Thanks Jan, Aayush, Yanis, Michael and Maxence for feedback and discussions.

A growing number of applications like Proof of passport, anon-aadhaar and zk-email use existing infrastructure from institutions to verify identities, respectively electronic passports, India's Aadhaar system and DKIM registries. As they are based on already existing Public Key Infrastructure, building privacy preserving nullifiers can be quite tricky.

We'll take the example of Proof of Passport, but what is discussed here applies to many other projects. Proof of Passport allows users to scan the NFC chip in their government-issued passport and generate zk proofs showing that they have a valid identity. It can be used for proof of humanity, sybil resistance and selective disclosure of private data like nationality or age.

We want to build a proof of passport SBT that displays information the user wants to disclose. It shouldn't let someone mint two SBTs with a single passport.

Naively, one can think of using the hash of the government's signature as a nullifier and storing it publicly. It does prevent reusage of a proof, but if the government keeps a database with all signature it has issued, it can link each nullifier with each user's address. As the ICAO specifications do not describe whether governments should do it or not, we can assume some governments keep those signatures. Additionally, anyone that gets access to the passport data before or after a mint can store it and deanonymize users. Same problem with zk-email: nullifying an email publicly deanonymizes the user to the mail server that signed the email.

Note that in the following constructions, we are not trying to hide the fact that a user minted a SBT from the government. This would be hard to do as the government has the same information as the user, and can just request a new SBT to see if the request is rejected or not. Instead, we're trying to at least hide the user's address from the government.

Using a trusted server

One approach would be to let applications manage user privacy themselves. Each application would have a server with its own key pair that can sign government attestations so that:

\begin{aligned} nullifier &= hash(server.sign(attestation))
\end{aligned}

The flow would be the following:

- The user extracts their government attestation from their passport and sends it to the application's trusted server
- The server signs the attestation and hashes the result to get the nullifier. If it has already been included on chain, it refuses the request and responds {false, \emptyset}

. If it's not already included, it responds with the {true, signature}

.

When the user submits the proof on chain, they shows that:

- The disclosed attributes (e.g. nationality) are valid given the government attestation
- The nullifier, a public output of the proof, is correctly computed as a hash of the server's signature of the attestation

The nullifier is then stored on chain.

The server has to block duplicate requests because if it doesn't, a government could pose as a user, send an attestation, get back a nullifier and identify the user's address. Note that the government can still know if a user has minted an SBT, but not their address.

The main drawback here is obvious: the server sees every government attestation, so it can link every user's attestation to its address. Also, there is a liveliness assumption on the server. This might be sufficient for some applications like a vote that happens only once. It can also be improved by keeping the secret in an SGX. But for long-lasting general purpose applications like the proof of passport SBT, it's insufficient.

Using two-party MPC

A better way to do it is using two-party MPC. This way, we can have the user and the server compute a circuit together so that the server does not learn the government attestation.

The circuit takes as inputs:

- The disclosed attributes, e.g. nationality (public)
- The government pubkey (public)

- The list of nullifiers already on chain (public)
- The server's public key (public)
- The server's private key (known only to the server)
- The government attestation (known only to the user)

Over MPC, the user and the server compute the signature of the attestation, hashe it and compare it with the nullifiers already posted. Just like before, if it's included the circuit returns {false, \emptyset}

and if it's not included it returns {true, nullifier}

. The user and the server then generate SNARKs proving they performed the MPC correctly and they can be verified on chain.

Note that we have to do the whole computation from disclosed attributes to nullifier over MPC and in one proof. If we tried to split it so that only the nullifier generation is done over MPC, the user's proof would have to disclose the government attestation or something deterministically derived from it, which would allow the government to identify them.

Now, the server can't link user's attestations to their addresses. However:

- If the server colludes with the government and shares its private key, they can deanonymize users.
- There is still a liveliness assumption on the server

Using n-way MPC

If we had something like iO, we could do an obfuscated inclusion check on chain. This way, if a government tried to provide the same proof as a user, it would be able to know if the SBT was already minted, but not to which address. We can try to emulate this by extending the MPC setup. It also relaxes the liveliness assumption on a single server.

A rough flow would be:

- Nodes join a network. This can be done on top of EigenLayer so as to slash nodes that misbehave.
- The circuit is computed by the user together with the nodes. This time, the server secret is a threshold secret reconstructed from t/N

nodes.

Again, each participant can prove correct computation using a snark. In particular, nodes show their secret is consistent over time by showing the hash of their secret is always the same. This guarantees the reconstructed secret is always the same, which makes the nullifier generation deterministic.

Signing the attestation with a threshold private key can be replaced with hashing it with the threshold secret for convenience.

To avoid putting a large burden on the nodes, it might be possible to adapt the design to something closer to Octopus.

Something very similar can be done with threshold FHE. The main difference is that instead of being part of the setup, the user encrypts the attestation with their FHE private key, makes their evaluation key public, and at the end of the process decrypts the encrypted nullifier before posting it on chain. Depending on the overhead of threshold FHE, this could be lighter for the user, but probably heavier for the nodes.

But this setup raise new issues:

- Leaking secrets is undetectable. A government corrupting t

nodes privately could deanonymize every user. It would be possible to slash nodes only if they make their secret public to everybody. We can still have some cryptoeconomic security, but it's insufficient if we assume governments are willing to forgo tens of millions in slashing rewards.

- There must be a way for nodes to rotate, so secrets need some way to be shared or at least added and removed.
- After some turnover, there will be a point at which nodes that stopped restaking or taking part in the computation can collude at no cost and deanonymize everyone.

Using embedded private keys

A growing number of official documents like new passports, most of EU's ID cards and Japanese Myna Cards not only contain a signed attestation from the issuing authority, but also their own public-private key pair. Authentication is done dynamically by asking the chip to sign a message. This is called active authentication, as opposed to passive authentication.

It prevents cloning by simply copying the attestation on a forged document.

Most smart cards are designed to generate their own private key so that it can never leave the chip. If a passport is made this way, the government attests to the public key without ever knowing the private key. In this case, we can construct a privacy preserving nullifier as the hash of a signature of a fixed message.

$$\begin{aligned} nullifier = hash(chip.sign(\text{"NULLIFIER"})) \end{aligned}$$

We can prove in a SNARK that the signing private key is correctly derived from the attestation, but it's impossible for the government to deanonymize the user without knowing their private key.

Some documents do not support Active Authentication but only Chip Authentication. Chip authentication does not allow for signing messages but performs semi-static ECDH. It involves a scalar multiplication on a secure elliptic curve so it can be used for nullifier generation in the same way.

This design is really cool because it doesn't only hide the address of the user, it can also make it impossible for the government to even know if a user has minted a SBT, as it can't produce the right signature.

Unfortunately:

- We do not currently know how much authorities generate the private keys on the chip vs embed them and store them in a registry. Please reach out if you know more about this.
- This doesn't work with passive attestations like anon-aadhaar, email DKIM signatures and older passports.
- This only works if the signature is deterministic. The ICAO docs allow for RSA or ECDSA to be used. If there is no way of choosing the random k

used in ECDSA, then it only works with RSA.

- It does not prevent someone accessing the document before or after the mint from recording the same signature and deanonymizing the user.

Other approaches

If some application needed to always combine multiple sources, we could nullify the combination using a hash of multiple attestations. This way, all institutions involved would have to collude to deanonymize users.

Another approach would be storing hard hashes (Argon2, PBKDF2 or scrypt) of nullifiers to make them harder to check. It would make checking a large set of people impractical, but an attacker looking for one person could easily find it.

Another approach would involve biometrics. The photo signed in the passport chip looks too low-resolution to do any significant zkml, but using FaceID or TouchID to sign deterministic messages from a smartphone's secure enclave might be promising while adding a relatively low trust assumption.

We are actively looking for better ideas. Please reach out if you have one!