Revokable branched outputs (Draft 2)

Abstract

While to this day Monero remains one of the top privacy coins in the cryptocurrency space it still lacks in several aspects, one important aspect that is drawing a lot of investment and development to other projects is smart contracts and scripting. While scripting as implemented in Ethereum and Bitcoin could not easily be implemented in Monero without heavily compromising the anonymity guarantees that Monero gives its users, increased interoperability or customizability of the behavior of outputs should still be sought after. Furthermore because of this lack of scripting time locked have so far only had very limited utility.

This proposal demonstrates a mechanism by which Monero transactions could gain more flexibility without compromising anonymity. These mechanisms would only use already existing cryptographic primitives that Monero already uses so there would also be a diminished development cost.

1 Proposal

1.1 Modification to transaction data

- every output requires an additional revocation public key sG (32 additional bytes)
- every input MLSAG ring signature needs to also be signed by the private revocation key s leading to the addition of the necessary ring signature values $r_{i,3}$ ($32 * m$ additional bytes)
- every input requires a revocation key image $\tilde{K}_r$ (32 additional bytes)
- since $\tilde{K}_r$ are unique and themselves will prevent double spending the normal key image $\tilde{K}$ of the one time address can be ommited (32 bytes less)

1.2 Modification to consensus rules

- transactions outputs that reuse public revocation keys sG are invalid
- transaction inputs that reuse revocation key images $\tilde{K}_r$ are invalid
- outputs may reuse the revocation key sG if they commit to the same value and are part of the same transaction
- the sum of output commitments no longer have to add up to 0 as long as the duplicate commitments use the same revocation key.

1.3 Modification of transaction construction

- t_r is the root output index, all outputs that share the same revocation key sG will also share the same t_r output index

1.3.1 Normal outputs

By “normal” outputs single recipient outputs are meant

- revocation public keys are constructed similarly to how one time addresses are generated just that additional salt is added to the hash input: $sG = \mathcal{H}_n(rK_B^v, "revocation\_key")G + K_B^s$

1.3.2 Multi-recipient outputs

before creating a multi-recipient output the recipients must perform a Diffie Hellman key exchange to create a shared private root spend key k_r^s. The different recipients can now derive their private root view key k_r^v and the corresponding public root keys (K_r^s, K_r^v) which they can distribute to potential senders. Using the public root keys of the recipients the sender can now compute the necessary values to construct the outputs:

- public revocation key: $sG = \mathcal{H}_n(rK_r^v, "revocation\_key")G + K_r^s$
- commitment mask: $y_{t_r} = \mathcal{H}_n("commitment\_mask", \mathcal{H}_n(rK_r^v, t_r))$
- encrypted amount: $amount_{t_r} = b_{t_r} \oplus_8 \mathcal{H}_n("amount", \mathcal{H}_n(rK_r^v, t_r))$

1.3.3 Atomic swaps

Atomic swaps are gone over in detail in section 3.2

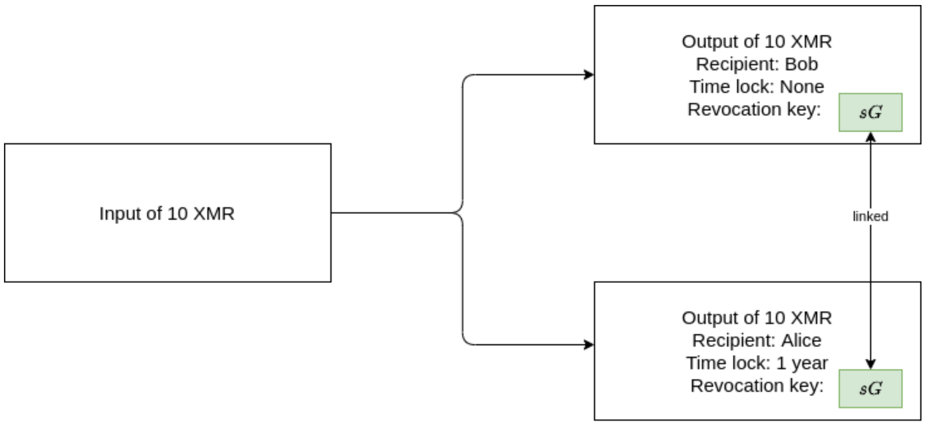
2 Anonymity

Since all transactions would have to have this additional revocation key and it would be part of the ring signature the general level of anonymity of transactions should remain the same. Furthermore this mechanism would allow the creation of decoy outputs possibly increasing anonymity. The ability to create more versatile transaction would likely lead to new heuristics but that is to be expected.

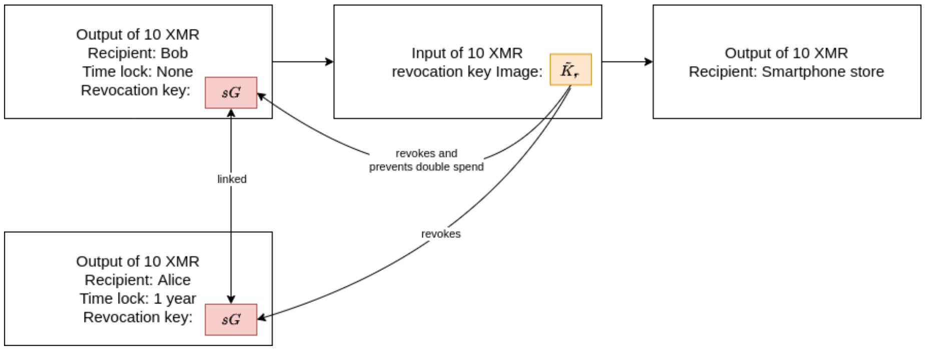
3 Utility

3.1 Timelocked inheritance

Let’s say Bob owns some XMR but he doesn’t trust his daughter so he can’t just give her his keys because she might take his funds while he’s alive. Nevertheless he wants his daughter to have his funds should anything happen to him. Let’s say Bob has a child Alice to which he wants his 10 XMR to go to if he were to be gone for more than a year, so he constructs a transaction allowing her to claim her output if he does not for a year. Since this is just a 2 party transaction with him being one of the parties he can create the secret non-interactively as follows: $s = \mathcal{H}_n(rK_A^s)$



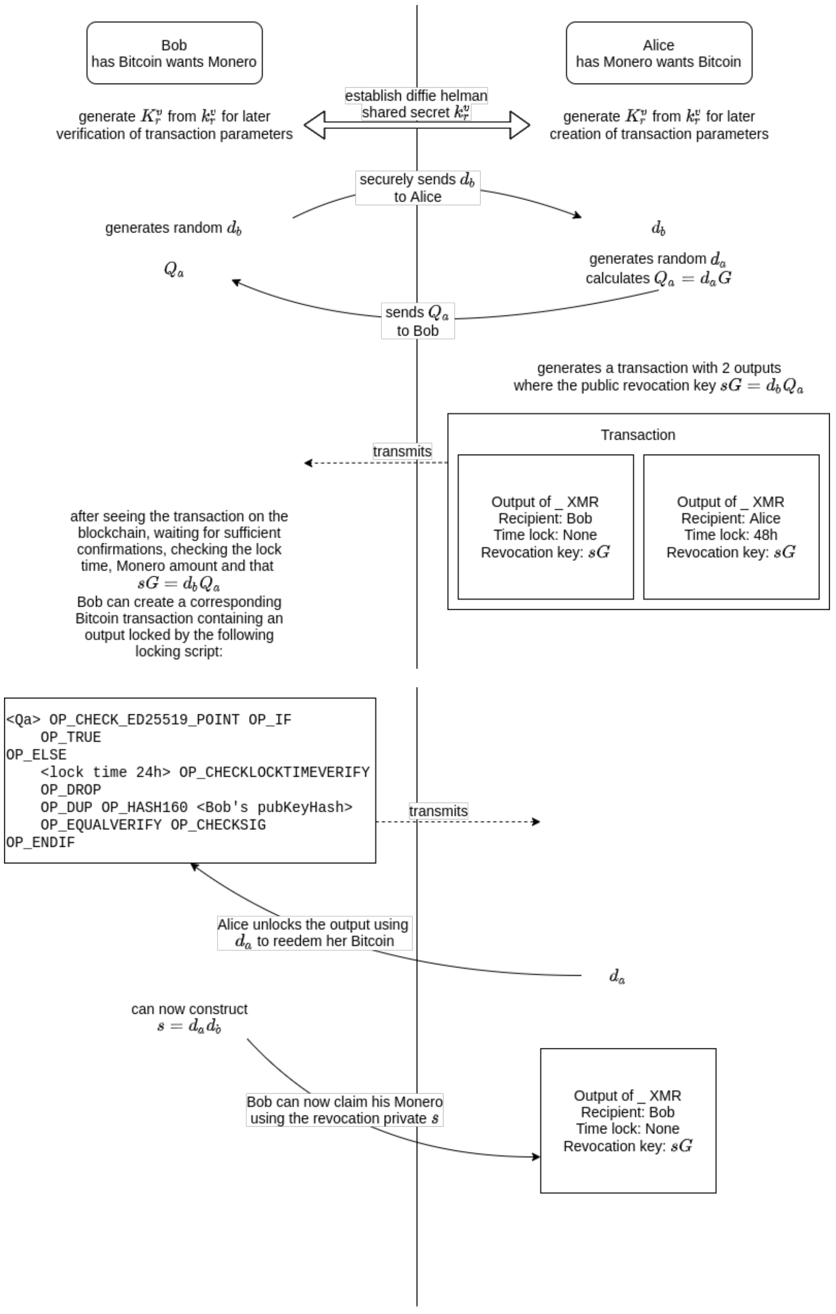
Now let's imagine Bob wants to use his money to pay for a smartphone:



Not only does Bob's spend revoke the time locked output to Alice it also prevents double spending without requiring the additional key image of the stealth address $\tilde{K}$ hence why it can be omitted for space savings. If Bob didn't spend his Monero for a year because say he unfortunately got hit by a bus Alice would be free to spend her output after the time lock expires.

3.2 Atomic swaps

A similar transaction construction as was just presented could also be used for atomic swaps, one would just need a smart contract capable of verifying a public/private EDCSA key pair. As of this writing Bitcoin's scripting language does not support any opcodes that allow it to do the necessary cryptographic calculations. So while directly swapping with Bitcoin might not be possible at the moment one could likely use an Ethereum smart contract as an intermediary since they are more flexible. For the purposes of demonstration the presented scenario will presume that Bitcoin has implemented a new opcode `OP_CHECK_ED25519_POINT` that takes a private and public EDCSA key and verifies that they belong to each other. In this scenario Bob has Bitcoin and wants Monero while Alice has Monero and wants Bitcoin. The price is already agreed upon. The following diagram visualizes the protocol:


Bob
has Bitcoin wants Monero
Alice
has Monero wants Bitcoin
establish diffie helman shared secret k_r^v
generate K_r^v from k_r^v for later verification of transaction parameters
generate K_r^v from k_r^v for later creation of transaction parameters
securely sends d_b to Alice
generates random d_b
d_b
Q_a
generates random d_a
calculates $Q_a = d_a G$
sends Q_a to Bob
generates a transaction with 2 outputs where the public revocation key $sG = d_b Q_a$
Transaction
Output of _ XMR
Recipient: Bob
Time lock: None
Revocation key: sG
Output of _ XMR
Recipient: Alice
Time lock: 48h
Revocation key: sG
transmits
after seeing the transaction on the blockchain, waiting for sufficient confirmations, checking the lock time, Monero amount and that $sG = d_b Q_a$ Bob can create a corresponding Bitcoin transaction containing an output locked by the following locking script:
transmits
Alice unlocks the output using d_a to reedem her Bitcoin
d_a
can now construct $s = d_a d_b$
Bob can now claim his Monero using the revocation private s
Output of _ XMR
Recipient: Bob
Time lock: None
Revocation key: sG

```
<Qa> OP_CHECK_ED25519_POINT OP_IF
    OP_TRUE
OP_ELSE
    <lock time 24h> OP_CHECKLOCKTIMEVERIFY
    OP_DROP
    OP_DUP OP_HASH160 <Bob's pubKeyHash>
    OP_EQUALVERIFY OP_CHECKSIG
OP_ENDIF
```

Since only d_a, sG and Q_a are publicly revealed an attacker cannot discover what amount is being transacted.

3.3 Escrow

Just as in the inheritance scenario one can combine this new capability with multisig wallets to create even more versatile escrow transactions.

4 Encoding

Entities which handle a lot of funds may not want the risk of an intermediary step of converting outputs from standard one address/multisig outputs to possibly more complex output structures. Therefore they'd likely want a way for the people who pay them to directly create the desired output structures but without unnecessary overhead. Therefore a standardized encoding is proposed:

data	type/size
network byte	1 byte
root public spend key	32 bytes
root public view key	32 bytes
amount of output branches	varint
output branches	output branch
checksum of entire string of data	4 bytes

output branch

data	type/size
time lock: first bit of the encoded value determines whether the value should be interpreted as an absolute time lock or relative to the current block (0: absolute; 1: relative)	varint
network byte	1 byte
public spend key	32 bytes
public view key	32 bytes

The whole string of bytes would then be encoded into base-58

Here's some example code of how this encoding could be computed:

```
from sha3 import keccak_256

BASE58_ALPHABET = '123456789ABCDEFGHJKLMNPQRSTUVWXYZabcdefghijkmnopqrstuvwxyz'
```

```python
data = {
    'root_address': '12281577bd1f0e4a21fa47158a7d6a5b469fe2b25cc89227b5c93bd95f94b2afec9c23
    'branches': [
        {
            'relative': True,
            'timelock': 1100,
            'address': '12281577bd1f0e4a21fa47158a7d6a5b469fe2b25cc89227b5c93bd95f94b2afec9
        },
        {
            'relative': False,
            'timelock': None,
            'address': '12fc2b43395408ba60e06409c953e639c64334b33145aa4c9b933a87f658c795372
        }
    ]
}

def encode_varint(x):
    data = []
    while x > 0x7f:
        data.append(x & 0x7f | 0x80)
        x >>= 7
    data.append(x & 0x7f)
    return bytes(data)


def b58encode(b: bytes) -> str:
    x = int.from_bytes(b, 'big')
    res = ''
    while x:
        x, dig = divmod(x, 58)
        res += BASE58_ALPHABET[dig]
    return res[::-1] or (b and '1')


def serialize_branch(branch):
    as_bin = ('1' if branch['relative'] else '0')\
        + bin(branch['timelock'] or 0)[2:]
    return encode_varint(int(as_bin, 2)) + bytes.fromhex(branch['address'])

def serialize(branches_data):
    buffer = bytes.fromhex(branches_data['root_address'])
    buffer += encode_varint(len(branches_data['branches']))

    for branch in branches_data['branches']:
        buffer += serialize_branch(branch)
    checksum = keccak_256(buffer).digest()[:4]

    return buffer + checksum

def encode_branch_data(branches_data):
    return b58encode(serialize(branches_data))
```

This results in approx. $72 + 72 * m$ bytes where m is the amount of addresses. For up to a few addresses this can still be stored in a QR-Code. The OpenAlias standard could be extended to permit another key value field by default so that one can create a more easily remembered alias.

Conclusion

While the addition of revocation keys would lead to an increase in transaction size it would allow for more versatile and complex transactions while preserving privacy. Furthermore all the outputs of these more complex transactions can be used as decoys in other transactions assuming that their time locks have expired at the time they are selected as decoys. The resulting interoperability and flexibility could lead to increased attention and use likely benefiting Monero and its users overall in the long run.