[bitcoin-dev] Taproot: Privacy preserving switchable scripting

Gregory Maxwell greg at xiph.org
Tue Jan 23 00:30:06 UTC 2018

```
Interest in merkelized scriptPubKeys (e.g. MAST) is driven by two main
areas: efficiency and privacy. Efficiency because unexecuted forks of
a script can avoid ever hitting the chain, and privacy because hiding
unexecuted code leaves scripts indistinguishable to the extent that
their only differences are in the unexecuted parts.

As Mark Friedenbach and others have pointed out before it is almost
always the case that interesting scripts have a logical top level
branch which allows satisfaction of the contract with nothing other
than a signature by all parties.  Other branches would only be used
where some participant is failing to cooperate. More strongly stated,
I believe that _any_ contract with a fixed finite participant set
upfront can be and should be represented as an OR between an N-of-N
and whatever more complex contract you might want to represent.

One point that comes up while talking about merkelized scripts is can
we go about making fancier contract use cases as indistinguishable as
possible from the most common and boring payments. Otherwise, if the
anonymity set of fancy usage is only other fancy usage it may not be
very large in practice. One suggestion has been that ordinary
checksig-only scripts should include a dummy branch for the rest of
the tree (e.g. a random value hash), making it look like there are
potentially alternative rules when there aren't really.  The negative
side of this is an additional 32-byte overhead for the overwhelmingly
common case which doesn't need it.  I think the privacy gains are
worth doing such a thing, but different people reason differently
about these trade-offs.

It turns out, however, that there is no need to make a trade-off.  The
special case of a top level "threshold-signature OR
arbitrary-conditions" can be made indistinguishable from a normal
one-party signature, with no overhead at all, with a special
delegating CHECKSIG which I call Taproot.

Let's say we want to create a coin that can be redeemed by either
Alice && Bob   or by CSV-timelock && Bob.

Alice has public A, Bob has pubkey B.

We compute the 2-of-2 aggregate key C = A + B.  (Simplified; to
protect against rogue key attacks you may want to use the MuSig key
aggregation function [1])

We form our timelock script S =  "<timeout> OP_CSV OP_DROP B OP_CHECKSIGVERIFY"

Now we tweak C to produce P which is the key we'll publish: P = C + H(C||S)G.

(This is the attack hardened pay-to-contract construction described in [2])

Then we pay to a scriptPubKey of [Taproot supporting version] [EC point P].

Now Alice and Bob-- assuming they are both online and agree about the
resolution of their contract-- can jointly form a 2 of 2 signature for
P, and spend as if it were a payment to a single party (one of them
just needs to add H(C||S) to their private key).

Alternatively, the Taproot consensus rules would allow this script to
be satisfied by someone who provides the network with C (the original
combined pubkey), S, and does whatever S requires-- e.g. passes the
CSV check and provides Bob's signature. With this information the
network can verify that C + H(C||S) == P.

So in the all-sign case there is zero overhead; and no one can tell
that the contract alternative exists. In the alternative redemption
branch the only overhead is revealing the original combined pubkey
and, of course, the existence of the contract is made public.

This composes just fine with whatever other merkelized script system
we might care to use, as the S can be whatever kind of data we want,
including the root of some tree.

My example shows 2-of-2 but it works the same for any number of
participants (and with setup interaction any threshold of
participants, so long as you don't mind an inability to tell which
members signed off).

The verification computational complexity of signature path is
obviously the same as any other plain signature (since its
indistinguishable). Verification of the branch redemption requires a
hash and a multiplication with a constant point which is strictly more
efficient than a signature verification and could be efficiently fused
into batch signature validation.

The nearest competitor to this idea that I can come up with would
supporting a simple delegation where the output can be spent by the
named key, or a spending transaction could provide a script along with
a signature of that script by the named key, delegating control to the
signed script. Before paying into that escrow Alice/Bob would
construct this signature. This idea is equally efficient in the common
case, but larger and slower to verify in the alternative spend case.
Setting up the signature requires additional interaction between
participants and the resulting signature must be durably stored and
couldn't just be recomputed using single-party information.

I believe this construction will allow the largest possible anonymity
set for fixed party smart contracts by making them look like the
simplest possible payments. It accomplishes this without any overhead
in the common case, invoking any sketchy or impractical techniques,
requiring extra rounds of interaction between contract participants,
and without requiring the durable storage of other data.


[1] https://eprint.iacr.org/2018/068
[2] https://blockstream.com/sidechains.pdf Appendix A
```
