Privacy-Preserving Proof of Assets

Status: DRAFT

August 30, 2024

1 Overview

We propose an algorithm to (and provide code to) prove, in zero knowledge, statements of the form (all numbers just being indicative examples):

> I own the private keys to a set of 10 taproot Bitcoin utxos out of all taproot utxos that are more than 0.05 BTC in size, whose total value is in the range 5.0 BTC to 7.0 BTC, but I am not revealing anything else about the individual utxos.

This concrete example illustrates quite well the potential use case of such a proving system: suppose a small company uses a reserve/treasury/holding of bitcoin as part of its relationship with customers (including the most direct case, where the company holds the customers' own bitcoin!), clients or business partners. In this case it may be very helpful to prove ownership of the coins, but it may also be very beneficial for security (and business confidentiality) to not reveal the on chain activities of the enterprise.

1.1 Earlier work

1.1.1 The 1000ft view

Approaches to proving facts about the bitcoin blockchain in zero-knowledge have to fall into one of three categories. Before listing them, consider these two issues that have to be addressed:

- Since the blockchain is public, and consists of very large sets of utxos, transactions, and blocks, proving statement S in ZK means "ZK in the context of some anonymity set S^*", which often realistically means *very* large sets; for example in this work, we envisage anonymity sets of 500K-2M utxos as typical.
- Since the bitcoin blockchain uses secp256k1, a non-pairing friendly curve, there is no way to use bilinear pairings and therefore systems like KZG polynomial commitments[9] or Groth16 and similar proving systems. Since also, most utxos are tied to scriptPubKeys that use hash functions, you have the additional problem of proving hash preimages in zero knowledge. These complexities result in very complicated and sometimes very slow proofs of statements about such large anonymity sets as was just mentioned.

Given this, it's easier to understand the three categories of approaches taken to tackle this difficult task:

1. Using only the ECDLP assumption on secp256k1, you can use standard Σ-protocol techniques to build proofs of arbitrarily complex statements in ZK. However even if the statements are fairly simple in structure (as is the case with Proof-of-Assets), the requirement of large anonymity set hits hard: such Σ-protocol based techniques result in proving, proof size and verifying all scale linearly with the anonymity set.

2. Still using only the ECDLP assumption on secp256k1, but using Bulletproofs which provides compact proofs, but linear verification. This can be a very helpful step up for some scenarios (see: Confidential Transactions as described in [3], but in itself does not aid performance as is required for huge anon. sets. However, when combined with an algebraic tree structure as in [2], which makes use of the lucky fact that there is a cycle over the curves secp256k1/secq256k1, we get logarithmic verification times. While this is structurally more complex than (1), it is always likely to win out unless the application does not require a very large anonymity set (a difficult scenario to envisage).

3. Using other assumptions "knowledge of exponent" and *-diffie-hellman on bilinear groups, (zk)SNARKs can be applied to this problem. Again, the problem here is the difficulty of "representing" the large anonymity set of group elements and hashes from the bitcoin blockchain, neither of which are expressible "natively" in a pairing-friendly group. This results in very large arithmetic circuits. As a result, it is possible to build such systems with fast verifiers and constant size proofs, but with a very large upfront proving cost (where it is even practical at all).

The seminal work "Provisions" addresses the same "proof of assets" scenario, as well as "proof of liabilities" (specifically for asset custodians), but using only standard Σ-protocol techniques (including proof of commitment to bit representations). Thus it is the perhaps canonical example of category (1) in the list above. Hence it is probably impractical for very large anonymity sets. To quote the paper itself:

> Provisions scales linearly in proof size, construction and verification time with respect to its inputs: the proof of assets scales with the size of the anonymity set ...

and though it is just an example implementation, the performance graph in Figure 2 of the paper illustrates the issue well; at an anon set of 500k the construction time is already reaching 15 minutes.

To be fair, as the paper notes, operations are parallelisable and can be ameliorated in practice, so that for some use cases, it may even be preferable to use this less sophisticated cryptographic construction.

The technique in this paper falls under category (2). If privacy of the proof's witness is important, very large "theoretical anonymity sets" will be highly desirable, and hence the logarithmic dependence on anonymity set size achieved by the techniques described in this paper, will be necessary.

Much more recent work "IZPR"[4] claims an even more substantial performance improvement, for similarly very large anonymity sets, and is thus probably a good example of category (3). It uses lookup tables, with constant size proofs of about 3kB and sub-second proving time, by doing preprocessing upfront (the prover of assets must in this case define the set of future possible public keys that will be assigned funds) to create proofs that can be aggregated with updates. This approach uses zkSNARKS and essentially defers all the "bad stuff" to its bootstrap phase, where the authors quote proving times of 4 hours +; the problem is the requirement to embed proofs of hash functions and ECDSA signatures over huge anon. sets within the arithmetic circuit. Overall, this author is not able to currently assess the potential of this approach, but it seems promising if hurdles in bootstrap can be overcome; however, we do wonder how much this approach's bootstrap problem could be ameliorated by restricting only the taproot set, thus removing the need to embed hashing into the arithmetic circuit, at a cost to anonymity set which may over time become very reasonable.

1.2 Why use a range not an exact total

A proof-of-assets, as an *auditing* function, may seem naturally to require an exact summation. However this is not compatible, realistically, with privacy.

If you prove ownership of exactly x satoshis, then finding sums of utxos that add up to exactly x, while it can in some cases be computationally infeasible, provides a route to exposing the exact utxos used in the proof where it is possible, destroying the intended privacy. And one should be cautious in assessing computability using worst case scenarios; often heuristics can take an intractable computation and make it practical. Thus we do not implement proof of exact total.

This illustrates something problematic: even though the *technique* used here is zero-knowledge proofs, we work in a scenario of leaking partial information (number of utxos; their total value) and so we can only argue for privacy over an estimated anonymity set; the protocol *as a whole* does not have a "zero knowledge property" even if sub-protocols within it, do indeed have such a property.

On the other hand, if a person/enterprise wants to prove ownership of *at least* x satoshis, and proves ownership of y s.t. $x < y < x + \delta$, the range of possible utxo sets *can*, in most normal scenarios, result in very large anonymity sets of sets of utxos.

1.3 High level outline of the properties of the algorithm

The algorithm is highly efficient in verification of the proofs, taking 10s to 100s of milliseconds if well optimised, and this time is little affected for utxo sets of up to millions. Note that only taproot utxos are supported, and that there are in total, as of this writing approximately 40M such utxos, of which a large majority are "dusty" in size and uneconomic. There are perhaps $\simeq$ 6M of normal size, which means "theoretical anonymity set" for such a privacy-preserving proof of reserve could go into the millions, as of today (but as we will see "effective anonymity set" will be a lot smaller than the theoretical one).

Proving can take seconds to tens of seconds on commodity hardware, but with appropriate parallelisation should not be impractical; in the most natural use case, since it is not clients on constrained devices that do proving operations, this should never be an issue.

Proof size will typically be in the 10-20kB range; for this application, it is hard to imagine proof size will ever be an issue (e.g. a company makes the proof available for download on a webpage). There is some linear dependence on the number of utxos used to prove, but this is heavily ameliorated with batching of bulletproofs.

Both prover and verifier need access to the filtered list of utxos, which is of course far larger for the desired large anonymity sets. As an example, a recent filtered list of taproot utxos greater than 500k sats in size, on mainnet, was about 20MB. *Creating* such filtered utxo lists *is* computationally intensive, since the set of utxos has size 180M as of this writing; while this is theoretically irrelevant to the performance characteristics of the algorithm described here, it is, in practice, something that must be addressed for usage. One approach is to distribute such lists to clients, along with easy methods to audit/verify correctness (which isn't very hard because the blockchain is accessible to all verifiers).

The algorithm is based principally on Curve Trees[2], and Bulletproofs[3]. The reason for this choice will be explained in the next section. Then, the remainder of the document will explain each subcomponent of the proving/verifying algorithms in detail, followed by a brief discussion of the privacy properties and performance properties of the algorithm.

2 The technical components of the algorithm

To repeat for convenience, but in more generality, we prove statements of this form:

> I own the private keys to a set of N taproot Bitcoin utxos out of all taproot utxos that are more than u satoshis in size, whose total value is in the range k satoshis to $k + 2^n$ satoshis, but I am not revealing anything else about the individual utxos.

2.1 Choice of anonymity set

We specify taproot because we want *spontaneously formed anonymity sets*, and this is possible with taproot because the public keys are published on the blockchain (for other outputs, covered by hashes, you would need very expensive proof-of-hash-preimage ZK machinery).

An alternative model would be to choose all keys that are reused, so that pubkey hashing does not prevent reading on the blockchain; this is the model that was suggested in [1], which was published way before taproot existed. However most user-level wallets do not reuse addresses, so the anonymity set is drastically reduced this way.

Another alternative model is to remove the requirement of *spontaneity.* In this model, "users" wishing to "take part" would have to explicitly provide public keys. In this case, we suggest, it may be better to have them use bitcoin (secp256k1) keys to sign over keys in a different group, one that supports pairings, so that much more compact proofs can be created using zkSNARKS and/or other algorithms from bilinear pairings can be used. An example of proving set membership this way can be found in Caulk [8], though there are many other possibilities.

In this document we treat spontaneity of forming large anonymity sets as a requirement, and so we focus on [2], which gives logarithmic scaling to verification and proof size while not requiring any of: trusted setup, hardness assumptions outside of ECDLP, or bilinear pairings (and thus pairing-friendly groups).

2.2 Select-And-Rerandomize for Set Membership

Note: this section glosses over details of how Curve Trees are constructed; read [2] for details.

[2] explains, in Section 3 how an algorithm "Select-and-rerandomize" (SaR for short) can be used to generate a "reblinded commitment" (why *re*-blinded? because we are assuming that points in general, in Curve Trees, are Pedersen commitments, which contain a blinding term already; though in fact, the leaves of Curve Trees may often be unblinded to start with).

In outline, SaR constructs a path of nodes to the root of the Curve Tree, proving that the provided commitment corresponds to an element of the set of leaves of the tree, but in zero-knowledge, because the components of the path are reblinded, and a ZKP (Bulletproofs) is attached to prove this statement.

Note that SaR is ZK but is *not* a proof of knowledge of secret key; it only says "this point was a leaf of the tree", it doesn't say "and I know the private key of the point", so that this must be included separately.

2.2.1 Constructing the Curve Tree.

Each leaf of the tree is constructed from the group element:

$$C_i = P_i + v_i J + r_p H$$

where P_i are the public keys of utxos, v_i are the values, in satoshis, of the (same) utxos, J and H are agreed generators formed using a NUMS technique such as hashing an agreed string (so that the relative discrete log between G and J is unknown, e.g., as is required for the soundness of these commitments), and finally r_p is a small integer required to make the point C_i "permissible" in the notation of [2]. The algorithm to increment this value to make C_i permissible is, again, agreed between the prover and verifier so that they both calculate the same value.

These commitments C_i are interpreted as Pedersen *vector* commitments (that happen to use a not-secret small blinding value of r_p). This classification will be important as we continue, but for now let's define a vector Pedersen commitment as:

$$C = \sum_j x_j G_j + rH$$

with $G_0 \dots G_{m-1}$ being other NUMS-generated generators, and with the opening of the commitment being the *vector* of secret values x_j along with the blinding/randomness r.
Thus we can see that both prover and verifier can construct the same tree T by the following series of steps:

- Agree on the specific blockheight so that the utxo set's state U is fixed.
- Apply a filter function $f(U)$ to retrieve a filtered utxo set U^*. Part of the filter is "must be segwit v1/taproot", but the filter may also apply age and value rules.
- Each $u_i \in U^*$ is read as a tuple (pubkey, value). The verifier and prover then construct $C_i = P_i + v_i G + r_p H$ and sort in some agreed way, e.g. lexicographically. The tree T is constructed from this ordered list, yielding a root ρ_{U^*}.

For each utxo u_i that the prover wants to prove over, he runs the standard SaR algorithm from [2] to generate a new reblinded commitment:

$$C_i^* = P + v_i G + rH$$

... with $r = r_p + r_r$, so that C_i^* is a proper blinded commitment, i.e. it is not correlatable with the original leaf of the publically known tree.

The SaR proof for one utxo concretely consists of:

- A bulletproof over secp256k1
- A bulletproof over secq256k1
- A list of nodes constituting a "path" from the leaf being proved over to the root, with each node being reblinded (this is a "Merkle path", effectively)

The first two are aggregations, in case the depth of the tree is ≥ 2, because a separate bulletproof must be constructed for each height in the tree (except the root), so in case we use a large depth like 10, we would aggregate each of the proofs at the same parity height (which is possible because they are proofs over the same curve).
A proof of this type is created for every utxo u_i in the set of N utxos that the prover chooses.
The 2 bulletproof components can be aggregated so that there is overall one bulletproof per elliptic curve; therefore the total set of elements in the proof here would be:

- A bulletproof over secp256k1 (size scaling logarithmically with N)
- A bulletproof over secq256k1 (size scaling logarithmically with N)
- A list of Merkle paths. This scales linearly with N.

The space overhead should therefore not be too bad, as the list of paths consists of $\simeq D \times N$ group elements, where D is the depth of the tree; it is independent of the branching factor of the tree; thus, for wide shallow trees (as tends to be optimal), it is very small (an example, it is not unusual to use $D = 2$, even for very large sets of utxos up to 1M, hence you would have only $2N$ group elements here).

2.3 (Bullet)proof of sum-in-range

Next, given the reblinded commitment C_i^*, the prover wants to prove knowledge of opening of this commitment, i.e. knowledge of witness values (x_i, v_i, r_i) $\forall i$, and at the same time, that the values v_i have the property that:

$$k < \sum_i v_i < k + 2^n$$

To prove the second part of that statement, the value range, the prover uses Bulletproofs. He starts, using "generalized bulletproofs" as per [5] by committing to the same i witness tuples (x, v, r) using i new Pedersen vector commitments:

$$Q_i = x_i G_1 + v_i G_2 + r_i H$$

He then commits to the sum (here V as per bulletproofs notation):

$$V = v_{\text{sum}} G + r_v H$$

then adding as a constraint to the arithmetic circuit, that:

$$v_{\text{sum}} = \sum_i v_i$$

Next, he proves that the committed sum v_{sum} is in range:
First, we define a function f_b:

$$f_b(v, i) = v \gg i \ \& \ 1$$

f_b extracts the i-th bit of a value v.
Next, iterating over i from $1 \ldots n$, we calculate variables:

$$b_i = f_b(v, i) \qquad a_i = 1 - f_b(v, i)$$

And using the structure of bulletproofs, assign a multiplication gate $a \cdot b = o$ for each of these bits. We then make additional arithmetic constraints that:

$$o = 0, \quad b + a - 1 = 0$$

By doing so, we have the bulletproof enforce that each value b_i is indeed a bit, since $x(1 - x) = 0$ holds if and only if $x \in 0, 1$.

Finally we add an arithmetic constraint that:

$$v_{\text{sum}} - k - \sum_i b_i 2^i = 0$$

which proves that the sum is of form $k + \alpha$ where α as a valid n-bit binary representation, as required.

2.4 Proof of multirepresentation

The previous section constructed i Pedersen vector commitments Q_i with the same witness tuples as the original reblinded leaf commitments C_i^* (that is, (x_i, v_i, r_i)).
We refer to this as "multirepresentation" in the sense that, for each index i, we are providing two representations of the witness with respect to two different base vectors (the first being G, J, H and the second being G_1, G_2, H, in this case).
The document [6] in this code repository details this proof, though it considers the more general case of n independent representation with respect to m bases, not specifically only $2, 3$. This problem can be seen as a special case of what is sometimes termed "generalised Schnorr protocols"; see e.g. [7] for details.
For the purposes of this paper, we consider that this step, included as part of our overall Proof-of-Assets, serves the two purposes:

- It serves as a proof of knowledge of the witness (x_i, v_i, r_i) for each utxo; with obviously the x_i being the one that matters.
- It proves that the unrevealed quantity v_sum that is the sum of the values in the witnesses for the C_i^* values, is in the stated range, indirectly, since we actually prove it for Q_i and then show that the witnesses are the same.

2.4.1 Performance

Since this special case of "multirepresentation" over two vectors of bases only requires 2 group elements and 3 + 1 (+1 for the hash challenge) scalars per utxo, it is, like the "path" component of the SaR proof, linear in N but still small for realistic scenarios. The proving and verifying time are negligible.

2.4.2 Proof of non-reuse

Proving that a set of unrevealed utxos have a sum of their values above a certain minimum would be useless if utxos could be repeated in the list; and without an additional step, this would indeed be possible; simply construct elements of the list $C_i^* = x_iG + v_iJ + r_iH$ and $C_j^* = x_iG + v_iJ + r_jH$, i.e. just change the blinding scalar.

To avoid this we simply stipulate an additional condition into the generalised Schnorr proof structure detailed above as "proof of representation". We agree on a further NUMS generator Q, and attach an additional requirement that a new commitment I_i, provided by the prover for each index utxo index i, satisfies $I_i = x_iQ$. For clarity, the verifier will be checking:

$$R_{I,i} + eI_i = s_{I,i}Q$$

This introduces a further two group elements and one scalar per proof (since unlike public keys, which form part of the precursor context of the proof, I must be included with the proof; see next).

Having provided I_i for each i in the proof, the verifier has a final check: if I_i values are not all distinct, the proof is invalid due to it having re-used the same public key. It is for this reason that, in the current algorithm, **reused**

addresses are not allowed in the proof of assets. This restriction could be removed with a fairly trivial tweak to the algorithm.

2.5 Effective anonymity sets

TODO

2.6 Performance results

TODO

3 References

1. Provisions: Privacy-preserving proofs of solvency for Bitcoin exchanges `https://eprint.iacr.org/2015/1008.pdf`, Bunz et. al. 2015
2. Curve Trees - Practical and Transparent Zero-Knowledge Accumulators `https://eprint.iacr.org/2022/756`, Campanelli, Hall-Andersen, Kamp 2022
3. Bulletproofs - Short Proofs for Confidential Transactions and more `https://eprint.iacr.org/2017/1066`, Bunz et al 2017
4. IZPR - Instant Zero-Knowledge Proof of Reserves, `https://eprint.iacr.org/2023/1156.pdf`, X. Fu et. al. 2023
5. `https://github.com/simonkamp/curve-trees/blob/main/bulletproofs/generalized-bulletproofs.md`
6. Proof of multirepresentation `https://github.com/AdamISZ/aut-ct/blob/auditing/auditor-docs/multirepresentation.pdf` 2024
7. Proof systems for general statements about discrete logarithms `https://crypto.ethz.ch/publications/files/CamSta97b.pdf` Camenisch, Stadler 1997
8. Caulk: Lookup Arguments in Sublinear Time `https://eprint.iacr.org/2022/621.pdf` Buterin et. al. 2022
9. Polynomial Commitments `https://cacr.uwaterloo.ca/techreports/2010/cacr2010-10.pdf` Kate, Zaverucha, Goldberg 2010