Provable Security of Public Blockchain Privacy-Preserving Schemes

October 7, 2022 Ziyu Meng

October 7, 2022

Abstract

There are many public blockchain privacy-preserving schemes nowadays, two of which are Zerocash and BlockMaze. Both them provide a way to protect the privacy issue in public blockchains, which use subtle combination of many cryptography primitives like zero-knowledge proofs scheme, encryption scheme, commitment scheme, etc. And thus the problem comes: How to prove the security? In the article, I try to give a simple explanation of how to prove the security of public blockchain privacy-preserving schemes by analyzing the security proof of Zerocash and BlockMaze.

1 Introduction

In public blockchains, transactions are transparent once put on blockchain. Therefore, information like: sender, receiver, transaction amount, extra message are visible by anyone. Privacy is subjective, for someone who wants to hide the account actions would regard "sender" as private, for someone who wants to hide the transaction amount would regard "transaction amount" as private, etc. And because of this, there is no unified definition of privacy in public blockchains.

But in general, we could give an informal definition of privacy in public blockchains: *The privacy in public blockchains includes 3 parts, which are account balance, transaction amount and linkage between senders and recipients.*

In order to protect the privacy, many schemes using zero-knowledge proofs are proposed. Here I take Zerocash and BlockMaze as examples.

2 How Zerocash and BlockMaze works?

Zerocash and BlockMaze protect privacy both by providing a "shield pool", we could image this as a opaque pool. And only what happens above the pool can

be viewed but what happens in the pool can not. In this way, users send "Mint" transaction to convert some public value into private value, and then they trade with private value, finally they withdraw the private value into public value.

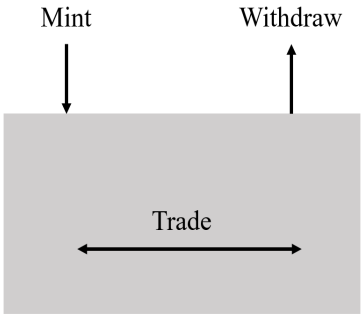

Figure 1: Overview of schemes.

When someone send "Mint" transaction, only sender address and mint amount are revealed (actually more than these two information are revealed, but temporary we only focus on these). When someone send "Trade" transaction, the linkage between sender and recipient is somehow hidden, specifically, it is achieved by commitment and zero-knowledge proofs, so that others could only link the recipient with the senders in the set. In the "Withdraw" transaction, only recipient address and withdraw amount are revealed.

It is pretty general in the description above, we now give a more detailed description. When someone mints, he actually constructs a commitment using his address, certain value amount, serial number and random number, and once the commitment is sent to the blockchain by him, we regard he mints some public value into private value. When someone trades, he construct a new commitment using recipient address, updated value, new serial number and new random number, and the recipient construct a zero-knowledge proof to show that in a set of transaction commitment, there is one for me. When someone Withdraw, he gives a zero-knowledge proof to show how one commitment is constructed, and by which he we regard he withdraw some private value into public value.

It is needed to declare that the real construction is not exactly follows the description above, I hope that could give an intuitive view of how Zerocash and BlockMaze works, for example, there is no "Withdraw" action in Zerocash, but there is still a way for user to withdraw private value into public value by "Pour" action defined in the paper.

3 How to prove the security of Zerocash and BlockMaze

In the last section, I have given an intuitive description of how Zerocash and BlockMaze works, but here the question comes: How do we prove the scheme

is secure? To answer this question, we need to clear first that what the security goal and the adversary capability are.

When we talk about a encryption scheme, we say it is IND-CPA secure or IND-CCA secure, which means that the adversary could not distinguish the ciphertext under the chosen plaintext attack and chosen ciphertext attack. It gives us a sense that the ciphertext could not reveal any information about the plaintext. But why this is important? The answer is related to the usage situation in the reality. In most cases, we encrypt a message because we don not want others see it. Thus, if an encryption scheme could leak some information about the plaintext, the adversary could use it to distinguish ciphertexts and we regard scheme unsecure.

So, what is the security goal of public blockchain privacy-preserving scheme? It is also related to the reality usage situation. For privacy-preserving scheme, we do not want let adversary get anything beyond the publicly-revealed information, by which information adversary may get private information like transaction amount, account balance, who we are trading with, etc. Then, for "Trade" algorithm ("Send" algorithm in BlockMaze and "Pour" algorithm in Zerocash), we want to make sure that adversary could not exploit it to generate an malicious but valid "Trade" transaction, for example, an adversary could construct a valid transaction but use an old serial number. Finally, for the aspect of blockchain itself, we do not want adversary get value out of air, which means all the input value should be equal to the output value plus adversary balance.

After this, we could give a formal definition of public blockchain privacy-preserving scheme's security goal:

Definition 1 *A public blockchain privacy-preserving scheme is secure if it satisfies ledger indistinguishability, transaction unlinkability, transaction non-melleability, and balance.*

For more detailed formal definition of four security goals above, please refer the Zerocash paper or BlockMaze paper.

3.1 Proof of Ledger Indistinguishability

Game-based definition is given as follows:

Definition 2 *For a public blockchain privacy-preserving scheme* Π*, we say that* Π *is* $\mathsf{L\text{-}IND}$ ***secure*** *if for every* $\mathcal{PPT}$ *adversary* $\mathcal{A}$ *and sufficiently large* λ*, there is*

$$\mathsf{Adv}^{\mathsf{L\text{-}IND}}_{\Pi,\mathcal{A}}(\lambda) < negl(\lambda)$$

where $\mathsf{Adv}^{\mathsf{L\text{-}IND}}_{\Pi,\mathcal{A}}(\lambda) := 2 \cdot \Pr[\mathsf{Exp}^{\mathsf{L\text{-}IND}}_{\Pi,\mathcal{A}} = 1] - 1$ *is* $\mathcal{A}$*'s advantage in the* $\mathsf{L\text{-}IND}$ *experiment.*

For experiment L-IND, there is an adversary $\mathcal{A}$ and a challenger $\mathcal{C}$. Given a uniform random bit b, the experiment outputs a bit b' at last. We see that if Π is secure, then the advantage of $\mathcal{A}$ should be negligible, see appendix how the formula of $\mathsf{Adv}^{\mathsf{L\text{-}IND}}_{\Pi,\mathcal{A}}(\lambda)$ comes from.

At the beginning of the experiment, $\mathcal{C}$ samples $b \in \{0,1\}$ at random, next initializes (using pp) two separate Π oracles, provides $\mathcal{A}$ with two ledgers $(L_{\mathsf{Left}}, L_{\mathsf{Right}})$, where $L_{\mathsf{Left}} := L_b$ is the current ledger in $\mathcal{O}^{\Pi}_b$, and $L_{\mathsf{Right}} := L_{1-b}$ is the current ledger in $\mathcal{O}^{\Pi}_{1-b}$. Then, $\mathcal{A}$ sends a pair of queries $\mathcal{Q}, \mathcal{Q}'$, which are consistent (basically, it means both queries are valid, of same type and have same public information). After receiving the queries, challenger then answer it with a_b, a_{1-b}. Then $\mathcal{A}$ outputs a bit $b \in \{0,1\}$, if $b = b'$ then experiment returns 1 otherwise returns 0.

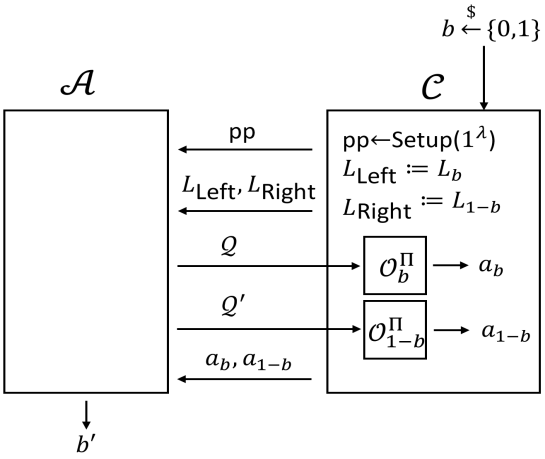

Figure 2: Experiment L-IND between $\mathcal{A}$ and $\mathcal{C}$

We will give a game-based proof to show that the advantage of $\mathcal{A}$ in the experiment is negligible. For game-based proof, it gives us a way to reduction step by step. For more detail information, please refer to the PowerPoint "Security Proofs using the Game-based Methodology" by David Pointcheval [2].

We first give a simulation game $\mathcal{G}_{sim}$ where the oracle and the challenger are all simulated by the simulator. Whenever adversary try to query challenger, it replies in simulated way. Basically, if the query is ***CreateAccount***, it generates a simulated key pair by replace the public key with random string, and records them for later use. Otherwise, if the query is zero-knowledge related, and the adversary queried for ***CreateAccount*** before, then the challenger simulated by replacing the serial number with random string, replacing the commitment with committing a random string, encrypting with new new generated keys. Note that the answer to adversary is computed independently of the bit b, since the process of challenger are all simulated, thus the output b' by adversary must be $\Pr[b = b'] = \frac{1}{2}$, and the advantage of adversary in $\mathcal{G}_{sim} = 0$.

For a series of games, in which the simulator simulates the generation of zero-knowledge proof, generation of ciphertext by encryption scheme, generation of serial number by pseudo random function, generation of commitment by commitment scheme, we will see that the advantage of adversary finally reduced to a combination of advantages for some cryptography primitives. Assuming the primitives are secure with security parameter λ for $\mathcal{PPT}$ adversary, then the advantage of adversary for these primitives are negligible. So that we can get a negligible advantage for adversary in experiment L-IND, which proves the security of our scheme Π in real world.

3.2 Proof of Transaction Non-malleability

We first give the definition of Transaction Non-malleability as follows:

Definition 3 *For a public blockchain privacy-preserving scheme* Π*, we say that* Π *is* TR-NM ***secure*** *if for every* $\mathcal{PPT}$ *adversary* $\mathcal{A}$ *and sufficiently large* λ*, there is*

$$\mathsf{Adv}^{\mathsf{TR\text{-}NM}}_{\Pi,\mathcal{A}}(\lambda) < negl(\lambda)$$

where $\mathsf{Adv}^{\mathsf{TR\text{-}NM}}_{\Pi,\mathcal{A}}(\lambda) := \Pr[\mathsf{Exp}^{\mathsf{TR\text{-}NM}}_{\Pi,\mathcal{A}} = 1] - 1$ *is* $\mathcal{A}$*'s advantage in the* L-IND *experiment.*

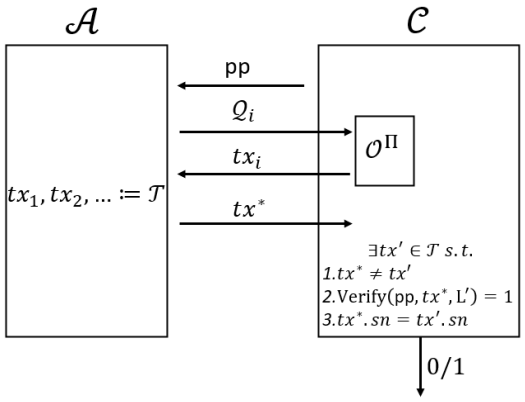

Figure 3: Experiment TR-NM between $\mathcal{A}$ and $\mathcal{C}$

For experiment TR-NM, there is an adversary $\mathcal{A}$ and a challenger $\mathcal{C}$, the experiment terminates with a binary output by $\mathcal{C}$. At the beginning of the experiment, $\mathcal{C}$ initializes (using pp) a Π oracle. And at the end of the experiment, $\mathcal{A}$ sends $\mathcal{C}$ a Trade transaction tx^* and $\mathcal{C}$ outputs 1 if and only if the following conditions holds: Let set $\mathcal{T}$ be transactions which are generated by $\mathcal{O}^{\Pi}$ to reply queries from $\mathcal{A}$, there exist $tx \in \mathcal{T}$ such that: (i) $tx = tx^*$;

(ii) $\mathsf{Verify}(\mathsf{pp}, tx^*, L') = 1$, where L' is the portion of ledger preceding tx; (iii) $tx^*.sn = tx.sn$.

We could see that, if $\mathcal{A}$ wants to win in the TR-NM experiment, the algorithm Verify is vital. Before giving how Verify works, we first see the construction of Trade transaction. (For the proof of transaction non-melleability, we use the scheme in Zerocash, because it separates signature from zero-knowledge proof so that we could prove the security more easily.). For Trade transaction in Zerocash, it could be parsed as $tx^{ZC}_{\mathsf{Trade}} : (root, sn^{old}, cm^{new}, v_{pub}, pk_{sig}, h, \pi, C, \sigma)$, where $sn^{old} = (sn_1^{old}, sn_2^{old})$, $cm^{new} = (cm_1^{new}, cm_2^{new})$, $h = (h_1, h_2)$, $C = (C_1, C_2)$ and $root$ is the root of commitment of sn^{old}, cm^{new} is the new commitment after trading, v_{pub} is the public output in Zerocash, pk_{sig} is the public key for signature, h is a pseudo random number used for transaction non-melleability security, π is zero-knowledge proof, C is ciphertext for encryption. For zero-knowledge proof π could be parses as $\pi : (root, sn^{old}, cm^{new}, v_{pub}, h_{sig}, h)$. For signature could be parses as $\sigma : \mathcal{S}_{sk_{sig}}(m)$. We now give the algorithm Verify as follows:

VerifyTransaction

- INPUTS:
  - public parameters pp
  - a (mint or pour) transaction tx
  - the current ledger L
- OUTPUTS: bit b, equals 1 iff the transaction is valid

1. If given a mint transaction $\mathsf{tx} = \mathsf{tx}_{\mathsf{Mint}}$:
   (a) Parse $\mathsf{tx}_{\mathsf{Mint}}$ as $(\mathsf{cm}, v, *)$, and $*$ as (k, s).
   (b) Set $\mathsf{cm}' := \mathsf{COMM}_s(v \| k)$.
   (c) Output $b := 1$ if $\mathsf{cm} = \mathsf{cm}'$, else output $b := 0$.
2. If given a pour transaction $\mathsf{tx} = \mathsf{tx}_{\mathsf{Pour}}$:
   (a) Parse $\mathsf{tx}_{\mathsf{Pour}}$ as $(\mathsf{rt}, \mathsf{sn}_1^{\mathsf{old}}, \mathsf{sn}_2^{\mathsf{old}}, \mathsf{cm}_1^{\mathsf{new}}, \mathsf{cm}_2^{\mathsf{new}}, v_{\mathsf{pub}}, \mathsf{info}, *)$, and $*$ as $(\mathsf{pk}_{\mathsf{sig}}, h_1, h_2, \pi_{\mathtt{POUR}}, \mathbf{C}_1, \mathbf{C}_2, \sigma)$.
   (b) If $\mathsf{sn}_1^{\mathsf{old}}$ or $\mathsf{sn}_2^{\mathsf{old}}$ appears on L (or $\mathsf{sn}_1^{\mathsf{old}} = \mathsf{sn}_2^{\mathsf{old}}$), output $b := 0$.
   (c) If the Merkle root rt does not appear on L, output $b := 0$.
   (d) Compute $h_{\mathsf{Sig}} := \mathsf{CRH}(\mathsf{pk}_{\mathsf{sig}})$.
   (e) Set $x := (\mathsf{rt}, \mathsf{sn}_1^{\mathsf{old}}, \mathsf{sn}_2^{\mathsf{old}}, \mathsf{cm}_1^{\mathsf{new}}, \mathsf{cm}_2^{\mathsf{new}}, v_{\mathsf{pub}}, h_{\mathsf{Sig}}, h_1, h_2)$.
   (f) Set $m := (x, \pi_{\mathtt{POUR}}, \mathsf{info}, \mathbf{C}_1, \mathbf{C}_2)$.
   (g) Compute $b := \mathcal{V}_{\mathsf{sig}}(\mathsf{pk}_{\mathsf{sig}}, m, \sigma)$.
   (h) Compute $b' := \mathsf{Verify}(\mathsf{vk}_{\mathtt{POUR}}, x, \pi_{\mathtt{POUR}})$, and output $b \wedge b'$.

Figure 4: Verify algorithm

Imagine you are the adversary, when you get a Trade transaction tx from the oracle, what will you do to win the TR-NM experiment? Intuitively, we could just copy a valid transaction and modify some information and let it pass the Verify algorithm. However, we could see in the 4 that, the simulation extractability of zero-knowledge proof π makes sure no one could generate a proof without the witness, and because of that, $\mathcal{A}$ could not modify almost all information of the tx. But, here is a vulnerability $\mathcal{A}$ may use, we see that Verify constructs x by copy $(root, sn^{old}, cm^{new}, v_{pub}, h)$ from tx and computes $h_{sig} := \mathsf{CRH}(pk_{sig})$. Then $\mathcal{A}$ may use h_{sig} to attack.

For the first case, considering there is a collision of hash function, then we could use it to construct a new transaction $tx^* := (root, sn^{old}, cm^{new}, v_{pub}, pk'_{sig}, h, \pi, C, \sigma)$ such that $\mathsf{CRH}(tx^*.pk'_{sig}) = \mathsf{CRH}(tx.pk_{sig})$. Under this case, tx^* satisfies all three conditions, and thus $\mathcal{A}$ wins. It is intuitive that it happens with negligible possibility, because Zerocash uses collision resistant hash function here.

For the later case, we specify $\mathcal{Q}_{CA} = \{a^1_{sk}, a^2_{sk}, ..., a^{q_{CA}}_{sk}\}$ as $\mathcal{A}$'s Trade queries to $\mathcal{C}$ and $\mathcal{Q}_P = \{pk^1_{sig}, pk^2_{sig}, ..., pk^{q_P}_{sig}\}$ as replies of $\mathcal{C}$ containing signature public keys.

Now we consider second case where $\mathcal{A}$ wins and case 1 does not happen. Consider there is $pk''_{sig} \in \mathcal{Q}_P$ such that $pk''_{sig} = pk_{sig}$, let this transaction be tx'', we say that $\mathcal{A}$ may use this to attack. We first prove that $tx^* \neq tx''$ with overwhelming probability. We do this by contradiction. First, since $\mathcal{A}$ wins, there is a transaction tx' in $\mathcal{T}$ such that $tx^* \neq tx'$ but share a same serial number. Therefore, if $tx^* = tx''$, then tx'' and tx' also share a same serial number, since serial numbers are generated by $\mathsf{PRF}(\rho)$, and ρ is a random number, so that $tx^* = tx''$ holds with negligible possibility. Next, since tx'' is valid, then $\mathsf{Verify}(tx''.pk_{sig}, tx''.m'', tx''.\sigma'') = 1$, note that for tx^*, $\mathsf{Verify}(tx^*.pk_{sig}, tx^*.m, tx^*.\sigma) = 1$ also holds. Since $tx^* \neq tx''$ with overwhelming possibility, then $(m, \sigma) \neq (m'', \sigma'')$ with overwhelming possibility. Considering that tx^* and tx'' share one signature public key, we say that $\mathcal{A}$ forgery a new signature σ'' for m'' with public key pk_{sig}. We say that $\mathcal{A}$ win the signature forgery game $\mathsf{SUF\text{-}1CMA}$ game with negligible possibility. So that $\mathcal{A}$ could not construct a valid transaction tx'' by modify (forgery) signature to win.

Consider third case where $\mathcal{A}$ wins and case 1,2 does not happen. Consider there is $h_i = \mathsf{PRF}^{pk}_a(i||h_{sig})$ for some $i \in \{1, 2\}$ and $a \in \mathcal{Q}_{CA}$. We should be clear that why we should consider this case, specifically, why $\mathcal{A}$ could use h to construct an attack. We consider this because h_{sig} may be vulnerable, thus h may also be vulnerable. We first give a game $\mathcal{G}_1$ where $\mathcal{C}$ simulates the key generation of zero-knowledge keys to get the zero-knowledge $\mathsf{trapdoor}$. Because the zero-knowledge scheme is perfect zero-knowledge, thus the possibility of case3 in $\mathcal{G}_1$ is equal to the possibility of case3 in real world. We do this reduction step for the later experiment construction. We next show that, if the possibility of case3 happens in $\mathcal{G}_1$ is not negligible, then we could construct an adversary $\mathcal{B}$ which could distinguish PRF from real random function RAND. We give the construction of experiment $\mathcal{G}_1$ as below. In the experiment, $\mathcal{B}$ first random select an index j and identifies a^{sk}_j, Then, for the interaction between $\mathcal{A}$ and $\mathcal{C}$ (with $\mathsf{trapdoor}$, $\mathcal{B}$ could generates zero-knowledge proof without witness), $\mathcal{B}$ check whether there is some data z is going to be computed as $\mathsf{PRF}_{a^{sk}_j}(z)$, then $\mathcal{B}$ queries oracle $\mathcal{O}$ which is either PRF or RAND. Finally, $\mathcal{A}$ outputs tx^*, and if $\mathcal{O}$ previously evaluated $\mathsf{PRF}_{a^{sk}_j}(i||h_{sig})$ then $\mathcal{B}$ aborts and outputs 1; Otherwise $\mathcal{B}$ evaluated $\mathsf{PRF}_{a^{sk}_j}(i||h_{sig})$ using oracle $\mathcal{O}$ and if result equals h_i, $\mathcal{B}$ outputs 1 otherwise outputs 0. We could conclude that, if case3 happens with non-negligible possibility, then $\mathcal{B}$ could win $\mathcal{G}_1$ win non-negligible possibility, since PRF is collision resistant, we say that case3 happens with negligible possibility.

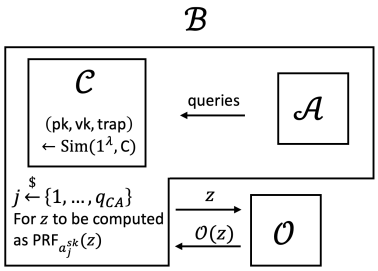

Figure 5: Experiment $\mathcal{G}_1$

Consider fourth case where $\mathcal{A}$ wins and case 1,2,3 does not happen. Consider $h_i \neq \mathsf{PRF}_a(i||h_{sig})$ for all $i \in 1, 2$ and $a \in \mathcal{Q}_{CA}$. We say that case4 happens with negligible possibility. If case4 happens with non-negligible possibility, then we could construct $\mathcal{B}$ that finds a collision of PRF, where $\mathcal{B}$ also uses $\mathcal{A}$ as subroutine.

3.3 Proof of Balance

We first give the definition of Balance as follows:

Definition 4 *For a public blockchain privacy-preserving scheme* Π*, we say that* Π *is* BAL ***secure*** *if for every* $\mathcal{PPT}$ *adversary* $\mathcal{A}$ *and sufficiently large* λ*, there is*

$$\mathsf{Adv}^{\mathsf{BAL}}_{\Pi,\mathcal{A}}(\lambda) < negl(\lambda)$$

where $\mathsf{Adv}^{\mathsf{BAL}}_{\Pi,\mathcal{A}}(\lambda) := \Pr[\mathsf{Exp}^{\mathsf{TR\text{-}NM}}_{\Pi,\mathcal{A}} = 1] - 1$ *is* $\mathcal{A}$*'s advantage in the* BAL *experiment.*

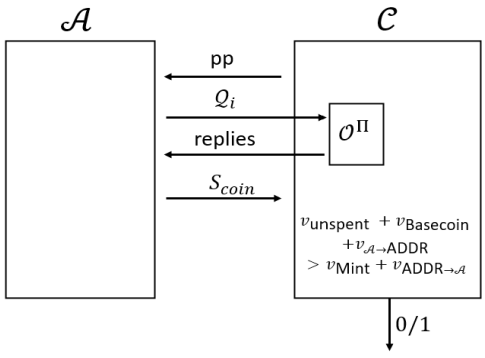

Figure 6: Experiment BAL between $\mathcal{A}$ and $\mathcal{C}$

For experiment BAL, there is an adversary $\mathcal{A}$ and a challenger $\mathcal{C}$, the experiment terminates with a binary output by $\mathcal{C}$. At the beginning of the experiment, $\mathcal{C}$ initializes (using pp) a Π oracle. And at the end of the experiment, $\mathcal{A}$ sends $\mathcal{C}$ a Set of coins S_{coin} and $\mathcal{C}$ outputs 1 if and only if the following conditions holds: coins in S_{coin} where $v_{\mathsf{unspent}} + v_{\mathsf{Basecoin}} + v_{\mathcal{A}\to\mathsf{ADDR}} > v_{\mathsf{Mint}} + v_{\mathsf{ADDR}\to\mathcal{A}}$, where v_{unspent} is the total value of all spendable coins in S_{coin}, v_{Basecoin} is the total value of public outputs placed by $\mathcal{A}$ on the ledger (by Insert query), $v_{\mathcal{A}\to\mathsf{ADDR}}$ is the total value sent by $\mathcal{A}$ to address in ADDR (the address set created by $\mathsf{CreateAddress}$ queries to $\mathcal{O}^{\pi}$, which represents the address of "honest" users), v_{Mint} is the total value of all coins minted by $\mathcal{A}$, $v_{\mathsf{ADDR}\to\mathcal{A}}$ is the total value of payments received by $\mathcal{A}$ from address in ADDR.

Intuitively, there are two methods for $\mathcal{A}$ to spend more public-output money than he owns, first is inserting transactions on the ledger and second is asking honest parties to insert such Insert transactions for him. Query Insert to the oracle is described detailed in appendix B.

We first modify BAL experiment in a way that does not affect $\mathcal{A}$'s view: $\mathcal{C}$ computes for every Trade transaction tx on the ledger L (in oracle) a witness $a = (\mathsf{path}_1, \mathsf{path}_2, \mathsf{c}_1^{old}, \mathsf{c}_2^{old}, \mathsf{addr}_{\mathsf{sk},1}^{old}, \mathsf{addr}_{\mathsf{sk},2}^{old}, \mathsf{c}_1^{new}, \mathsf{c}_2^{new})$ for zk-SNARK instance x corresponding to tx. So that C obtains an augmented ledger $(L, \vec{a})$ where a_i is the witness of instance x_i for i-th Trade transaction in L.

For the modified ledger $(L, \vec{a})$, we say it is *balanced* if the following conditions holds:

1. Each Trade transaction tx contains openings of two distinct input coin commitments and they are the output commitment of Trade or mint transaction that precedes tx on L.

2. No two Trade transactions contain openings of the same coin commitment.

3. For each Trade transaction, the input value implied in input commitment equal to the output value implied in output commitment plus the public output value.

4. For each Trade transaction, if the i-th input commitment is also a output commitment of mint transaction on L, then the value implied in this input commitment is equal to the public value in this mint transaction; if the i-th input commitment is a output commitment of a Trade transaction on L, then the value implied in these two commitment are same.

5. For each Trade transaction that is inserted by $\mathcal{A}$, if the i-th input commitment is the output of earlier mint or Trade transaction, then the i-th output public address is not contained in ADDR.

In light of above, it suffices to argue that augmented ledger induced by the modified BAL experiment is balanced with all but negligible probability. Suppose, by way of contradiction, that $\mathcal{A}$ induces an augmented ledger $(L, \vec{a})$ with non-negligible possibility that is not balanced, in this way, $(L, \vec{a})$ will violates at

one of five conditions above. We give all five violations to show that $\mathcal{A}$ could not give a unbalanced ledger violating five conditions with non-negligible possibility.

First, we assume $\mathcal{A}$ wins and contradicts condition 1 is non-negligible. This means $\mathcal{A}$ inserts a Trade transaction tx such that: (i) with same input commitments; or (ii) there is a input commitment that has no corresponding output coin commitment in any Trade or mint transaction that precedes tx on L. However, for the validity of Trade transaction implies that two input serial numbers are distinct and the witness contains two valid authentication path path_1 and path_2. For (i) if two input commitments are same, then it contradicts with the fact that two input serial numbers implies two distinct openings of commitments. This violates the binding property of the commitment scheme COMM. For (ii), if there is a commitment does not previously appear in L, and because path of this commitment is valid, thus there must be a collision for CRH.

Second, we assume $\mathcal{A}$ wins and contradicts condition 2 is non-negligible. This means two Trade transactions use one same input commitment and revealing two serial number. Since two Trade transaction are valid, thus these two serial number must be different, however, it contradicts the face that one same input commitment is used.

Third, we assume $\mathcal{A}$ wins and contradicts condition 3 is non-negligible. In this case, contradiction is obvious, since the Trade transaction is valid, then the input value implied in input commitment must be equal to the output value implied in output commitment plus the public output value.

Fourth, we assume $\mathcal{A}$ wins and contradicts condition 4 is non-negligible. In this case, L contains a Trade transaction that one of its input commitments opens to be value v and another Trade (or) mint transaction opens this commitment to be value v' that different from v. Since COMM is binding, thus v must be equal to v'.

Fifth, we assume $\mathcal{A}$ wins and contradicts condition 5 is non-negligible. In this case, L contains an inserted Trade transaction that spends the output of a previous transaction whose public address is in ADDR. Since the previous Trade transaction is valid, thus the witness gives a valid output public address, we could show that there is a adversary $\mathcal{B}$ could use $\mathcal{A}$ to distinguish a random oracle from PRF with non-negligible possibility.

References

[1] Guan Z, Wan Z, Yang Y, et al. BlockMaze: An efficient privacy-preserving account-model blockchain based on zk-SNARKs[J]. IEEE Transactions on Dependable and Secure Computing, 2020.

[2] `https://www.di.ens.fr/david.pointcheval/Documents/Slides/s2009_catania.pdf`

A Advantage of $\mathcal{A}$ in L-IND experiment

We here give another definition of advantage in L-IND experiment first:

Definition 5 *The advantage of adversary $\mathcal{A}$ in* L-IND *experiment for public blockchain privacy-preserving scheme Π is:*

$$\mathsf{Adv}^{\text{L-IND}}_{\Pi,\mathcal{A}}(\lambda) = \Pr[\mathsf{Exp}^{\text{L-IND-0}}_{\Pi,\mathcal{A}}(\lambda) = 1] - \Pr[\mathsf{Exp}^{\text{L-IND-1}}_{\Pi,\mathcal{A}}(\lambda) = 1]$$

where $\mathsf{Exp}^{\text{L-IND-0}}_{\Pi,\mathcal{A}}$ *and* $\mathsf{Exp}^{\text{L-IND-1}}_{\Pi,\mathcal{A}}$ *are same as* $\mathsf{Exp}^{\text{L-IND}}_{\Pi,\mathcal{A}}$ *except they return what $\mathcal{A}$ outputs.*

We see that for an innocent adversary $\mathcal{A}$, in the experiment $\mathsf{Exp}^{\text{L-IND}}_{\Pi,\mathcal{A}}$ there is at least possibility of $1/2$ for him to win, thus we could see that advantage of $\mathcal{A}$ is 0, since $\mathsf{Exp}^{\text{L-IND-0}}_{\Pi,\mathcal{A}} = \mathsf{Exp}^{\text{L-IND-1}}_{\Pi,\mathcal{A}} = 1/2$. But, for a normal adversary, it prefer $b' = 0$ or $b' = 1$ more, since he could get some information through the experiment. So by subtracting the possibility in $\mathsf{Exp}^{\text{L-IND-0}}_{\Pi,\mathcal{A}}$ and $\mathsf{Exp}^{\text{L-IND-1}}_{\Pi,\mathcal{A}}$ we could get the advantage of $\mathcal{A}$.

$$\begin{aligned}
&\Pr[\mathsf{Exp}^{\text{L-IND}}_{\Pi,\mathcal{A}}(\lambda) = 1] \\
&\quad = \Pr[b = b'] \\
&\quad = \Pr[b = b' | b' = 1] \cdot \Pr[b' = 1] + \Pr[b = b' | b' = 0] \cdot \Pr[b' = 0] \\
&\quad = \Pr[b = 1 | b' = 1] \cdot \tfrac{1}{2} + \Pr[b = 0 | b' = 0] \cdot \tfrac{1}{2} \\
&\quad = \Pr[b = 1 | b' = 1] \cdot \tfrac{1}{2} + (1 - \Pr[b = 1 | b' = 0]) \cdot \tfrac{1}{2} \\
&\quad = \tfrac{1}{2} + \tfrac{1}{2}(\Pr[b = 1 | b' = 1 - \Pr[b = 1 | b' = 0]) \\
&\quad = \tfrac{1}{2} + \tfrac{1}{2}(\mathsf{Exp}^{\text{L-IND-0}}_{\Pi,\mathcal{A}}(\lambda) = 1] - \Pr[\mathsf{Exp}^{\text{L-IND-1}}_{\Pi,\mathcal{A}}(\lambda) = 1]) \\
&\quad = \tfrac{1}{2} + \tfrac{1}{2} \cdot \mathsf{Adv}^{\text{L-IND}}_{\Pi,\mathcal{A}}(\lambda)
\end{aligned}$$

By the equations above, we could get that:

$$\mathsf{Adv}^{\text{L-IND}}_{\Pi,\mathcal{A}}(\lambda) := 2 \cdot \Pr[\mathsf{Exp}^{\text{L-IND}}_{\Pi,\mathcal{A}} = 1] - 1$$

B Queries to Oracle $\mathcal{O}^{\pi}$ and Replies

The oracle $\mathcal{O}^{\pi}$ is initialized by a list of public parameters pp and maintains states. Internally, $\mathcal{O}^{\pi}$ stores: (i) a ledger L; (ii) a set of address key pairs ADDR; (iii) a set of coins COIN. $\mathcal{O}^{\pi}$ accepts different types of queries and each query causes different updates to L, ADDR, COIN and outputs. For queries Q to $\mathcal{O}^{\pi}$, we describe it oracle's behavior as below:

We should see from the the 7 that, there is a "special" Query named Insert that could insert a valid transaction to the ledger, we should notice that there is no Insert algorithm in real. We see that the Insert query here is for proof of Balance use.

- $Q = (\textbf{CreateAddress})$
  1. Compute $(\mathsf{addr}_{\mathsf{pk}}, \mathsf{addr}_{\mathsf{sk}}) := \mathsf{CreateAddress}(\mathsf{pp})$.
  2. Add the address key pair $(\mathsf{addr}_{\mathsf{pk}}, \mathsf{addr}_{\mathsf{sk}})$ to ADDR.
  3. Output the address public key $\mathsf{addr}_{\mathsf{pk}}$.

  The ledger L and coin set COIN remain unchanged.

- $Q = (\textbf{Mint}, v, \mathsf{addr}_{\mathsf{pk}})$
  1. Compute $(\mathbf{c}, \mathsf{tx}_{\mathsf{Mint}}) := \mathsf{Mint}(\mathsf{pp}, v, \mathsf{addr}_{\mathsf{pk}})$.
  2. Add the coin $\mathbf{c}$ to COIN.
  3. Add the mint transaction $\mathsf{tx}_{\mathsf{Mint}}$ to L.
  4. Output $\perp$.

  The address set ADDR remains unchanged.

- $Q = (\textbf{Pour}, \mathsf{idx}_1^{\mathsf{old}}, \mathsf{idx}_2^{\mathsf{old}}, \mathsf{addr}_{\mathsf{pk},1}^{\mathsf{old}}, \mathsf{addr}_{\mathsf{pk},2}^{\mathsf{old}}, \mathsf{info}, v_1^{\mathsf{new}}, v_2^{\mathsf{new}}, \mathsf{addr}_{\mathsf{pk},1}^{\mathsf{new}}, \mathsf{addr}_{\mathsf{pk},2}^{\mathsf{new}}, v_{\mathsf{pub}})$
  1. Compute rt, the root of a Merkle tree over all coin commitments in L.
  2. For each $i \in \{1, 2\}$:
     (a) Let $\mathsf{cm}_i^{\mathsf{old}}$ be the $\mathsf{idx}_i^{\mathsf{old}}$-th coin commitment in L.
     (b) Let tx_i be the mint/pour transaction in L that contains $\mathsf{cm}_i^{\mathsf{old}}$.
     (c) Let $\mathbf{c}_i^{\mathsf{old}}$ be the first coin in COIN with coin commitment $\mathsf{cm}_i^{\mathsf{old}}$.
     (d) Let $(\mathsf{addr}_{\mathsf{pk},i}^{\mathsf{old}}, \mathsf{addr}_{\mathsf{sk},i}^{\mathsf{old}})$ be the first key pair in ADDR with $\mathsf{addr}_{\mathsf{pk},i}^{\mathsf{old}}$ being $\mathbf{c}_i^{\mathsf{old}}$'s address.
     (e) Compute path_i, the authentication path from $\mathsf{cm}_i^{\mathsf{old}}$ to rt.
  3. Compute $(\mathbf{c}_1^{\mathsf{new}}, \mathbf{c}_2^{\mathsf{new}}, \mathsf{tx}_{\mathsf{Pour}}) := \mathsf{Pour}(\mathsf{pp}, \mathsf{rt}, \mathbf{c}_1^{\mathsf{old}}, \mathbf{c}_2^{\mathsf{old}}, \mathsf{addr}_{\mathsf{sk},1}^{\mathsf{old}}, \mathsf{addr}_{\mathsf{sk},2}^{\mathsf{old}}, \mathsf{path}_1, \mathsf{path}_2, v_1^{\mathsf{new}}, v_2^{\mathsf{new}}, \mathsf{addr}_{\mathsf{pk},1}^{\mathsf{new}}, \mathsf{addr}_{\mathsf{pk},2}^{\mathsf{new}}, v_{\mathsf{pub}}, \mathsf{info})$.
  4. Verify that $\mathsf{VerifyTransaction}(\mathsf{pp}, \mathsf{tx}_{\mathsf{Pour}}, L)$ outputs 1.
  5. Add the coin $\mathbf{c}_1^{\mathsf{new}}$ to COIN.
  6. Add the coin $\mathbf{c}_2^{\mathsf{new}}$ to COIN.
  7. Add the pour transaction $\mathsf{tx}_{\mathsf{Pour}}$ to L.
  8. Output $\perp$.

  If any of the above operations fail, the output is $\perp$ (and $L, \mathsf{ADDR}, \mathsf{COIN}$ remain unchanged).

- $Q = (\textbf{Receive}, \mathsf{addr}_{\mathsf{pk}})$
  1. Look up $(\mathsf{addr}_{\mathsf{pk}}, \mathsf{addr}_{\mathsf{sk}})$ in ADDR. (If no such key pair is found, abort.)
  2. Compute $(\mathbf{c}_1, \ldots, \mathbf{c}_n) \leftarrow \mathsf{Receive}(\mathsf{pp}, (\mathsf{addr}_{\mathsf{pk}}, \mathsf{addr}_{\mathsf{sk}}), L)$.
  3. Add $\mathbf{c}_1, \ldots, \mathbf{c}_n$ to COIN.
  4. Output $(\mathsf{cm}_1, \ldots, \mathsf{cm}_n)$, the corresponding coin commitments.

  The ledger L and address set ADDR remain unchanged.

- $Q = (\textbf{Insert}, \mathsf{tx})$
  1. Verify that $\mathsf{VerifyTransaction}(\mathsf{pp}, \mathsf{tx}, L)$ outputs 1. (Else, abort.)
  2. Add the mint/pour transaction tx to L.
  3. Run **Receive** for all addresses $\mathsf{addr}_{\mathsf{pk}}$ in ADDR; this updates the COIN with any coins that might have been sent to honest parties via tx.
  4. Output $\perp$.

  The address set ADDR remains unchanged.

Figure 7: Oracle $\mathcal{O}^{\pi}$ behavior to queries