# Hours of Horus: Keyless Cryptocurrency Wallets

Dionysis Zindros

Stanford University and Harmony

**Abstract.** We put forth a *keyless wallet*, a cryptocurrency wallet where money can be spent using a password alone, and no private keys are required. It requires a smart contract blockchain. We propose a scheme in which the user uses an OTP authenticator seed to generate a long series of time-based OTP passwords for the foreseeable future. These are encrypted and organized in a Merkle tree whose root is stored in a smart contract. The user can spend funds at any time by simply visually providing the current OTP password from an air gapped device. These OTPs can be relatively short: Just 6 alphanumeric characters suffice. Our OTP scheme can work in proof-of-stake as well as static and variable difficulty proof-of-work blockchains. The low-entropy passwords and OTPs in our scheme are protected from brute force attacks by requiring that an adversary accompany any attempt by a transaction on the chain. This quickly incurs enormous economic costs for the adversary. Thus, we develop the first decentralized *rate limiting* scheme. We use *Witness Encryption* (WE) to construct a timelock encryption scheme in which passwords are encrypted from past into future blocks by leveraging the NP-language having proof-of-work or proof-of-stake performed as the witness. Witness Encryption is a currently impractical cryptographic primitive, but our scheme may become practical as these primitives are further developed.

## 1 Introduction

The management of cryptocurrency [50] wallet private keys is a hassle. Can we get rid of them and replace them with a simple short password or a rotating 6-digit one-time password (OTP) [48,49]? Users are more familiar with this model, but this seems, at first sight, impossible to achieve: The blockchain is public infrastructure, and anyone has access to the public keys and smart contracts [14, 53] governing the conditions under which one can spend. Any short password or OTP will be easily broken by an offline brute force attack [52].

Perhaps unexpectedly, it *is* possible to build brute force resilient wallets by leveraging the blockchain infrastructure itself. We build the first *keyless cryptocurrency wallet*. It operates as follows. Alice initially uses her mobile wallet to generate a high entropy OTP seed. This seed is used to generate a large amount of time-based OTPs (with, say, hourly resolution), which are encrypted and collected into a Merkle tree. The wallet creates a smart contract containing the Merkle tree root on the blockchain and outputs a wallet address to which money can be deposited at any time. The internal nodes of the Merkle tree are posted on a public, high availability location such as IPFS [10] and can also be kept by Alice in any untrusted device, if desired for availability. Alice then disconnects the mobile wallet and keeps it completely air gapped and offline. At any time, Alice can use the offline device to generate a time-based OTP. Without plugging in the offline device via USB or connecting it to the Internet, Alice visually copies the short (perhaps 6 alphanumeric characters long) OTP that appears on the device's screen into her online computer. The wallet on her online computer can then be used to input a target address and amount to be transferred. This second construction allows the user to spend money *at any time*. As the OTPs are very short, this wallet is highly usable. After the initial OTP seed generation, the seed is kept in an air gapped device, ensuring any bugs in the hardware or software cannot be abused to steal it.

Critical to the security of the construction is ensuring that no adversary can brute force the short password or OTPs. Towards that goal, we devise a new cryptographic mechanism to secure cryptocurrency wallet passwords from offline brute forcing attempts. Any adversary who wishes to brute force these passwords *must* do so through the chain itself and record the attempt in a transaction. As such, these attempts are governed by the limitations of the chain: Each transaction costs gas to perform. This gives rise to the first *decentralized rate limiting* mechanism. Through appropriate cryptoeconomic parametrization, we ensure that the adversary will, in expectation, and with any desired probability, lose much more money than they will win out of brute forcing attempts. The parametrization dictates the length of the password based on current transaction gas costs and the capital to be protected.

To achieve this property, we leverage the fact that the network is performing proof-of-work [25] (or proof-of-stake [39]) in a predictable rate in expectation [13]. We use *Witness Encryption* (WE) [31] to encrypt the password in such a way that it can only be decrypted using the *future* proof-of-work/stake that will be performed by the network. As such, the encryption is a *Timelock Encryption* [51] in which the miners function in tandem [41] to decrypt the submitted password. This decryption is a by-product of the proof-of-work/stake they are performing anyway. The miners do not need to know that the passwords have been timelock encrypted. The security of timelock encryption ensures that the passwords will not be decryptable prior to the chain progressing a certain number of blocks. Our security argument stands upon five pillars:

1. a secure *extractable* Witness Encryption scheme,
2. a secure underlying blockchain (with *Common Prefix*),
3. a *preimage/collision* resistant hash function;
4. a secure *pseudorandom* OTP scheme, and
5. a rational adversary.

Our constructions could, in principle, be deployed to any smart-contract–enabled proof-of-work/stake chain such as Ethereum. In particular, we do not require any modifications to the Ethereum consensus mechanism or smart contract virtual machine (EVM). The best known instantiation of the Witness Encryption primitive, which the Timelock Encryption instance makes use of, requires the use of *multilinear maps*. Multilinear maps are (approximately) constructible using ideal lattices. Unfortunately, this construction currently remains impractical. Until such constructions are built, our scheme is of theoretical interest.

**Our contributions.** The contributions of this paper are summarized as follows:

- We introduce the first timelock-based *OTP* wallet, with OTP length of just 6 alphanumeric characters. The funds can be spent any time just by providing the OTP password from an air gapped device.
- We put forth the first *decentralized rate limiting* scheme. The scheme protects the user from brute force attacks by an adversary by requiring all attempts to be recorded on the chain.

Secondary contributions include the first instantiation of timelock encryption applied to proof-of-stake blockchains and variable difficulty proof-of-work blockchains, explored in the appendix. Lastly, our security argument, in the appendix, uses a hybrid approach which combines a high-entropy cryptographic parameter — in which classical cryptographic security is ensured with overwhelming probability — with a low-entropy cryptoeconomic parameter whose role is to ensure the attack is uneconomical for a rational adversary. This novel proof methodology may be of independent interest in analyzing blockchain protocols, which often compose cryptography and economics.

Our security assumptions pertain to pseudorandomness, hash security, and lattice-based cryptography. Therefore our wallets are also quantum-resistant.

**Related work.** Witness Encryption was introduced by Garg et al. [31] using lattice-based approximate multilinear maps [30]. It was lated improved theoretically [40] and implementation-wise [2], and attacked [1, 15, 16, 18, 20, 34]. A follow-up lattice-based approach [32] was also attacked [19]. Integer-based multilinear maps were proposed [21, 22] and attacked [15,17,47]. Current advancements [42] seem immune to such attacks. Timelock Encryption was introduced by Rivest et al. [51]. Using Witness Encryption and proof-of-work for timelocking was proposed by Liu et al. [41]. Cryptocurrency applications were discussed by Miller [46]. An overview of wallets is given by Karantias [35], and of hardware wallets by Karakostas et al. [3]. The use of OTP [48,49] for wallets was explored in SmartOTP [33]. Password-based wallets appeared as *brain wallets* [52].

## 2 Preliminaries

**Blocks and chains.** The proof-of-work (PoW) [25] blockchain consists of block headers $B = \langle \mathsf{ctr}, \mathsf{tx}, s \rangle$ each of which contains a *nonce* $\mathsf{ctr}$, a short Merkle Tree [43] root of transaction data $\mathsf{tx}$, and a pointer $s$ to the previous block in the chain [28]. The value $H(B)$, where $H$ is a hash function modelled as a random oracle [8], is used as the $s'$ to include in the next block. Each block satisfies the PoW equation $H(B) \leq T$, where $T$ is the *mining target*. In the *static difficulty model* [27, 28], $T$ is assumed to be a constant (we discuss the variable difficulty model in the appendix).

To address blocks within chain $\mathcal{C}$, we use $\mathcal{C}[i]$ to mean the $i^{\text{th}}$ (zero-based) block from the beginning and $\mathcal{C}[-i]$ to mean the $i^{\text{th}}$ (one-based) block from the end. $\mathcal{C}[0]$ indicates *genesis* and $\mathcal{C}[-1]$ the current tip. $|\mathcal{C}|$ denotes the chain length. We use $\mathcal{C}[i{:}j]$ to denote the subchain starting at the $i^{\text{th}}$ block (inclusive) and ending at the $j^{\text{th}}$ block (exclusive). Omitting $i$ takes the range to the beginning, while omitting $j$ takes the range to the end. Similarly, we use $\mathcal{C}\{A{:}Z\}$ to denote the subchain starting at block $A$ (inclusive) and ending at block $Z$ (exclusive).

Each honest party keeps a local chain $\mathcal{C}$, which may be different from the others. It is known [28] these chains cannot deviate much: The *Common Prefix* property establishes that they share a long common prefix and only deviate with forks of length up to a constant $k \in \mathbb{N}$. Formally, if at any round $r_0$ an honest party $P_0$ has adopted chain $\mathcal{C}_0$, then at any round $r_1 > r_0$, any honest party $P_1$ will have adopted a chain $\mathcal{C}_1$ with

$\mathcal{C}_0[: -k]$ a prefix of $\mathcal{C}_1$. This gives rise to ledger *safety*: Any transaction that appears prior to $C[-k]$ is *confirmed*, and will eventually appear at the same position in the chains of all honest parties.

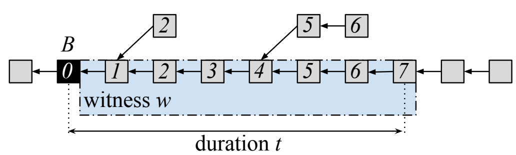
The chain of an honest party grows with a certain rate, which is bounded from below and above with overwhelming probability. This is known as the *Chain Growth* property (see [28] for proof for the lower bound; a proof of the upper bound is found in the Appendix). This gives rise to *liveness*: A transaction submitted to the network will eventually appear confirmed to all honest parties after at most $\ell \in \mathbb{N}$ blocks. The two security parameters $k$ and $\ell$ that govern the evolution of the chain are polynomial in the underlying cryptographic security parameter $\kappa$, but constant in the execution time.

**Timelock encryption.** Timelock encryption allows us to *timelock* a secret so that it can be unlocked at a prespecified date and time $t$ in the future, but not prior. It consists of a *timelock* algorithm $\mathsf{timelock}(m, t)$ that takes message $m$ and timestamp $t$ after which decryption should be possible, and returns ciphertext $c$ encrypted for time $t$, and a *timeunlock* algorithm $\mathsf{timeunlock}(c, w)$ that takes a ciphertext $c$ encrypted using $\mathsf{timelock}$ and a witness $w$ illustrating that indeed time $t$ has passed, and returns message $m$. When the time $t$ has elapsed, it becomes easy to compute a *witness* which is not possible to compute earlier. The $\mathsf{timeunlock}$ function can be called with this witness $w$ for time $t$, and it returns the original message $m$. Prior to time $t$, timelock encryption security ensures that ciphertexts corresponding to the encryptions of two plaintexts $m_1$ and $m_2$ are indistinguishable.

**Witness encryption.** To construct timelock encryption, we use *Witness Encryption* (WE) [41]. In a WE scheme, a plaintext is encrypted into a ciphertext that can be decrypted *only* if a solution to a computational problem is given. More concretely, the Witness Encryption scheme is parametrized by an NP language $\mathcal{L}$ (a decision problem) and an associated relation $\mathcal{R}$ (which verifies a solution). For each instance $x \in \mathcal{L}$, there exists a witness $w$ such that $x\mathcal{R}w$. For non-instances $x \notin \mathcal{L}$, no such witness exists. The relation $\mathcal{R}$ is polynomially computable. A witness encryption scheme consists of an *encryption* algorithm $\text{WE.}\textsc{Enc}_{\mathcal{R}}(m, x)$ that takes plaintext $m$ and instance $x$ and returns ciphertext $c$ encrypted for this problem instance, and a *decryption* algorithm $\text{WE.}\textsc{Dec}_{\mathcal{R}}(c, w)$ that takes ciphertext $c$ and witness $w$ and returns the decrypted plaintext $m$ as long as $x\mathcal{R}w$. *Correctness* mandates that, whenever $x\mathcal{R}w$, it holds that $\text{WE.}\textsc{Dec}_{\mathcal{R}}(\text{WE.}\textsc{Enc}_{\mathcal{R}}(m, x), w) = m$. On the other hand, *security* mandates that an adversary given $c = \text{WE.}\textsc{Enc}_{\mathcal{R}}(m, x)$ can extract information about $m$ only if she can also produce (through a helper *extractor* machine) a $w$ such that $x\mathcal{R}w$, except with negligible probability. A correct and secure scheme allows a party to decrypt *if and only if* she can solve the problem instance by providing a witness.

To construct timelock encryption using witness encryption, the problem statement asks for the existence of a series of blockchain work nonces that solve the proof-of-work equation, as illustrated in Figure 1. The instantiation of timelock encryption using witness encryption begins by identifying the chain tip $B$. The timelock time $t$ is expressed in chain time: We ask that a certain number of blocks must have been mined on top of $B$ in order for the secret to become decryptable. The witness encryption NP language contains the integers $t \in \mathbb{N}$ indicating that there exists a block with additional block height $t$ descending from the known block $B$. The witness consists of a series of nonces $\mathsf{ctr}_i$ and transaction root hashes $\mathsf{tx}_i$ such that $B_0 = B$ and $B_i = \langle \mathsf{ctr}_i, \mathsf{tx}_i, H(B_{i-1}) \rangle$ and $H(B_i) \leq T$, where $i$ ranges from 1 to $t$. Therefore, the timelock function $\mathsf{timelock}(m, t)$ is defined as $\mathsf{WE.Enc}_{\mathcal{R}}(m, x)$ where $\mathcal{R}$ corresponds to the relation checking the validity of the blockchain and $x$ corresponds to the number of blocks $t$ as well as the current chain tip $B$. The unlock function $\mathsf{timeunlock}(c, w)$ is defined as $\mathsf{WE.Dec}_{\mathcal{R}}(c, w)$ under the same relation $\mathcal{R}$ where $w$ consists of the sequence of $\mathsf{ctr}_i$ and $\mathsf{tx}_i$ (c.f., [41]).

**Fig. 1.** Timelock implemented using the moderately hard NP language of chain discovery. The problem instance $x = (B, t)$ requests $t = 7$ blocks on top of $B$. The witness consists of block headers produced sequentially on top of $B$, irrespective of any temporary forks.




What is the outcome of encrypting secrets in this manner? Whenever a secret is encrypted for block time $t$ following block $B$, the secret remains hidden until time $t$ has arrived. The secret cannot be decrypted prior to that time, because decrypting it would require the decrypting party to produce (through an extractor) a witness $w$ which is a blockchain of height $t$ extending $B$. However, due to the *Common Prefix* property of the blockchain, no adversary can do that much sooner than the honest parties converge to that height (even offline!). Furthermore, the *chain growth* rate is bounded both *above* and *below* by a certain *velocity*, and so, while we do not know its exact growth rate, we can give an estimate of how quickly it will grow. With sufficient time elapsed, the miners will produce a witness anyone can use to decrypt the secret. The result is that no one knows the secret prior to the desired block height, but everyone knows it afterwards. Because the adversary can have a chain that is leading by up to $k$ blocks, she has an advantage in decrypting the secret slightly ahead of the honest parties: The secret begins *leaking* to the adversary at block time $t - k$. This will require us to establish certain time bounds in our construction. Our security proof hinges on the *common prefix* property: If an adversary can decrypt the witness encrypted ciphertext *much* earlier than the honest parties, she will need to have produced a chain which significantly deviates from the honest parties' chain, but this is improbable.

Contrary to timelock schemes that require the interested party to devote compute power to decrypt the secret over time, the scheme using blockchain witnesses allows any party (who can remain offline and have limited compute power) to take advantage of the scheme.

## 3 A Password Wallet

We start by building a password-based wallet without private keys. This construction will be a stepping stone for the next. In this construction, we have a severe limitation: The wallet can only be used to spend *once*, and at a *predetermined time*. Once the wallet has been used, it cannot be reused with the same password. Furthermore, the wallet becomes unusuable if the funds have not been spent prior to the maturity date.

From a user point of view, the wallet works as follows. Initially, Alice chooses a secret password with $\lambda$ bits of entropy. We will determine this $\lambda$ later, but let us say, with foresight, that it will be enough to have a password just 6 alphanumeric characters long. Alice also chooses a *maturity date*, a timestamp in the future (expressed as chain height), and uses her wallet software to generate a smart contract which she then posts on the chain. This generates a public wallet address for Alice that she can use to receive multiple payments prior to the maturiy date. The wallet software can then be discarded and no secret information needs to be kept by Alice, beyond the secret password that she remembers, and the public contract that remains on the chain. No private keys are stored anywhere. A short period before the maturity date arrives, Alice uses the wallet software to connect to the chain network, and enters her password and desired destination. The software issues two transactions to the chain:

First, a $\mathsf{tx}_{\mathsf{commit}}$ transaction, which lets Alice illustrate prior knowledge of her secret password; and, second, a $\mathsf{tx}_{\mathsf{reveal}}$ transaction, in which Alice proves that her previous commitment indeed corresponds to the secret password committed on the chain. The first transaction is posted strictly prior to the maturity date, while the second transaction is posted on or after the maturity date.

**Algorithm 1** A password-only wallet with a maturity date.

```
contract PasswordWallet
    BLOCK_DELAY ← 2k
    c ← ⊥; t_1 ← ⊥
    commitments ← ∅
    function construct(c̄, t̄_1)
        c ← c̄
        t_1 ← t̄_1
    end function
    function commit(z)
        require(block.number < t_1 − BLOCK_DELAY)
        commitments[z] ← true
    end function
    function reveal(sk, salt, α_to, w)
        z ← H(⟨sk, salt, α_to⟩)
        require(commitments[z])
        require(WE.Dec_R(c, w) = sk)
        α_to.transfer(address(this).balance)
    end function
end contract
```

The smart contract is illustrated in Algorithm 1. It consists of three methods: a construct method, called when the wallet is initialized; a commit method, called shortly prior to the maturity date; and a reveal method, called after the maturity date. These two last methods are used for spending.

The interaction with the wallet is illustrated in Algorithm 2. When Alice wishes to deploy her wallet, she begins by generating a password $sk \xleftarrow{\$} \{0,1\}^{\lambda}$. She also chooses a future timestamp at which she will be able to spend her money. She submits this information to her software wallet. The software wallet connects to the blockchain and observes the current stable tip $B = \mathcal{C}[-k]$ and its height $t_0$. Alice's timestamp choice is translated to a future block height $\mathit{\Delta} \in \mathbb{N}$ which denotes how far in the future, in block height after $t_0$, she wants to spend her money: If $\mathit{\Delta} = 100$, the money will be spendable when the chain reaches height $t_0 + \mathit{\Delta}$. We set $t_1 = t_0 + \Delta$ to be the height at which spending becomes possible. The software wallet constructs the contract of Algorithm 1 by broadcasting its construction transaction $\mathsf{tx}_{\mathsf{construct}}$ to the network. The constructor accepts two parameters, $t_1$ and $c$. The $c$ parameter is a timelock-encrypted ciphertext of her password. Concretely, Alice's software wallet sets $c = \mathsf{timelock}(sk, t)$ by invoking $\mathsf{WE.Enc}_{\mathcal{R}}(sk, x)$. Here, $\mathcal{R}$ denotes the block-validated relation described in the preliminaries. The problem instance $x = (B, t)$ is the tuple consisting of the latest known stable block and the maturity height. Observe now that the ciphertext $c$ which is published on the smart contract and known to the adversary is a ciphertext which can only be decrypted after $t$ blocks have been mined on top of block $B$. The transaction returns a wallet address $pk$ at which she can receive money prior to the maturity height.

**Algorithm 2** Interacting with the password wallet.

$\mathsf{BLOCK\_DELAY} \leftarrow 2k$
$pk \leftarrow \bot$ ▷ Published so that money can be received
$B \leftarrow \bot$ ▷ Published on insecure public storage
$t_1 \leftarrow \bot$ ▷ The maturity date
**upon** initialize(t) **do**
    $sk \xleftarrow{\$} \{0,1\}^{\lambda}$ ▷ Password is generated with low entropy $\lambda$
    $B \leftarrow \mathcal{C}[-k]$ ▷ Latest stable block
    $x \leftarrow (B, t)$ ▷ NP language problem instance
    $c \leftarrow \mathsf{WE.Enc}_{\mathcal{R}}(sk, x)$
    $pk \leftarrow \mathsf{PasswordWallet.construct}(c, t)$
    $t_1 \leftarrow t$
    **return** $sk$ ▷ The user must remember this
**end upon**
**upon** spend($sk, \alpha_{\mathsf{to}}$) **do**
    ▷ *At any time prior to* $|\mathcal{C}| < t_1 - \ell - 2k$
    $\mathsf{salt} \xleftarrow{\$} \{0,1\}^{\kappa}$ ▷ Generate short-lived high-entropy salt
    $z \leftarrow H(\langle sk, \mathsf{salt}, \alpha_{\mathsf{to}} \rangle)$
    $\mathsf{PasswordWallet.commit}(z)$
    **wait until** $|\mathcal{C}| = t_1$
    $w \leftarrow \mathcal{C}\{B{:}\}$
    $\mathsf{PasswordWallet.reveal}(sk, \mathsf{salt}, \alpha_{\mathsf{to}}, w)$
**end upon**

To spend her money, Alice runs the wallet software anew and inputs her public wallet address $pk$, her password $sk$, and destination address $\alpha_{\mathsf{to}}$. The wallet software does not have any information beyond this. The software runs an SPV (or full) node which observes a chain $\mathcal{C}$. At any time before its local chain reaches height $t_1 - \ell - 2k$, the wallet generates a new high-entropy salt $\mathsf{salt} \xleftarrow{\$} \{0,1\}^\kappa$ (where $\kappa$ is a security parameter in the order of 128). This salt is short-lived ($\ell - 2k$ blocks) and must survive until the chain reaches height $t_1$. It then creates transaction $\mathsf{tx}_{\mathsf{commit}}$ which contains a commitment $z$ evaluated as $z = H(\langle sk, \mathsf{salt}, \alpha_{\mathsf{to}}\rangle)$. This transaction is submitted to the smart contract by invoking *commit*. Due to liveness, the transaction is confirmed in a block with height at most $t_1 - 2k$. The contract records the commitment, as the requirement in Line 10 is satisfied, and stores it in the *commitments* set.

After the local chain of the wallet reaches height $t_1$, the software gathers the block headers $\mathcal{C}[t_0{:}t_1]$ to construct a timelock witness $w$. It then creates a transaction $\mathsf{tx}_{\mathsf{reveal}}$ which invokes the reveal method of the smart contract and includes the plaintext password $sk$, which now becomes public, the plaintext salt, which also becomes public, the target address $\alpha_{\mathsf{to}}$, and the witness $w$. The reveal method checks that the submitted data corresponds to the previous commitment, and that the stored encrypted password $c$ timelock decrypts to the provided password $sk$. If so, it sends the money to $\alpha_{\mathsf{to}}$.

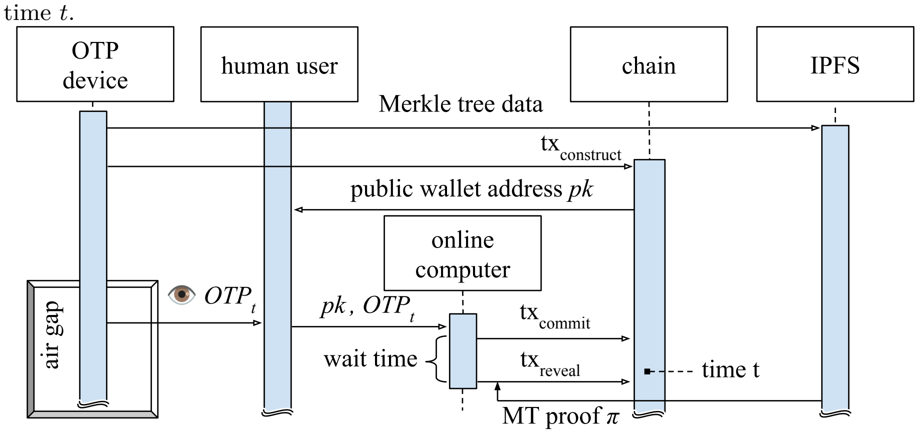
The commitments variable is a *set* to avoid denial of service attacks. If the honest party submits a *commit* transaction, the adversary should not be able to overwrite this. Therefore, the wallet stores *all* commit transactions in the set and checks that the correct one is revealed afterwards. The reason why the target address $\alpha_{\mathsf{to}}$ is placed in the hash commitment is to avoid front running attacks [23]. The target address is *tied* to the knowledge of the password. If the adversary replays the commitment, she cannot change the target address.

We now give a high-level overview of the correctness and security of this scheme. The *correctness* property of the wallet mandates that the honest wallet user can create a valid spending transaction, i.e., a transaction which executes *reveal* to completion. The *security* property mandates that the adversary cannot create a valid transaction. These properties, together, ensure that the honest user can spend her money, while the adversary cannot.

On the one hand, the scheme is correct, because the honest user can always create the commit and reveal transactions in order, and, due to liveness, these cannot be censored. When time $t_1$ arrives, the smart contract can verify the veracity of the claims and issue the final transfer. On the other hand, the scheme is secure, because, prior to time $t_1$ the adversary does not hold a chain of length $t_1$. Without such a chain, the adversary cannot distinguish a correct from an incorrect guess, due to the security of witness encryption. Any guess the adversary makes is a good as any. However, all of these guesses must be committed to the smart contract sufficiently before time $t_1$ arrives. And, so, the adversary must choose to blindly submit some guesses and hope that some of them are correct. This very soon becomes uneconomical, even if the password length is quite short. We give more details on the security of the scheme in the Analysis section of the appendix.

One detail to note here is that the time $t_1$ is given in block time. Because the rate of chain growth can vary, the honest party must monitor the chain and ensure that block height $t_1$ has not passed. This is one additional limitation of this scheme that makes it unusable. Additionally, the password can only be used once. In the next section, we lift these limitations.

**Fig. 2.** The sequence diagram of the OTP wallet. After initialization, the OTP device becomes air gapped and the user submits the OTP visually to an online computer at time $t$.



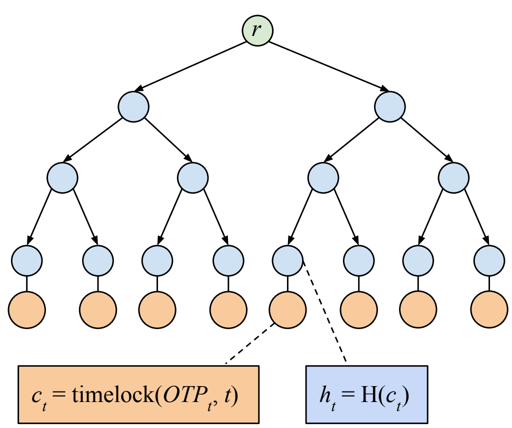

## 4 An OTP Wallet

The password-based construction has an important limitaton: The money can only be spent once at a prespecified date. This makes the wallet unusable. Using the previous construction as a stepping stone, we now move on to describe our *Hours of Horus* scheme. This is an OTP-based scheme in which the OTP is used as the *single factor* for wallet access, without any need of private keys.

The workflow of the OTP wallet is illustrated as a sequence diagram in Figure 2. At the beginning, Alice initializes a time-based OTP device (such as a mobile phone app) which generates and stores an OTP seed (leftmost column). Upon this generation, the device also generates a smart contract, which is constructed and submitted to the chain through a transaction $\mathsf{tx}_{\mathsf{construct}}$. This transaction generates a wallet address *pk* to which payers can send money for Alice. The OTP device also constructs a Merkle tree containing a large number of encrypted future OTPs and submits them to IPFS [10] (or other persistent storage service) publicly for availability (rightmost column). Both *pk* and all internal Merkle tree nodes are public. After this initial phase, the OTP device becomes air gapped.

At any time Alice wishes to spend, she visually consults her OTP device which displays a time-based OTP key. Using a (newly booted) online computer, she creates a transaction $\mathsf{tx}_{\mathsf{commit}}$ in which she commits to the $OTP$, the amount she wishes to spend, and the target address. The wallet waits a short amount of time before submitting the final $\mathsf{tx}_{\mathsf{reveal}}$ to confirm the spending. This transaction is accompanied by a Merkle tree proof-of-inclusion constructed using the IPFS data. The wallet then releases the payment to the desired address.

**Fig. 3.** Each $\mathsf{OTP}_t$ is timelocked with time $t$. All timelock ciphertexts $c_t$ are organized into a Merkle Tree whose root is $r$.




The contract deployed as a wallet is illustrated in Algorithm 3. The interaction with the contract by the user is illustrated in Algorithm 4.

The constructor accepts a parameter $r$ denoting a Merkle tree root. This is constructed by generating a large number (MAX_TIME) of time-based OTPs for the foreseeable future. For example, to support a wallet with a lifetime of 100 years with an hourly OTP resolution, 876,000 codes need to be generated. Let $OTP_t$ denote the OTP for future time $t \in \mathbb{N}$ (in the example, $t$ ranges from 1 to 876,000). These are generated from the OTP seed in the OTP device by invoking the pseudorandom function $\mathcal{G}(\mathsf{seed}, t)$ whose output has $\lambda$ bits of entropy. Each such OTP is then timelock encrypted for time $t$, multiplied by the expected hourly production rate of the blockchain (the *hourly* resolution is an arbitrary choice that can be made differently, giving rise to a tradeoff between how much data must be stored on IPFS versus how often the user can spend her money). Specifically, the software computes $c_t = \mathsf{timelock}(\mathsf{OTP}_t, t)$, (setting $c_t = \mathsf{WE.Enc}_{\mathcal{R}}(\mathsf{OTP}_t, x)$, where $x = (B, t)$). All of these $c_t$ are then organized into a Merkle tree as illustrated in Figure 3 using the hash function $H$ whose root is $r$. This $r$ is submitted to the constructor.

**Algorithm 3** Hours of Horus: A short OTP wallet.

```
contract OTPWallet
    BLOCK_DELAY ← 2k
    r ← ⊥
    spent ← ∅
    commitments ← ∅
    function construct(r̄)
        r ← r̄
    end function
    function commit(z, t)
        require(t > block.number + BLOCK_DELAY)
        commitments[z][t] ← true
    end function
    function reveal(OTP, salt, α_to, amount, c_t, w, t, π)
        h ← H(⟨OTP, salt, α_to, amount⟩)
        require(commitments[z][t])
        require(¬spent[t])
        require(MT.Ver(c_t, t, r, π))
        require(WE.Dec_R(c_t, w) = OTP)
        spent[t] ← true
        to.transfer(amount)
    end function
end contract
```

When Alice wishes to spend, she calls *commit* by issuing a $\mathsf{tx}_{\mathsf{commit}}$ transaction. The method takes parameters $z$ and $t$. Here, $z$ is the com-

**Algorithm 4** Interacting with the Hours of Horus wallet.

1: BLOCK_DELAY $\leftarrow 2k$
2: seed $\leftarrow \bot$ ▷ After generation, remains air gapped
3: $pk \leftarrow \bot$ ▷ Published so that money can be received
4: $c \leftarrow [\,]$ ▷ Published on insecure public storage (e.g., IPFS)
5: $B \leftarrow \bot$ ▷ Published on insecure public storage
6: **upon** initialize **do**
7:     seed $\xleftarrow{\$} \{0,1\}^\kappa$ ▷ Seed is generated with high entropy $\kappa$
8:     $B \leftarrow \mathcal{C}[-k]$ ▷ Latest stable block
9:     **for** $t \leftarrow 1$ to MAX_TIME **do**
10:         $\mathsf{OTP}_t \leftarrow \mathcal{G}(\mathsf{seed}, t)$ ▷ Time $t$ OTP with low entropy $\lambda$
11:         $x \leftarrow (B, \mathsf{HOURLY\_BLOCK\_RATE} \cdot t)$
12:         $c_t \leftarrow \mathsf{WE.Enc}_{\mathcal{R}}(\mathsf{OTP}_t, x)$
13:         $c \leftarrow c \,\|\, c_t$
14:     **end for**
15:     $r \leftarrow \mathsf{MT.build}(c)$
16:     $pk \leftarrow \mathsf{OTPWallet.construct}(r)$
17: **end upon**
18: **upon** spend($\alpha_{\mathsf{to}}$, amount) **do**
19:     $t_1 \leftarrow |\mathcal{C}| + \ell + \mathsf{BLOCK\_DELAY}$
20:     salt $\leftarrow \{0,1\}^\kappa$ ▷ Generate short-lived high-entropy salt
21:     $\mathsf{OTP}_t \leftarrow \mathcal{G}(\mathsf{seed}, t_1)$
22:     $z \leftarrow H(\langle \mathsf{OTP}_t, \mathsf{salt}, \alpha_{\mathsf{to}}, \mathsf{amount} \rangle)$
23:     $\mathsf{OTPWallet.commit}(z, t_1)$
24:     **wait until** $|\mathcal{C}| = t_1$
25:     $w \leftarrow \mathcal{C}\{B{:}\}$
26:     $\pi \leftarrow \mathsf{MT.prove}(c_t, c)$
27:     $\mathsf{OTPWallet.reveal}(\mathsf{OTP}_t, \mathsf{salt}, \alpha_{\mathsf{to}}, \mathsf{amount}, c_t, w, t_1, \pi)$
28: **end upon**

mitment $z = H(\langle \mathsf{OTP}, \mathsf{salt}, \alpha_{\mathsf{to}}, \mathsf{amount} \rangle)$. For $t$, Alice looks at her local chain $\mathcal{C}$, obtains its length $|\mathcal{C}|$ and evaluates $t = |\mathcal{C}| + \ell + 2k$. So, $t$ is a block height at least $\ell + 2k$ blocks in the future. As liveness ensures this transaction will confirm within $\ell$ blocks, the condition on Line 10 will succeed. The contract records the pair $(z, t)$ in the *commitments* set. Alice waits for her chain to grow to a height of $t$ blocks, at which point she issues the $\mathsf{tx}_{\mathsf{reveal}}$ transaction calling the *reveal* method. She reveals the OTP (no longer useful to any adversary), the salt, the destination address, and the amount to transfer. These are accompanied by proof-of-inclusion $\pi$ at position $t$ in the Merkle Tree whose root $r$ is recorded in the contract obtained from the data stored on IPFS (anyone can compute this). Additionally, it is accompanied by a witness that time $t$ has passed by providing the chain portion $\mathcal{C}\{B{:}\}$ (this can also be computed by anyone). The contract verifies the provided data are included in previous commitments, that the Merkle tree proof is valid for the specified position, and that the timelock encryption of the provided OTP corresponds to the given ciphertext.

The honest party will always succeed in creating a valid spending transaction. To see this, note that the party begins creating the commit transaction at time $t - \ell - 2k$. Due to liveness, it becomes confirmed at block $t - 2k$ at most, and so the check in Line 10 will pass. The reveal transaction will be called with the corresponding data and release the funds.

To see why an adversary cannot create a valid spending transaction beyond random guessing, we note that any adversary can either provide a commit transaction prior to block $t - 2k$, or afterwards. If she provides it prior, then the timelock scheme will protect the secret, and so the spending transaction will include a random OTP guess. On the other hand, if she provides it afterwards, it will not be accepted due to the time delay enforced in Line 10. Consult the Analysis section for a more complete argument.

## 5 Conclusion

We presented the first wallets that work securely without private keys, developing a wallet in which the user spends with an OTP from an air gapped device. We proved our scheme secure through a hybrid cryptographic/cryptoeconomic argument which may be of independent interest (in the appendix). The cryptoeconomic analysis led to a short OTP code: Just 6 alphanumeric characters suffice even for large capital of seven figures and with a conservative economic margin of 90% capital loss for the adversary. Our calculations were also conservative with respect to fees.

We extended our scheme to work in the proof-of-stake model, as well as variable difficulty proof-of-work model (in the appendix). We are the first to extend timelock encryption to proof-of-stake and to effectively use it for the variable proof-of-work case as well (previous considerations [41] considered the variable difficulty case, but did not account for the fact that the decryption time will be varying with the miner population adjustments).

As far as we know, our work is the first to build any useful protocol, and certainly to construct wallets, on top of timelock encryption and blockchains. We believe that timelock encryption and witness encryption is a promising cryptographic direction and, once established, will prove to be cornerstones of future protocol development for blockchains.

## Appendix

## A Analysis

We now give a more complete analysis of the scheme. First, let us prove that the Password-based wallet of Algorithm 1 is correct.

**Theorem 1 (Password Wallet Correctness (Informal)).** *Let the blockchain have* liveness *and* safety*, and let the witness encryption scheme* $\mathsf{WE}$ *be* correct*. An honest party spending at block height* $t_1 - \ell - 2k$ *or earlier will generate a valid spending transaction for Algorithm 1.*

*Proof (Sketch).* The contract is created when $B = \mathcal{C}[-k]$ is stable. Due to safety, all the future chains will be extending this block. The contract is initialized with $x = (B, t)$ by issuing the $\mathsf{tx}_{\mathsf{construct}}$ transaction. Due to liveness, this transaction is confirmed within $\ell$ blocks. The honest user then creates a transaction $\mathsf{tx}_{\mathsf{commit}}$ when her own chain has length $|\mathcal{C}| = t_1 - \ell - 2k$. Due to liveness, this transaction becomes confirmed for all honest parties after $\ell$ blocks have elapsed, and is placed in position $\mathcal{C}[t_1 - 2k]$ or earlier. Therefore, Line 10 of the method *commit* succeeds. When $|\mathcal{C}| = t_1$, the honest user calls *reveal*, passing $w$. Due to the correctness of the witness encryption scheme, the decryption succeeds. The password and salt revealed match the ones committed. Due to liveness, this transaction becomes confirmed.

The correctness of the OTP-based scheme is similar.

**Theorem 2 (OTP Wallet Correctness (Informal)).** *Let the blockchain have* liveness *and* safety*, and let the witness encryption scheme* $\mathsf{WE}$ *be* correct*. An honest party spending multiple times prior to* $\mathsf{MAX\_TIME} - \ell - 2k$ *will succeed in creating valid transactions. in Algorithm 3.*

*Proof (Sketch).* The proof is the same as above, with the difference that the value $t$ is provided at the $\mathsf{commit}$ time, not the $\mathsf{construct}$ time. The argument that Line 10 will be successful remains the same due to liveness.

Our security analysis is in a hybrid *cryptographic* and *cryptoeconomic* setting. In the system described, we have two security parameters. First, we have the *cryptographic* security parameter $\kappa$ ($\approx 128$ bits), which determines the security of the hash function, the security of the witness encryption scheme, and the security of the blockchain (in terms of liveness, safety, and common prefix). The probability of failure is negligible in this parameter. Any breakage in this parameter can be catastrophic for the system and can potentially provide the adversary with gains without any cost. Secondly, we have the much shorter *cryptoeconomic* security parameter $\lambda$ ($\approx$ 35 bits) which denotes the entropy of the chosen user password $sk$ or the length of each OTP $\mathsf{OTP}_t$. While this parameter is hopelessly short from a cryptographic point of view (and $2^{-35}$ is nothing but negligible), we will use it to establish a lower bound in the economic cost of an attack. In particular, we will tweak this parameter so that the return-on-investment of an attack can be made arbitrarily close to $-100\%$. The result will be that the adversary can make the probability of success non-negligible, but at an economic cost which renders such attempts irrational.

We begin by stating our Decentralized Rate Limiting lemma, which establishes that an adversary must necessarily submit transactions to the blockchain in order to have any non-negligible probability of success. The probability of success is determined by the number of transactions submitted by the adversary and made persistent by the system. Based on this result, we will determine the cryptoeconomic parametrization ($\lambda$) required to make the system economically infeasible to attack.

**Lemma 1 (Decentralized Rate Limiting (Informal)).** *Consider a static difficulty proof-of-work blockchain with* safety *and* common prefix*. Let the hash function $H$ be collision-resistant and preimage-resistant, and let the witness encryption scheme $\mathsf{WE}$ be a* secure witness encryption with witness extractability*. A PPT adversary who submits fewer than $g$ transactions that are eventually confirmed by all honest parties has a probability of achieving a valid spending transaction in Algorithm 1 upper bounded by $\frac{g}{2^\lambda} + \mathsf{negl}(\kappa)$.*

*Proof (Sketch).* In order for the adversary to have a valid transaction, she must have created a $\mathsf{tx}_{\mathsf{reveal}}$ in which she passes a password $sk$, a $\mathsf{salt}$ and a $\alpha'_{\mathsf{to}}$ address which is different from the honestly provided $\alpha_{\mathsf{to}}$ address. This $\mathsf{reveal}$ transaction must be confirmed into the chain $\mathcal{C}$ adopted by a verifier honest party $P_v$ and have matching data with a previous $\mathsf{tx}'_{\mathsf{commit}}$ transaction which was placed earlier in $\mathcal{C}$. Additionally, $\mathsf{tx}'_{\mathsf{commit}}$ must be in $\mathcal{C}[t_1 - 2k]$ or earlier (due to the check in Line 10). Due to the collision resistance of $H$, the respective commit transaction must be different from the one ($\mathsf{tx}_{\mathsf{commit}}$) provided by the honest party, as $\alpha_{\mathsf{to}} \neq \alpha'_{\mathsf{to}}$.

Let $r_c$ denote the round during which the spender honest party $P_s$ broadcasts their $\mathsf{tx}_{\mathsf{commit}}$ transaction to the network, and let $r_z$ denote the last round during which *all* honest parties have chains with length of at most $t_1 - k$.

Let us consider all adversarially generated commit transactions $\mathsf{tx}^i_{\mathsf{commit}}$ ($i \geq 1$) that are eventually reported as *stable* by $P_v$ (the adversary can also create transactions that do not make it in the chain of $P_v$, but we will not count these). For these transactions, let us consider the round $r_i$ during which each of these transactions $\mathsf{tx}^i_{\mathsf{commit}}$ was created.

**Case 1:** $r_i < r_c$. Since the honest spender has not yet submitted a commitment, the only information that the adversary has is the ciphertext $c$. If at this round the adversary can distinguish between $sk$ and any other plaintext in $\{0,1\}^\lambda$ with probability non-negligible in $\kappa$, then, due to the witness extractability of $\mathsf{WE}$, an extractor can extract a witness $w$ attesting to the existence of a chain of height $t_1$. But in that case, we can perform a computational reduction to an adversary that breaks the common prefix property of the chain by producing a chain of height $t_1$ at round $r_i$ when the honest party $P_v$ has adopted a chain of length only $t_1 - \ell - 2k$. This breaks the common prefix assumption.

**Case 2:** $r_c \leq r_i \leq r_z$. In this case, the honest spender has broadcast a commitment to the network, but there are no chains of length $t_1$. The adversary now holds both the timelocked ciphertext $c$ and the commitment $z$. Again the adversary should not be able to distinguish between $sk$ and any other plaintext in $\{0,1\}^\lambda$, except with probability negligible in $\kappa$ (recall that the $\mathsf{salt}$ is kept secret and has $\kappa$ bits of entropy). Otherwise, we can either perform a reduction to a common-prefix-breaking adversary making use of witness extractability, or we can perform a reduction to a preimage-resistance-breaking adversary.

**Case 3:** $r_i > r_z$. By the definition of $r_z$, in round $r_i$ there must exist an honest party with a chain of length at least $t_1 - k$. By the common prefix property, all other honest parties have a chain of length exceeding $t_1 - 2k$.

Let us consider what happens in all of these three cases. In the first two cases, any *single* guess that the adversary places into a transaction can be correct with probability $\frac{1}{2^\lambda} + \mathsf{negl}(\kappa)$. In the third case, while the adversary can potentially guess with better probability (due to the chain reaching its leakage point $t_1 - k$), any such transactions can never make it into the chain eventually adopted by $P_v$, as the check in Line 10 will fail.

As the transactions that eventually make it into the chain of $P_v$ were all generated prior to $r_z$, the probability that each of them is a valid spending transaction is upper bounded by $\frac{1}{2^\lambda} + \mathsf{negl}(\kappa)$. If the adversary submits at most $g$ such transactions, and applying a union bound, the overall probability of success is $g(\frac{1}{2^\lambda} + \mathsf{negl}(\kappa)) = \frac{g}{2^\lambda} + \mathsf{negl}(\kappa)$.

The above Lemma is identical for our other construction. We state it for completeness.

**Lemma 2 (OTP Decentralized Rate Limiting (Informal)).** *Consider a static difficulty proof-of-work blockchain with* safety *and* common prefix. *Let the hash function $H$ be collision-resistant and preimage-resistant, and let the witness encryption scheme $\mathsf{WE}$ be a* secure witness encryption with witness extractability. *Let $\mathcal{G}$ be a secure pseudorandom function $\{0,1\}^\kappa \times \mathbb{N} \longrightarrow \{0,1\}^\lambda$. A PPT adversary who submits fewer than g transactions that are eventually confirmed by all honest parties has a probability of achieving a valid spending transaction in Algorithm 3 upper bounded by $\frac{g}{2^\lambda} + \mathsf{negl}(\kappa)$.*

*Proof (Sketch).* The proof here is identical to Lemma 1, noting that each of the different $\mathsf{OTP}_t$ essentially gives rise to independent attack paths to the adversary. Due to the pseudorandom nature of the OTP scheme, any previous OTPs do not reveal any information to our polynomial adversary. An adversary making a spending attempt has a probability of $\frac{1}{2^\lambda}$ of succeeding in each attempt, unless it can be reduced to a collision-resistance-breaking adversary, a common-prefix-breaking adversary, or an adversary breaking the pseudorandomness of the OTP scheme. But all of these events are negligible in $\kappa$.

At this point, we have established that the probability of success is negligible in both parameters $\kappa$ and $\lambda$. However, we will keep the parameter $\lambda$ short, and we will make the $\kappa$ parameter reasonably long ($\kappa = 128$). Setting $\lambda = \kappa$ would, of course, give sufficient security. The reason for separating these two parameters is that the $\lambda$ parameter affects the usability of the system: It is the number of characters that must be remembered by the user in the case of a password, or the number of characters that must be visually copied by the user in the case of an OTP.

In the above result, we have expressed the probability as a sum of two terms: $\frac{g}{1^\lambda} + \mathsf{negl}(\kappa)$. This reflects the nature of the two parameters: We opt to calculate the *concrete* probability with respect to $\lambda$, but only give an asymptotic probability with respect to $\kappa$. This treatment hints at our intentions: Our high-level argument was to condition the system to the overwhelming events that there will be no cryptographic breakage in the hash function, common prefix property, blockchain safety, blockchain liveness, and OTP pseudorandomness. Conditioned under these events, the concrete probability as a function of $\lambda$ allows us to make an argument of why any attack is uneconomical. This gives rise to our (cryptoeconomic) security theorem.

**Theorem 3 (Cryptoeconomic Security (Informal)).** *Consider a chain with fee $f$ per transaction. If the wallet of Algorithm 1 or Algorithm 3 is used with a maximum capital of $V$, then the parametrization $\lambda > \log \frac{V}{f}$ yields a negative expectation of income for the adversary, with overwhelming probability in $\kappa$. Additionally, the expected return-on-investment for this adversary is at most $\frac{V}{f 1^\lambda} - 1$.*

*Proof (Sketch).* Consider an adversary who submits $g$ transactions that are eventually confirmed by every honest party. This adversary is irrevocably investing a capital of $gf$ for this attack. By Lemma 1, the adversary has a probability of success upper bounded by $\frac{g}{1^\lambda}$ (with overwhelming probability in $\kappa$). The expected income for this adversary is at most $\mathbb{E}[\mathsf{income}] \leq V \frac{g}{1^\lambda} - gf$. Taking $\lambda > \log \frac{V}{f}$, we obtain $\mathbb{E}[\mathsf{income}] < 0$. The expected return-on-investment is $\frac{\mathbb{E}[\mathsf{income}]}{gf} - 1$.

In this scheme, we can set $\lambda$ big enough to make the return-on-investment as close to $-100\%$ as we want. If we want the return-on-investment to be $-1 + \epsilon$ for some $\epsilon \in (0, 1]$, we let $\lambda = \log \frac{V}{f\epsilon}$. In short, we can make the adversary lose an amount arbitrarily close to all their money in expectation.

To consider some concrete parametrization of the scheme, let us assume that we wish to establish a target $-90\%$ ($\epsilon = 0.1$) expected return-on-investment for the adversary in a wallet where we want to store up to $V$ = \$100,000 in capital at any point in time. Consider a blockchain where the fees per transaction are[1] at least $f$ = \$1.60. We obtain $\lambda = \log \frac{V}{f\epsilon} = \log_2 625{,}000 < 20$ bits. This corresponds to just 6 numerical characters (base 10), or just 4 alphanumeric characters (base 58). A standard OTP authenticator such as Google's Authenticator application is therefore appropriate for such parameters. Increasing the maximum capital that will be stored in the wallet by three orders of magnitude to \$100,000,000 requires 5 alphanumeric characters instead.

[1] This price corresponds to Ethereum–fiat prices and gas fees for simple transfer transactions in May 2021. As smart contract transactions are significantly more expensive, this is a conservative estimation for the fees.

## B Proof-of-Stake

Contrary to proof-of-work blockchains, a proof-of-stake chain progresses in *slots* (prefixed time durations) during which parties can create blocks or remain silent. As in proof-of-work, each block header $B_i$ consists of $\langle \mathsf{tx}_i, s_i \rangle$, but now does not include a $\mathsf{ctr}_i$. The blocks created at each slot are accompanied by a signature $\sigma_i$ created by a designated leader for the slot. A proof $\pi_i$ illustrating the designated leader is the rightful one is also broadcast together with the block. The probability that a party becomes a leader at a given slot is roughly proportional to the stake they hold within the system. These proofs of leadership are different depending on the system and can be the random outcome of a multiparty computation, as in the *Ouroboros* [39] system, or a verifiable random function [45] evaluated on this randomness, as in the *Ouroboros Praos* [24] construction. In the first system, each slot is allocated to precisely one party, and the production of no blocks, or two competing blocks in the same slot, indicates adversarial behavior. In the second system, it is possible that a slot is allocated to multiple honest parties, or no parties at all. These details do not affect our scheme, as long as the following property is maintained: For any $2k$ consecutive slots, at least $k+1$ slots are allocated to an honest party. Additionally, we will assume that the common prefix property holds here, too.

As in proof-of-work, the chain is split into epochs. At the end of each epoch, a multiparty computation is performed to determine the randomness value for the next epoch based on the stake distribution during the current epoch. Different systems use different MPCs. Our only requirement is that these MPCs provide some evidence $u_e$ that the randomness for epoch $e$ is $\rho_e$. This evidence must be polynomially checkable in retrospect. This requirement is satisfied in proof-of-stake blockchains, as it is this evidence that allows new nodes to bootstrap correctly [5].

In this section, we adapt our OTP wallet construction to the proof-of-stake setting (the password wallet can also be adapted likewise). For concreteness, we describe a construction for the *Ouroboros* [39] and *Ouroboros Praos* [24] systems, but our results are extensible to other systems as well (such as Snow White [11] and Algorand [44]).

The construction does not change much from the proof-of-work case, so we only provide a sketch of the construction here. The smart contract remains identical, except for the moderately hard NP language describing the existence of a proof-of-work witness. More concretely, the problem instance $x$ is now $(\rho, sl, D, t)$, where $\rho$ denotes the randomness of the current epoch, $sl$ denotes the slot during which block $B$ (the most recently known stable block $\mathcal{C}[-k]$) was generated, $D$ denotes the stake distribution during the current epoch, and $t$ denotes the future time. While the proof-of-stake chains also enjoy the common prefix property, unfortunately, we cannot simply take any blockchain that has length $t$ following $B$, because the adversary can create blockchains of arbitrary length. The proof-of-stake system ensures that such chains are not taken into account by checking that any blockchain received on the network does not contain blocks that were issued in future slots [39]. However, we cannot incorporate this check in the form of an NP language, as we do not have access to a clock.

Instead, we rely on a critical property of the proof-of-stake system that states that, in any consecutive $2k$ slots, at least $k+1$ will be honestly allocated. Therefore, we reinterpret the parameter $t$ to mean the *number of slots* after block $B$ instead of the *number of blocks*. The witness $w$ consists of two parts: *block* data and *epoch* data. The block data contains a sequence $(\sigma_1, H_1, \pi_1, sl_1), (\sigma_2, H_2, \pi_2, sl_2), \cdots, (\sigma_d, H_d, \pi_d, sl_d)$ of signatures $\sigma_i$ each with their corresponding slot $sl_i$, with $sl_i > sl$ and a proof of leadership $\pi_i$. As in the proof-of-work case, no transaction data is verified. The epoch data contains a sequence $(\rho_1, u_1), (\rho_2, u_2), \cdots, (\rho_e, u_e)$ spanning all the epochs starting from the epoch of slot $sl$ up until the epoch of slot $sl + t$. For each of these, the randomness $\rho_j$ and evicence $u_j$ of the multiparty computation leading to it (typically a collection of signatures) is included.

The relation $\mathcal{R}$ polynomially verifies that all the signatures $\sigma_i$ correctly sign their respective plaintext $H_i$, that the proofs of leadership $\pi_i$ are correct, and that the evidence $u_j$ for the randomness $\rho_j$ of each epoch is correct. Critically, it also checks that, for every window of length $2k$ slots, at least $k+1$ blocks have been provided.

This completes the basic scheme. We can improve upon this scheme by noting that blocks and signatures for everything but the most recent epoch are not necessary, as long as the randomness and its evidence for each epoch is given. This evidence can be made quite short using ATMs schemes, in which the evidence consists of aggregate threshold signatures (c.f., [37]). In such an optimization, a constant amount of bits is required per epoch. Blocks only need to be presented for the last epoch in order to have better time granularity. However, here, too, some pruning can occur: It is sufficient that only $k+1$ blocks and signatures pertaining to the most recent $2k$ slots of the most recent epoch are presented. The relation $\mathcal{R}$ can then simply check the evidence for each epoch randomness, that the

$k+1$ signatures are correct, that they all fall within a $2k$ window, and that the slot during which the last such block was generated is $t$. Again, the witness encryption can be composed with a zk-SNARK to make the witnesses constant size.

Contrary to proof-of-work where the velocity of the chain is unknown, despite bounded, in the proof-of-stake case we have a much better grasp on how quickly the time $t$ will be reached, as it is a slot number. While the adversary still enjoys some early leakage ($k$ slots early), the timelocked data will be available at the prespecified time. In the proof-of-work case, it is possible that the blockchain growth rate will increase or decrease due to the stochastic nature of block production. As such, the proof-of-stake scheme is naturally fitting to the timelock problem.

## C Variable Difficulty Proof-of-Work

In the variable difficulty model, the target $T$ is adjusted based on how the chain evolves. Concretely, the chain is split into *epochs* of constant block length $m$ each. At the end of each epoch, the timestamp at the end of the chain is noted and the mining target is adjusted with the aim of keeping the expected block production rate constant. The way $T$ is adjusted is algorithmically determined, and it is important that it follows certain rules. While we will not articulate the exact rules, we remark that the new value $T'$ must fall within a range $\tau T, \frac{1}{\tau}T$, where $\tau \in (0,1)$ (for example, Bitcoin sets $\tau = \frac{1}{4}$). This critical condition is necessary to avoid certain attacks [7].

The proof-of-work OTP wallet described in the main body is suitable for the *static difficulty* model in which the mining target $T$ is not adjusted and remains constant. Real blockhchain systems adjust their $T$ parameter dynamically in every epoch [29]. Our construction in this section will work in both models.

The construction for the dynamic difficulty proof-of-work model is similar to the static difficulty proof-of-work construction, with a key difference in the NP language used for witness encryption. The key idea is that, instead of encrypting for a chain descending from $B$ and consisting of $t$ *blocks* in the future, we need to encrypt for a chain descending from $B$ and consisting of blocks that have together accumulated a total of $t$ *difficulty*. More concretely, the witness encryption problem statement $x = (B, t, T_0, r_0, \nu)$ contains $B$ and $t$ as before, but now also contains the term $T_0$, the difficulty of the chain at the point when timelock encryption took place, the round $r_0$ during which the first block of the current epoch was generated, as well as $\nu$, the position of block $B$ within its current epoch ($\nu = (|C| - k) \bmod m$, where $m$ denotes the fixed epoch length).

The format of the witness $w$ is now a sequence of block headers in the form $\langle T_i, \mathsf{ctr}_i, \mathsf{tx}_i, H(B_{i-1}), r_i \rangle$, where $\mathsf{ctr}_i$, $\mathsf{tx}_i$ and $H(B_{i-1})$ are as before, and, additionally, $T_i$ is the individual block's mining target and $r_i$ is the round during which it was mined (following the notation of Garay et al. [29]). The relation $\mathcal{R}$ checks that the witness provided forms a chain that begins at the last known stable block $B$, that every block satisfies the dynamic difficulty proof-of-work equation $H(B_i) \leq T_i$, and that difficulty has been adjusted correctly. Specifically, for the difficulty adjustment, it checks that for all $i \geq 2$, if $i - \nu \bmod m \neq 1$, then $T_{i-1} = T_i$ (ensuring difficulty was not improperly adjusted internally within the epoch of $B$ or any subsequent epochs). It also checks that the rounds provided are increasing $r_i < r_{i+1}$, and ensures that the difficulty at the epoch borders $i-1$, $i$ with $i - \nu \bmod m \neq 1$ and $i > 1$ has been correctly adjusted by verifying that $T_i = \min(\max(T'_i, \frac{1}{\tau}T'_i), \tau T'_i)$, where $T'_i = \frac{r_{i-1} - r_{i-m}}{a} T_{i-1}$ is the unclamped target, and the term $a$ indicates the expected block production rate of the system in rounds [29]. To achieve security with overwhelming probability, and not just in expectation, in $\kappa$, it is imperative that the $\tau$ bounds are also checked by $\mathcal{R}$ (see Bahack [7] for more details on a tail attack). Lastly, the relation checks that the difficulty is sufficient, as required by the $t$ parameter. To do this, the difficulty of each block in the witness is summed up to discover the cummulative difficulty of the fork, checking that $\sum_{\langle T_i, \_, \_, \_, \_ \rangle \in w} \frac{1}{T_i} \geq t$.

Now that the precise NP language has been established, a couple of things need to be changed in our protocol. First of all, at the time the OTPs are generated, $\mathsf{MAX\_TIME}$ no longer indicates the maximum lifetime of the wallet (in chunks of $\mathsf{HOURLY\_BLOCK\_RATE}$ blocks), but the maximum total *difficulty* accumulated during the lifetime of the wallet. So we rename it to $\mathsf{MAX\_DIFFICULTY}$. This parameter is sensitive in case the difficulty *increases*. Hence, the value must be *increased* sufficiently (at the cost of increased IPFS storage needs) to cover for all foreseeable difficulty adjustments for the expected lifetime of the wallet. One way to do this is to look at past difficulty adjustment trends and extrapolate them to the future for the number of years the wallet is to be usable. In any case, this prediction does not need to be perfect: In the unfortunate case that the OTPs are close to becoming exhausted, which can easily be observed by inspecting the chain as it evolves, the wallet can be sunset by moving the funds to a new wallet with a new lifetime.

Next, the value HOURLY_BLOCK_RATE no longer indicates the number of blocks generated in one hour, but the amount of difficulty that must be accumulated before the next OTP can be utilized. So we rename it to OTP_ROTATION_DIFFICULTY. This parameter is sensitive in case the difficulty *decreases*. Hence, this value must be *decreased* sufficiently (at the cost of increased IPFS storage needs) to allow the user to spend as quickly as desired. As difficulty typically does not decrease, one way to do this is to look at the previous HOURLY_BLOCK_RATE parameter and multiply it by the current difficulty to obtain a lower bound for the future. If one can predict a lower bound for how much future difficulty increases, it is also possible to timelock encrypt with non-uniform difficulty: The difference in the difficulty used to witness encrypt two early consecutive OTPs can be smaller than the difference in difficulty used to witness encrypt two later consecutive OTPs. The precise mechanism to do this effectively depends on the cryptocurrency and empirical measurements.

Lastly, the smart contract must be modified in the security-critical line that ensures that $t$ is sufficiently in the future. In the static difficulty, $t$ counts the number of blocks (or slots in the proof-of-stake case), but here it is counting difficulty. Therefore, it cannot be compared to block.number, and the $2k$ delay (which also counts blocks) cannot be readily applied. Instead, we must use a new variable[2] block.cumdiff, the cummulative difficulty collected by the blockchain if all the difficulty from genesis to the current block is summed up. Additionally, the $2k$ factor must be weighted by the current difficulty $\frac{1}{\mathsf{block.T}}$, where block.T indicates the mining target of the current block.

The algorithms for the variable difficulty OTP wallet appear in Algorithm 5 and 6.

[2] This block property is not currently available in Ethereum Solidity, but it is available in web3 as `block.totalDifficulty`. It is an easily implementable solution, but can even be incorporated into a smart contract within the current infrastructure without any forks [36].

**Algorithm 5** Hours of Horus in variable difficulty.

```
contract OTPWallet
    BLOCK_DELAY ← 2k
    r ← ⊥; spent ← ∅; commitments ← ∅
    function construct(r̄)
        r ← r̄
    end function
    function commit(z, t)
        require(t > block.number + BLOCK_DELAY/block.T)
        commitments[z][t] ← true
    end function
    function reveal(OTP, salt, α_to, amount, c_t, w, t, π)
        h ← H(⟨OTP, salt, α_to, amount⟩)
        require(commitments[z][t])
        require(MT.Ver(c_t, t, r, π))
        require(WE.Dec(c_t, w) = OTP)
        to.transfer(amount)
    end function
end contract
```

**Algorithm 6** Interacting with variable difficulty Hours of Horus.

```
BLOCK_DELAY ← 2k
seed ← ⊥; pk ← ⊥; c ← [ ]; B ← ⊥
upon initialize do
    seed ← {0, 1}^κ                         ▷ Seed is generated with high entropy κ
    B ← C[−k]                                ▷ Latest stable block
    for t ← 1 to MAX_DIFFICULTY do
        OTP_t ← G(seed, t)                   ▷ Time t OTP with low entropy λ
        x ← (B, OTP_ROTATION_DIFFICULTY · t)
        c ← c ∥ WE.Enc(OTP_t, x)
    end for
    r ← MT.build(c)
    pk ← OTPWallet.construct(r)
end upon
upon spend(α_to, amount) do
    t_1 ← ⌈C.cumdiff + (ℓ + BLOCK_DELAY)/C[−1].T⌉
    salt ← {0, 1}^κ                         ▷ Generate short-lived high-entropy salt
    OTP_t ← G(seed, t_1)
    z ← H(⟨OTP_t, salt, α_to, amount⟩)
    OTPWallet.commit(z, t_1)
    wait until C.cumdiff = t_1
    w ← C{B:}
    π ← MT.prove(c_t, c)
    OTPWallet.reveal(OTP_t, salt, α_to, amount, c_t, w, t_1, π)
end upon
```

The argument for the correctness of the scheme and the security of the scheme remains the same. Some remarks about the security portion are in order. First, recall that any blockchain protocol does not accept blocks with timestamps in the future. In the static difficulty model, this was not important, but in the proof-of-stake and in the variable difficulty model, it is something to consider. In particular, for the variable difficulty case, if the adversary constructs blocks timestamped with future rounds, she can cause the difficulty to drop more than it would be possible in a real-world execution. However, this does not bless the adversary with more mining power. Additionally, such chains will not be mined on by honest parties (because they are considered invalid, as of yet), and so they will only be extended by the adversary. The effect is the contrary of a difficulty raising [7] attack: The total difficulty accumulated as the target difficulty is artificially decreased becomes concentrated to its expected value. Hence, the minority adversary, who does not win in expectation, has an even lower probability of accumulating the difficulty goal described by $x$ in this futuristic chain. Therefore, we shall not be concerned about this behavior.

**The bounded delay model.** The above high-level analysis, as well as the more detailed analysis of the static case in Section A, was in the synchronous setting. However, all the proofs made direct use of high-level chain properties such as the common prefix property, safety, and liveness. The use of the rounds $r_c$, $r_i$, and $r_z$ to split time into chunks in the proof of Lemmas 1 and 2 is material to the proof. However, these rounds are defined based on transaction broadcast events and lengths attained by local honest chains. In a setting where the parties incur an unknown bounded delay $\varDelta$ (which satisfies certain conditions [27]), the properties of the chain still hold, albeit with worse parameters $k$ and $\ell$, and the same security proof remains valid.

## D Discussion

Having completed the presentation of our schemes in both static and variable proof-of-work, as well as in proof-of-stake, we now discuss a couple of remarks (and shortcomings) of our scheme.

**Large witnesses.** The witnesses to the problem instances of the moderately hard NP language describing chain creation can grow linearly together with the chain. To reduce witness size, a (zero knowledge) proof of knowledge such as a zk-SNARK [9] can be used. In this case, instead of witness encrypting against a decryptor who “knows a witness consisting of a list of chain headers that satisfy the blockchain properties," one can instead witness encrypt against a decryptor who "knows a witness consisting of a zero-knowledge proof of knowledge attesting to the knowledge of chain headers that satisfy the blockchain properties." This composition of witness encryption and zero-knowledge proofs allows the witnesses presented to the blockchain to become constant size. For further details, refer to [41].

**Using standard time-based OTP.** A standard time-based OTP cannot be used by the user of our protocol, because the chain, as a stochastic process, may have grown faster or slower than expected. The OTP device must know the current height of the blockchain to be able to reveal the correctly indexed OTP (which will reside at OTP index $|\mathcal{C}| + \ell + 2k$). One practical way to achieve this is to have the mobile wallet (or a block explorer) display the current block height, which can then be inputted by the user to the OTP application.

**Security of the online computer.** One critical point of infrastructure is the online computer to which the user inputs their OTP code. If that computer becomes compromised, it can change the target address and amount that the user is inputting and deplete the wallet. One practical mechanism to cut the user's losses is to establish an hourly limit in the amount that can be spent by the wallet. The simplest way is to add an assertion in Algorithm 3 that ensures the amount spent in every *reveal* call is limited. In such a case, the compromised computer can only steal the user's funds *once*, and up to the specified hourly limit, before being detected. More complex schemes can introduce limits for various periods of time, and the contract would have to keep track of how much money has been spent in every period of time.

**Two-factor wallets.** The OTP scheme can be used either as a single-factor (as described in the main body) or as a second factor combined with a private key if desired. It is an effective second factor because, if either, but not both, of the private key or the OTP device become compromised, the wallet remains secure.

**A bounty for the miners.** In our analysis, we have considered a rational adversary who is only allowed to allocate her capital into taking guesses for the user passwords and OTPs and holds a minority of the adversarial power. This worldview is slightly myopic. An adversary with a large capital operating in an open world can also use this money for other purposes such as bribing miners. In fact, let us take a step back and reconsider the *honest majority* assumption of the chain which allowed us to conclude that the properties of common prefix, safety, and liveness hold. What if the miners are not honest, but rational, instead? In this case, the properties do not hold (it is known that the honest protocol is not a Nash equilibrium [26], although it may be close to it [38]). In our case of keyless wallets, the wallet functions as a *bounty* to the miner who creates a long chain fork: If an adversary can violate the common prefix property, then she can, as far as the chain is concerned "go back in time." In such an attack, after the secrets become timelock decryptable, the adversary creates a long chain reorganization and resubmits the correct guess to the wallet. As the reorganization was long, the delay check in the smart contract succeeds and the trial is correctly committed to the chain, granting the adversary the prize. This can be dangerous.

However, It can be argued that such bounties can be created by the adversary herself: If she double spends her money, she creates an incentive for herself to go back in time and reclaim it. But there is a crucial difference: The adversary can only spend her own money in the double spending case. Namely, although in a double spending case, the party receiving the money was harmed by the chargeback, it is the adversary's money that is being double spent, not someone else's. There is another critical difference here: While a single adversary can create such bounties for herself, keyless wallets are *universal* bounties claimable by any miner. We remark, however, that a double spending adversary can twist the double spending attack in a way that makes it a universal bounty: The adversary first creates a legitimate transaction spending some of her own money and receives, say, fiat money in exchange. She then creates a double spending transaction in an alternative fork: That double spending transaction pays 50% of her money to herself, and the rest to the miner who confirms the given block. In this case, all miners are incentivized to confirm this transaction and fork.

Therefore, we argue that the existing blockchain systems are not very different in the way incentives are aligned as compared to our proposed wallet. Nevertheless, as highlighted in the analysis, the proposed wallet does indeed have a different security model from a standard wallet: It is not purely cryptographic, but a cryptographic/cryptoeconomic hybrid. A quantified analysis of the (ir)rationality of conducting a common prefix attack is explored by Bonneau et al. [12].

**Temporary dishonest majority.** Our analysis assumed adversarial minority throughout the execution. However, this may not necessarily be the case. Blockchains have faced situations where the adversarial power has temporary majority spikes, even though the adversary generally controls only a minority [4,6]. One of the arguments protecting from double spending stems from the ability of the user to set their own local $k$ parameter when they consider which transaction to accept as confirmed. The parameter $k$ is not a global parameter of the system, but it can be set by each user individually at the time of payment. If there are rumours or evidence that the chain may be under attack, the user can delay accepting payments. This is not the case for our protocol. While the user can set the critical $2k$ delay when instantiating the contract, and this is a local choice, this choice cannot be changed later. If an adversary attains majority after the contract has been instantiated, she will be able to roll back the chain sufficiently to steal the user's money. This limitation of the system must be taken into account when deciding about the $k$ parameter of the wallet: The parameter must not only withstand current adversarial bounds, but adversarial bounds through the lifetime of the wallet. One mechanism to deal with this issue is to migrate from a wallet with a small $k$ to a different wallet with a more conservative value when evidence appears that the chain may become attacked in the near future. If the blockchain network cannot be trusted to maintain honest majority, and the adversarial majority spike length is unpredictable, the money cannot be left and forgotten as in a key-based wallet.

**Detectability of brute force attacks.** One of the advantages of our system is that brute force attacks are not only economically infeasible, but they are also detectable. If an adversary submits a brute force attempt to the wallet, a *commit* transaction will appear on the chain that the honest user will see. As such, the user can decide to move their funds out of their wallet if they observe such behavior. This benefit stems from the fact that the brute forcing of user passwords and OTPs cannot be done offline, but must necessarily be made on the chain. Our rate limiting scheme is therefore not just enabling limiting, but also detection. It is the first scheme of its kind that works in a decentralized manner on-chain.

## E Auxiliary Theorems

The following theorem establishes that chains cannot grow too quickly. It uses the notation adopted from the backbone series [28, 29].

**Theorem 4 (Chain Growth Bound).** *In a typical execution, consider a round $r_0$ during which the longest chain that exists on the network has a height of $h_0$. Then at round $r_1 > r_0$, let $h_1$ denote the height of the longest chain that exists on the network. The chain cannot grow too quickly:*

$$\Pr[h_1 - h_0 > (1+\epsilon)(r_1 - r_0)nq\frac{T}{2^\kappa}] \leq \exp(-\Omega(\kappa(r_1 - r_0)))$$

*Proof.* Let us consider the case where the adversary uses all her queries (if the adversary does not use all of her queries, we can force her to do so at the end of her round and ignore the results). Then there will be $nq$ queries per round in total, and $(r_1 - r_0)nq$ queries across all the rounds in $r_1 - r_0$. In typical executions, the longest chain on the network can only grow if a query is successful. The probability of success of a query is $\frac{T}{2^\kappa}$. The random variable $h_1 - h_0$ is hence upper bounded by a Binomial distribution with parameters $\frac{T}{2^\kappa}$ and $(r_1 - r_0)nq$, which has expectation $(r_1 - r_0)nq\frac{T}{2^\kappa}$. Applying a Chernoff bound with error $\epsilon$, we obtain the desired result.

We include the Chernoff bound, referenced at a high level throughout this paper, for completeness.

**Theorem 5 (Chernoff bounds).** *Let* $\{X_i : i \in [n]\}$ *are mutually independent Boolean random variables, with* $\Pr[X_i = 1] = p$*, for all* $i \in [n]$*. Let* $X = \sum_{i=1}^n X_i$ *and* $\mu = pn$*. Then, for any* $\delta \in (0,1]$*,*

$$\Pr[X \leq (1-\delta)\mu] \leq e^{-\delta^2\mu/2} \text{ and } \Pr[X \geq (1+\delta)\mu] \leq e^{-\delta^2\mu/3}.$$

## F Acknowledgements

The author thanks Pyrros Chaidos, Alexander Chepurnoy, and Dimitris Karakostas for reading early versions of this paper and providing useful feedback towards a clearer narrative.

## References


1. M. R. Albrecht, S. Bai, and L. Ducas. A subfield lattice attack on overstretched NTRU assumptions - cryptanalysis of some FHE and graded encoding schemes. pages 153–178, 2016.
2. M. R. Albrecht, C. Cocis, F. Laguillaumie, and A. Langlois. Implementing candidate graded encoding schemes from ideal lattices. pages 752–775, 2015.
3. M. Arapinis, A. Gkaniatsou, D. Karakostas, and A. Kiayias. A formal treatment of hardware wallets. In *International Conference on Financial Cryptography and Data Security*, pages 426–445. Springer, 2019.
4. G. Avarikioti, L. Käppeli, Y. Wang, and R. Wattenhofer. Bitcoin security under temporary dishonest majority. In *23rd Financial Cryptography and Data Security (FC)*. Springer, 2019.
5. C. Badertscher, P. Gaži, A. Kiayias, A. Russell, and V. Zikas. Ouroboros genesis: Composable proof-of-stake blockchains with dynamic availability. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, pages 913–930, 2018.
6. C. Badertscher, P. Gazi, A. Kiayias, A. Russell, and V. Zikas. Consensus redux: distributed ledgers in the face of adversarial supremacy. Technical report, Cryptology ePrint Archive, Report 2020/1021, 2020.
7. L. Bahack. Theoretical Bitcoin attacks with less than half of the computational power (draft). Cryptology ePrint Archive, Report 2013/868, 2013. `http://eprint.iacr.org/2013/868`.
8. M. Bellare and P. Rogaway. Random oracles are practical: A paradigm for designing efficient protocols. In *Proceedings of the 1st ACM conference on Computer and communications security*, pages 62–73. ACM, ACM, 1993.
9. E. Ben-Sasson, A. Chiesa, E. Tromer, and M. Virza. Succinct non-interactive arguments for a von neumann architecture. Cryptology ePrint Archive, Report 2013/879, 2013. `http://eprint.iacr.org/2013/879`.
10. J. Benet. IPFS: Content addressed, versioned, p2p file system. *arXiv preprint arXiv:1407.3561*, 2014.
11. I. Bentov, R. Pass, and E. Shi. Snow white: Provably secure proofs of stake. *IACR Cryptology ePrint Archive*, 2016:919, 2016.
12. J. Bonneau, J. Clark, and S. Goldfeder. On bitcoin as a public randomness source. *IACR Cryptology ePrint Archive*, 2015:1015, 2015.
13. J. Bonneau, A. Miller, J. Clark, A. Narayanan, J. A. Kroll, and E. W. Felten. SoK: Research Perspectives and Challenges for Bitcoin and Cryptocurrencies. In *Security and Privacy (SP), 2015 IEEE Symposium on*, pages 104–121. IEEE, 2015.
14. V. Buterin et al. A next-generation smart contract and decentralized application platform. *white paper*, 2014.
15. J. H. Cheon, K. Han, C. Lee, H. Ryu, and D. Stehlé. Cryptanalysis of the multilinear map over the integers. pages 3–12, 2015.
16. J. H. Cheon, J. Jeong, and C. Lee. An algorithm for NTRU problems and cryptanalysis of the GGH multilinear map without a low level encoding of zero. Cryptology ePrint Archive, Report 2016/139, 2016. `http://eprint.iacr.org/2016/139`.
17. J. H. Cheon, C. Lee, and H. Ryu. Cryptanalysis of the new CLT multilinear maps. Cryptology ePrint Archive, Report 2015/934, 2015. `http://eprint.iacr.org/2015/934`.
18. J.-S. Coron, C. Gentry, S. Halevi, T. Lepoint, H. K. Maji, E. Miles, M. Raykova, A. Sahai, and M. Tibouchi. Zeroizing without low-level zeroes: New MMAP attacks and their limitations. pages 247–266, 2015.

19. J.-S. Coron, M. S. Lee, T. Lepoint, and M. Tibouchi. Cryptanalysis of GGH15 multilinear maps. pages 607–628, 2016.
20. J.-S. Coron, M. S. Lee, T. Lepoint, and M. Tibouchi. Zeroizing attacks on indistinguishability obfuscation over CLT13. pages 41–58, 2017.
21. J.-S. Coron, T. Lepoint, and M. Tibouchi. Practical multilinear maps over the integers. pages 476–493, 2013.
22. J.-S. Coron, T. Lepoint, and M. Tibouchi. New multilinear maps over the integers. pages 267–286, 2015.
23. P. Daian, S. Goldfeder, T. Kell, Y. Li, X. Zhao, I. Bentov, L. Breidenbach, and A. Juels. Flash boys 2.0: Frontrunning in decentralized exchanges, miner extractable value, and consensus instability. In *2020 IEEE Symposium on Security and Privacy (SP)*, pages 910–927. IEEE, 2020.
24. B. David, P. Gaži, A. Kiayias, and A. Russell. Ouroboros praos: An adaptively-secure, semi-synchronous proof-of-stake protocol. Cryptology ePrint Archive, Report 2017/573, 2017. `http://eprint.iacr.org/2017/573`. To appear at EUROCRYPT 2018.
25. C. Dwork and M. Naor. Pricing via processing or combatting junk mail. In *Annual International Cryptology Conference*, pages 139–147. Springer, Springer, 1992.
26. I. Eyal and E. G. Sirer. Majority is not enough: Bitcoin mining is vulnerable. In *International Conference on Financial Cryptography and Data Security*, pages 436–454. Springer, 2014.
27. J. Garay, A. Kiayias, and N. Leonardos. The bitcoin backbone protocol: Analysis and applications (revised 2019). Cryptology ePrint Archive, Report 2014/765, 2014. `https://eprint.iacr.org/2014/765`.
28. J. A. Garay, A. Kiayias, and N. Leonardos. The bitcoin backbone protocol: Analysis and applications. In E. Oswald and M. Fischlin, editors, *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, volume 9057 of *LNCS*, pages 281–310. Springer, Apr 2015.
29. J. A. Garay, A. Kiayias, and N. Leonardos. The bitcoin backbone protocol with chains of variable difficulty. In J. Katz and H. Shacham, editors, *Annual International Cryptology Conference*, volume 10401 of *LNCS*, pages 291–323. Springer, Aug 2017.
30. S. Garg, C. Gentry, and S. Halevi. Candidate multilinear maps from ideal lattices. pages 1–17, 2013.
31. S. Garg, C. Gentry, A. Sahai, and B. Waters. Witness encryption and its applications. pages 467–476, 2013.
32. C. Gentry, S. Gorbunov, and S. Halevi. Graph-induced multilinear maps from lattices. pages 498–527, 2015.
33. I. Homoliak, D. Breitenbacher, O. Hujnak, P. Hartel, A. Binder, and P. Szalachowski. SmartOTPs: An air-gapped 2-factor authentication for smart-contract wallets. In *Proceedings of the 2nd ACM Conference on Advances in Financial Technologies*, pages 145–162, 2020.
34. Y. Hu and H. Jia. Cryptanalysis of GGH map. pages 537–565, 2016.
35. K. Karantias. Sok: A taxonomy of cryptocurrency wallets. Technical report, IACR Cryptology ePrint Archive, 2020: 868, 2020.
36. K. Karantias, A. Kiayias, and D. Zindros. Smart contract derivatives. In *The 2nd International Conference on Mathematical Research for Blockchain Economy*. Springer Nature, 2020.
37. A. Kiayias, P. Gaži, and D. Zindros. Proof-of-stake sidechains. In *IEEE Symposium on Security and Privacy*. IEEE, IEEE, 2019.

38. A. Kiayias, E. Koutsoupias, M. Kyropoulou, and Y. Tselekounis. Blockchain mining games. In *Proceedings of the 2016 ACM Conference on Economics and Computation*, pages 365–382, 2016.
39. A. Kiayias, A. Russell, B. David, and R. Oliynykov. Ouroboros: A Provably Secure Proof-of-Stake Blockchain Protocol. In J. Katz and H. Shacham, editors, *Annual International Cryptology Conference*, volume 10401 of *LNCS*, pages 357–388. Springer, Springer, Aug 2017.
40. A. Langlois, D. Stehlé, and R. Steinfeld. GGHLite: More efficient multilinear maps from ideal lattices. pages 239–256, 2014.
41. J. Liu, T. Jager, S. A. Kakvi, and B. Warinschi. How to build time-lock encryption. *Designs, Codes and Cryptography*, 86(11):2549–2586, 2018.
42. F. Ma and M. Zhandry. The mmap strikes back: obfuscation and new multilinear maps immune to clt13 zeroizing attacks. In *Theory of Cryptography Conference*, pages 513–543. Springer, 2018.
43. R. C. Merkle. A digital signature based on a conventional encryption function. In *Conference on the Theory and Application of Cryptographic Techniques*, pages 369–378. Springer, 1987.
44. S. Micali. ALGORAND: the efficient and democratic ledger. *CoRR*, abs/1607.01341, 2016.
45. S. Micali, M. O. Rabin, and S. P. Vadhan. Verifiable random functions. pages 120–130, 1999.
46. A. Miller. IRC logs of #bitcoin-wizards on 2015-03-13 on FreeNode. `https://download.wpsoftware.net/bitcoin/wizards/2015-03-13.html`, 03 2015. Accessed: 2021-05-13.
47. B. Minaud and P.-A. Fouque. Cryptanalysis of the new multilinear map over the integers. Cryptology ePrint Archive, Report 2015/941, 2015. `http://eprint.iacr.org/2015/941`.
48. D. M'Raihi, S. Machani, M. Pei, and J. Rydell. Totp: Time-based one-time password algorithm. RFC 6238, RFC Editor, 5 2011.
49. D. M'Raihi, M. Bellare, F. Hoornaert, D. Naccache, and O. Ranen. Hotp: An hmac-based one-time password algorithm. RFC 4226, RFC Editor, 12 2005.
50. S. Nakamoto. Bitcoin: A peer-to-peer electronic cash system. 2008.
51. R. L. Rivest, A. Shamir, and D. A. Wagner. Time-lock puzzles and timed-release crypto. 1996.
52. M. Vasek, J. Bonneau, R. Castellucci, C. Keith, and T. Moore. The bitcoin brain drain: Examining the use and abuse of bitcoin brain wallets. In *International Conference on Financial Cryptography and Data Security*, pages 609–618. Springer, 2016.
53. G. Wood. Ethereum: A secure decentralised generalised transaction ledger. *Ethereum Project Yellow Paper*, 151:1–32, 2014.