역시 매우 적은 양의 토큰이 포함된 노트를 이용하여 거래에 응하여 정직한 Maker들의 이용을 방해할 수 있다. zk-DEX에서 거래는 Taker가 등록된 이상 다른 Taker를 수용할 수 없고, 거래가 청산되더라도 역시 실질적인 거래량이 0에 가깝기 때문에 Maker는 해당 거래를 청산하기 전까지 실질적인 거래 활동을 할 수 없게 된다.

### 5.2.1 Solution

의도적으로 매우 낮은 거래량을 이용하여 공격하는 이러한 문제를 해결하기 위한 방법 중 하나는 거래량을 제한하는 것이다. 예를 들어 거래를 등록할 때 최소 거래량을 등록할 수 있게 하는데, Maker와 Taker 모두 해당 거래량보다 많거나 같은 양만큼을 거래에 사용하도록 증명하게 하는 것이다. 이 경우 Maker와 Taker 모두 사전에 거래될 수 있는 최소한의 양을 파악할 수 있기 때문에 위의 문제를 완화할 수 있게 된다. 하지만 이는 노트에 포함된 토큰의 양에 대한 정보의 일부를 노출하게 되어 프라이버시를 다소 희생하게 된다는 한계가 존재한다.

## 6 Further Research

### 6.1 Privacy Enhancement

현재 zk-DEX의 프로토콜에서는 노트의 소유주 신원에 대한 정보와 노트에 포함된 토큰의 양이 노출되지 않지만, 어떤 인풋 노트가 사용되었는지에 대한 정보는 공개가 된다. 따라서 구체적인 정보는 알 수 없지만 Deposit부터 Withdraw까지 일련의 노트 생성 기록 자체는 추적할 수 있게 된다.

이를 개선하기 위해 Hopper, Ethereum 9<sup>3/4</sup>과 같이 머클트리와 연결불가능한 태그를 활용할 수 있다. 모든 노트 해시들을 머클트리로 저장하고, 해당 노트 해시들을 표현하는 연결불가능한 태그를 둔다. 또한 기존에 노트를 사용하기 위해서는 인풋 노트를 공개해야 했던 것과 달리, 해당 노트 해시가 머클트리에 저장되어 있고, 해당 노트의 연결 불가능한 태그가 사용되지 않았다는 것을 영지식 증명을 통해 검증한다. 이를 통해 zk-DEX에서도 노트의 사용기록 자체도 추적 불가능하게 만들 수 있다.

### 6.2 Brokerage

zk-DEX의 거래의 편의성을 개선하기 위해 중개인을 둘 수도 있다. 사용자들은 특정 중개인에게 거래의 요구조건과 함께 노트를 중개인에게 전달한다. 이를 받은 중개인은 위임받은 거래들을 매칭시키거나, 혹은 다른 중개인과 거래를 체결할 수 있다. 브로커는 사용자로부터 노트를 전달받은 시점부터 모든 희망 거래량을 사전에 파악할 수 있으므로, 거래를 추가적으로 청산하는 과정 없이 완료할 수 있다.

중요한 점은 이 과정들이 중개인을 신뢰하지 않더라도 안전하게 수행될 수 있다는 점이다. 사용자들은 중개 서비스를 이용할 때 자신 소유의 노트 해시를 스마트 컨트랙트에 등록하고, 거래 완료 후 생성되는 스마트 노트의 노트 해시를 강제할 수 있게 된다. 즉, 중개인은 사용자들의 자산을 위탁 관리하는 것이 아니며, 단순히 중개하는 역할만 수행할 뿐이다.

물론 중개인을 이용하는 순간부터 프라이버시를 완전히 보장받을 수는 없게 된다. 중개인이 역할을 수행하기 위해서는 각 노트의 정보를 모두 사전에 알고 있어야만 하기 때문이다.

### 6.3 Delegated Proving

zk-DEX에서 모든 거래 과정들은 영지식 증거를 생성해야 하기 때문에 사용자가 연산력이 낮은 장치를 이용할 경우 증거를 생성하는데 소요되는 시간이 매우 길어져 사실상 프로토콜을 사용할 수 없게 될 수 있다.

이에 대한 가능한 해결책 중 하나는 증거를 생성하는 연산을 제3자에게 위임하는 것이다. 증거를 빠른 시간 내에 생성할 수 없는 사용자들은 연산력이 충분한 다른 이에게 증거를 생성하는 필요한 입력값들을 전달하여 연산을 위임할 수 있다. 이 경우 연산을 위임함과 동시에 이에 필요한 입력값들이 모두 전송되기 때문에 연산을 위임받은 이는 거래자의 신원과 거래량등을 파악할 수 있게 된다.

때문에 ZEXE에서 다루었던 것과 같이 위임하는 연산의 범위를 구분할 필요가 있다. ZEXE에서는 트랜잭션에 대한 연산을 모두 위임할 경우 사용자의 트랜잭션 서명도 위임하게 되므로, 비밀키가 노출될 수 있는 점을 지적하였다. 또한 이를 해결하기 위해 서명을 제외한 부분의 연산에 대한 영지식 증명만 제3자가 수행하고, 사용자는 오직 서명에 대한 영지식 증명만 수행하여 이를 재귀 영지식 증명(Recursive Zero Knowledge Proof)[**recursive-zkp**]을 통해 검증하는 방법을 제안하였다.

zk-DEX에서도 이와 유사한 방식으로, 노트의 소유주에 대한 검증과 이를 제외한 부분에 대한 검증을 분리하여 사용자의 비밀키 노출 없이 연산을 위임할 수 있다. 이를 통해 사용자는 프라이버시를 최대한 보장받으면서 효율적으로 연산을 위임할 수 있다.

#### 6.4 Interoperation with other DEXs

zk-DEX는 기존의 다른 DEX 생태계와도 상호 운용될 수 있다. 이 과정에서 zk-DEX는 일종의 다크풀(Dark Pool)로써 다른 DEX에게 익명화된 유동성을 제공할 수 있고, 동시에 다른 DEX로부터 유동성을 제공받을 수 있게 된다.

이는 zk-DEX에 외부의 DEX와 연결하는 일종의 유동성 공급자(Liquidity Provider)인 릴레이어(Relayer)를 두는 방법을 통해 가능하다. 릴레이어는 사용자로부터 거래를 희망하는 노트와 거래조건 등을 전달 받게 되며, 이는 모두 암호화되어 스마트 컨트랙트에 기록된다. 릴레이어가 해당 조건에 맞는 노트를 사용자에게 전달하였을 경우 사용자가 전달한 노트의 소유권과 수수료 수익을 얻게 된다. 릴레이어는 사용자의 거래조건에 맞게 외부 DEX에서 별도로 거래를 진행하여야 하며, 프라이버시를 보장하기 위해 이 과정에서 사용자의 노트를 토큰으로 변환하여 사용할 수 없다. 따라서 해당 거래에 사용된 노트에 대한 정보는 오직 소유주인 사용자와 릴레이어만 알 수 있다. 또한 이 모든 과정들은 스마트 컨트랙트를 통해 신뢰 없이 진행될 수 있다는 점에 유의하라.

## 7 Conclusion

zk-DEX는 영지식 증명을 활용한 두번에 걸친 조건부 거래를 통해 탈중앙 거래소에서 완전한 프라이버시를 지원할 수 있게 하였다. 또한 별도의 암호화된 통신 채널을 두어 사전에 거래 당사자들이 거래조건을 합의하고, 필요한 정보들을 교환하는 대신 오직 탈중앙 거래소만을 통해서 편리하게 거래를 진행할 수 있도록 하였다. 물론 거래의 정산을 거래소 시스템이 아닌 사용자가 직접 해야 하기 때문에, 거래가 완료되기까지 시간이 소요될 수 있다. 또한 악의적으로 매우 낮은 거래량이 포함된 거래를 등록하여 다른 사용자에게 불편을 줄 수 있다. 이러한 점들은 이를 완화할 수 있는 해결책이 있음에도 불구하고 여전히 zk-DEX의 내재적인 문제점이라고 할 수 있다. 하지만 그럼에도 불구하고 zk-DEX는 프라이버시가 보장된 거래를 편리하게 할 수 있기 때문에, 이를 필요로 하는 사용자들에게 분명히 도움이 될 수 있을 것이라 생각한다.

## References

- [1] *A Next-Generation Smart Contract and Decentralized Application Platform.* URL: <https://github.com/ethereum/wiki/wiki/White-Paper>.
- [2] Arpit Agarwal. *ZkDai Design Doc.* URL: [https://docs.google.com/document/d/1z3ZRLLD-wvgERe\\_K05VhqxEJDqiBd3xxfL01pYk7Z0k/edit](https://docs.google.com/document/d/1z3ZRLLD-wvgERe_K05VhqxEJDqiBd3xxfL01pYk7Z0k/edit).
- [3] Doug Alexander. *Crypto CEO Dies Holding Only Passwords That Can Unlock Millions in Customer Coins.* URL: <https://www.bloomberg.com/news/articles/2019-02-04/crypto-exchange-founder-dies-leaves-behind-200-million-problem>.
- [4] argentlabs. *Introducing Hopper: Mobile web-friendly privacy for Ethereum.* URL: <https://medium.com/argenthq/introducing-hopper-mobile-web-friendly-privacy-for-ethereum-d02a8c400dad>.
- [5] *Cryptocurrency market cap.* URL: <https://coinmarketcap.com/>.
- [6] *Decentralized exchange.* URL: <https://en.bitcoinwiki.org/wiki/DEXes>.
- [7] *Decentralized exchange trading volume.* URL: <https://dex.watch/>.
- [8] *DECENTRALIZED VS CENTRALIZED EXCHANGES: ADVANTAGES AND DISADVANTAGES.* URL: <https://hacken.io/research/education/decentralized-and-centralized-exchanges-advantages-vs-disadvantages-aa9a27da4584/>.
- [9] Tom Elvis Jedusor. *MimbleWimble Origin.* URL: <https://github.com/mimblewimble/docs/wiki/MimbleWimble-Origin>.
- [10] Brandon Kite. *zk-dex.* URL: <https://github.com/Voxelot/zk-dex>.
- [11] Wanseob Lim. *Ethereum 9/4: Send ERC20 privately using Mimblewimble and zk-SNARKs.* URL: <https://ethresear.ch/t/ethereum-9-send-erc20-privately-using-mimblewimble-and-zk-snarks/6217>.
- [12] Adriana Lopez-Alt, Eran Tromer, and Vinod Vaikuntanathan. *On-the-Fly Multiparty Computation on the Cloud via Multikey Fully Homomorphic Encryption.* URL: <https://eprint.iacr.org/2013/094>.
- [13] Om Malviya, Madhav Verma, and Tejinder Singh Mor. *Blockchain and Scalability.* URL: [https://www.researchgate.net/publication/327000219\\_Blockchain\\_and\\_Scalability](https://www.researchgate.net/publication/327000219_Blockchain_and_Scalability).
- [14] *Merkle Mountain Ranges.* URL: <https://github.com/opentimestamps/opentimestamps-server/blob/master/doc/merkle-mountain-range.md>.
- [15] *Mt. Gox.* URL: [https://en.wikipedia.org/wiki/Mt.\\_Gox](https://en.wikipedia.org/wiki/Mt._Gox).
- [16] Satoshi Nakamoto. *Bitcoin: A Peer-to-Peer Electronic Cash System.* URL: <https://bitcoin.org/bitcoin.pdf>.
- [17] *Ring signature.* URL: [https://en.wikipedia.org/wiki/Ring\\_signature](https://en.wikipedia.org/wiki/Ring_signature).

- [18] Amir Herzberg Ryan Henry and Aniket Kate. *Blockchain Access Privacy: Challenges and Directions*. URL: [https://www.researchgate.net/profile/Amir\\_Herzberg/publication/326855148\\_Blockchain\\_Access\\_Privacy\\_Challenges\\_and\\_Directions/links/5b80adb292851c1e12304c1Blockchain-Access-Privacy-Challenges-and-Directions.pdf](https://www.researchgate.net/profile/Amir_Herzberg/publication/326855148_Blockchain_Access_Privacy_Challenges_and_Directions/links/5b80adb292851c1e12304c1Blockchain-Access-Privacy-Challenges-and-Directions.pdf).
- [19] *Stealth address*. URL: <https://monero.stackexchange.com/questions/1500/what-is-a-stealth-address/1506#1506>.
- [20] Kendrick Tan. *Introducing Heiswap - An Ethereum Mixer*. URL: [https://kndrck.co/posts/introducing\\_heiswap/](https://kndrck.co/posts/introducing_heiswap/).
- [21] Patrick P. Tsang and Victor K. Wei. *Short Linkable Ring Signatures for E-voting, E-cash and Attestation*. URL: <https://eprint.iacr.org/2004/281.pdf>.
- [22] Dr Zachary J. Williamson. *The Aztec Protocol*. URL: <https://github.com/AztecProtocol/AZTEC/blob/master/AZTEC.pdf>.
- [23] *Zero-knowledge proof*. URL: [https://en.wikipedia.org/wiki/Zero-knowledge\\_proof](https://en.wikipedia.org/wiki/Zero-knowledge_proof).