Totally not well-formed yet.

A user's assets in Crosswise v1

The following items are the **all and only and exclusive assets** that a user can have on Crosswise v1. These assets changes as the user calls transactions, like transfer, swap, add/remove liquidity, and deposit/withdraw.

- Crss: Crss in user's account (wallet)
- CrssBnb: Crss/Bnb LP token in user's account (wallet)
- CrssBusd: Crss/Busd LP token in user's account (wallet)
- CrssFarm: Crss staked in Masterchef's account, user's share thereof
- CrssBnbFarm: Crss/Bnb LP token staked in Masterchef's account, user's share thereof
- CrssBusdFarm: Crss/Busd LP token staked Masterchef's account, user's share thereof
- CrssVest: Crss staked in xCrss's account, user's share thereof

The amount reference of there items are as follows, respectively:

- Crss: Crss.balanceOf(user)
- CrssBnb: Crss/Bnb.balanceOf(user)
- CrssBusd: Crss/Busd.balanceOf(user)
- CrssFarm: Masterchef.userInfo[0][user].amount
- CrssBnbFarm: Masterchef.userInfo[1][user].amount
- CrssBusdFarm: Masterchef.userInfo[2][user].amount
- CrssVest: xCrss.balanceOf(user)

User's transaction-asset history

Now user's asset state is a tuple of the following form:

```
assets = ( Crss, CrssBnb, CrssBusd, CrssFarm, CrssBnbFarm, CrssBusdFarm, CrssVest
)
```

A user's transaction has the following form:

```
transaction = ( number, txHash, blocknumber, timestamp, actionName,
actionParameters )
```

If a user's transaction 'tx' results in a new assets values 'assets', the relationship is denoted as follows:

```
assets = r(tx)
```

A user's transaction-asset history has the following form:

```
[ (tx0, assets0), (tx1, assets1), (tx2, assets2), ...]

where i-th assets = r(i-th tx)
```

Compensation

Pre-attack snapshot has the following form:

```
{ assets(user, time) | for all user, time = attack time }

where
- a user is a distinct account
- assets(user, time) is the user's assets at the time time
```

Options of compensation

Option1: Compensation amount = pre-attack snapshot

This option compensates a user an amount of **assets(user, attack start time)**

- Pros: Simplest
- Cons: We will be over-compensating. This option assumes that users completely lost their assets by the attack. The pre-attack snapshot is the highest possible loss a user can suffer from the attack. Not all users, though, suffer the highest possible loss. Lots of users sold their sold (liquidated) their assets, often at a lower price than the snapshot price, to minimize the loss.

Option2: Compensation amount = Value(pre-attack snapshot) - Value(post-attack net income) - Value(current asset)

This option compensates a user an amount of **Value(pre-attack snapshot) - Value(post-attack net income) - Value(current asset) post-attack net income** is the net income that the user gained by **selling or buying** after the attack. **current asset** is the assets the user retains currently.

Note1: If the user did not sell or buy after attack, then the compensation amount is **Value(pre-attack snapshot) - Value(current asset)** Note2: If the user has no current assets, then the compensation amount is **Value(pre-attack snapshot) - Value(post-attack net income)** Note3: If the user liquidated all his assets at a fallen price after attack, then the amount is **Value(pre-attack snapshot) - Value(post-attack net income)**, where **Value(post-attack net income) is a positive number**, and we have to compensate less than the snapshot. Note4: If the user sold his assets at a higher price than the snapshot, then the amount is **negative** and we should be compensated by the user, though this is impossible.

- Pros: It IS precise and shout be fare to both the project and community. We have to spend less for compensation than in Option1, as we will be tracking all the value transfers from hand to hand.
- Cons: The function 'Value' is hard to define clearly and acceptable-to-all, especially for liquidity.

Option3: A practical version of Option2

This option will simplify the function 'value' that is introduced in Option2.

- Pros: Practical, yet it can be fair to both the project and community and we will spend less for compensation
- Cons: Precision will be more or less compromised, compared to Option2.

Option3 discussed - how to simplify the 'Value' function

The term Value(pre-attack snapshot)

We know: Compensation amount = Value(pre-attack snapshot) - Value(post-attack net income) - Value(current asset)

This section discusses the term value(pre-attack snapshot).

By the definition, and as the asset items are all and only and exclusive, we get the following **basic equation**:

Value(asset) = Value(Crss) + Value(CrssBnb) + Value(CrssBusd) + Value(CrssFarm) + Value(CrssBnbFarm) + Value(CrssBusdFarm) + Value(CrssVest)

(Please refer to the above definition of the terms Crss, CrssBnb, CrssBusd, CrssFarm, CrssBnbFarm, CrssBusdFarm, and CrssVest.)

Value of Crss

There are three types of Crss value in the **basic equation**:

- Value(Crss): The value of Crss kept in the user's account
- Value(CrssFarm): The value of Crss kept in MasterChef, which the user staked to gain rewards.
- Value(CrssVest): The value of Crss kept in xCrss, which came from the user's rewards and will be vested to the user as time elapsed.

Suppose there is a warning of attack and the user withdraw 'CrssFarm', then 'CrssFarm' will be added to 'Crss'. The user will also withdraw 'CrssVest' to add to 'Crss'. Note we can **ignore** the vesting age of 'CrssVest' **because the age is at most 7 days while the vesting period is 150 days**

Now, **Value(Crss) + Value(CrssFarm) + Value(CrssVest)** is simplified to **Value(amount(Crss) + amount(CrssFarm) + amount(CrssVest))**, which can have a readable form: **Value(Crss + CrssFarm + CrssVest)** if we compromise the rigorous notations.

Valuse of CrssBnb LP tokens

There are two types of CrssBnb LP token value in the **basic equation**:

- Value(CrssBnb)
- Value(CrssBnbFarm)

Again, suppose there is a warning of attack and the user withdraw 'CrssBnbFarm', then it will be added to 'CrssBnb'. Similarly with the above, **Value(CrssBnb) + Value(CrssBnbFarm)** can be simplified to

Value(CrssBnb + CrssBnbFarm)

Valuse of CrssBusd LP tokens

There are two types of CrssBusd LP token value in the **basic equation**:

- Value(CrssBusd)
- Value(CrssBusdFarm)

Again, suppose there is a warning of attack and the user withdraw 'CrssBusdFarm', then it will be added to 'CrssBusd'. Similarly with the above, **Value(CrssBusd) + Value(CrssBusdFarm)** can be simplified to **Value(CrssBusd + CrssBusdFarm)**

The basic equation has now the following form:

Value(asset) = Value(Crss + CrssFarm + CrssVest) + Value(CrssBnb + CrssBnbFarm) + Value(CrssBusd + CrssBusdFarm)

Value of CrssBnb LP token

Suppose the user remove the liquidity represented by the LP tokens at the warning of incoming attack, then

Value(CrssBnb LP) = Value(LP.Crss) + Value(LP.Bnb)

where LP.Crss is the Crss tokens underlying the LP token, and LP.Bnb is the Bnb coins underlying the LP token.

Similarly, **Value(CrssBusd LP) = Value(LP.Crss) + Value(LP.Busd)**

The basic equation has not the following equation:

```
Value(asset) = Value(Crss + CrssFarm + CrssVest)
+ Value([CrssBnb + CrssBnbFarm].Crss) + Value([CrssBnb + CrssBnbFarm].Bnb)
+ Value([CrssBusd + CrssBusdFarm].Crss) + Value([CrssBusd + CrssBusdFarm].Busd)

= Value(Crss + CrssFarm + CrssVest) + Value([CrssBnb + CrssBnbFarm].Crss) +
Value([CrssBusd + CrssBusdFarm].Crss)
+ Value([CrssBnb + CrssBnbFarm].Bnb)
+ Value([CrssBusd + CrssBusdFarm].Busd)

= Value(Crss + CrssFarm + CrssVest + [CrssBnb + CrssBnbFarm].Crss + [CrssBusd +
CrssBusdFarm].Crss)
+ Value([CrssBnb + CrssBnbFarm].Bnb)
+ Value([CrssBusd + CrssBusdFarm].Busd)
```

Note1. The basic equation has three types of value: Crss, Bnb, and Busd, where **the values of Bnb and Busd is almost independent of the attack.** Note2. The Bnb and Busd terms in the equation can be easily computed if only we have the snapshot. Note3. The Crss term in the equation, or the Value of Crss, depends **on the price and amount and the liquidity on the pools.** The liquidity part poses difficulty to compute. So we will bring it from v1 to v2. (See below)

The term Value(post-attack net income)

We know: Compensation amount = Value(pre-attack snapshot) - Value(post-attack net income) - Value(current asset)

This section discusses the term Value(post-attack net income).

post-attack net income is **the net income of a user coming from his post-attack transaction activities**.

- We can retrieve all the transaction records from the block chain.
- A user's post-attack transactions, on the Crosswise v1 DeFi, consists of add/remove liquidity and sell/buy Crss.
- Sell/buy Crss should be via swap.
- post-attack net income should be quoted by Bnb or Busd, because there are only Crss/Bnb and Crss/Busd pools.

So, the **post-attack net income** is

- the net amount of Bnb (or Busd) the user gained by swapping with/for Crss.
- the residue effect is left over in the form of **current asset**

The term Value(current asset)

We know: Compensation amount = Value(pre-attack snapshot) - Value(post-attack net income) - Value(current asset)

This section discusses the term Value(current asset). This should be similar to the discussion of the term Value(pre-attack snapshot), as both deals with the same form 'assets' on Crosswise v1.

The current assets of a user can be retrieved form the block chian.

Composition

We know: Compensation amount = Value(pre-attack snapshot) - Value(post-attack net income) - Value(current asset)

We discussed:

- **Value(pre-attack snapshot), which is decomposed into Value(Crss), Bnb, and Busd**
- **Value(post-attack net income), which has the form Bnb and/or Busd**
- **Value(current asset), which is decomposed into Value(Crss), Bnb, and Busd**

It is clear that:

- The Bnb and Busd, coming from Value(post-attack net income) and Value(current asset), **cancels the Bnb and Busd**, coming from Value(pre-attack snapshot), reducing the amount of Bnb and Busd to compensate to users. This is computable.
- The Value(Crss), coming from Value(current asset) **should cancel the Value(Crss)**, coming from Value(pre-attack snapshot), reducing the amount of Crss to compensate to users. The computability of this item needs some more discussion.

What we achieved in the above discussion is:

- **We checked our intuition of compensation in a bit formal, rigorous discussion**
- It's now more likely that we can **reduce the compensation amount and be fair** to both the community and us.

Study is going on, until we get program code.