Micropayments, Now!
Probabilistic Micropayments in Bitcoin

Jeremy Rubin

November 26, 2015

In this paper, I introduce a probabilistic micropayments scheme which works without any modification to the Bitcoin Core Protocol.

Base Protocol

To create a probabalistic payment for amount M from some sender Alice to a recipient Bob with probability $\frac{1}{n}$.

Let X_A, X_B be (secret) random variables on $[0, n-1]$ for Alice and Bob with probability $P(X_* = i) = \frac{1}{n}$.

Have Alice and Bob select random strings S_* of length $X_* + 32$ bytes. These strings should also be kept secret. Alice and Bob should share commitments $H(S_*)$.

Have Alice and Bob construct a transaction as below:

```
TXIN:
> M*n from Alice
TXO 1:
amount: M*n
probPay1: op_dup op_hash160  <hash S_A> op_equalverify
          op_rot op_dup op_hash160 <hash S_B> op_equalverify
          op_drop op_nip
probPay2: op_size  op_rot op_size op_nit op_numequal
probPay3: op_rot op_dup op_hash160 op_rot
          op_if
              <hash K2>
          op_else
              <hash K1>
          op_endif
          op_nip op_equalverify op_checksig
probPay: {probPay1} {probPay2} {probPay3}
scriptSig: [signature] [key K2] [data S_B] [data S_A] {probPay}
scriptPubKey: OP_HASH160 [20-byte-hash of {probPay} ] OP_EQUAL
TXO N>1: (Only needed if change is needed)
```

Then Alice should release the preimage of S_A to Bob. If Bob sees that TXO 1 was spendable, he should broadcast it to network and wait for it to confirm.

If TXO 1 is spendable to Bob, then Bob should publish it. If it is spendable to Alice, then it can be ignored as it just spends back to herself.

Essentially what the above script does is checks the hashed committed values preimage, and using op_numequalverify has the following probability of success:

$$P(X_A = X_B) = \sum_{0 \cdots n-1} P(X_A = i, X_B = i) = \sum_{0 \cdots n-1} P(X_A = i) P(X_B = i)$$

In the honest case, where each party chooses X_* with probability $\frac{1}{n}$,

$$P(X_A = X_B) = 1/n$$

In the dishonest case,

$$P(X_A = X_B) = \sum_{0 \cdots n-1} P(X_A = i) P(X_B = i) = \frac{1}{n} \sum_{0 \cdots n-1} P(X_A = i) = \frac{1}{n}$$

Therefore, if one party is honest then the protocol is fair. So Alice and Bob can both be sure that as long as they draw their length honestly, the transaction will execute $\frac{1}{n}$ of the time.

Double Spend Attack

The above protocol has a weakness: nothing prevents Alice from trying to double spend the coin before Bob can spend it. Any time Alice makes a payment which ends up not in her favor, she will scramble to issue another transaction or perhaps one with higher fee. We will explore two potential mitigations to this problem.

Attempt 1: Probabalistic Release

Instead of ignoring the case where it spends back to Alice, Bob should, with a probability of $\frac{1}{(n-1)}$, publish the transaction. Therefore the probability of publishing the transaction is $\frac{1}{n}$ irrespective of who it was spent to[1]. Because of this, Alice cannot condition her behavior on this signal. This is not quite true because if Alice plays the strategy of always attempt Replace-By-Fee (RBF) double spend if published, then her loss is 0[2]. The key is that should Alice use replace by fee, $\frac{n-1}{n}$ of the time it will be a loss of Min-RBF-Amount. However, because this is small, that doesn't mean much of an expect loss.

[1] as $\frac{1}{n} = \frac{n-1}{n} \frac{1}{n-1}$

[2] however her payments will also not work $\frac{2}{n}$ of the time

Attempt 2: Punish Script

To eliminate the RBF potential, we have Alice also issue to Bob a punishing transaction can be used to eliminate RBF usefullness in the malicious case.

```
TXIN:
> c from Bob
> M*n from Alice
TXO 1:
amount: M*n+c
probPay1: op_dup op_hash160  <hash S_A> op_equalverify
          op_rot op_dup op_hash160 <hash S_B> op_equalverify
          op_drop op_nip
probPay2: op_size  op_rot op_size op_nit op_numequal
probPay3: op_if
              op_true
          op_else
              op_rot op_dup op_hash160 <hash K1>
              op_equalverify op_checksig
          op_endif
probPay: {probPay1} {probPay2} {probPay3}
scriptSig: [signature] [key K2] [data S_B] [data S_A] {probPay}
scriptPubKey: OP_HASH160 [20-byte-hash of {probPay} ] OP_EQUAL
TXO N>1: (Only needed if change is needed)
```

Bob cannot use such a script maliciously against Alice, unless Alice has behaved dishonestly. Alice cannot use the script maliciously against Bob.

Proof: In the case where the payment did not spend to Bob, Alice recovers funds even if Bob posts this script.

In the case where it spent to Bob, Bob has no incentive to use such a transaction as he gets nothing.

Alice cannot use the script maliciously because Alice does not have the script (as long Bob signs last and keeps the transaction hidden from Alice).

Miners would need to be aware of the Probabilistic Payment scheme should see what such a transaction will be worth M*n to them if they see one, as it would only be useful to post in a case where Alice tried to double spend. (Generous miners could even attempt to return funds to Bob). It is in the interest of miners to become aware of this rule as they can get more fees. However, this is not a consensus critical change.

Less Draconian

As a modification, we can generate the set of transaction as follows [3], and Bob can issue the "best fit" one based on how much RBF Alice attempts.

[3] this can be done efficiently with a sighash type that allows for the final signer to designate amounts

```
for i in 1..n by \phi:
TXIN:
> c from Bob
> M*n from Alice
TXO 1:
amount: M*i+c
    probPay1: op_dup op_hash160  <hash S_A> op_equalverify
              op_rot op_dup op_hash160 <hash S_B> op_equalverify
              op_drop op_nip
    probPay2: op_size  op_rot op_size op_nit op_numequal
    probPay3: op_if
                  op_true
              op_else
                  op_rot op_dup op_hash160 <hash K1>
                  op_equalverify op_checksig
              op_endif
probPay: {probPay1} {probPay2} {probPay3}
scriptSig: [signature] [key K2] [data S_B] [data S_A] {probPay}
scriptPubKey: OP_HASH160 [20-byte-hash of {probPay} ] OP_EQUAL

TXO 2:
amount: M*(n-1)
    probPay1: op_dup op_hash160  <hash S_A> op_equalverify
              op_rot op_dup op_hash160 <hash S_B> op_equalverify
              op_drop op_nip
    probPay2: op_size  op_rot op_size op_nit op_numequal
    probPay3: op_rot op_dup op_hash160 op_rot
              op_if
                  <hash K2>
              op_else
                  <hash K1>
              op_endif
              op_nip op_equalverify op_checksig
probPay: {probPay1} {probPay2} {probPay3}
scriptSig: [signature] [key K2] [data S_B] [data S_A] {probPay}
scriptPubKey: OP_HASH160 [20-byte-hash of {probPay} ] OP_EQUAL
TXO N>1: (Only needed if change is needed)
```

Acknowledgments

Thanks to Neha Narula for a great conversation about such topics from which this was derived, and Bryan Bishop for helping locate the prior work on this topic.

References

[a] https://bitcointalk.org/index.php?topic=62558.0.

[d] https://blog.ethereum.org/2014/10/21/scalability-part-2-hypercubes/.

[f] https://download.wpsoftware.net/bitcoin/bitcoin-probabilistic-payments.pdf.

[g] https://botbot.me/freenode/bitcoin-wizards/2014-12-18/?msg=27851314&page=1.

[h] https://bitcointalk.org/index.php?topic=201920.0.

[i] https://eprint.iacr.org/2013/784.pdf.

[j] http://diyhpl.us/~bryan/papers2/bitcoin/Secure%20multi-party%20computation%20with%20identifiable%20abort.pdf.

[k] http://diyhpl.us/~bryan/papers2/bitcoin/Publicly%20auditable%20secure%20multi-party%20computation.pdf.

[s] http://lists.linuxfoundation.org/pipermail/bitcoin-dev/2013-May/002564.html.