Algorithm 3 Our uncensorable proof-of-burn protocol for Bitcoin P2PKH.

```
function GenBurnAddr_H(1^κ, t)
    th ← H(t)
    th' ← th ⊕ 1                          ▷ Key perturbation
    return th'
end function
function BurnVerify_H(1^κ, t, th')
    return (GenBurnAddr_H(1^κ, t) = th')
end function
```

This 'security' parameter —?

Basically a commitment scheme.

Otherwise, this is fairly straight forward..

Algorithm 4 The Bitcoin P2PKH algorithm, parameterized by a signature scheme $S = (\mathsf{Gen}, \mathsf{Sig}, \mathsf{Ver})$.

```
function GenAddr_{S,H}(1^κ)
    (pk, sk) ← Gen(1^κ)
    pkh ← H(pk)
    return (pkh, sk)
end function
function SpendVerify_{S,H}(m, σ, pkh)
    (pk, σ') ← σ
    return (H(pk) = pkh ∧ Ver(m, σ', pk))
end function
```

-> Already studied.

of Bitcoin in Appendix A). GenAddr uses S to generate a keypair and hashes the public key to generate the public key hash. A tuple consisting of the public key hash and the secret key is returned. SpendVerify takes a spending transaction m, a scriptSig σ and a public key hash pkh. The scriptSig should contain the public key pk corresponding to pkh such that $H(pk) = pkh$ and a valid signature σ' for the spending transaction m [2]. If these conditions are met, the function returns true, otherwise it returns false. The blockchain address protocol is illustrated in Algorithm 4.

4 Comparison

We now compare three alternatives for proof-of-burn proposed in previous work against our scheme: OP_RETURN, P2SH OP_RETURN and nothing-up-my-sleeve. These schemes are instances of our burn primitive.

We study whether the aforementioned schemes satisfy binding, unspendability and uncensorability. Additionally, we compare them on how

easily they translate to multiple cryptocurrencies, a property we call *flexibility*, as well as whether a standard *user friendly* wallet can be used to burn money. A summary of our comparison is illustrated on Table 1.

Table 1. Comparison between proof-of-burn schemes.

	Binding	Flexible	Unspendable	Uncensorable	User friendly
OP_RETURN	•		•		
P2SH OP_RETURN	•		•	•	•
Nothing-up-my-sleeve		•	•	•	•
$a \oplus 1$ (this work)	•	•	•	•	•

Binding simply due to a key

OP_RETURN. Bitcoin supplies a native OP_RETURN [5] opcode. The Bitcoin Script interpreter deems an output **unspendable** when this opcode is encountered. The tag is included directly in the Bitcoin Script, hence the scheme is **binding** by definition. This Bitcoin-specific opcode is **inflexible** and does not translate to other cryptocurrencies such as Monero [31]. It is trivially **censorable**. However, the output is prunable, benefiting the network. Standard wallets **do not provide a user friendly interface** for such transactions. Any provably failing [28] Bitcoin Script can be used in OP_RETURN's stead.

P2SH OP_RETURN. An OP_RETURN can be used as the redeemScript for a Pay to Script Hash (P2SH) [4] address. **Binding** and **unspendability** are accomplished by the collision resistance of the hash function RIPEMD160 ∘ SHA256. Similarly to OP_RETURN this scheme is **inflexible**. From the one-wayness of the hash function it is **uncensorable**. Finally, the scheme is **user friendly** since any wallet can create a burn transaction.

Also in my notes!

Appendix: Don't

Nothing-up-my-sleeve. An address is manually crafted so that it is clear it was not generated from a regular keypair. For example, the all-zeros address is considered nothing-up-my-sleeve [4]. The scheme is **not binding**, as no tag can be associated with such a burn, and **flexible** because such an address can be generated for any cryptocurrency. It is hard to obtain a public key hashing to this address, thus funds sent to it are **unspendable**. On the other hand, because a widely known address is used, the scheme is **censorable**. Finally, the address is a regular recipient and any wallet can be used to fund it, making it **user friendly**.

[4] The Bitcoin address 1111111111111111111114oLvT2 encodes the all-zeros string and has received more than 50,000 transactions dating back to Aug 2010.