Payshuffle: Adjusting output amount of Cashshuffle without introducing linkage and creating liquidity via improving mechanism from Joinmarket

@im_uname v0.1

Motivation

The Coinshuffle protocol, based on Coinjoin first descibed by Gregory Maxwell and specified in detail by Ruffing et. al. in 2014 is a powerful privacy tool that breaks linkage between bitcoin outputs, implemented on the BTC chain (Shufflepuff on Mycelium) and BCH chain (Cashshuffle on Electron-cash). The Coinshuffle protocol breaks linkage in “shuffled” outputs by adjusting one output per participant to identical amounts in the same transaction, hence obfuscating the linkage of each output to their parent inputs; given n participants, there will be n identical-amount outputs, which can belong to any of the participants informatically without additional information.

Coinshuffle, though powerful and trustless in nature, introduces a complication in usability: in order to improve matching liquidity, in practical implementations only a few standardized amounts of shuffled outputs are accepted. This creates additional complications when used in practice: a shuffled output cannot be expected to match a payment request exactly, making it inevitable that a change will be left over. The change will eventually have to be either merged with another input from the same owner, creating linkage to said input, or given away to a party that does not specify amount (such as donations), increasing cost of anonymity.

Joinmarket, an initiative by @belcher, attempts to get around the liquidity and amount limitations by implementing a market of “takers” and “makers”: “takers” can specify Coinjoin amounts, while paying “makers”, who provide liquidity, a small fee in their respective change outputs. As we will see below, though, this introduces a significant weakness in linkage analysis if the payee is compromised. In order to mitigate this weakness it is commonplace for Joinmarket users to seek “tumbling” several times before using the output for payments; this unfortunately removes its usefulness for mass adoption, as the obfuscating transaction can no longer be directly used for payments.

Contribution

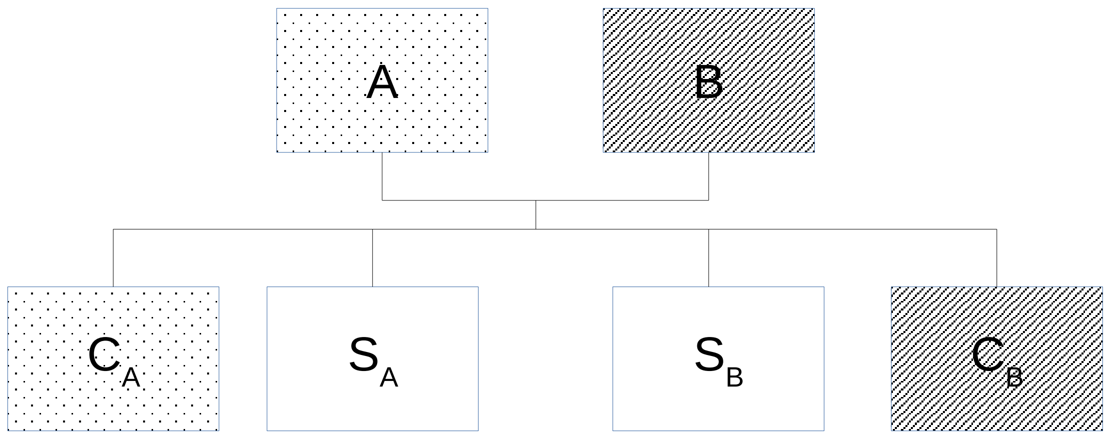
I propose a method, dubbed Payshuffle, that builds on the concept of Joinmarket by introducing a third output, separate from the change and payment outputs, that pays the “makers” providing liquidity. The mechanism prevents tracing of payments from payee to change addresses, hence making it suitable for direct deployment in paying wallets, unlike Joinmarket. Unlike Coinshuffle, the shuffled output amount can be specified, hence making it usable for mass adoption.

The problem

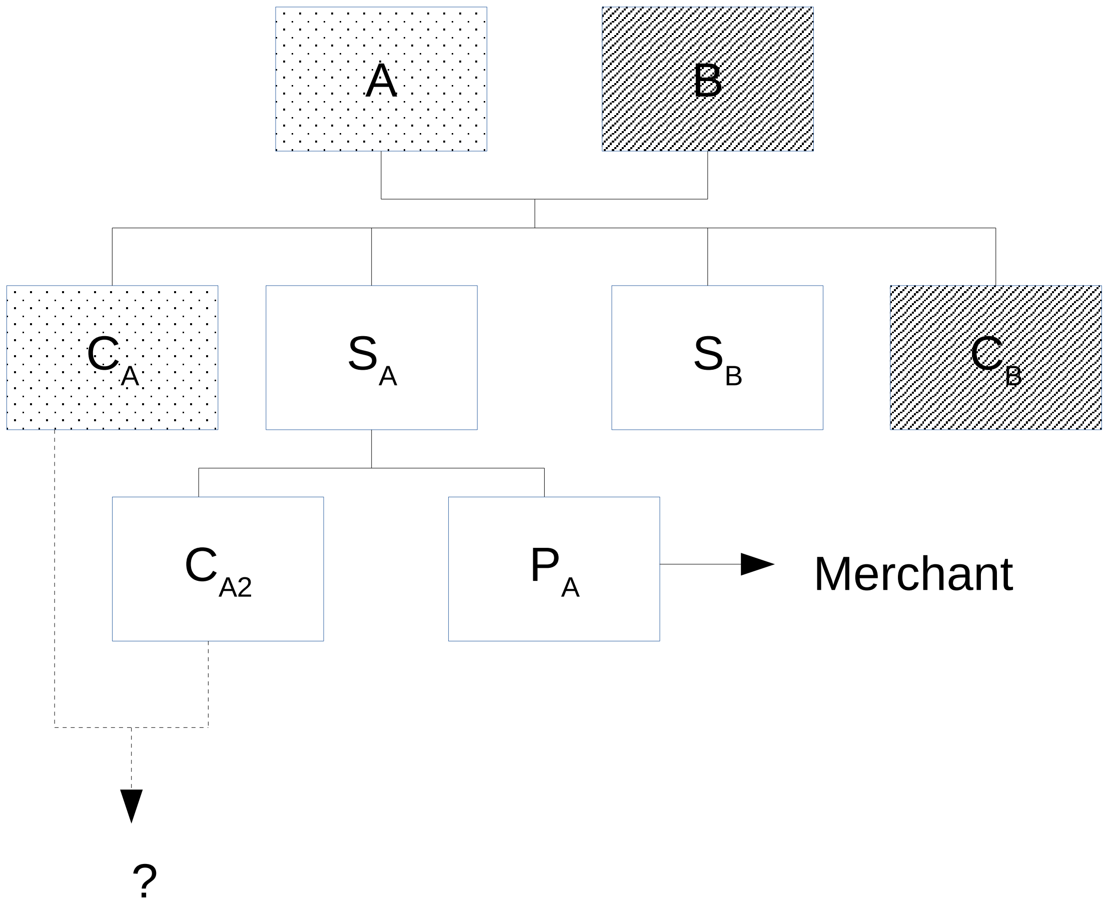
Coinshuffle

The draft here only describes the algorithm to construct output amounts and linkage implications. For trustless network peer-discovery and information exchange mechanism, refer to Ruffing et. al. 2014.

We begin by revisiting the outputs of Coinshuffle, consider Alice and Bob having inputs A and B each they wish to anonymize.

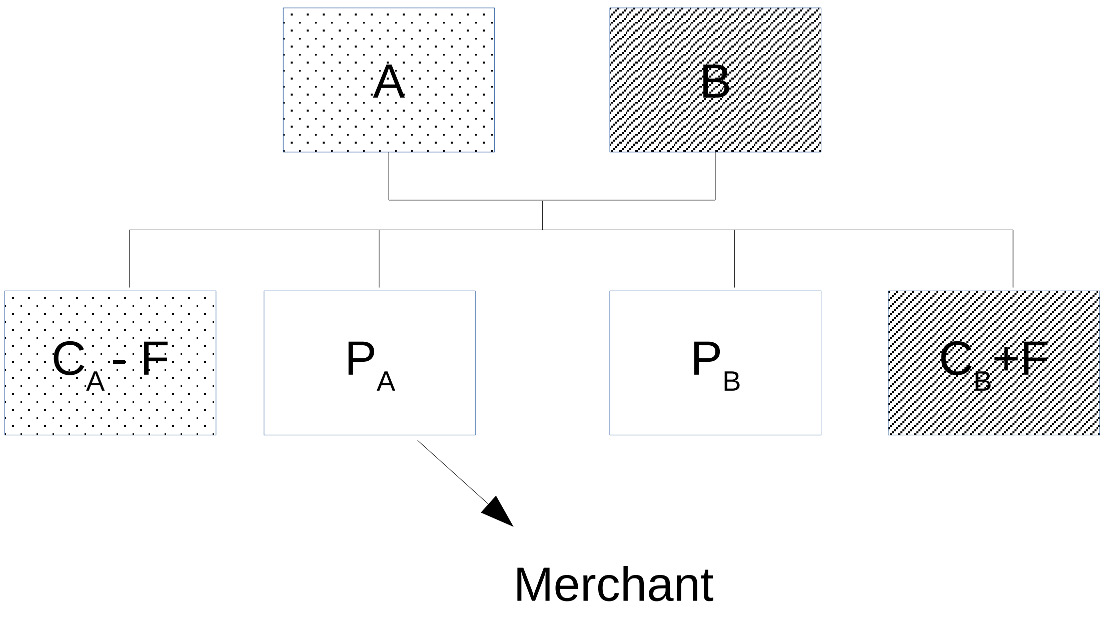


In the figure, the two shuffled outputs S have identical amounts, hence an observer surveying the blockchain cannot tell which output belongs to Alice or Bob. The change addresses C_A and C_B can be linked to the original inputs, indicated by their shades. The amounts are not adjustable though, so after Alice makes use of her shuffled coins, she now faces a dilemma:



Alice's change C_{A2} is now too little to spend. She must then either discard it (incurring cost), or combine it with her other outputs as shown above, creating linkage and destroying privacy of even P_A.

Joinmarket



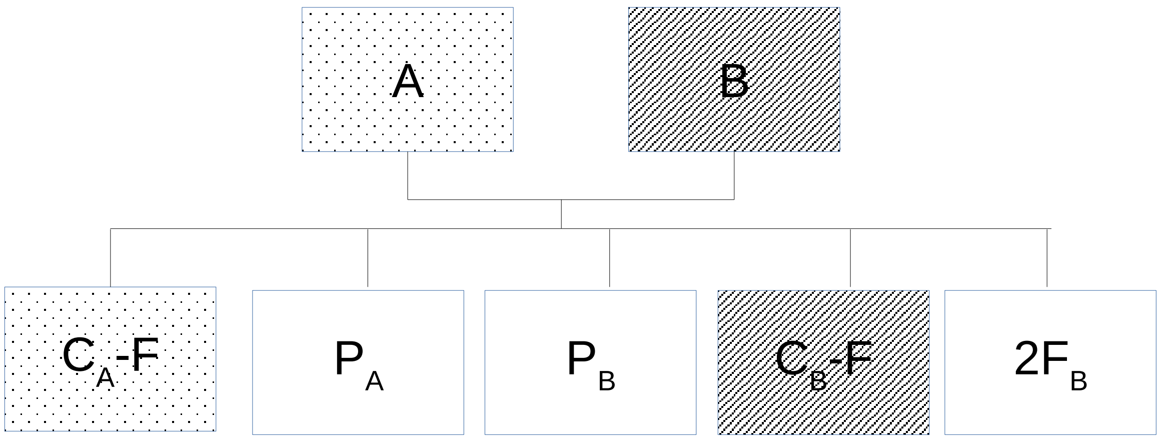
In contrast to Coinshuffle, Joinmarket introduces a “taker” (Alice in this figure) and “maker” (Bob in this figure). Alice specifies her desired output amount PA, and Bob complies, getting paid a small convenience fee F for his liquidity. Since Alice is now able to specify output amount P_A, the merchant can be paid! If desired, Alice can pay multiple makers even, making her unlinked P_A obfuscated among even more peers.

This method, though, have a glaring weakness: None of the makers are likely to pay merchants most of the time; the likelihood to get paid passively without risk dictates that they will simply offer their coins and tumble over and over again (i.e. P_B will likely not go to a merchant). Furthermore, their linked change addresses can be readily identified via the attached fee F. Hence if the merchant is compromised, the payment can, in practice, be traced to Alice’s output $C_{A\,-}$ F, simply by the fact that $P_A + C_A - F < A$ while $P_B + C_B + F > B$. The fee-paying party is the one doing the payment.

To mitigate this, Joinmarket takers typically go through several tumbling steps before using their coins, preventing the market from being adopted for direct payments, as well as reducing the number of takers willing to pay for the trouble and hurting liquidity as a whole.

The Method

In order to solve the dilemmas above, I propose a method to add another output to obfuscate linkage in a single transaction. Consider the following figure:



Just like in Joinmarket, Bob is compensated by a small fee; except the fee is now no longer included in his linkable change (his linkable change is, in fact, *deducted* the same amount of fee as Alice!). Rather, a new output $2F_B$ is created, under Bob's control. The new output that includes nothing but fees cannot be immediately linked to either Alice or Bob; all we know is one of them is the maker and controls the output, but we do not know which.

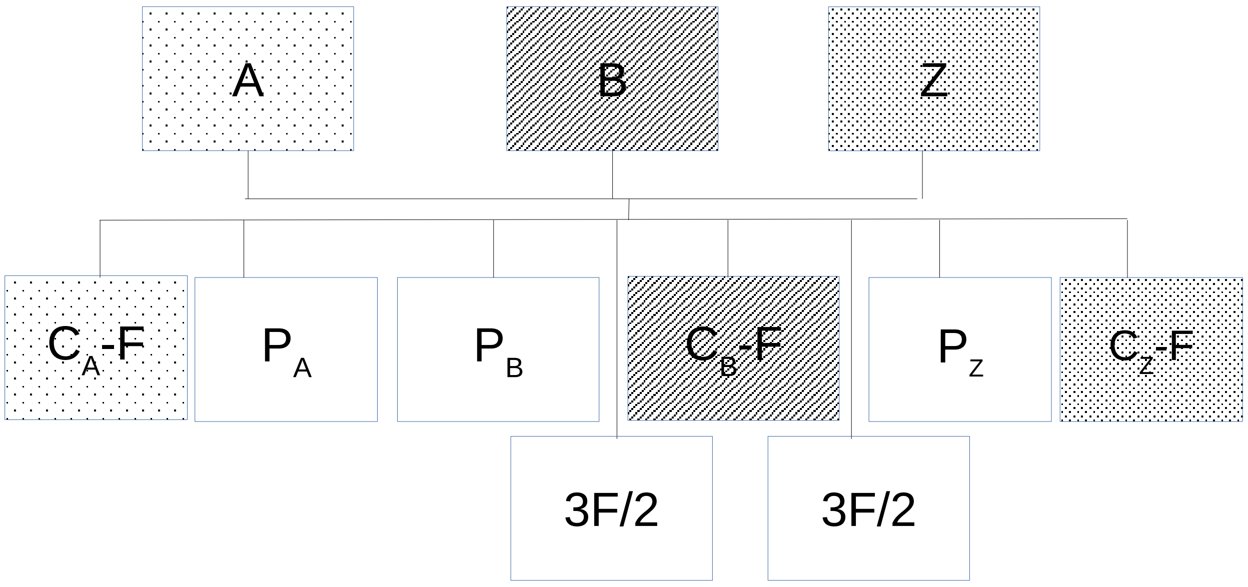
By deducting equal amounts from the linkable outputs C_A and C_B , we have additionally made sure that a merchant paid by P_A now cannot deduce that the most likely person who has paid him (the taker) must be Alice. The one-step transaction is therefore obfuscated (taint reduced by 50%) when directly used to pay merchants.

The fee output $2F_B$ can be made arbitrarily low as long as it is worth the transaction fee of sweeping it and above the dust limit; note that unlike Joinmarket, it is an additional output that needs to be swept, hence it cannot be low beyond dust limit and transaction fee limitations, and is best used on a blockchain where the fee is consistently low (such as Bitcoin Cash).

Also note that Alice's obfuscation can be broken if Bob subsequently proceeds to join $2F_B$ and C_B-F into a single output; it is therefore optimal for Bob to keep his (unlinked) fee outputs in a separate wallet from his linkable change. Keeping the fee output separate yields additional benefits; after collecting enough fees from being a maker, Bob can sweep the fees into a single output that is unlinked to himself, and proceed to use it with privacy benefits.

Additional participants

Like Coinshuffle which is trivially scalable to more than two parties, Payshuffle can also acquire additional participants to improve privacy for all involved. However, unlike Coinshuffle, only one party in each transaction can be the "taker" who specified the payment output, the rest will have to be makers that simply provide liquidity and privacy for a fee. Consider Alice (taker), Bob and Zach (both makers):



In this case the payment P_A is obfuscated via ambiguous linkage to C_A, C_B or C_Z, maintaining privacy. Bob and Zack gets paid 3F/2 each in unlinked fee outputs.

Considering the low fees on the Bitcoin Cash chain, this should be cost-effective for Alice to make a Shuffled payment involving a high number of participants, considering the eventual change she might have to discard in Coinshuffle for privacy can both be a hassle and be much higher than the compensation she pays to makers.

Wallet integration

In order to drive mass adoption of the privacy system, I propose that a common wallet (e.g. Electron-cash) can be modified to do the following:

1. At start, create two wallets. One of the wallets is a "common" wallet used for most private payments, the other are used to strictly collect fees until sweeping, and cannot be linked via depositing or joining inputs with the common wallet conveniently.

2. When the common wallet received a deposit, it begins separating said deposit into several outputs. The owner can specify how much of the contents he wishes to put into a maker pool; the wallet automatically seeks takers while idle with mechanisms similar to Joinmarket and Coinshuffle, and will continuously deposit fees into the fee wallet from such activities.

3. When the owner desires an immediate private payment, he/she uses available funds from the common wallet and becomes a taker instead.

4. Once in a while, the option of "pay from fees" becomes available; the owner can use the accumulated fees in his fee wallet to make an unlinked private payment. As each fee output is expected to be in very small increments, the payment can utilize only the needed number of outputs, generating very minimal change. For maximum privacy, the owner can even specify a donation address of his desire to remove that small change.

Limitations

Pool poisoning

A possible problem with Coinshuffle is that a dedicated attacker with patience and resources can put up large amounts of coins to "poison" the pool; by controlling most liquidity available, the attacker can isolate and link individual shufflers with ease. This is further complicated by the fact that Coinshuffle can be inconvenient to use, reducing the number of "honest" users in the pool making the attack easier.

Joinmarket improves upon the liquidity problem by providing an incentive for makers: as profit-minded makers become available, it raises the difficulty for an attacker to control a majority of the pool. Its difficulty to use and remain private in practice, however, limits the effectiveness of this measure.

As Payshuffle is designed to allow one-step obfuscation and payment, we expect it to be usable in popular wallets directly; automated makers integrated into such wallets shall provide a larger pool of liquidity than the two previous methods, further raising difficulty for a poison attacker.

Denial of service

Since a Coinjoin transaction needs to be passed around and signed by all involved parties in order to be broadcast, an enterprising attacker can perhaps complicate the shuffle process by refusing to sign either as a maker or a taker, denying the payment experience and hindering adoption. As a maker, the attacker can directly prevent payment; as a taker, the attacker can tie up valuable maker liquidity, denying them profits while leaving less liquidity for potential takers.

A possible mitigation for such an attack is for clients to remember failed transactions, and the exact input party who refuse to sign such a failed transaction. By lowering the priority of such outputs when choosing partners, a DoS attacker going against a sufficiently large pool of liquidity can be expected to requie rotating among more outputs as his attacks go on to be known by more clients.