Proof-of-Burn

Kostis Karantias[1], Aggelos Kiayias[1,3], and Dionysis Zindros[1,2]

[1] IOHK
[2] University of Athens
[3] University of Edinburgh

Abstract. *Proof-of-burn* has been used as a mechanism to destroy cryptocurrency in a verifiable manner. Despite its well known use, the mechanism has not been previously formally studied as a primitive. In this paper, we put forth the first cryptographic definition of what a proof-of-burn protocol is. It consists of two functions: First, a function which generates a cryptocurrency address. When a user sends money to this address, the money is irrevocably destroyed. Second, a verification function which checks that an address is really unspendable. We propose the following properties for burn protocols. *Unspendability*, which mandates that an address which verifies correctly as a burn address cannot be used for spending; *binding*, which allows associating metadata with a particular burn; and *uncensorability*, which mandates that a burn address is indistinguishable from a regular cryptocurrency address. Our definition captures all previously known proof-of-burn protocols. Next, we design a novel construction for burning which is simple and flexible, making it compatible with all existing popular cryptocurrencies. We prove our scheme is secure in the Random Oracle model. We explore the application of destroying value in a legacy cryptocurrency to bootstrap a new one. The user burns coins in the source blockchain and subsequently creates a proof-of-burn, a short string proving that the burn took place, which she then submits to the destination blockchain to be rewarded with a corresponding amount. The user can use a standard wallet to conduct the burn without requiring specialized software, making our scheme user friendly. We propose burn verification mechanisms with different security guarantees, noting that the target blockchain miners do not necessarily need to monitor the source blockchain. Finally, we implement the verification of Bitcoin burns as an Ethereum smart contract and experimentally measure that the gas costs needed for verification are as low as standard Bitcoin transaction fees, illustrating that our scheme is practical.

1 Introduction

Since the dawn of history, humans have entertained the defiant thought of money burning, sometimes literally, for purposes ranging from artistic effect [9] to protest [23], or to prevent it from falling into the hands of pirates [22,12]. People did not shy away from the practice in the era of

cryptocurrencies. Acts of money burning immediately followed the inception of Bitcoin [24] in 2009, with the first recorded instance of intentional cryptocurrency destruction taking place on August 2010 [29], a short three months after the first real-world transaction involving cryptocurrency in May 2010 [8]. For the first time, however, cryptocurrencies exhibit the unique ability for money burning to be provable retroactively in a so-called *proof-of-burn*.

First proposed by Iain Stewart in 2012 [28], proof-of-burn constitutes a mechanism for the destruction of cryptocurrency irrevocably and provably. The ability to create convincing proofs changed the practice of money burning from a fringe act to a rational and potentially useful endeavour. It has since been discovered that metadata of the user's choice—a so-called *tag*—can be uniquely ascribed to an act of burning, allowing each burn to become tailored to a particular purpose. Such protocols have been used as a consensus mechanism similar to proof-of-stake (Slimcoin [25]), as a mechanism for establishing identity (OpenBazaar [26,34]), and for notarization (Carbon dating [13] and OpenTimestamps [30]). A particularly apt use case is the destruction of one type of cryptocurrency to create another. In one prolific case, users destroyed more than 2,130.87 BTC ($1.7M at the time, $21.6M in today's prices) for the bootstrapping of the Counterparty cryptocurrency [1].

While its adoption is undeniable, there has not been a formal treatment for proof-of-burn. This is the gap this work aims to fill.

Our contributions. A summary of our contributions is as follows:

(i) **Primitive definition.** Our definitional contribution introduces proof-of-burn as a cryptographic primitive for the first time. We define it as a protocol which consists of two algorithms, a burn address *generator* and a burn address *verifier*. We put forth the foundational properties which make for secure burn protocols, namely *unspendability*, *binding*, and *uncensorability*. One of the critical features of our formalization is that a tag has to be bound cryptographically with any proof-of-burn operation.

(ii) **Novel construction.** We propose a novel and simple construction which is flexible and can be adapted for use in existing cryptocurrencies, as long as they use public key hashes for address generation. To our knowledge, all popular cryptocurrencies are compatible with our scheme. We prove our construction secure in the Random Oracle model [6].

(iii) **Bootstrapping mechanism.** We propose a cryptocurrency proof-of-burn bootstrapping mechanism which for the first time does not require target blockchain miners to connect to external blockchain networks.

Our mechanism in principle allows burning from any proof-of-work-based cryptocurrency.
(iv) **Experimental results.** We provide a compehensively tested production grade implementation of the bootstrapping mechanism in Ethereum written in Solidity, which we release as open source software. Our implementation can be used to consume proofs of burn of a source blockchain within a target blockchain. We provide experimental measurements for the cost of burn verification and find that, in current Ethereum prices, burn verification costs $0.28 per transaction. This allows coins burned on one blockchain to be consumed on another for the purposes of, for example, ERC-20 tokens creation [32].

Workflow. A user who wishes to burn her coins generates an address which we call a *burn address*. This address encodes some user-chosen metadata called the *tag*. She then proceeds to send any amount of cryptocurrency to the burn address. After burning her cryptocurrency, she proves to any interested party that she irrevocably destroyed the cryptocurrency in question.
Properties. We define the following properties for a proof-of-burn protocol:

- **Unspendability.** No one can spend the burned cryptocurrency.
- **Binding.** The burn commits only to a single tag.
- **Uncensorability.** Miners who do not agree with the scheme cannot censor burn transactions.

Finally, we consider the *usability* of a proof-of-burn protocol important: whether a user is able to create a burn transaction using her regular cryptocurrency wallet.
Notation. We use $\mathcal{U}(S)$ to denote the uniform distribution obtained by sampling any item of the finite set S with probability $\frac{1}{|S|}$. We denote the support of a distribution $\mathcal{D}$ by $[\mathcal{D}]$. We also use $[n]$ to denote the set of integers from 1 to n. We denote the empty string by ϵ and string concatenation by $\|$.

2 Defining Proof-of-Burn

We now formally define what a proof-of-burn protocol is. Let κ be the security parameter. The protocol consists of two functions $\mathsf{GenBurnAddr}$ and $\mathsf{BurnVerify}$ and works as follows. Alice first generates an address $\mathsf{burnAddr}$ to which she sends some cryptocurrency. The address is generated by invoking $\mathsf{GenBurnAddr}(1^\kappa, t)$ and encodes information contained in a tag t,

a string of Alice's choice. When the transaction is completed, she gives the transaction and tag to Bob who invokes $\mathsf{BurnVerify}(1^\kappa, t, \mathsf{burnAddr})$ to verify she irrevocably destroyed the cryptocurrency while committing to the provided tag.

Definition 1 (Burn protocol). *A* burn *protocol Π consists of two functions* $\mathsf{GenBurnAddr}(1^\kappa, t)$ *and* $\mathsf{BurnVerify}(1^\kappa, t, \mathsf{burnAddr})$ *which work as follows:*

- $\mathsf{GenBurnAddr}(1^\kappa, t)$*: Given a tag $t \in \{0,1\}^*$, generate a* burn address*.*
- $\mathsf{BurnVerify}(1^\kappa, t, \mathsf{burnAddr})$*: Given a tag $t \in \{0,1\}^*$ and an address* $\mathsf{burnAddr}$*, return* true *if and only if* $\mathsf{burnAddr}$ *is a burn address and correctly encodes t.*

We require that the burn scheme is *correct*.

Definition 2 (Correctness). *A burn protocol Π is* correct *if for all $t \in \{0,1\}^*$ and for all $\kappa \in \mathbb{N}$ it holds that* $\mathsf{BurnVerify}(1^\kappa, t, \mathsf{GenBurnAddr}(1^\kappa, t)) = \mathsf{true}$*.*

With foresight, we remark that the implementation of $\mathsf{GenBurnAddr}$ and $\mathsf{BurnVerify}$ will typically be deterministic, which alleviates the need for a probabilistic correctness definition.

Naturally, for $\mathsf{GenBurnAddr}$ to generate addresses that "look" valid but are unspendable according to the blockchain protocol requires that the burn protocol respects its format. We abstract the address generation and spending verification of the given system into a *blockchain address protocol*:

Definition 3 (Blockchain address protocol). *A* blockchain address protocol *Π_α consists of two functions* $\mathsf{GenAddr}$ *and* $\mathsf{SpendVerify}$*:*

- $\mathsf{GenAddr}(1^\kappa)$*: Returns a tuple* $(\mathsf{pk}, \mathsf{sk})$*, denoting the cryptocurrency address* pk *(a public key) used to receive money and its respective secret key* sk *which allows spending from that address.*
- $\mathsf{SpendVerify}(\mathsf{m}, \sigma, \mathsf{pk})$*: Returns* true *if the transaction m spending from receiving address pk has been authorized by the signature σ (by being signed by the respective private key).*

We note that, while the blockchain address protocol is not part of the burn protocol, the *security* properties of a burn protocol Π will be defined *with respect to* a blockchain address protocol Π_α.

Algorithm 1 The challenger for the burn protocol game-based security.

function SPEND-ATTACK$_{\mathcal{A},\Pi}(\kappa)$
    $(t, m, \sigma, pk) \leftarrow \mathcal{A}(1^\kappa)$
    return ($\mathsf{BurnVerify}(1^\kappa, t, pk) \wedge \mathsf{SpendVerify}(m, \sigma, pk)$)
end function

Algorithm 2 The challenger for the burn protocol game-based security.

function BIND-ATTACK$_{\mathcal{A},\Pi}(\kappa)$
    $(t, t', \mathsf{burnAddr}) \leftarrow \mathcal{A}(1^\kappa)$
    return ($t \neq t' \wedge \mathsf{BurnVerify}(1^\kappa, t, \mathsf{burnAddr}) \wedge \mathsf{BurnVerify}(1^\kappa, t', \mathsf{burnAddr})$)
end function

These two functionalities are typically implemented using a public key signature scheme and accompanied by a respective signing algorithm. The signing algorithm is irrelevant for our burn purposes, as burning entails the inability to spend. As the format of m is cryptocurrency-specific, we intentionally leave it undefined. In both Bitcoin and Ethereum, m corresponds to transaction data. When a new candidate transaction is received from the network, the blockchain node calls $\mathsf{SpendVerify}$, passing the public key pk, which is the address spending money incoming to the new transaction m, together with a signature σ, which signs m and should be produced using the respective secret key.

To state that the protocol generates addresses which cannot be spent from, we introduce a game-based security definition. The unspendability game SPEND-ATTACK is illustrated in Algorithm 1.

Definition 4 (Unspendability). *A burn protocol Π is* unspendable *with respect to a blockchain address protocol Π_α if for all probabilistic polynomial-time adversaries $\mathcal{A}$ there exists a negligible function $\mathsf{negl}(\kappa)$ such that* $\Pr[\text{SPEND-ATTACK}_{\mathcal{A},\Pi}(\kappa) = \mathsf{true}] \leq \mathsf{negl}(\kappa)$.

It is desired that a burn address encodes one and only one tag. Concretely, given a burn address $\mathsf{burnAddr}$, $\mathsf{BurnVerify}(1^\kappa, t, \mathsf{burnAddr})$ should only evaluate to true for a single tag t. The game BIND-ATTACK in Algorithm 2 captures this property.

Definition 5 (Binding). *A burn protocol Π is* binding *if for all probabilistic polynomial-time adversaries $\mathcal{A}$ there is a negligible function $\mathsf{negl}(\kappa)$ such that* $\Pr[\text{BIND-ATTACK}_{\mathcal{A},\Pi}(\kappa)] \leq \mathsf{negl}(\kappa)$.

We note here that the correctness and binding properties of a burn protocol are irrespective of the blockchain address protocol it was designed for.

We are now ready to define what constitutes a *secure proof-of-burn protocol*.

Definition 6 (Security). *Let Π be a correct burn protocol. We say Π is* secure *with respect to a blockchain address protocol Π_α if it is* unspendable *and* binding *with respect to Π_α.*

The aforementioned properties form a good basis for a burn protocol. We observe that it may be possible to detect whether an address is a burn address. While this is desirable in certain circumstances, it allows miners to censor burn transactions. To mitigate this, we propose *uncensorability*, a property which mandates that a burn address is indistinguishable from a regular address if its tag is not known. During the execution of protocols which satisfy this property, when the burn transaction appears on the network, only the user who performed the burn knows that it constitutes a burn transaction prior to revealing the tag. Naturally, as soon as the tag is revealed, *correctness* mandates that the burn transaction becomes verifiable.

Definition 7 (Uncensorability). *Let $\mathcal{T}$ be a distribution of tags. A burn protocol Π is* uncensorable *if the distribution ensembles $\{(pk, sk) \leftarrow \mathsf{GenAddr}(1^\kappa); pk\}_\kappa$ and $\{t \leftarrow \mathcal{T}; pk \leftarrow \mathsf{GenBurnAddr}(1^\kappa, t); pk\}_\kappa$ are computationally indistinguishable.*

3 Construction

We now present our construction for an uncensorable proof-of-burn protocol. To generate a burn address, the tag t is hashed and a perturbation is performed on the hash by toggling the last bit. Verifying a burn address $\mathsf{burnAddr}$ encodes a certain tag t is achieved by invoking $\mathsf{GenBurnAddr}$ with tag t and checking whether the result matches $\mathsf{burnAddr}$. If it matches, the $\mathsf{burnAddr}$ correctly encodes t. Our construction is illustrated in Algorithm 3.

We outline the blockchain address protocol for Bitcoin Pay to Public Key Hash (P2PKH) [2], with respect to which we prove our construction secure and uncensorable in Section 5. It is parametrized by a secure signature scheme S and a hash function H (for completeness, we give a construction which includes the concrete hash functions and checksums

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

function $\mathsf{GenBurnAddr}_H(1^\kappa, t)$
    $th \leftarrow H(t)$
    $th' \leftarrow th \oplus 1$ ▷ Key perturbation
    return th'
end function
function $\mathsf{BurnVerify}_H(1^\kappa, t, th')$
    return $(\mathsf{GenBurnAddr}_H(1^\kappa, t) = th')$
end function

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

function $\mathsf{GenAddr}_{S,H}(1^\kappa)$
    $(pk, sk) \leftarrow \mathsf{Gen}(1^\kappa)$
    $pkh \leftarrow H(pk)$
    return (pkh, sk)
end function
function $\mathsf{SpendVerify}_{S,H}(m, \sigma, pkh)$
    $(pk, \sigma') \leftarrow \sigma$
    return $(H(pk) = pkh \wedge \mathsf{Ver}(m, \sigma', pk))$
end function

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)**	•	•	•	•	•

`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 $\texttt{RIPEMD160} \circ \texttt{SHA256}$. 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.

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.

5 Analysis

We now move on to the analysis of our scheme. As the scheme is deterministic, its correctness is straightforward to show.

Theorem 1 (Correctness). *The proof-of-burn protocol Π of Section 3 is* correct*.*

Proof. Based on Algorithm 3, $\mathsf{BurnVerify}(1^\kappa, t, \mathsf{GenBurnAddr}(1^\kappa, t)) = \mathsf{true}$ if and only if $\mathsf{GenBurnAddr}(1^\kappa, t) = \mathsf{GenBurnAddr}(1^\kappa, t)$, which always holds as $\mathsf{GenBurnAddr}$ is deterministic. □

We now state a simple lemma pertaining to the distribution of Random Oracle outputs.

Lemma 1 (Perturbation)**.** *Let $p(\kappa)$ be a polynomial and $F : \{0,1\}^\kappa \longrightarrow \{0,1\}^\kappa$ be a permutation. Consider the process which samples $p(\kappa)$ strings $s_1, s_2, \ldots, s_{p(\kappa)}$ uniformly at random from the set $\{0,1\}^\kappa$. The probability that there exists $i \neq j$ such that $s_i = F(s_j)$ is negligible in κ.*

We will now apply the above lemma to show that our scheme is unspenable.

Theorem 2 (Unspendability). *If H is a* Random Oracle*, then the protocol Π of Section 3 is* unspendable*.*

Proof. Let $\mathcal{A}$ be an arbitrary probabilistic polynomial time SPEND-ATTACK adversary. $\mathcal{A}$ makes at most a polynomial number of queries $p(\kappa)$ to the Random Oracle. Let MATCH denote the event that there exist $i \neq j$ with $s_i = F(s_j)$ where $F(s) = s \oplus 1$.

If the adversary is successful then it has presented t, pk, pkh such that $H(pk) = pkh$ and $H(t) \oplus 1 = pkh$. Observe that $\text{SPEND-ATTACK}_{\mathcal{A},\Pi}(\kappa) = \mathsf{true} \Rightarrow \text{MATCH}$. Therefore $\Pr[\text{SPEND-ATTACK}_{\mathcal{A},\Pi}(\kappa)] \leq Pr[\text{MATCH}]$. Apply Lemma 1 on F to obtain $\Pr[\text{SPEND-ATTACK}_{\mathcal{A},\Pi}(\kappa)] \leq \mathsf{negl}(\kappa)$. □

We note that the security of the signature scheme is not needed to prove unspendability. Were the signature scheme of the underlying cryptocurrency ever found to be *forgeable*, the coins burned through our scheme would remain unspendable. We additionally remark that the choice of the permutation $F(x) = x \oplus 1$ is arbitrary. Any one-to-one function beyond the identity function would work equally well.

Preventing proof-of-burn. It is possible for a cryptocurrency to prevent proof-of-burn by requiring every address to be accompanied by a

Algorithm 5 The collision adversary $\mathcal{A}^*$ against H using a proof-of-burn BIND-ATTACK adversary $\mathcal{A}$.

```
1: function A*_A(1^κ)
2:     (t, t', _) ← A(1^κ)
3:     return (t, t')
4: end function
```

proof of possession [27]. To the best of our knowledge, no cryptocurrency features this.

Next, our binding theorem only requires that the hash function used is collision resistant and is in the standard model.

Theorem 3 (Binding). *If H is a* collision resistant *hash function then the protocol of Section 3 is* binding.

Proof. Let $\mathcal{A}$ be an arbitrary adversary against Π. We will construct the Collision Resistance adversary $\mathcal{A}^*$ against H.

The collision resistance adversary, illustrated in Algorithm 5, calls $\mathcal{A}$ and obtains two outputs, t and t'. If $\mathcal{A}$ is successful then $t \neq t'$ and $H(t) \oplus 1 = H(t') \oplus 1$. Therefore $H(t) = H(t')$.

We thus conclude that $\mathcal{A}^*$ is successful in the COLLISION game if and only if $\mathcal{A}$ is successful in the BIND-ATTACK game.

$$\Pr[\text{BIND-ATTACK}_{\mathcal{A},\Pi}(\kappa) = \mathsf{true}] = \Pr[\text{COLLISION}_{\mathcal{A}^*,H}(\kappa) = \mathsf{true}]$$

From the collision resistance of H it follows that $\Pr[\text{COLLISION}_{\mathcal{A}^*,H} = \mathsf{true}] < \mathsf{negl}(\kappa)$. Therefore, $\Pr[\text{BIND-ATTACK}_{\mathcal{A},\Pi} = \mathsf{true}] < \mathsf{negl}(\kappa)$, so the protocol Π is binding. □

We now posit that no adversary can predict the public key of a secure signature scheme, except with negligible probability. We call a distribution *unpredictable* if no probabilistic polynomial-time adversary can predict its sampling. We give the formal definition, with some of its statistical properties, in Appendix B.2.

Lemma 2 (Public key unpredictability)**.** *Let $S = (\mathsf{Gen}, \mathsf{Sig}, \mathsf{Ver})$ be a secure signature scheme. Then the distribution ensemble $X_\kappa = \{(sk, pk) \leftarrow \mathsf{Gen}(1^\kappa); pk\}$ is unpredictable.*

The following lemma shows that the output of the random oracle is indistinguishable from random if the input is unpredictable (for the

complete proofs see Appendix B.3). For reference, the definition of computational indistinguishability is included in Appendix B.1.

Lemma 3 (Random Oracle unpredictability)**.** *Let $\mathcal{T}$ be an unpredictable distribution ensemble and H be a Random Oracle. The distribution ensemble $X = \{t \leftarrow \mathcal{T}; H(t)\}$ is indistinguishable from the uniform distribution ensemble $\mathcal{U}(\{0,1\}^\kappa)$.*

Theorem 4 (Uncensorability). *Let $S = (\mathsf{Gen}, \mathsf{Sig}, \mathsf{Ver})$ be a* secure signature scheme*, H be a* Random Oracle*, and $\mathcal{T}$ be an unpredictable tag distribution. Then the protocol of Section 3 instantiated with $H, S, \mathcal{T}$ is* uncensorable*.*

Proof. Let X be the distribution ensemble of public keys generated using $\mathsf{GenAddr}$ and Y that of keys generated using $\mathsf{GenBurnAddr}$.

From Lemma 2 the distribution of public keys generated from S is unpredictable. The function $\mathsf{GenAddr}$ samples a public key from S and applies the random oracle H to it. Applying Lemma 3, we obtain that $X \approx_c \mathcal{U}(\{0,1\}^\kappa)$.

The function $H'(x) = H(x) \oplus 1$ is a random oracle (despite not being independent from the random oracle H). Since $\mathcal{T}$ is unpredictable, and applying Lemma 3 with random oracle H', we obtain that $Y \approx_c \mathcal{U}(\{0,1\}^\kappa)$.

By transitivity, X and Y are computationally indistinguishable. □

From the above, we conclude that the tags used during the burn process must be unpredictable. If the tag is chosen to contain a randomly generated public key from a secure signature scheme, or its hash, Lemmas 2 and 3 show that sufficient entropy exists to ensure uncensorability. Our cross-chain application makes use of this fact.

6 Consumption

Over the last 5 years there has been an explosion of new cryptocurrencies. Unfortunately, it is hard for a new cryptocurrency to gain traction. Without traction, no market depth ensues and a cryptocurrency has difficulty getting listed in exchanges. But without being listed in exchanges, a cryptocurrency cannot gain traction.

This chicken-and-egg situation presents the need for a solution that circumvents exchanges and allows users to acquire the cryptocurrency directly. We propose utilizing proof-of-burn to allow users to obtain capital on a new *target* cryptocurrency by burning a legacy *source* cryptocurrency. The target blockchain may support burning from multiple sources.

Workflow. A user wishes to acquire a target cryptocurrency. She uses her target address as a tag to generate a source burn address. She then sends an amount of source cryptocurrency to that address. She submits a proof of this burn to a smart contract [11] on the target chain, where it is verified and she is credited an equivalent amount of currency. Proof-of-burn verification happens in either a centralized manner which is lighter on computation, or in a decentralized manner using Non-Interactive Proofs of Proof-of-Work (NIPoPoWs) [19,20,18,17,10]. Target miners need not be connected to every other source blockchain network. We call this property *miner-isolation* and propose methods to achieve it.

We now describe how a target smart contract verifies a burn took place on the source chain. We make use of the Proof-of-Work Sidechains mechanism [21] in which they propose a generic information transfer construction. We tailor it towards our purposes for proof-of-burn transfers. We call the user the *prover* and the smart contract the *verifier*. The prover wishes to convince the verifier that an event occurred on the source chain. We define an event as a simple value transfer described by a transaction id txid, a receiving address addr and an amount. Simple value transfers are supported by all cryptocurrencies, allowing a verifier to process burns from a wide range of source blockchains. Note that this event type does not yet distinguish between burn and non-burn addresses.

A verifier checks an event occurred on the source chain by ensuring its transaction is contained in a stable block [15,16] in the best source chain. Specifically, the following data are supplied to the smart contract as a proof:

- tx: The transaction which contains the burn on the source chain.
- b: The block header for the block which contains tx.
- τ_{tx}: An inclusion proof showing $\mathsf{tx} \in b$.
- τ_{b}: A proof that b is contained in the best (i.e., most proof-of-work) source chain and is stable.

We assume the source blockchain provides a function $\mathsf{verify\text{-}tx}(\mathsf{addr}, \mathsf{amount}, b, \mathsf{tx}, \tau_{\mathsf{tx}})$ which can be written in the smart contract language of the target blockchain and verifies the validity of a source transaction. It takes a source address addr, an amount, a block b, a transaction tx and a proof τ_{tx} for the inclusion of tx in b. It returns true if tx contains a transfer of amount to addr and the proof τ_{tx} is valid.

The proof τ_{tx} is usually a Merkle Tree inclusion proof. More concretely, in Bitcoin, each block header contains a commitment to the set of transaction ids in the block in the form of a Merkle Tree root. Ethereum

stores a similar commitment in its header — the root of a Merkle–Patricia Trie [33].

For verifying that a provided block b belongs to the best source blockchain and is stable, we assume the existence of a function in-best-chain(b). We explore how it can be implemented in the "Verifying block connection" paragraph below.

Bootstraping mechanism. Being able to verify events, we can grant target cryptocurrency to users who burn source cryptocurrency. After burning on the source blockchain, the user calls the claim function with the aforementioned event and a proof for it. This function ensures that the event provided is valid and has not been claimed before (i.e. no one has been granted target cryptocurrency for this specific event in the past), that it corresponds to the transaction tx and that the block b is stable, belongs to the best source chain and contains tx. Then, after verifying by invoking BurnVerify that the receiving address of the event is a burn address where the tag is the function caller's address, it releases the amount of coins burned in the form of an ERC-20 token. We present the contract burn-verifier with this capability in Algorithm 6.

Algorithm 6 A contract for verifying burns from the source chain. This smart contract runs within the target blockchain.

```
1: contract burn-verifier extends crosschain; ERC20
2:     mapping(address ⇒ uint256) balances
3:     claimed-events ← ∅
4:     function claim(e, b, τ_tx)
5:         block-ok ← in-best-chain(b)
6:         tx-ok ← verify-tx(e.addr, e.amount, b, e.tx, τ_tx)
7:         event-ok ← e ∉ claimed-events
8:         if block-ok ∧ tx-ok ∧ event-ok ∧ BurnVerify(msg.sender, e.addr) then
9:             claimed-events ← claimed-events ∪ {e}
10:            balances[msg.sender] += e.amount
11:        end if
12:    end function
13: end contract
```

In the interest of keeping this implementation generic we assume that the user receives a token in return for his burn. However, instead of minting a token, the target cryptocurrency could allow the burn verifier contract to mint native cryptocurrency for any user who successfully claims an event. This would allow the target cryptocurrency to be bootstrapped entirely though burning as desired.

Verifying block connection. We now shift our attention to the problem of verifying a block belongs in the best source chain. We provide multiple ways of implementing the aforementioned in-best-chain method.

Direct observation. Miners connect to the source blockchain network and have access to the best source chain. A miner can thus evaluate if a block is included in that chain and is stable. This mechanism does not provide miner-isolation. It is adopted by Counterparty.

NIPoPoWs. Verifying block connection can be achieved through NIPoPoWs, as in [21]. We remark that with this setup a block connection proof may be considered valid provisionally, but there needs to be a period in which the proof can be disputed for the smart contract to be certain for the validity of the proof. Specifically, when a user performs a claim, they have to put down some collateral. If they have provided a valid NIPoPoW, a contestation period begins. Within that period a challenger can dispute the provided proof which – provided that the dispute is successful – would turn the result of in-best-chain to false, abort the claim and grant the challenger the user's collateral. If the contestation period ends with the proof undisputed, then in-best-chain evaluates to true, the collateral gets returned to the user and the claim is performed successfully.

Federation. A simpler approach is to allow a federation of n nodes monitoring the source chain to vote for their view of the best source chain. This construction works under the assumption that the majority of the federation is honest.

The best source chain is expressed as the root $\mathcal{M}$ of a Merkle Tree containing the chain's stable blocks as leaves. Each federation node connects to both blockchain networks, calculates $\mathcal{M}$ and submits their vote for it every time a new source chain block is found. When a majority of $\lfloor \frac{n}{2} \rfloor + 1$ nodes agrees on the same $\mathcal{M}$, it is considered valid.

Having a valid $\mathcal{M}$, a verifier verifies a Merkle Tree inclusion proof τ_{b} for $b \in \mathcal{M}$ and is certain the block provided is part of the best source chain and is stable. This approach is illustrated in Algorithm 7. The more suitable Merkle Mountain Range [10] data structure can be used to store $\mathcal{M}$ in place of regular Merkle Trees, as they constitute a more efficient append-only structure.

7 Empirical Results

In order to evaluate our consumption mechanisms, we implement the federated consumption mechanism in Solidity. We provide a concrete implementation of the burn-verifier contract described in Algorithm 6. We

Algorithm 7 A in-best-chain implementation which verifies that a block b is included in the best source chain using the federation mechanism. $\mathcal{M}$ denotes the latest MMR approved by the federation majority.

votes $\leftarrow \emptyset$
best-idx $\leftarrow 0$
$\mathcal{M} \leftarrow \epsilon$
function vote$_{\text{FED}}(m, \sigma, pk)$
    if $pk \in$ FED $\wedge$ Ver(m, σ, pk) **then** ▷ Check that pk is a valid federation member
        $(\mathcal{M}^*, \text{idx}) \leftarrow m$
        votes$[m] \leftarrow$ votes$[m] \cup \{pk\}$
        if $|\text{votes}[m]| \geq \lfloor \frac{|\text{FED}|}{2} \rfloor + 1 \wedge \text{idx} > \text{best-idx}$ **then**
            $\mathcal{M} \leftarrow \mathcal{M}^*$ ▷ Update accepted MMR
            best-idx $\leftarrow$ idx
        end if
    end if
end function
function in-best-chain$_{\mathcal{M}}(b, \tau_{\text{b}})$
    return Ver$\mathcal{MT}(\mathcal{M}, b, \tau_{\text{b}})$
end function

implement the crosschain parent contract from [21]. We verify transaction data by making use of the open source bitcoin-spv library [3]. Finally, the federation mechanism for verifying block connection is employed. The members of the federation can vote on their computed checkpoints using the vote function.

We release our implementation as open source software under the MIT license[5]. The implementation is production-ready and fully tested with 100% code coverage.

At the time of writing we obtain the median gas price of 6.9 gwei and the price of Ethereum in US Dollars at $170.07. The cost of gas in USD is calculated by the formula $gas * 1.173483 * 10^{-6}$ rounded to two decimal places.

Method	Gas cost	Equivalent in USD
vote	50103 gas	$0.06
submit-event-proof	157932 gas	$0.19
claim	78267 gas	$0.09
Total claim cost	262817 gas	$0.28

For the end user to prove an event and claim her burn, the cost is thus $0.28. Comparatively, for a Bitcoin transaction to be included in the next block at the time of writing a user has to spend $0.77.

[5] https://github.com/decrypto-org/burn-paper/tree/master/experiment.

References

1. Counterparty. Available at: https://counterparty.io/.
2. Developer guide - bitcoin. Available at: https://bitcoin.org/en/developer-guide.
3. summa-tx/bitcoin-spv: utilities for bitcoin spv proof verification on other chains. Available at: https://github.com/summa-tx/bitcoin-spv/.
4. Gavin Andresen. BIP 0016: Pay to Script Hash. Available at: https://github.com/bitcoin/bips/blob/master/bip-0016.mediawiki, Jan 2012.
5. Massimo Bartoletti and Livio Pompianu. An analysis of Bitcoin OP_RETURN metadata. In *International Conference on Financial Cryptography and Data Security*, pages 218–230. Springer, 2017.
6. Mihir Bellare and Phillip Rogaway. Random oracles are practical: A paradigm for designing efficient protocols. In V. Ashby, editor, *ACM CCS 93*, pages 62–73. ACM Press, November 1993.
7. Manuel Blum, Paul Feldman, and Silvio Micali. Non-interactive zero-knowledge and its applications (extended abstract). In *20th ACM STOC*, pages 103–112. ACM Press, May 1988.
8. Joseph Bonneau, Andrew Miller, Jeremy Clark, Arvind Narayanan, Joshua A. Kroll, and Edward W. Felten. SoK: Research perspectives and challenges for bitcoin and cryptocurrencies. In *2015 IEEE Symposium on Security and Privacy*, pages 104–121. IEEE Computer Society Press, May 2015.
9. Chris Brook. *K Foundation Burn a Million Quid*. Ellipsis Books, 1997.
10. Benedikt Bünz, Lucianna Kiffer, Loi Luu, and Mahdi Zamani. Flyclient: Superlight clients for cryptocurrencies. *IACR Cryptology ePrint Archive*, 2019:226, 2019.
11. Vitalik Buterin et al. A next-generation smart contract and decentralized application platform. *white paper*, 2014.
12. Cicero. *De Inventione*, page 133. Kessenger Publishing, 2004.
13. Jeremy Clark and Aleksander Essex. Commitcoin: Carbon dating commitments with bitcoin. In *International Conference on Financial Cryptography and Data Security*, pages 390–398. Springer, 2012.
14. Apoorvaa Deshpande and Yael Kalai. Proofs of ignorance and applications to 2-message witness hiding. *IACR Cryptology ePrint Archive*, 2018:896, 2018.
15. Juan A. Garay, Aggelos Kiayias, and Nikos Leonardos. The bitcoin backbone protocol: Analysis and applications. In Elisabeth Oswald and Marc Fischlin, editors, *EUROCRYPT 2015, Part II*, volume 9057 of *LNCS*, pages 281–310. Springer, Heidelberg, April 2015.
16. Juan A. Garay, Aggelos Kiayias, and Nikos Leonardos. The bitcoin backbone protocol with chains of variable difficulty. In Jonathan Katz and Hovav Shacham, editors, *CRYPTO 2017, Part I*, volume 10401 of *LNCS*, pages 291–323. Springer, Heidelberg, August 2017.
17. Kostis Karantias. Enabling NIPoPoW Applications on Bitcoin Cash. Master's thesis, University of Ioannina, Ioannina, Greece, 2019.
18. Kostis Karantias, Aggelos Kiayias, and Dionysis Zindros. Compact Storage of Superblocks for NIPoPoW Applications. In *Mathematical Research for Blockchain Economy - 1st International Conference MARBLE 2019*. Springer, 2019.
19. Aggelos Kiayias, Nikolaos Lamprou, and Aikaterini-Panagiota Stouka. Proofs of proofs of work with sublinear complexity. In Jeremy Clark, Sarah Meiklejohn, Peter Y. A. Ryan, Dan S. Wallach, Michael Brenner, and Kurt Rohloff, editors, *FC 2016 Workshops*, volume 9604 of *LNCS*, pages 61–78. Springer, Heidelberg, February 2016.

20. Aggelos Kiayias, Andrew Miller, and Dionysis Zindros. Non-Interactive Proofs of Proof-of-Work, 2017.
21. Aggelos Kiayias and Dionysis Zindros. Proof-of-Work Sidechains. In *International Conference on Financial Cryptography and Data Security*. Springer, 2019.
22. Diogenes Laertius. *Lives of the Eminent Philosophers, Book II*, chapter 8, page 100. Oxford University Press, 2018.
23. Steven E Landsburg. *The Armchair Economist (revised and updated May 2012): Economics & Everyday Life*. Simon and Schuster, 2007.
24. Satoshi Nakamoto. Bitcoin: A peer-to-peer electronic cash system. Available at: https://bitcoin.org/bitcoin.pdf, 2008.
25. P4Titan. Slimcoin a peer-to-peer crypto-currency with proof-of-burn. Available at: https://www.doc.ic.ac.uk/~ids/realdotdot/crypto_papers_etc_worth_reading/proof_of_burn/slimcoin_whitepaper.pdf, May 2014.
26. Sam Patterson. Proof-of-burn and reputation pledges. Available at: https://www.openbazaar.org/blog/proof-of-burn-and-reputation-pledges/, Aug 2014.
27. Thomas Ristenpart and Scott Yilek. The power of proofs-of-possession: Securing multiparty signatures against rogue-key attacks. Cryptology ePrint Archive, Report 2007/264, 2007. http://eprint.iacr.org/2007/264.
28. Iain Stewart. Proof of burn - bitcoin wiki. Available at: https://en.bitcoin.it/wiki/Proof_of_burn, Dec 2012.
29. suppp. Interesting bitcoin address, bitcoin forum post. Available at: https://bitcointalk.org/index.php?topic=237143.0, 2013.
30. Peter Todd. Opentimestamps: Scalable, trustless, distributed timestamping with bitcoin (2016). Available at: https://petertodd.org/2016/opentimestamps-announcement, 2016.
31. Nicolas Van Saberhagen. Cryptonote v2.0. Available at: https://cryptonote.org/whitepaper.pdf, 2013.
32. Fabian Vogelsteller and Vitalik Buterin. Erc 20 token standard. Available at: https://github.com/ethereum/EIPs/blob/master/EIPS/eip-20.md, 2015.
33. Gavin Wood. Ethereum: A secure decentralised generalised transaction ledger. *Ethereum project yellow paper*, 151:1–32, 2014.
34. Dionysis Zindros. Trust in decentralized anonymous marketplaces. Master's thesis, National Technical University of Athens, Athens, Greece, 2016.

Appendix

Our Appendix is structured as follows. In Appendix A we show how our scheme can be adopted to the practical implementation details of Bitcoin. In Appendix B we provide the formal definitions and proofs of our claims. Finally, in Appendix C we discuss potential future directions in relaxing the Random Oracle assumption and propose a scheme for which we show some desirable properties in the Common Random String model [7].

A Deployment to Bitcoin

Algorithm 8 The Bitcoin blockchain address protocol, including the engineering details of checksums and practical hash implementation.

function GenAddr()
  $(pk, sk) \leftarrow \mathsf{Gen}()$
  $pkh \leftarrow \texttt{RIPEMD160}(\texttt{SHA256}(\texttt{0x04} \,\|\, pk))$
  $\mathsf{addr} \leftarrow \texttt{0x00} \,\|\, pkh$ ▷ Magic byte indicating *mainnet*
  $\mathsf{checksum} \leftarrow \texttt{SHA256}(\texttt{SHA256}(\mathsf{addr}))[:4]$ ▷ Keep the first 4 bytes
  return $\texttt{base58}(\mathsf{addr} \,\|\, \mathsf{checksum})$
end function

The scheme described above works for a generic P2PKH cryptocurrency and can be adapted to any cryptocurrency. We illustrate its suitability by giving a precise construction for Bitcoin, taking into account the engineering details that are behind the generation of a Bitcoin P2PKH address. A comparable approach can be used to generate Ethereum addresses or others.

The way Bitcoin generates P2PKH addresses is illustrated in Algorithm 8. Here, Gen generates an elliptic curve public key (of fixed key size $\kappa = 256$). After the elliptic curve public key is generated, it is marked by a magic number and subsequently hashed by the so-called `HASH160` algorithm, which consists of evaluating `RIPEMD160` on the `SHA256` of the public key. The resulting hash is additionally prefixed by a magic number indicating that the execution is taking place on the main net (and not the test net), and the final address, together with a checksum, are encoded using `base58` to obtain the final address.

Our burn algorithm follows the same structure for address generation, ensuring that the magic numbers and checksums validate correctly. In this construction, the hash function which is modelled as a random

Algorithm 9 The key perturbation algorithm which generates a provable proof-of-burn address which validates under Bitcoin.

```
1: function GenBurnAddr(t)
2:     th ← RIPEMD160(SHA256(t))
3:     th′ ← th ⊕ 0x01                          ▷ Key perturbation
4:     addr ← 0x00 ‖ th′                        ▷ Magic byte indicating mainnet
5:     checksum ← SHA256(SHA256(addr))[: 4]     ▷ Keep the first 4 bytes
6:     return base58(addr ‖ checksum)
7: end function
```

oracle is the `HASH160` algorithm. The algorithm is illustrated in Algorithm 9 and works as follows. Given a tag t, the user derives a 160-byte hash $th = \texttt{RIPEMD160}(\texttt{SHA256}(t))$ which looks like a public key hash. The least significant bit of th is then flipped to achieve unspendability. This produces the 20-byte *perturbated hash* th'. The perturbated hash is then prefixed with `0x00` to designate that we're working on the Bitcoin mainnet as usual. The checksum is calculated and appended to it, and the result is `base58` encoded into a Bitcoin address which correctly validates.

B Full proofs

In this section, we give the full proofs of our claims. Appendix B.1 proves some facts about computationally indistinguishable distributions. In Appendix B.2, we introduce unpredictable distributions and show that public keys are unpredictable. In Appendix B.3, we prove some facts about the statistical properties of random oracles, including Lemma 1 from which the unspendability of our scheme follows and Lemma 3 from which the uncensorability of our scheme follows.

B.1 Computational indistinguishability

We review the definition of computational indistinguishability between two distributions X and Y. Define the cryptographic game illustrated in Algorithm 10. Computational indistinguishability mandates that no adversary can win the game, except with negligible probability.

Definition 8 (Computational indistinguishability). *Two distribution ensembles* $\{X_\kappa\}_{\kappa\in\mathbb{N}}$ *and* $\{Y_\kappa\}_{\kappa\in\mathbb{N}}$ *are* computationally indistinguishable *if for every probabilistic polynomial-time adversary* $\mathcal{A}$*, there exists a negligible function* $\mathsf{negl}(\kappa)$ *such that* $\Pr[\textsc{dist-game}_{\mathcal{A},X,Y}(\kappa) = \mathsf{true}] < \mathsf{negl}(\kappa)$.

Algorithm 10 The challenger for computational indistinguishability.

function DIST-GAME$_{\mathcal{A},D_0,D_1}(\kappa)$
  $b \xleftarrow{\$} \{0,1\}$
  $z \leftarrow D_b$
  $b^* \leftarrow \mathcal{A}(z, 1^\kappa)$
  return $(b = b^*)$
end function

It is clear that applying an efficiently computable function to indistinguishable distributions preserves indistinguishability.

Algorithm 11 The distinguisher $\mathcal{A}^*$ between distributions X, Y which makes use of a distinguisher $\mathcal{A}$ between X' and Y'.

function $\mathcal{A}^*_{X,Y,f}(z, 1^\kappa)$
  return $\mathcal{A}(f(z))$
end function

Lemma 4 (Indistinguishability preservation)**.** *Given two computationally indistinguishable distribution ensembles $\{X_\kappa\}_{\kappa\in\mathbb{N}}$ and $\{Y_\kappa\}_{\kappa\in\mathbb{N}}$, let $\{f_\kappa\}_{\kappa\in\mathbb{N}}$, be an efficiently computable family of functions $X_\kappa \longrightarrow Y_\kappa$. Then the distribution ensembles $X' = \{f_\kappa(X_\kappa)\}_{\kappa\in\mathbb{N}}$ and $Y' = \{f_\kappa(Y_\kappa)\}_{\kappa\in\mathbb{N}}$ are computationally indistinguishable.*

Proof. Let $\mathcal{A}$ be a probabilistic polynomial-time distinguisher between X' and Y'. Consider the probabilistic polynomial-time distinguisher $\mathcal{A}^*$ between X and Y illustrated in Algorithm 11. Then $\Pr[\text{DIST-GAME}_{\mathcal{A}^*}(\kappa) = \mathsf{true}] = \Pr[\text{DIST-GAME}_{\mathcal{A}}(\kappa) = \mathsf{true}]$. As $\Pr[\text{DIST-GAME}_{\mathcal{A}^*}(\kappa) = \mathsf{true}] \leq \frac{1}{2} + \mathsf{negl}(\kappa)$, therefore $\Pr[\text{DIST-GAME}_{\mathcal{A}}(\kappa) = \mathsf{true}] \leq \frac{1}{2} + \mathsf{negl}(\kappa)$. □

B.2 Unpredictable distributions

We call a distribution ensemble *unpredictable* if no polynomial-time adversary can guess its output. The cryptographic predictability game is illustrated in Algorithm 12 and the security definition is given below.

Definition 9 (Unpredictable distribution). *A distribution ensemble $\{X_\kappa\}_{\kappa\in\mathbb{N}}$ is* unpredictable *if for all probabilistic polynomial-time adversaries $\mathcal{A}$ there is a negligible function $\mathsf{negl}(\kappa)$ such that*

$$\Pr[\text{PREDICT}_{\mathcal{A},X}(\kappa) = \mathsf{true}] < \mathsf{negl}(\kappa)\,.$$

Algorithm 12 The challenger for the distribution predictor.

```
1: function PREDICT_{A,X}(κ)
2:     x ← X
3:     x* ← A(1^κ)
4:     return (x = x*)
5: end function
```

We observe that, if each element of a distribution appears with negligible probability, then the distribution must be unpredictable.

Lemma 5 (Negligible unpredictability)**.** *Consider a distribution ensemble* $\{X_\kappa\}_{\kappa\in\mathbb{N}}$ *and a negligible function* $\mathsf{negl}(\kappa)$*. If*

$$\max_{x\in[X_\kappa]} \Pr_{x^*\leftarrow X_\kappa}[x^* = x] \leq \mathsf{negl}(\kappa)\,,$$

then X *is unpredictable.*

Proof. Consider a probabilistic polynomial-time adversary $\mathcal{A}$ which predicts X_κ. The adversary is not given any input beyond 1^κ, hence the distribution of its output is independent from the choice of the challenger. Therefore

$$\Pr[\textsc{predict}_{\mathcal{A},X}(\kappa) = \mathsf{true}] = \sum_{x'\in[X]} \Pr_{x\leftarrow X}[\mathcal{A}(\kappa) = x' \wedge x = x'] =$$

$$\sum_{x'\in[X]} \Pr[\mathcal{A}(\kappa) = x'] \Pr_{x\leftarrow X}[x = x'] \leq \mathsf{negl}(\kappa) \sum_{x'\in[X]} \Pr[\mathcal{A}(\kappa) = x'] \leq \mathsf{negl}(\kappa)\,.$$

□

Finally, we observe that public keys generated from secure signature schemes must be unpredictable.

Algorithm 13 The existential forgery $\mathcal{A}$ which tries to guess the secret key through sampling.

```
1: function A_S(1^κ, pk)
2:     (pk, sk) ← Gen(1^κ)
3:     return (ϵ, Sig(sk, ϵ))
4: end function
```

Lemma 2 (Public key unpredictability)**.** *Let $S = (\mathsf{Gen}, \mathsf{Sig}, \mathsf{Ver})$ be a secure signature scheme. Then the distribution ensemble $X_\kappa = \{(sk, pk) \leftarrow \mathsf{Gen}(1^\kappa); pk\}$ is unpredictable.*

Proof. Let $p = \max_{\widehat{pk} \in [X_\kappa]} \Pr_{pk \leftarrow X_\kappa}[pk = \widehat{pk}]$. Consider the existential forgery adversary $\mathcal{A}$ illustrated in Algorithm 13 which works as follows. It receives pk as its input from the challenger, but ignores it and generates a new key pair $(pk', sk') \leftarrow \mathsf{Gen}(1^\kappa)$. Since the two invocations of Gen are independent,

$$\begin{aligned}
\Pr[pk = pk'] &\geq \max_{\widehat{pk} \in [X_\kappa]} \Pr[pk = \widehat{pk} \wedge pk' = \widehat{pk}] \\
&= \max_{\widehat{pk} \in [X_\kappa]} \Pr[pk = \widehat{pk}] \Pr[pk' = \widehat{pk}] \\
&= \max_{\widehat{pk} \in [X_\kappa]} \left(\Pr[pk = \widehat{pk}]\right)^2 = p^2 .
\end{aligned}$$

The adversary checks whether $pk = pk'$. If not, it aborts. Otherwise, it uses sk' to sign the message $m = \epsilon$ and returns the forgery $\sigma = \mathsf{Sig}(sk, m)$. From the correctness of the signature scheme, if $pk = pk'$, then $\mathsf{Ver}(pk, \mathsf{Sig}(sk, m)) = \mathsf{true}$ and the adversary is successful. Since the signature scheme is secure, $\Pr[\mathsf{Sig\text{-}forge}^{cma}_{\mathcal{A},S}] = \mathsf{negl}(\kappa)$. But $\Pr[pk = pk'] \leq \Pr[\mathsf{Sig\text{-}forge}^{cma}_{\mathcal{A},S}]$ and therefore $p \leq \sqrt{\Pr[pk = pk']} \leq \mathsf{negl}(\kappa)$. Applying Lemma 5, we deduce that the distribution ensemble X_κ is unpredictable. □

B.3 Random Oracle properties

In this section, we state some statistical properties of the Random Oracle, which are useful for the proofs of our main results.

Lemma 1 (Perturbation)**.** *Let $p(\kappa)$ be a polynomial and $F : \{0,1\}^\kappa \longrightarrow \{0,1\}^\kappa$ be a permutation. Consider the process which samples $p(\kappa)$ strings $s_1, s_2, \ldots, s_{p(\kappa)}$ uniformly at random from the set $\{0,1\}^\kappa$. The probability that there exists $i \neq j$ such that $s_i = F(s_j)$ is negligible in κ.*

Proof. Let MATCH denote the event that there exist $1 \leq i \neq j \leq p(\kappa)$ such that $s_i = F(s_j)$. Let $\text{MATCH}_{i,j}$ denote the event that $s_i = F(s_j)$. Apply a union bound to obtain $\Pr[\bigcup_{i \neq j} \text{MATCH}_{i,j}] \leq \sum_{i \neq j} \Pr[\text{MATCH}_{i,j}]$. But $\Pr[\text{MATCH}_{i,j}] = 2^{-p(\kappa)}$ and therefore $\Pr[\text{MATCH}] \leq \sum_{i \neq j} 2^{-p(\kappa)} \leq p^2(\kappa) 2^{-p(\kappa)}$. □

Algorithm 14 The predictor $\mathcal{A}^*$ of the distribution X which makes use of a distinguisher $\mathcal{A}$ between X and $U(\{0,1\}^\kappa)$.

$i \leftarrow 0$
$Q \leftarrow \emptyset$ ▷ Record of all random oracle queries
function $H'_H(x)$
    $i \leftarrow i + 1$
    $Q[i] \leftarrow H(x)$
    return $Q[i]$
end function
function $\mathcal{A}^*_{X,\mathcal{A}}(1^\kappa)$
    $b \xleftarrow{\$} \{0,1\}$
    if $b = 0$ **then**
        $z \leftarrow X$
        $j \xleftarrow{\$} [r]$
    else
        $z \leftarrow U(\{0,1\}^\kappa)$
    end if
    $b^* \leftarrow \mathcal{A}^{H'}(z)$
    if $b = 1 \vee j > i$ **then**
        return FAILURE
    end if
    return Q[j]
end function

Lemma 3 (Random Oracle unpredictability)**.** *Let $\mathcal{T}$ be an unpredictable distribution ensemble and H be a Random Oracle. The distribution ensemble $X = \{t \leftarrow \mathcal{T}; H(t)\}$ is indistinguishable from the uniform distribution ensemble $\mathcal{U}(\{0,1\}^\kappa)$.*

Proof. Let $\mathcal{A}$ be an arbitrary polynomial distinguisher between X and $\mathcal{U}(\{0,1\}^\kappa)$. We construct an adversary $\mathcal{A}^*$ against $\text{PREDICT}_\mathcal{T}$. Let r denote the (polynomial) maximum number of random oracle queries of $\mathcal{A}$. The adversary $\mathcal{A}^*$ is illustrated in Algorithm 14 and works as follows. Initially, it chooses a random bit $b \xleftarrow{\$} \{0,1\}$ and sets $Z = X$ if $b = 0$, otherwise sets $Z = \mathcal{U}(\{0,1\}^\kappa)$. It samples $z \leftarrow Z$. If $b = 0$, then z is chosen by applying $\mathsf{GenAddr}$ which involves calling the random oracle H with some input pk. It then chooses one of $\mathcal{A}$'s queries $j \xleftarrow{\$} [r]$ uniformly at random. Finally, it outputs the input received by the random oracle during the j^{th} query of $\mathcal{A}$.

We will consider two cases. Either $\mathcal{A}$ makes a random oracle query containing pk, or it does not. We will argue that, if $\mathcal{A}$ makes a random oracle query containing pk with non-negligible probability, then $\mathcal{A}^*$ will

be successful with non-negligible probability. However, we will argue that, if $\mathcal{A}$ does not make the particular random oracle query, it will be unable to distinguish X from $\mathcal{U}(\{0,1\}^\kappa)$.

Let QRY denote the event that $b = 0$ and $\mathcal{A}$ asks a random oracle query with input pk. Let x denote the random variable sampled by the challenger in the predictability game of $\mathcal{A}^*$. Let EXQRY denote the event that $b = 0$ and $\mathcal{A}$ asks a random oracle query with input equal to x. Observe that, since the input to $\mathcal{A}$ does not depend on x, we have that $\Pr[\text{EXQRY}] = \Pr[\text{QRY}]$. As j is chosen independently of the execution of $\mathcal{A}$, conditioned on EXQRY the probability that $\mathcal{A}^*$ is able to correctly guess which query caused EXQRY will be $\frac{1}{r}$. Therefore we obtain that $\Pr[\text{PREDICT}_{\mathcal{A}^*,\mathcal{T}}(\kappa) = \mathsf{true}] = \frac{1}{r}\Pr[\text{EXQRY}] = \frac{1}{r}\Pr[\text{QRY}]$. As $\Pr[\text{PREDICT}_{\mathcal{A}^*,\mathcal{T}}(\kappa) = \mathsf{true}] \leq \mathsf{negl}(\kappa)$ and r is polynomial in κ, we deduce that $\Pr[\text{QRY}] \leq \mathsf{negl}(\kappa)$.

Consider the computational indistinguishability game depicted in Algorithm 10 in which the distinguisher gives a guess b^* attempting to identify the origin b of its input. If $b = 0$, then the distinguisher $\mathcal{A}$ receives a truly random input $pkh = H(pk)$. If the distinguisher does not query the random oracle with input pk, the input of the distinguisher is truly random and therefore $\Pr[b^* = 0|b = 0|\neg\text{QRY}] = \Pr[b^* = 0|b = 1]$.

Consider the case where $b = 0$ and apply total probability to obtain

$$\begin{aligned}
&\Pr[b^* = 0|b = 0] = \\
&\Pr[b^* = 0|\text{QRY}]\Pr[\text{QRY}] + \Pr[b^* = 0|b = 0|\neg\text{QRY}]\Pr[\neg\text{QRY}] \\
\leq &\Pr[b^* = 0|\text{QRY}]\Pr[\text{QRY}] + \Pr[b^* = 0|b = 0|\neg\text{QRY}] \\
\leq &\Pr[\text{QRY}] + \Pr[b^* = 0|b = 0|\neg\text{QRY}]
\end{aligned}$$

Then $\Pr[\text{DIST-GAME}_{\mathcal{A},X,\mathcal{U}(\{0,1\}^\kappa)} = \mathsf{true}] = \Pr[b = b^*]$ is the probability of success of the distinguisher. Applying total probability we obtain

$$\begin{aligned}
\Pr[b = b^*] &= \Pr[b = b^*|b = 0]\Pr[b = 0] + \Pr[b = b^*|b = 1]\Pr[b = 1] \\
&= \frac{1}{2}(\Pr[b^* = 0|b = 0] + \Pr[b^* = 1|b = 1]) \\
&\leq \frac{1}{2}(\Pr[\text{QRY}] + \Pr[b^* = 0|b = 0|\neg\text{QRY}] + \Pr[b^* = 1|b = 1]) \\
&= \frac{1}{2}(\Pr[\text{QRY}] + \Pr[b^* = 0|b = 1] + \Pr[b^* = 1|b = 1]) \\
&= \frac{1}{2}(\Pr[\text{QRY}] + \Pr[b^* = 0|b = 1] + (1 - \Pr[b^* = 0|b = 1])) \\
&= \frac{1}{2}(1 + \Pr[\text{QRY}]) \leq \frac{1}{2} + \mathsf{negl}(\kappa)
\end{aligned}$$

□

C Relaxing the Random Oracle assumption

The construction presented above works for P2PKH and achieves its unspendability and uncensorability in the Random Oracle model. In this section, we discuss alternative constructions which work without requiring the Random Oracle model.

The simplest blockchain address protocol is the Pay to Public Key (P2PK) protocol which, in contrast to P2PKH does not hash the public key to generate an address. Instead, the address is literally the public key and spending verification simply checks the validity of a signature. This protocol is illustrated in Algorithm 15.

Algorithm 15 The blockchain address P2PK algorithm, parameterized by a signature scheme $S = (\mathsf{Gen}, \mathsf{Sig}, \mathsf{Ver})$.

1: **function** $\mathsf{GenAddr}_S(1^\kappa)$
2: $\quad (pk, sk) \leftarrow \mathsf{Gen}(1^\kappa)$
3: $\quad$ **return** (pk, sk)
4: **end function**
5: **function** $\mathsf{SpendVerify}_S(m, \sigma, pk)$
6: $\quad$ **return** $\mathsf{Ver}(m, \sigma', pk)$
7: **end function**

Without the Random Oracle model, our construction must be tailored to the signature scheme in order to ensure uncensorability, as our addresses must look similar to public keys generated by the scheme. We describe a burn scheme which can work for (EC)DSA signatures, as used in most cryptocurrencies today. Our scheme is unconditionally correct and binding in the standard model. We provide evidence of uncensorability in the Common Random String model, assuming the DLOG problem is hard and a collision resistant hash function exists. Additionally, we provide evidence that our scheme is unspendable in the Common Random String model and that no generic unspendable construction is possible in the standard model.

Initially, a κ-order multiplicative group $\mathbb{G}$ of order q and a generator g are selected and let the Common Random String be a random group element $h = g^y$ for some $y \in [q]$. Due to the self-reducibility of the DLOG problem, if DLOG is difficult in the group, an adversary will not be able to

Algorithm 16 Our proof-of-burn protocol for P2PK using a Common Random String h representing a group element in which DLOG is difficult and parameterized by a collision resistant hash function H.

function $\mathsf{GenBurnAddr}_H(1^\kappa, t)$
    return $hg^{H(t)}$
end function
function $\mathsf{BurnVerify}_H(1^\kappa, t, th)$
    return $(\mathsf{GenBurnAddr}(1^\kappa, t) = th)$
end function

Algorithm 17 The random discrete log solver $\mathcal{A}^*$ which makes use of an adversary $\mathcal{A}$ which recovers the spending key corresponding to $hg^{H(t)}$.

function $\mathcal{A}^*_{\mathcal{A},H}(h)$
    $(t, z) \leftarrow \mathcal{A}(h)$
    return $z - H(t)$
end function

find the logarithm y of the random group element, except with negligible probability.

Our scheme is illustrated in Algorithm 16. GenBurnAddr hashes the tag t and treats $H(t)$ as the exponent, calculates the public key $g^{H(t)}$ and blinds it using the factor h. As before, BurnVerify regenerates the burn address from t and ensures it has been calculated correctly.

Correctness holds unconditionally.

Theorem 5 (Correctness). *The proof-of-burn protocol Π of Algorithm 16 is* correct*.*

Proof. Based on Algorithm 16, $\mathsf{BurnVerify}(1^\kappa, t, \mathsf{GenBurnAddr}(1^\kappa, t)) = \mathsf{true}$ if and only if $\mathsf{GenBurnAddr}(1^\kappa, t) = \mathsf{GenBurnAddr}(1^\kappa, t)$, which always holds as GenBurnAddr is deterministic. □

As evidence towards unspendability, we now remark that it is difficult for an adversary to obtain the secret key corresponding to the public key $hg^{H(t)}$ needed to produce signatures. We therefore conjecture that our scheme is unspendable.

Lemma 6 (Logarithm ignorance)**.** *If h is a* Common Random String *and assuming the DLOG problem is hard, no probabilistic polynomial-time*

Algorithm 18 The collision adversary $\mathcal{A}^*$ against H using a proof-of-burn BIND-ATTACK adversary $\mathcal{A}$.

```
1: function A*_A(1^κ)
2:     (t, t', _) ← A(1^κ)
3:     return (t, t')
4: end function
```

adversary can produce (t, z) *such that* $g^z = hg^{H(t)}$*, except with negligible probability in* κ*.*

Proof. Suppose $\mathcal{A}$ is a probabilistic polynomial-time adversary which produces (t, z) with probability of success $p = \Pr[g^z = hg^{H(t)}]$. We construct the adversary $\mathcal{A}^*$ which invokes $\mathcal{A}$ illustrated in Algorithm 17 and finds the logarithm of h. Conditioned on the event that $\mathcal{A}$ is successful, we have that $g^z = hg^{H(t)} \Rightarrow g^z = g^{y+H(t)} \Rightarrow y \equiv z - H(t) \pmod{q}$, so $\mathcal{A}^*$ is successful. Therefore $\Pr[\mathcal{A}^*(h) = y] = p$. But $\Pr[\mathcal{A}^*(h) = y]$ is negligible. □

This observation illustrates the useful fact that, if a single group element with unknown logarithm is provided, an arbitrary number of such group elements can be found and logarithm ignorance can be proven.

Proofs-of-ignorance. There are other constructions which can give similar results. In fact, recent work on *proofs-of-ignorance* [14] has shown that any NP language can support proofs-of-ignorance, which are a prerequisite for our need of unspendability (as inability to produce signatures mandates ignorance of the private key). Therefore, we conjecture that such constructions are possible using any secure signature scheme in which the secret key constitutes a witness for the fact that the public key is an element of an NP language. Additionally, they argue that such constructions are not possible in the standard model given non-uniform probabilistic polynomial-time adversaries, supporting our construction in the Common Random String model. Whether burn constructions in the Standard Model exist against uniform probabilistic polynomial-time adversaries remains a question for future work.

Theorem 6 (Binding). *If the hash function* H *is collision resistant and its range lies in* $[q]$ *where* q *denotes the group order of* $\mathbb{G}$*, then the proof-of-burn protocol* Π *of Algorithm 16 is* binding.

Proof. Let $\mathcal{A}$ be a probabilistic polynomial-time binding adversary against the protocol Π. We construct the probabilistic polynomial-time collision

Algorithm 19 The collision adversary $\mathcal{A}$ against H which samples from an unpredictable distribution $\mathcal{T}$.

```
1: function A_{H,T}(1^κ)
2:     t_1 ← T
3:     t_2 ← T
4:     return (t_1, t_2)
5: end function
```

adversary $\mathcal{A}^*$ against the hash function H. The adversary $\mathcal{A}^*$ is illustrated in Algorithm 18 and works as follows. It invokes $\mathcal{A}$ which returns a triplet $(t, t', \mathsf{burnAddr})$, then returns the collision (t, t'). Let p denote the probability that $\mathcal{A}$ is successful.

Conditioned on the event that $\mathcal{A}$ is successful, it holds that $hg^{H(t)} = hg^{H(t')}$ and $t \neq t'$. This implies that $g^{H(t)} = g^{H(t')}$, which in turns yields $H(t) \equiv H(t') \pmod{q}$. Since the range of H lies in $[q]$, this constitutes a collision and $\mathcal{A}^*$ is successful.

We thus conclude that $\mathcal{A}^*$ is successful in the COLLISION game if and only if $\mathcal{A}$ is successful in the BIND-ATTACK game.

$$\Pr[\text{BIND-ATTACK}_{\mathcal{A},\Pi} = \mathsf{true}] = \Pr[\text{COLLISION}_{\mathcal{A}^*,H} = \mathsf{true}]$$

From the collision resistance of H it follows that $\Pr[\text{COLLISION}_{\mathcal{A}^*,H} = 1] < \mathsf{negl}(\kappa)$. Therefore, $\Pr[\text{BIND-ATTACK}_{\mathcal{A},\Pi} = \mathsf{true}] < \mathsf{negl}(\kappa)$, so the protocol Π is binding. □

We now give some evidence towards the uncensorability of our scheme. The following lemma expands on the results of Lemma 3 without making use of the Random Oracle model.

Lemma 7 (Collision resistant unpredictability)**.** *Let H be a collision resistant hash function and $\{\mathcal{T}\}_{\kappa\in\mathbb{N}}$ be an efficiently samplable unpredictable distribution ensemble. Then the distribution ensemble $X_\kappa = \{t \leftarrow \mathcal{T}; H(t)\}$ is unpredictable.*

Proof. Consider the collision adversary $\mathcal{A}$ against the hash function H illustrated in Algorithm 19 which samples t_1 and t_2 independently from $\mathcal{T}_\kappa$ and hopes for a collision. Let COLL_{h^*} denote the event that $H(t_1) =$

$H(t_2) = h^*$. Applying total probability

$$\begin{aligned}
&\max_{h^* \in [X_\kappa]} \Pr[\textsc{Coll}_{h^*}] \\
= &\max_{h^* \in [X_\kappa]} (\Pr[\textsc{Coll}_{h^*} | t_1 = t_2] \Pr[t_1 = t_2] + \Pr[\textsc{Coll}_{h^*} | t_1 \neq t_2] \Pr[t_1 \neq t_2]) \\
&\leq \max_{h^* \in [X_\kappa]} \Pr[\textsc{Coll}_{h^*} | t_1 = t_2] \Pr[t_1 = t_2] \\
&+ \max_{h^* \in [X_\kappa]} \Pr[\textsc{Coll}_{h^*} | t_1 \neq t_2] \Pr[t_1 \neq t_2] \\
&\leq \max_{h^* \in [X_\kappa]} \Pr[t_1 = t_2] + \max_{h^* \in [X_\kappa]} \Pr[\textsc{Coll}_{h^*} | t_1 \neq t_2] \Pr[t_1 \neq t_2] \\
&= \Pr[t_1 = t_2] + \max_{h^* \in [X_\kappa]} \Pr[\textsc{Coll}_{h^*} \wedge t_1 \neq t_2] \,.
\end{aligned}$$

Therefore $\max_{h^* \in [X_\kappa]} \Pr[\textsc{Coll}_{h^*} \wedge t_1 \neq t_2] \geq \max_{h^* \in [X_\kappa]} \Pr[\textsc{Coll}_{h^*}] - \Pr[t_1 = t_2]$. We have that

$$\begin{aligned}
\Pr[\textsc{collision}_{\mathcal{A}}(\kappa) = \mathsf{true}] &= \sum_{h^* \in [X_\kappa]} \Pr[\textsc{Coll}_{h^*} \wedge t_1 \neq t_2] \\
\geq \max_{h^* \in [X_\kappa]} \Pr[\textsc{Coll}_{h^*} \wedge t_1 \neq t_2] &\geq \max_{h^* \in [X_\kappa]} \Pr[\textsc{Coll}_{h^*}] - \Pr[t_1 = t_2] \,.
\end{aligned}$$

Since $\Pr[\textsc{collision}_{\mathcal{A}}(\kappa) = \mathsf{true}] \leq \mathsf{negl}(\kappa)$ and $\Pr[t_1 = t_2] \leq \mathsf{negl}(\kappa)$, therefore $\max_{h^* \in [X_\kappa]} \Pr[\textsc{Coll}_{h^*}] \leq \mathsf{negl}(\kappa)$. Because $H(t_1)$ and $H(t_2)$ are chosen independently,

$$\max_{h^* \in [X_\kappa]} \Pr_{x \leftarrow X_\kappa}[x = h^*] = \sqrt{\max_{h^* \in [X_\kappa]} \Pr[\textsc{Coll}_{h^*}]} \leq \mathsf{negl}(\kappa) \,.$$

Applying Lemma 5, we deduce that the distribution ensemble X_κ is unpredictable. □

Unfortunately, a merely unpredictable distribution on the exponent does not allow us to prove uncensorability. However, we can prove uncensorability if we assume the hash function maps the tag distribution to the uniform distribution $\mathcal{U}([q])$ of the exponents of $\mathbb{G}$, which is an assumption closely related to the Random Oracle. We leave the relaxation of this additional assumption for future work.

Theorem 7 (Uncensorability). *Let $\mathcal{T}$ be an efficiently samplable unpredictable tag distribution and H be a hash function such that $\{t \leftarrow \mathcal{T}_\kappa; H(t)\} \approx_c \mathcal{U}([q])$ where q denotes the order of the group $\mathbb{G}$. Then the proof-of-burn protocol Π of Algorithm 16 is* uncensorable *with respect to blockchain address protocol Π_α of Algorithm 15.*

Proof. Apply Lemma 4 to the computationally indistinguishable distribution ensembles $X = \{t \leftarrow \mathcal{T}_\kappa; H(t)\}$ and $Y = \mathcal{U}([q])$ mapped through the function $f(x) = g^x$. The resulting distributions, g^X and g^Y are indistinguishable. The distribution g^Y is the distribution of public keys generated by GenAddr. As multiplication by h constitutes a permutation of the group, the distribution g^Y is identical to the distribution hg^Y. Hence g^X and hg^Y are indistinguishable. □

Trusted setup. We remark here that we *do not* require a trusted setup. In particular, for the selection of the protocol parameters, we do not generate a Common Reference String g^y by selecting a random y and computing g^y, as this would require ensuring y is destroyed. Instead, we select a random group element h directly, which is possible in many finite groups. As an example of such a construction in practice, a point can be selected on the secp256k1 elliptic curve by starting with an X coordinate corresponding to a well-known number such as $X = \texttt{SHA256}$("Whereof one cannot speak, thereof one must be silent") and incremented until a solution of the elliptic curve equation exists for Y, then taking the positive such Y and using the point $h = (X, Y)$.

Perturbation of group element labels. Yet another scheme that can potentially realize the above properties is the burn address generation by evaluating $(g^{H(t)}) + 1$, where the $+1$ does not pertain to the group operation, but operates on the label of the group element. For example, in the primed order group $\mathbb{Z}_p^*$, the $+1$ operation can be taken to be literally the next integer. Such a scheme is clearly correct and binding. Its uncensorability is comparable to our above scheme. Lastly, its unspendability, given appropriate restrictions ($t \neq 0$) seems to intuitively hold: It is hard to know the logarithm of both a group element and its next. Whether this is provable in the Generic Group Model or using appropriate hardness assumptions is left for future work.

D Acknowledgements

Research partly supported by H2020 Project Priviledge, # 780477. The authors wish to thank Dimitris Karakostas, Sandro Coretti-Drayton and Andrianna Polydouri who read early versions of this paper and provided helpful suggestions.