Protecting Quantum Procrastinators with Signature Lifting
A Case Study in Cryptocurrencies

Or Sattath[1] and Shai Wyborski[2,1]

[1] Department of Computer Science, Ben-Gurion University of the Negev, Israel
sattath@bgu.ac.il
[2] School of Computer Science and Engineering, The Hebrew University of Jerusalem, Israel
shai.wyborski@mail.huji.ac.il

Abstract. Current solutions to quantum vulnerabilities of widely used cryptographic schemes involve migrating users to post-quantum schemes *before* quantum attacks become feasible. This work deals with protecting *quantum procrastinators*: users that failed to migrate to post-quantum cryptography in time.

To address this problem in the context of digital signatures, we introduce a technique called *signature lifting*, that allows us to lift a deployed pre-quantum signature scheme satisfying a certain property to a post-quantum signature scheme that uses the *same* keys. Informally, the said property is that a post-quantum one-way function is used "somewhere along the way" to derive the public-key from the secret-key. Our constructions of signature lifting relies heavily on the post-quantum digital signature scheme Picnic (Chase et al., CCS'17).

Our main case-study is cryptocurrencies, where this property holds in two scenarios: when the public-key is generated via a key-derivation function or when the public-key hash is posted instead of the public-key itself. We propose a modification, based on signature lifting, that can be applied in many cryptocurrencies for securely spending pre-quantum coins in presence of quantum adversaries. Our construction improves upon existing constructions in two major ways: it is not limited to pre-quantum coins whose ECDSA public-key has been kept secret (and in particular, it handles all coins that are stored in addresses generated by HD wallets), and it does not require access to post-quantum coins or using side payments to pay for posting the transaction.

Table of Contents

1 Introduction

The canonical solution for quantum attacks on widely used cryptographic schemes is to migrate users to post-quantum schemes before quantum attacks become feasible. The focus of this work is *quantum procrastinators*, namely, users who remain dependent on the security of pre-quantum cryptography in the era of scalable quantum computation.

There are many possible reasons for procrastinators to exist:

- Most notably, the tendency of individuals and organizations to procrastinate: some users would not act in a timely manner. Even after the vulnerability is identified and a solution is available, it might take a long time for organizations to adapt to changes due to security risks. For example, recall the WEP standard for securing local wireless networks. Even though the vulnerabilities of WEP to were identified and fixed by the succeeding WPA standard within months, it took about a decade for the majority of systems their systems due to critical vulnerabilities. Criminals were able to exploit these delays to steal millions of credit cards [Bar21, Section 6.2.1].
- An unexpected technological breakthrough, perhaps done in secret, could make quantum attacks faster than expected.
- Post-quantum technologies have not been battle-tested and scrutinized to a fraction of the extent widely used cryptographic schemes have, and are still rapidly evolving and improving in terms of security and performance, which incentivizes delaying adoption.

In this work, we focus on users who rely on pre-quantum signature schemes whose public-key is known to the adversary. To help such procrastinators, we introduce post-quantum *signature lifting*: a technique for lifting an already deployed pre-quantum signature scheme to a post-quantum signature scheme *with the same keys*, given that a post-quantum one-way function is applied "somewhere along the way" when computing the public-key from the secret-key. This property does not hold for commonly used signature schemes such as ECDSA or Schnorr signatures. However, there are several applications of digital signature that could be recast such that it does hold. The two scenarios we consider are a) when a hash of the public-key is provided rather than the public-key itself, and b) when the secret-key is generated using a key-derivation function.

As a case study, we consider the setting of cryptocurrencies. In this setting, coins are bound to public signature keys, and any attempt to spend them must be signed by the corresponding secret-key. To allow safely spending pre-quantum coins, we exploit the fact that, in many cryptocurrencies, the addresses of most coins correspond to either scenario a) or b) above.

1.1 Case Study: Cryptocurrency Procrastinators

Bitcoin, Ethereum, and most other cryptocurrencies rely on the security of signature schemes that are susceptible to quantum attacks, such as the ECDSA and Schnorr signature schemes. Even an unspent transaction (UTXO) whose public-key is kept secret, while only its hash is available on the blockchain, is susceptible to attacks while the transaction spending it is in the mempool [But13].

We consider the following question:

Question 1: how can cryptocurrencies maximize the number of coins that procrastinators could securely spend?

We say that a method to spend pre-quantum UTXOs is *quantum-cautious* if it is secure even in the presence of quantum adversaries. Current quantum-cautious spending methods [BM14,SIZ^{+}18,IKS19] allow cautiously spending pre-quantum UTXOs that are *hashed*, that is, whose public-key has not leaked (e.g., leakage can occur by using the same address for several UTXOs and only spending some of them). However, these methods have a considerable drawback: to pay a fee to the miner for including the transaction, the user must either use a side payment or have access to post-quantum UTXOs.

Our cautious spending methods provide two major improvements. First, they allow cautiously spending hashed pre-quantum UTXOs without requiring a side-payment or a post-quantum UTXO. Second, they extend the set of cautiously spendable UTXOs to include:

- UTXOs whose address was generated by an HD wallet, *even if* the public-key has leaked,
- UTXOs whose public-key has leaked *even if* it was not generated by a key-derivation function, as long as the adversary is unaware that the public-key was not generated by a key-derivation function, and
- UTXOs whose secret-key was *lost*, but the fact that it was lost was kept secret by the user.

However, to make the latter two types spendable, all procrastinators who own pre-quantum UTXOs whose public-keys have leaked must be *online* periodically (e.g., once a year) to notice attempts at stealing their coins.

We stress that a large majority of bitcoin arguably resides in UTXOs whose address is a derived public-key (or a hash thereof). In particular, all UTXOs generated by a Hierarchical Deterministic (HD) wallet [Wui13] fall under this category. As of 2022, all bitcoin.org recommended wallets are HD wallets, and have been HD wallets since as early as 2015[3]. According to an analysis by CryptoQuant [Cry22], as of September 2022, UTXOs generated during and after 2015 contain more than 16.5 million bitcoins.

Furthermore, in the newly deployed Bitcoin update Taproot [WNT20], the public-key of the owner of a UTXO is posted to the blockchain in the clear. Hence, current methods are unable to quantum-cautiously spend UTXOs generated by following the Taproot specification, whereas our methods can. Taproot was deployed in November 2021, and its adoption has been steadily increasing since. As of October 2022, about 1% of Bitcoin's transaction throughput is spent to Taproot addresses [Tra22].

Our methods can be deployed in either *restrictive* mode, in which some funds of honest procrastinators are effectively burned, or *permissive* mode, in which those funds are spendable but require a large *deposit* as well as a very long *challenge period* (during which anyone can *challenge* the transaction by posting a *fraud proof*). Hence, it is desirable to defer deploying these methods as much as reasonable. Furthermore, since the current state of affairs is that quantum computers are not an immediate concern, users will tend to procrastinate as they will only realize that quantum threats are relevant *after* their coins are stolen. Both considerations motivate the next question:

Question 2: how can we warn users *in advance* that quantum adversaries will emerge in the near future? Can we make this warning *in consensus*?

Towards this goal, we present *quantum canaries*, a mechanism for signaling the emergence of quantum adversaries in a way that could be read off the blockchain. Quantum canaries are puzzles designed to be infeasible for classical computers but solvable for quantum computers whose scale is significantly smaller than required to attack ECDSA signatures. We propose to use quantum canaries to determine the block from which restrictions on spending pre-quantum UTXOs will take effect.

1.2 Signature Lifting

A key ingredient in our proposed solutions is a novel technique we call *signature lifting*. While we apply this technique to cryptocurrencies, it is useful in various scenarios and therefore is of independent interest. We overview lifted signatures in the current section and defer a more formal treatment to Section 3.

Chase et al. [CDG+17] present *Picnic*: a construction that transforms any post-quantum one-way function f into a post-quantum signature scheme $Picnic(f)$. In the scheme $Picnic(f)$, the secret-key is generated by uniformly sampling a point x in the domain of f, and the corresponding public-key is $f(x)$.

Our contribution is the observation that in some cases, by instantiating the Picnic scheme with an appropriate function, we can *lift* an existing *pre*-quantum signature scheme to a *post*-quantum signature scheme *with the same keys.*

Loosely speaking, the required property is that a post-quantum one-way function is applied "somewhere along the way" when generating the public-key from the secret key.

One setting where this property is satisfied is when users post the hash of their public key instead of the public-key itself. This scenario is common in Bitcoin and other cryptocurrencies, where unspent coins

[3] As could be verified by tediously looking them up one by one.

correspond to *addresses*. The address is typically a public-key of a signature scheme, and spending the coins requires signing a message with the corresponding secret-key. Commonly, the address is not an ECDSA public-key, but a hash thereof, and the user only reveals the public-key when spending the coin.

As we explain in Section 3.2, we can interpret this use case as a slight modification of ECDSA: instead of using the ECDSA public-key pk as a public-key, the hash thereof $H(\mathsf{pk})$ is used; when signing a message, pk is attached to the ECDSA signature σ. Since we assume the hash function is post-quantum one-way, we can lift the resulting scheme to obtain a quantum-cautious method for spending pre-quantum UTXOs. However, this scheme is only secure assuming the user never published a message signed by the non-lifted scheme, as the secret-key of the lifted scheme is part of the signature of the non-lifted scheme.

In Section 3.4 we apply signature lifting to the scenario where a *key-derivation function* (see Section 2.6) is applied to a *seed* to derive a *master secret-key* for a hierarchical deterministic (HD) wallet (see Section 2.7).

The challenge in this setting is that the function mapping the master secret-key to the public-key also depends on a derivation path, and typically many different derivation paths are applied to the same master secret-key. We overcome this by only applying the Picnic transform to the key-derivation function, and providing the master secret-key and derivation path along with the Picnic signature. When verifying a signature, one verifies the Picnic signature (proving knowledge of the seed), and also verifies that the public-key can be derived from the master secret-key using the provided derivation path.

The resulting scheme provides a method for safely spending UTXOs whose addresses were generated by HD wallets, provided that the key-derivation function is post-quantum collision resistant (which does hold for all key-derivation functions we are aware of, including the one used in Bitcoin wallets, since all such construction apply a hash at some point), and that the underlying hash function used to derive addresses from the master secret-key is modeled as a quantum random oracle.

We provide a full security proof in Theorem 1. Loosely speaking, Proposition 3 implies that we need only consider three types of attacks:

- The adversary can create their own seed and master secret-key, and find a derivation path from their master secret-key to an existing public-key. We show that such an attack implies a collision in the hash function.
- The adversary manages to *extend* the derivation path. This is similar to the previous attack, except the derivation path in the signature is a *suffix* of the derivation path produced by the adversary. The analysis of this attack is similar to the previous one.
- In the third type of attack, the adversary *truncates* the given derivation path. That is, the derivation path they provide is a *suffix* of the derivation path in the signature. Given a derivation path and a master secret-key, it is trivial to create a master secret-key for a suffix (by only applying a part of the path), so the argument used in the previous forms of attack does not carry over. In order to prove this attack infeasible, we first prove in Section 3.3 that given an adversary that can forge a Picnic signature, it is possible to extract a preimage of the master secret-key with respect to the key-derivation function. We use this property to show that in this scenario it is also possible to recover a collision in the hash function.

1.3 Quantum Canaries

In the early 19th century, miners used caged canaries to notice leakages of deadly gasses that were otherwise hard to detect. The miners would go into the mine carrying a cage with a canary, and as long as the canary lived, they would assume that the air was safe to breathe. Using canaries over human beings had two advantages: loss of canary life was considered much preferable to loss of human life, and canaries were more sensitive to lethal gasses such as carbon monoxide, so the death of a canary signified the presence of hazardous gasses well before the human holding the cage would be exposed to a lethal dosage.

Inspired by this solution, in Section 4 we propose to set up *quantum canaries* on the blockchain. Quantum canaries are bounties locked away behind classically infeasible challenges devised to require quantum computers considerably smaller than those required to steal unsafe coins. The cost of paying the bounty (be it by raising funds from the community or by a hard-fork to mint bounty funds) is far smaller than the consequences of a full-blown quantum thief capable of stealing unsafe coins. Killing the canary (that is, claiming the bounty) requires a much smaller "dosage" of quantum capability. Most importantly, the canary would act as a global beacon providing a warning that quantum adversaries are about to emerge.

Adversaries capable of killing the canary and claiming the bounty might rather wait for their quantum technology to ripen sufficiently for stealing coins. However, while they wait, another adversary could emerge that would kill the canary, depriving the original adversary of *both* the bounty and the loot. This dynamic incentivizes quantum entities to claim the bounty. We provide an analysis of this dynamic in Section 4.5.

Since this act of ornithicide is recorded on the blockchain, policies could be implemented with respect to the well-being of the canary, such as "do not accept transactions spending naked UTXOs if the canary has been dead for at least ten thousand blocks."

Remark 1. While completing this work, we became aware that quantum canaries were already proposed by Justin Drake in the Ethereum research forum [Dra18] as a general method to protect the network from exploits discovered in widely used cryptographic primitives. We retain the discussion about canaries as it is useful for our overall solution, and since we expand upon the discussion in [Dra18] in two regards: in Section 4.2 we argue that canaries need to be as similar as possible to the scheme they are protecting, pointing out existing research particularly relevant to the special case where the canary is used to protect ECDSA signatures from quantum adversaries; in Section 4.5 we provide a game theoretic analysis of adversaries competing for the bounty furnished by the quantum canaries. We also borrowed the name "quantum canaries" from [Dra18] and adapted the text accordingly. We thank Andrew Miller for bringing this to our attention.

The idea of posting UTXOs that could be spent by solving a cryptographic challenge was already suggested and implemented by Peter Todd in [Tod13]. He created several UTXOs, each only spendable by providing a collision in a particular hash function, including SHA1, RIPEMD-160, SHA-256 and several compositions thereof. Of these UTXOs, only the first has been claimed so far.

1.4 Quantum-Cautious Spending

Quantum-cautious spending is any method for spending a pre-quantum UTXO in a way that is secure even in the presence of quantum adversaries.

Even though we try to be as general as possible, some of our results may be hard to adjust, for example, when considering cryptocurrencies with advanced smart-contract capabilities, such as Ethereum, which may have more complicated logic; or privacy-oriented cryptocurrencies, such as ZCash, where special attention is given to privacy and anonymity—aspects which we completely ignore in this work.

Types of UTXO*s.* We first identify several useful subsets of the UTXO set, defined by the information available to the owner of the UTXO and to an adversary trying to steal the coins in the UTXO. We always assume that the adversary has no access to the seed (see Section 2.6 and Section 2.7) used to generate the address of the UTXO (that is, we consider anyone holding the derivation seed an "owner" of the UTXO). We summarize the properties of these sets in Table 1.

- **Hashed** A UTXO whose secret-key is known to the owner and whose public-key is not known to the adversary.
- **Derived** A UTXO whose owner knows a derivation seed from which the signature keys were derived.
- **Naked** A non-derived UTXO whose public-key is known to the adversary, but the adversary does not know that the UTXO is non-derived.
- **Lost** A UTXO whose secret-key is not known to anyone, but whose public-key is known to the adversary, but the adversary does not know the user does not know the secret-key. Some of our methods allow owners of lost UTXOs to recover their funds (see Section 5.2.3).
- **Stealable** A UTXO whose derivation seed is not known to the owner (either because they lost it or because the signature keys were not derived from a key-derivation function), and whose public-key is known to the adversary as well as the fact that the owner does not have the derivation key. Such UTXOs are called stealable as they could be safely claimed by a quantum adversary if unrestrictive FawkesCoin

is implemented (see Section 5.2.2). If permissive FawkesCoin is implemented (see Section 5.2.3), these funds could also be stolen by *classical* adversaries.

- **Doomed** A UTXO whose secret-key and public-key are not known to anyone. We call such UTXO doomed as no one (including a quantum adversary) can spend these with any of our methods.

Remark 2. Note that most pairs of the two sets above are disjoint, with hashed and derived UTXOs being the only exception. If a UTXO is both hashed and derived, the user could treat it as any of the two when spending it.

		Hashed	Derived	Naked	Lost	Stealable	Doomed
Owner	Derivation Seed	✓✗	✓	✗	(✗)	✗	(✗)
	Secret-Key	✓	(✓)	✓	✗	✓✗	(✗)
	Public-Key	(✓)	(✓)	(✓)	✓	✓	✗
Adv.	Public-Key	✗	✓✗	✓	✓	✓	(✗)
	Non-derived/lost	✓✗	(✗)	✗	✗	✓	✓✗

Table 1: The different types of UTXOs are defined by the information available to the owner and an adversary. Each column represents a subset of the UTXO set, and each row represents a particular datum regarding this UTXO and whether it is known to the owner/adversary. The non-derived/lost row, in particular, represents a scenario where the adversary knows that one of the following holds: either the UTXO address was not generated by an HD wallet, or the owner lost access to the corresponding secret-key (note these scenarios are not mutually exclusive). ✓✗ indicates that the property described in the corresponding row is irrelevant to the type defined by the column (e.g., a UTXO whose public-key is known to the user but not known to the adversary is hashed regardless of whether it was derived or not), a value is parenthesized to indicate that it could be inferred from other rows in the same column (e.g., in a derived UTXO the owner knows the derivation seed, so they can derive the secret- and public-keys).

FawkesCoin (Section 5.2). The first method we propose is a variation of the FawkesCoin protocol of [BM14]. This is the first method that allows spending some pre-quantum UTXOs quantum cautiously, though it requires the user to have access to post-quantum UTXOs.

As was noted in [But13], spending hashed UTXOs directly is not quantum-cautious, as a quantum adversary listening to the mempool could recover the public-key from the transaction and use it to post a competing transaction spending the same UTXO. Stewart et al. [SIZ$^+$18] propose overcoming this by adopting the commit-wait-reveal approach of [BM14]: the user first *commits* a transaction, then *waits* a prescribed amount of blocks after it was posted to the blockchain, and finally *reveals* the transaction.

By requiring all users to follow this procedure, it becomes unfeasible for an adversary exposed to the revealed transaction to steal the UTXO, as they would have to complete a commit-wait-reveal cycle before the honest revealed transaction is included in the blockchain.

The main challenge in implementing this approach is to allow miners to collect fees in a way that prevents spam attacks and does not create many small unspendable UTXOs. The only preexisting secure methods solve this issue by expecting the user to pay the commitment fees by using either a post-quantum UTXO or a side payment. We adopt the former approach for the purpose of the current method.

We extend the FawkesCoin protocol by adding two new modes of operation, namely *unrestrictive* and *permissive* modes. These modes allow spending UTXOs that are not cautiously spendable using the original (restrictive) design of FawkesCoin. The two new modes of operation work similarly to optimistic rollups [Eth22], so we follow the terminology used therein.

- **Restrictive (Section 5.2.1)** This mode allows users to spend hashed UTXOs using the public-key, and derived UTXOs using the derivation seed. The advantage of this mode of operation is that it minimizes quantum loot. The disadvantage is that it can only support spending hashed and derived UTXOs (so implementing restrictive FawkesCoin as the *only* way to spend pre-quantum UTXOs will make all pre-quantum UTXOs which are neither hashed nor derived unspendable indefinitely).
- **Unrestrictive (Section 5.2.2)** This mode further allows users to spend naked UTXOs. However, to do so they are required to provide a *deposit* as large as the value of the UTXOs, and then the transaction goes into a long *challenge period* (in Section 5.2.6 we discuss the length of the waiting period and propose to set it to one year) during which the rightful owner can claim the coins in the UTXO *and* the deposit by posting a *fraud proof* showing that they hold the derivation seed. The purpose of the deposit is to make it risky to try stealing a UTXO not known to be stealable.
  The disadvantage of unrestrictive mode is that owners of derived UTXOs that are not hashed have to occasionally (e.g. at least once a year) scan the blockchain for attempts to spend their money.
- **Permissive (Section 5.2.3)** This mode further allows users to spend lost UTXOs. To achieve this, we allow users to claim naked UTXOs without signing the transaction. This allows recovering of lost UTXOs. Like in unrestrictive mode, we require a deposit for spending a naked UTXO, making it risky to try to claim a naked UTXO without knowing that it is lost. Permissive mode has the advantage that lost UTXOs are no longer quantum loot. This method allows spending UTXOs without presenting a signature, whereby implementing it requires a hard-fork.

Using restrictive mode minimizes the quantum loot available to a quantum adversary. One might argue that even though unrestrictive and permissive modes allow more honest users to spend their coins, they have a negative side-effect, namely, unrestrictive and permissive modes increase the available quantum loot over restrictive mode. However, this is not quite the case, since holders of derived UTXOs can falsely declare their UTXOs lost with the purpose of baiting attackers to place a deposit they could claim. We discuss this further in Section 5.2.4. Note that permissive mode could be modified to allow spending hashed UTXOs whose keys were lost. However, we advise against this approach as it would require *all* holders of pre-quantum UTXOs, including hashed UTXOs, to be actively cautious against attempts on their money. Also note that permissive mode could be implemented regardless of quantum adversaries as means for recovering lost funds, though this approach also has the drawback of requiring users to be online to maintain the safety of their money.

Lifted Spending (Section 5.3). *Lifted spending* is the first quantum-cautious spending method that does not require having access to any post-quantum UTXO or using a side-payment.

In lifted spending we use signature lifting (see Section 3) to sign pre-quantum UTXOs with post-quantum security. We use signature lifting in two ways; one intended to spend hashed UTXOs whereas the other intended to spend derived UTXOs (naturally, a UTXO which is both hashed and derived could be spent either way).

The disadvantage of lifted spending is that the signature sizes are prohibitively large. For this reason, we do not recommend implementing lifted spending directly (see Section 1.7).

Lifted FawkesCoin (Section 5.4). *Lifted FawkesCoin* is the first method to allow cautious spending without access to post-quantum UTXOs *while keeping the transaction size small.*

Lifted FawkesCoin combines FawkesCoin and lifted spendings and enjoys from the benefits of both. In this method, in order to spend a UTXO, the user creates a commitment like they would in FawkesCoin. They post the commitment to the mempool alongside a large lifted signature used as a *proof of ownership*. Unlike non-lifted FawkesCoin, when including the commitment to the blockchain, the miner also posts the UTXO in the clear. The UTXO is then considered *committed* and can not be spent before the commitment is resolved.

This protects miners from attackers attempting to spam the block space by committing to the same UTXO several times. We discuss this more in Section 5.2.1.

After the commitment has been posted, the spender has a limited time to post a reveal of the commitment to the blockchain. If they fail to do so, the miner can claim the entire UTXO by posting the proof of ownership and is thus protected from spamming attacks. Note that the proof of ownership is never posted to the blockchain if both parties are honest and follow the protocol. Hence, the transaction size is about the same size as current transactions spending *pre*-quantum UTXOs.

Note that a miner can commit a UTXO even without a proof of ownership, preventing the owner from spending it. To prevent this, we also set a time limit for the miner to post the proof of ownership, and require that the miner leaves a *miner's deposit* proportional to the value of the UTXO. The deposit will be awarded to the owner of the UTXO in case its commitment expires before the miner provides a proof of ownership. The deposit makes such delay attacks extremely costly for the miner, and the costs are given to the attacked user as reimbursement, effectively eliminating this drawback.

The disadvantage of Lifted FawkesCoin over FawkesCoin is that, is that miners are not likely to include transactions whose value is less than the fee required to post a proof of ownership, as they risk losing money in case the commitment is not revealed. Hence, the spendability threshold of this method remains as high as the cost of posting a lifted signature.

1.5 Properties and Comparison of Spending Methods

In this section, we compare various properties of our quantum-cautious spending methods. Table 2 summarizes the methods for quantum-cautious spending. We briefly explain the properties by which we compare them. We use Bitcoin as a concrete example and use its current state as a basis for comparison. In particular, we consider the number of confirmation blocks in the "current state" to be six blocks, which is the default number of blocks required by the standard Bitcoin wallet.

Cautiously Spendable. A pre-quantum UTXO is considered *cautiously-spendable* with respect to a given method if implementing this method allows spending it in a way that is not vulnerable to quantum attackers. We use ✗ to denote the subtle scenario where a UTXO can be stolen, but an attempt to do so requires the adversary to take a considerable monetary risk. This happens in unrestrictive and permissive FawkesCoin for naked and lost UTXOs. If an adversary claims such a UTXO, they will manage to steal the coins therein. However, if an adversary claims a non-hashed derived UTXO, the rightful owner of the UTXO would post a fraud-proof, and the adversary would lose her deposit. Hence, the owners of naked and lost UTXOs are protected by the adversary's inability to distinguish them from derived UTXOs.

Confirmation Blocks. The number of blocks a user must wait after a transaction has been posted to the network before the transaction is considered confirmed. We write the multiplicative factor by which the number of blocks increases compared with the current state (where we consider the current confirmation times in Bitcoin to be 6 blocks). In methods where spending a UTXO requires posting two messages to the blockchain (a commit and a reveal), we count the number of blocks since the first message.

Transaction Size Increase. The size of a transaction, compared with the size of a current transaction spending ECDSA signed UTXOs. Multiplicative notation $\times x$ means that the size of a transaction is x times that of a current transaction. Additive notation $+x$ means that spending several UTXOs is possible by adding a fixed amount of data larger than a current transaction by a factor of x. In particular, the amount of added data does not depend on the number of UTXOs spent.

Spendability Threshold. The minimal value a UTXO should have so it could be spent using this method without the user or the miner risking losing money. The values in the table represent the factor of increase compared to the current spendability threshold. In most methods, this is the same as the cost of spending the transaction (i.e., the transaction fee). However, in Lifted FawkesCoin, the spendability threshold is actually much higher. This follows since the UTXO must be valuable enough to cover the costs of posting a proof

	Cautiously Spendable				Confirmation Blocks								
	Hashed	Derived	Naked	Lost	Hashed/Derived	Naked	Lost	Transaction Size Increase	Spendability Threshold	Works Without PQ UTXOs	No Delay Attacks	Soft-Fork	Offline Users Not Risked
Current State	✗	✗	✗		×1			×1	×1	✓	✓		✓
FawkesCoin													
Restrictive	✓	✓			∼ ×10			∼ +10	∼ ×1	✗	✓	✓	✓
Unrestrictive	✓	✓	⍻		∼ ×10	∼ $\times 10^3$		∼ +10	∼ ×1	✗	✓	✓	✗
Permissive	✓	✓	⍻	⍻	∼ ×10	∼ $\times 10^3$	∼ $\times 10^3$	∼ +10	∼ ×1	✗	✓	✗	✗
Lifted Spending	✓	✓			×1			∼ $\times 10^3$	∼ $\times 10^3$	✓	✓	✗	✓
Lifted FawkesCoin	✓	✓			∼ ×10			∼ ×1	∼ $\times 10^3$	✓	⍻	✗	✓

Table 2: A comparison of the quantum-cautious spending methods with the current state as a baseline. Each row represents a method, and the columns represent different properties of the methods reviewed below the table. Each method is compared to the one preceding it except if there is an arrow designating otherwise. Green circles and red boxes designate what we consider the most significant advantages and disadvantages respectively of each method with respect to the method it is being compared to. Cells that are not applicable (e.g., using the current state to spend a lost UTXO) are left blank. The meaning of each column is discussed below. The ⍻ symbol indicates that the property corresponding to the column is partially satisfied in a manner explicated in the description of the relevant column. The table headers and cells hyperlink to relevant sections in the text.

of ownership in case the user fails to reveal the transaction. In methods where many UTXOs may be spent at once (e.g., in non-Lifted FawkesCoin, one can post *many* commitments in the payload of a *single* post-quantum UTXO), we consider the threshold in the "many UTXOs" limit. We stress that spendability threshold is not the same as dust (see p. 17): dust refers to UTXOs whose value is in the same order of magnitude as it would cost to spend them, *regardless* of what method is used; on the other hand, a UTXO could be valuable enough to be above the spendability threshold of one method while being below the spendability threshold of another method.

Works Without Post-Quantum UTXO*s.* Some of our methods require the user to have a post-quantum UTXO (namely, a UTXO whose address is associated with a post-quantum signature public key), for paying fees or deposits. Such methods are marked with ✗, while methods that work even without a post-quantum UTXO are marked with ✓.

No Delay Attacks. A *delay attack* is a way to make an arbitrary UTXO unspendable for a fixed but meaningful period of time (say, a few hours) at a low cost (say, the cost of posting a transaction on the blockchain). We do not consider methods that allow cheap delay attacks suitable. Hence, no row in Table 2 has ✗ in this column. Lifted FawkesCoin is the only method marked with ⍻ to indicate that in this method, delay attacks are possible but are automatically detected, and the subject of the attack is compensated by the perpetrator.

Soft-fork. Is true if the method can be implemented as a soft-fork, that is, in a way that does not require the entire network to upgrade their nodes (see p. 16).

Offline Users not Risked. Some methods require owners of derived UTXOs which are not hashed to monitor the network for attempts to spend their coins. We use ✓ in methods for which this is *not* the case.

1.6 Compatibility of Methods

The properties in Table 2 only hold when each method is implemented separately. Carelessly implementing several methods together might compromise their security. For example, allowing lifted spending and FawkesCoin simultaneously renders FawkesCoin insecure: a quantum attacker who sees the revealed FawkesCoin transaction in the mempool can try to front-run by creating a competing lifted signature transaction.

Another example is when using FawkesCoin and Lifted FawkesCoin together. Spending a derived UTXO in Lifted FawkesCoin requires revealing the master secret-key msk. An adversary listening to the mempool can use msk to derive parent-keys she could use to spend any UTXO in the wallet using FawkesCoin. Hence, if FawkesCoin and Lifted FawkesCoin are implemented together, FawkesCoin must be modified such that the seed from which msk was derived must be posted instead of the parent key, which has the disadvantage of requiring the user to commit to their *entire wallet* before they can start revealing it.

For that purpose, we propose *not* to implement lifted spending directly, and to segregate FawkesCoin and Lifted FawkesCoin into *epochs*, where in each epoch only one of the solutions is available. We further require users of Lifted FawkesCoin to complete a commit-wait-reveal cycle within the epoch.

However, as the example above shows, segregating the solutions into epochs is insufficient, and some more steps are required for safely combining them. We discuss this further in Section 5.5.

We stress that as long as any method that requires users to be online is implemented, this requirement holds even if other methods without this requirement are implemented.

1.7 Our Proposed Solution

The purpose of this section is to propose a concrete solution to questions 1 and 2 above by combining the methods described above.

We propose the following:

- Set up a quantum canary (see Section 4) with a bounty of 20,000 BTC. The bounty should be freshly minted for that purpose.
- Define the *quantum era* to the period starting 8,000 blocks (about two months) after the canary is killed.
- Once the quantum era starts:
  - Activate FawkesCoin (see Section 5.2) and Lifted FawkesCoin (see Section 5.4). Set up a rotation (see Section 1.6) such that the FawkesCoin epoch is 1,900 blocks long and the Lifted FawkesCoin epoch is 500 blocks long.
  - Directly spending pre-quantum UTXOs becomes prohibited. That is, FawkesCoin and Lifted FawkesCoin become the only methods for spending pre-quantum UTXOs.
  - The mode of operation for FawkesCoin is restrictive for any naked UTXO whose *address* was posted to the blockchain prior to 2013, and permissive for the rest of the UTXOs.

A core component in our solution is Lifted FawkesCoin, which requires a hard-fork (see p. 16). The hard-fork could be implemented at any stage and does not require waiting for the quantum era. A hard-fork is also required to implement permissive FawkesCoin, and makes it easier to set up the bounty for the canary (see Section 4.3)..

Once the quantum era commences, all naked UTXOs whose public-keys leaked before 2013 would be burned, unless they were spent before the quantum era. Hence, the number of blocks we should wait for after the canary has been killed before the quantum era should be sufficient for all such UTXOs to be migrated. On the other hand, we do not want to wait too long since longer waiting times increase the risk that the quantum adversary will scale sufficiently to loot pre-quantum UTXOs before the quantum era starts. At the start of 2013, the *entire* UTXO set included about 4 million UTXOs. We hence use 4 million as a rough upper bound on the number of UTXOs at risk of being burned. Assuming about half of the block space is spent on signatures (see Section 2.5), spending 4 million UTXOs should require about 800 blocks. However, we should not spend the waiting time too close to 800 blocks since 1. we should account for a possible increase in the number of UTXOs whose public-keys leaked before 2013, and 2. it is plausible that once the canary is killed, there will be a rush to spend pre-quantum UTXOs (including those who are not at the risk of burning) to post-quantum addresses, and there should be sufficient leeway for owners of soon-to-be-burned UTXOs to spend them in this scenario. We thus suggest a waiting period of 8,000 blocks (or about two months) as, on the one hand, it seems unreasonable that quantum computers will increase in scale during this period, and on the other, it gives owners of soon-to-be-burned UTXOs ample time to spend their transactions.

The 1,900 block epoch for FawkesCoin gives a 1,800 block window for posting commitments. Commitments can not be posted during the last 100 blocks, as this will not leave enough blocks in the epoch to complete the required waiting time.

For similar reasons, the last 300 blocks of a Lifted FawkesCoin epoch could not contain reveals since we have a 100 blocks wait period, 100 blocks during which the user can post a reveal, and 100 blocks during which the miner can post a proof of ownership in case the user failed to post a reveal. Thus, a 500 blocks Lifted FawkesCoin period gives a 200 block window for posting commitments.

We propose implementing FawkesCoin along with Lifted FawkesCoin, with the intention that a user with no access to post-quantum UTXOs could use Lifted FawkesCoin initially to migrate pre-quantum UTXOs to post-quantum UTXOs, and then use these post-quantum UTXOs to pay the FawkesCoin fees in future transactions. We thus expect that users will gradually shift from using Lifted FawkesCoin to using non-Lifted FawkesCoin.

We do not recommend enabling permissive mode before the quantum era since that would require users to be online (see Section 5.2.2). We stress that permissive (or even unrestrictive) mode should not be used at all for UTXOs prior to 2013 since HD wallets were only introduced in 2013, so *all* naked UTXOs prior to 2013 are known to be non-derived and are thus stealable.

Enabling unrestrictive mode in the quantum era for UTXOs made after 2013 is desirable, as it increases the set of spendable pre-quantum UTXOs while not increasing the available quantum loot (see Section 5.2.4). Since unrestrictive mode already requires users to be online and Lifted FawkesCoin already requires a hard-fork, there is no downside to enabling permissive mode.

Note that we do *not* recommend directly implementing lifted spending, as this method does not provide a meaningful benefit over Lifted FawkesCoin, and is incompatible with non-Lifted FawkesCoin (see Section 1.6).

Remark 3. For concreteness, our proposal contains some "magic numbers" such as the length of the delay period for FawkesCoin, the funding of the canary, the length of the dispute period, the delay between claiming the canary and commencing the quantum era, etc. We have done our best to pick reasonable values and justify their magnitude (based on the magic numbers used in Bitcoin, estimations of the size of the UTXO set and post-quantum signatures, and so on). We note that all these values could be easily tuned, as they only mildly affect the functionality and security of the network.

1.8 Related Work

The loot currently waiting for a quantum adversary on the Bitcoin network was explored in [IKK20].

The observation that secure signature schemes could be constructed from a one-way function (not necessarily a cryptographic hash function) given reliable time-stamping (or append-only log) was first made by Anderson et al. [ABC+98], who used it to construct *Fawkes signatures*. A crucial property of this construction is that the signature only requires a single preimage as a key and a single image as a signature, unlike Lamport signatures and similar hash-based one-time signatures whose signatures are typically much larger.

Bonneau and Miller [BM14] observe that the blockchain itself could be used as a reliable time-stamping service (at least for blocks sufficiently temporally separated). Upon this observation, they construct the *FawkesCoin* ledger protocol, which uses the Fawkes signatures of [ABC+98] instead of ECDSA signatures.

Bonneau and Miller concede that the security properties of Bitcoin are preferable to their protocol (especially in the way it responds to chain forks) but argue that their protocol could be combined with Bitcoin to mitigate “a catastrophic algorithmic break of discrete log on the curve P-256 or rapid advances in quantum computing.” The authors note the problem of paying fees for transfer messages and suggest a few possible solutions. However, these solutions can only cautiously spend hashed UTXOs and do not allow using the spent UTXO to pay the transaction fees. We discuss this further in Section 5.4. Our *restrictive FawkesCoin* method (when further restricted to only support hashed UTXOs) is mostly based on Bonneau and Miller’s approach.

Applying the ideas of [BM14] to solve the problem of safely migrating funds from a pre-quantum to a post-quantum UTXO was first discussed in [SIZ+18,IKS19] (the authors thereof point out several threads from Twitter and the Bitcoin developer mailing list to which they attribute first suggesting this approach) by means of *key surrogacy*. They use the techniques of [BM14] to build a mechanism allowing a user to publicly attach to any existing pre-quantum public-key a post-quantum *surrogate key*, so that the network would expect UTXOs signed with the pre-quantum key to pass verification with respect to the post-quantum key. This approach achieves some minor improvements upon FawkesCoin but still only allows spending hashed UTXOs and fails to address the problem of paying the fees from the pre-quantum UTXO.

Coladangelo and Sattath [CS20, Section 4.1] originally proposed the idea of allowing users to claim lost coins by leaving a deposit, which could be claimed by the rightful owner in case of a theft attempt. We use the same approach in unrestrictive and permissive FawkesCoin (see Section 5.2).

The technique of using arbitrary post-quantum one-way functions to construct post-quantum digital signatures was presented in [CDG+17], we review this work in more depth Section 3. A follow-up work [KZ20] reduces the signature size considerably.

The canary introduced in Section 1.3 has some shared features with bug bounties. Breidenbach et al. [BDTJ18] introduces *Hydra*, a systematic approach for bug bounties for smart-contracts. Their approach is tailored to finding implementation bugs in smart contracts and, therefore, cannot be used in our setting. Ref. [RMD+21] presents and analyzes a framework for prediction markets, which can be used to estimate the future risk of an attack on ECDSA, which would break a cryptocurrency. The motivation and goals for their prediction market are far removed from the quantum canary discussed in this work.

1.9 Acknowledgements

We would like to thank Andrew Miller for valuable discussions and, in particular for pointing out the existence of quantum canaries, and Ori Newman and Elichai Turkel for their invaluable feedback and discussion.

This work was supported by the Israel Science Foundation (ISF) grant No. 682/18 and 2137/19, and by the Cyber Security Research Center at Ben-Gurion University.

This work has received funding from the European Research Council (ERC) under the European Union’s Horizon 2020 research and innovation programme (grant agreement No 756482).

2 Preliminaries and Definitions

2.1 Digital Signatures

A *digital signature scheme* is a cryptographic primitive that allows users to sign messages such that only they can sign the message, but anyone can verify the authenticity of the message. In digital signature schemes, the signer creates a *secret signature key* that can be used to sign messages and a *public-key* that can be used to verify these messages.

2.1.1 Security of Digital Signatures The security notion we consider is *existential unforgeability under chosen message attack* (EUF-CMA). In this notion, the adversary is given access to the public-key as well as access to a *signing oracle* which she could use to sign any message she wants. Her task is to produce a signature for any message she did not use the oracle to sign. A scheme is EUF-CMA *secure* if an efficient adversary cannot achieve this task with more than negligible probability (for a more formal treatment of the security of digital signatures we refer the reader to [Gol04,KL14]).

Extending the security notions of digital signatures to accommodate general quantum adversaries is not quite straightforward. The main difficulty is in the setting where the adversary has access to sign messages of their own, and they can sign a *superposition* of messages. The established classical notions of security do not generalize directly to this setting, since the notion of "a message she did not use the oracle to sign" becomes ill-defined when discussing superimposed queries.

However, in the setting of cryptocurrencies, the signature oracle reflects the adversary's ability to read transactions signed by the same public-key off the blockchain. This ability is captured even when assuming that the adversary, albeit quantum, may only ask the oracle to sign classical messages. A scheme that remains secure against quantum adversaries with classical oracle access is called *post-quantum* EUF-CMA secure. For brevity, we use the term EUF-CMA security to mean post-quantum EUF-CMA security, unless stated otherwise.

2.1.2 Elliptic Curve based Signatures The ECDSA and Schnorr signature schemes are based on a particular mathematical object called an *elliptic curve*. Bitcoin uses the ECDSA signature scheme instantiated with the secp256k1 curve, which is considered to admit 128-bit security against *classical* attackers [Bit22a]. The Taproot update [WNT20] replaces ECDSA with Schnorr signatures instantiated with the same secp256k1 curve, which is also considered to admit 128-bit security [WNR20].

While the way signatures are produced and verified in both these schemes is beyond the scope of the current work, we do require some understanding of how keys are generated. Fortunately, the key generation procedure is the same for ECDSA and Schnorr signatures (provided they were instantiated with the same curve and the same basis point).

Given an elliptic curve, one can give the element of the curve the structure of an abelian group known as the *elliptic curve group*. Recall that given any group $\varGamma$, we can define for any element $G \in \varGamma$ and any natural number n the group element $n \cdot G$ by defining $0 \cdot G = 0_\varGamma$ (where $0_\varGamma$ is the identity element of $\varGamma$) and $n \cdot G = (n-1) \cdot G + G$. We let $o(\varGamma)$ be the order of the group and say that G is a *generator* of $\varGamma$ if $\{n \cdot G \mid n = 0, \ldots, o(\varGamma)\} = \varGamma$. The secp256k1 curve has the property that it has a prime order, whereby *any* element besides the identity is a generator.

Instantiating a signature scheme requires specifying not only an elliptic curve group $\varGamma$, but also a fixed generator G of the group. Given the curve and generator, the secret-key is a uniformly random number $\mathsf{sk} \leftarrow \{0, \ldots, o(\varGamma)\}$ and the matching public-key is $\mathsf{pk} = \mathsf{sk} \cdot G$. As we further discuss in Section 4.2, the considerations behind choosing the curve and generator are highly involved, and are relevant both to the security and the efficiency of the resulting scheme.

To recover the public-key from a secret-key, one needs to be able to compute sk from G and $\mathsf{pk} = \mathsf{sk} \cdot G$. Differently stated, one needs to compute the *discrete logarithm* $\log_G(\mathsf{pk})$. The *hardness of discrete logarithm* is the assumption that solving such equations is infeasible in the average case. As we'll shortly review, quantum computers disobey this assumption, which is the root of the problem at hand.

In practice, elements of ECDSA are encoded into binary strings, for a group element $H \in \varGamma$, let $\widetilde{H}$ denote its binary representation. We define the function $\mathsf{PK}^{\mathsf{EC}}(\mathsf{sk}) = \widetilde{\mathsf{sk} \cdot G}$, and note that it is an injective function with domain $\{0, \ldots, o(\varGamma)\}$. Furthermore, $\mathsf{PK}^{\mathsf{EC}}$ is a *group homomorphism* from $\mathbb{Z}_{o(\varGamma)}$ to $\varGamma$: $\mathsf{PK}^{\mathsf{EC}}(k+l) = (k+l) \cdot G = k \cdot G + l \cdot G = \mathsf{PK}^{\mathsf{EC}}(k) + \mathsf{PK}^{\mathsf{EC}}(l)$.

In practice the secret key sk is not sampled uniformly from $\{0, \ldots, o(\varGamma)\}$ but from $\{0,1\}^{\log_2 \lceil o(\varGamma) \rceil}$ (that is from strings of the minimal length required so that each possible number in $0, \ldots, o(\varGamma)$ has a unique representation. However, the elliptic curve group is typically chosen such that almost all keys have a unique representation. For example, in the secp256k1 curve sk is given as a string of 256 bits whereas there are more than $2^{256} - 2^{64}$ group elements, so the fraction of elements of $\mathbb{Z}_{o(g)}$ which admit two representations is smaller than 2^{-192}. In practice, this is overlooked and a key is generated by choosing a uniformly random string. We also overlook this detail and assume that $\mathsf{PK}^{\mathsf{EC}}$ is injective.

The binary representation $\tilde{H}$ of a curve point H has several different formats with different properties. In practice in general, and Bitcoin in particular, different formats appear in different contexts. Most of the current work work is agnostic to how the string is actually formatted, and treat $\mathsf{PK}^{\mathsf{EC}}$ as a bijection onto its domain. The only place where we appeal to the format is in Section 2.7, where we use the fact that in HD wallets following the specification of BIP-32 [Wui13], sk is encoded as a 32 byte string whereas pk is encoded as a 33 byte string (and in particular it is impossible that $\mathsf{sk} = \mathsf{PK}^{\mathsf{EC}}(\mathsf{sk})$).

2.1.3 Quantum Attacks

Shor's algorithm [Sho94] was the first to break previously considered unbreakable cryptographic schemes by efficiently factoring large numbers. Shor's techniques were generalized in [BL95] to solve the discrete logarithm problem in arbitrary groups, including elliptic groups, thus proving that ECDSA and Schnorr signatures are *not* post-quantum (i.e., the number of bits of security of these schemes over any group is polylogarithmic in the number of bits required to describe an arbitrary element of the group). We refer the reader to [LK21] for a fairly comprehensive survey of the state-of-the-art algorithms optimized to solve the discrete logarithm problem in several types of elliptic curve groups[4].

When designing quantum canaries (see Section 4), we are concerned with curves similar to the secp256k1 curve used in Bitcoin. The secp256k1 curve is over a prime field and is given in Weierstrass form. Fortunately, quantum algorithms for solving discrete logarithms over prime curves given in Weierstrass form were analyzed by several authors. The first concrete cryptanalysis is given by Roetteler et al. [RNSL17], which provide an explicit algorithm optimized to minimize the required number of logical qubits. The algorithm in [RNSL17] is vastly improved by Häner et al. [HJN+20], who provide a more efficient algorithm and also consider space-time trade-offs. They provide algorithms minimizing either the number of logical qubits, the number of T gates, or the depth of the circuit. Unfortunately, [HJN+20] do not analyze any of these metrics as a function of the size of the field, but rather compute them for several known curves and compare the result with [RNSL17].

2.1.4 Post-Quantum Signature Scehems

To address the quantum vulnerability of contemporary signature schemes, the United States National Institute of Standards and Technology (NIST) invited protocol designers to submit post-quantum signature schemes. In July 2022, NIST announced that three post-quantum signatures will be standardized: CRYSTALS-Dilithium [DKL+18], FALCON [FHK+18], and SPHINCS+ [BHK+19].

A survey of the chosen schemes and their performance is available in [RWC+21]. The blog posts [Wes21,TC22,Wig22] provide a more approachable (yet less formal) survey of the state-of-the-art post-quantum signature schemes in general, as well as the NIST curves.

2.2 The Random Oracle Model

When considering the security of constructions which involve explicit hash functions such as SHA-256 or SHA-512, it is difficult to make formal arguments about their security. A common way to overcome this is using the *random oracle model*. The random oracle model assumes that there is an oracle H accessible to anyone, such that $\mathsf{H}(x) = f_{|x|}(x)$ where for each n the function f_n was uniformly sampled from the set of functions from $\{0,1\}^n$ to $\{0,1\}^\ell$ (where ℓ is fixed). The *quantum* random oracle model further assumes that users have access to the unitary $|x, y\rangle \mapsto |x, y \oplus \mathsf{H}(x)\rangle$.

It is common practice in cryptography to analyze constructions assuming that random oracles are used instead of hash functions, and then instantiate them with hash functions considered secure for the application at hand. For a more extensive introduction to the random oracle model and the quantum random oracle model we refer the reader to [BS20, Section 8.10.2] and [BDF+11] respectively.

Random oracles have a particularly useful property that if applied to a "sufficiently random" distribution, the output is indistinguishable from uniformly random. The measure of randomness appropriate for our settings is that of *guessing probability*:

[4] The "type" of a curve is determined by the field it is defined above, the form in which the parametric equation is given, and the types of coordinates used

Definition 1 (Guessing probability). *Let $\mathcal{S}$ be a finite distribution, the* guessing probability *of $\mathcal{S}$ is $\gamma_{\mathcal{S}} \max_s \mathbb{P}[t = s \mid t \leftarrow \mathcal{S}]$.*

That is, the guessing probability is the probability of the most likely output of sampling from $\mathcal{S}$[5]. It is called the *guessing* probability as it is also the best probability with which an adversary can guess the outcome of sampling from $\mathcal{S}$, as one can prove that always guessing the most likely outcome is an optimal strategy.

Note that a distribution can have a very small guessing probability and still be easily distinguishable from uniformly random. For example, consider the distribution over (s, pw) where s is a uniformly random string of length n and pw is some fixed string of length k. Then $\gamma_{(s,pw)} = 2^{-n}$, though it is easily distinguishable from a random string of length $n + k$.

Proposition 1 ([BS20] Theorem 8.10, adapted). *Let $\mathcal{S}$ be a finite distribution. If H is modeled as a random oracle and $\gamma_{\mathcal{S}} = \mathsf{negl}(\lambda)$ then the distribution $\mathsf{H}(\mathcal{S})$ is computationally indistinguishable from a random distribution.*

Throughout this work, we use H to represent the hash used in the construction we analyzed (which is always either SHA-256 or SHA-512), and assume H is modeled as a random oracle. In practice, SHA-256 and SHA-512 are based on a construction called the Merkle–Damgård transform, which is known to be an unsuitable replacement for a random oracle in some particular settings. In Appendix A we expand on the properties of Merkle–Damgård and justify our modeling thereof as a random oracle.

2.3 Bitcoin and Blockchain

In this section, we overview some of the aspects of Bitcoin relevant to our discussion. We assume the reader is familiar with core concepts of Bitcoin such as transactions and UTXOs, chain reorganizations, etc. For a review of these concepts, we refer the reader to [NBF+16].

Coinbase Cooldown. In a reorg scenario, most transactions removed by the reorg can be reposted to the mempool, and any transaction posted by an honest user will eventually be included in the new chain. The only scenario where a transaction becomes invalid is if a dishonest user attempts a double spend by posting conflicting transactions to the two sides of the fork. However, this is no longer the case for reverted coinbase transactions, as their validity relies on the block that included them. Reverting a coinbase transaction renders it, and all transactions spending coins minted in that coinbase transaction, invalid. To avoid a scenario where a valid transaction made by an honest user becomes invalid, Bitcoin imposes a cooldown of one hundred blocks before the coinbase transaction can be spent (see the Bitcoin developer guide).

Software Forks. A *software fork* is a change to the code that nodes are expected to run that affects the conditions under which a block is considered valid. There are two types of software forks, a *soft-fork* and a *hard-fork*. The difference is that in a hard-fork, there exist blocks that the new version considers valid while the old version does not. In order to adopt a soft-fork, only the miners are required to update their nodes, and non-mining nodes will operate correctly even with the outdated version. A hard-fork, on the other hand, causes the chain to split into two separate chains that can not accept each other's blocks. Notable examples of hard-forks include Bitcoin Cash (forked from Bitcoin) and Ethereum Classic (forked from Ethereum).

We illustrate the two types of forks with two scenarios which occur in our methods:

- Paying the coin in a UTXO to a different address than the one specified in the transaction (e.g., paying the UTXO of a non-revealed commitment to a miner who posted a proof of ownership in Lifted FawkesCoin, see Section 5.4). This behavior can be implemented in a soft-fork by not spending to transaction to its intended address, but rather spending it to a new anyone-can-spend UTXO. Updated miners who enforce the rules of the protocol will know to consider attempts to spend the anyone-can-spend UTXO to any address but the one dictated by the protocol invalid. However, from the point of view of non-updated nodes, spending the UTXO to *any* address is valid.

[5] Readers familiar with the min-entropy function H_{min} might notice that $\gamma_{\mathcal{S}} = 2^{-H_{\min}(\mathcal{S})}$

- Spending a UTXO without including a valid signature (e.g., spending a lost UTXO in permissive FawkesCoin, see Section 5.2.3). This scenario can not be implemented in a soft-fork, as currently there is no way to spend a UTXO without including a signature.

Dust. UTXOs whose value is too small to cover the costs of spending them are called *Dust*. The difference between dust and unspendability is that a UTXO can be spendable with respect to one spending method but not the other. For a UTXO to be considered *dust*, it must not be spendable using *any* method available. Since the transaction fees fluctuate, the set of dust UTXOs changes in time.

Any changes to the operation of Bitcoin that increase the transaction sizes will also incur an increase in the amount of dust. In particular, current post-quantum signature schemes require much larger keys and/or signature sizes than the ECDSA scheme currently used in Bitcoin. Moreover, variations on the Bitcoin protocol can cause the dust threshold to increase beyond the transaction fee, as indeed happens in Lifted FawkesCoin (see Section 5.4).

2.4 Quantum Threats on Blockchains

Roughly speaking, quantum computers affect Bitcoin on two different fronts: quantum mining and attacks on pre-quantum cryptography.

A common misconception is that quantum computers have no drastic effects on Bitcoin mining (beyond increased difficulty due to Grover's quadratic speedup). This was debunked in [Sat20,LRS19]. Our work is orthogonal to aspects related to quantum mining, especially since quantum attacks on secp256k1 are projected to occur a few years before quantum mining starts [ABL$^+$18].

The most immediate risk is in the form of UTXOs with leaked public-keys. A quantum adversary could use the public-key to sign arbitrary messages and, in particular, spend any UTXO whose address is this public-key. The public-key can be exposed in a variety of ways, including (but not limited to):

- P2PK UTXOs: when Bitcoin just launched, all transactions would contain the ECDSA public-key of the recipient. Hence, all UTXOs in the UTXO set would contain the public-key used to verify transactions spending them. That is, *none* of the UTXOs were hashed. In 2009, in order to conserve space, P2PKH transactions which only contain a hash of the public-key were introduced. P2PKH soon became the standard, and as of today, no P2PK UTXOs are created. However, the UTXO set still contains a few old P2PK UTXOs whose balance totals about 2 million bitcoin [Del22].
- Reused addresses: while considered bad practice, reusing the same address for several UTXOs is a very common habit. A reused address in itself is not quantumly compromised, but once such a UTXO is spent, the remaining UTXOs with the same address become compromised. A survey by Deloitte estimates [Del22] that at least 2 million bitcoins are stored in reused UTXOs whose key has been exposed this way. A survey by BitMex [Bit22b] estimates that about half of the Bitcoin transaction throughput is to reused addresses[6].
- Taproot: the Taproot update to Bitcoin [WNT20] uses a new form of UTXO called P2TR in which the public-key itself is posted like in P2PK UTXOs. Taproot was created with the intention to become the new standard and replace other spending methods, which might greatly increase the amount of leaked public-keys. Since its deployment in November 2021 the adoption of Taproot has been steadily increasing and as of October 2022, about 1% of newly created UTXOs are P2TR [Tra22].
- Software forks: when a chain forks into two independent chains (such as Bitcoin and Bitcoin Cash, or Ethereum and Ethereum classic), any UTXO from before the splitting point coexists on both chains. Thus, in order to spend money on one of the chains, the owner must expose her key, which could be used by an attacker to spend the owner's corresponding UTXO on the other chain. See [IKK20] for an analysis of address reuse across Bitcoin and Bitcoin Cash.

[6] It should be noted that reusing addresses poses security risks beyond quantum vulnerabilities – particularly compromising the anonymity of users – and is generally considered a bad security practice.

2.5 Using Post-Quantum Signatures on the Blockchain

The most straightforward way to address quantum threats is to introduce post-quantum signatures. However, this requires that all users migrate their UTXOs to post-quantum UTXOs *before* a quantum adversary emerges.

According to bitcoinvisuals.com, the average Bitcoin transaction in the last two years is about 600 bytes long and contains three inputs. Each such input contains a compressed ECDSA signature and an ECDSA public key whose combined length is at most 103 bytes. Hence, at full capacity, about half of the block space is used for signature data.

As of the time of writing, there are about 85 million UTXOs in the Bitcoin UTXO set [Blo22]. Assuming a block size of one megabyte and a block delay of 10 minutes, it follows that migrating every single UTXO to a post-quantum address would require about five months. Though, even if it was possible to rally all users to migrate their UTXOs, a quantum adversary could still loot lost UTXOs.

Another crucial aspect of post-quantum signature schemes is their increased storage and computational requirements compared to current signatures. The most immediate consequence is the increase in public-key and signature lengths, which directly decreases the throughput of Bitcoin.

The most space-efficient post-quantum signature endorsed by NIST is the Falcon512 scheme, whose combined public-key and signature size is 1532 bytes, which is about 15 times larger than the ECDSA equivalent, making the average transaction about eight times larger. Crystal-DILITHIUM is the second-best scheme in terms of combined public-key and signature storage, with a combined length of 2740 bits, inflating the size of an average transaction by a factor of about 14.

2.6 Key-Derivation functions

A *password-based key-derivation function* (PBKDF) is a function used to derive a secret-key for a cryptographic application from a string that is not necessarily uniformly random (e.g. the key also contains a user selected password, without assuming anything about the distribution the password was sampled from) called the *seed*. For a formal introduction we refer the reader to [BS20, Section 8.10].

The property we require from a PBKDF is that the resulting key is indistinguishable from random, provided that the input seed was sampled from a distribution with sufficient guessing probability (recall Definition 1). Proposition 1 establishes that a random oracle is a PBKDF.

Most Bitcoin wallets use a particular PBKDF specified in [PRVB13], where the seed is comprised of a user-chosen password and a uniformly random binary string (encoded in the form of a *mnemonic phrase* for the sake of human readability). The PBKDF therein serves another purpose: to provide some protection for the user from an adversary which has access to the binary string but not to the password (as the string is usually stored inside the wallet, but the user is required to input the password with each use). This is achieved by making the PBKDF computationally heavy to make dictionary attacks on the password less feasible. Towards this end, the PBKDF described therein applies 2048 iterations of $\mathsf{SHA\text{-}512}$ to the input.

2.7 Hierarchical Deterministic wallets

Bitcoin HD wallets [Wui13] allow a user to only store one *master secret-key* from which many ECDSA key-pairs can be derived. The master secret-key itself is usually derived using a PBKDF (see Section 2.6).

All keys derived by the wallet are actually *extended keys*. An extended secret-key is of the form $\mathsf{xsk} = (\mathsf{sk}, c)$ where sk is an ECDSA secret-key and c is a pseudo-random string. The corresponding public-key is $\mathsf{xpk} = (\mathsf{pk}, c)$ where $\mathsf{pk} = \mathsf{sk} \cdot G$ is the corresponding public-key and c is the same string (note that this key-derivation is appropriate for EC-Schnorr as well, as its key generation procedure is identical to ECDSA).

Given xsk one can derive its ith *child key*. Child keys come in two flavors: *hardened* and *non-hardened*. We let $\mathsf{xsk}_{i,h}$ and $\mathsf{xsk}_{i,nh}$ denote the hardened and non-hardened ith children of xsk respectively, and $\mathsf{xpk}_{i,h}$ and $\mathsf{xpk}_{i,nh}$ denote their corresponding extended-public keys. We will soon describe two *derivation procedures* H^h and H^{nh} such that $\mathsf{xsk}_{(i,p)} = \mathsf{H}^p(\mathsf{xsk}, i)$. Once defined, it is straightforward to check that non-hardened public-keys have the advantage that they can be derived from the master public-key (or any other public-key along the derivation path) without knowledge of the secret master-key, but non-hardened secret-keys have the disadvantage that they could be used to recover the master secret-key from the master public-key. Hence, hardened and non-hardened keys have different use cases.

These procedures are defined such that it is possible to compute $\mathsf{xpk}_{i,nh}$ from xpk without knowing xsk, but a classical adversary cannot feasibly compute $\mathsf{xpk}_{i,h}$ from xpk. Non-hardened keys have

We call a pair $s = (i,p)$ where $p \in \{h, nh\}$ a *derivation step*. A *derivation path* P is a sequence of derivation steps. We recursively define $\mathsf{xsk}_\emptyset = \mathsf{msk}$ and $\mathsf{xsk}_{P,(i,p)} = \mathsf{H}^p(\mathsf{xsk}_P, i)$.

Given a parent extended secret-key, $\mathsf{xsk} = (\mathsf{sk}, c)$ we derive the ith non-hardened child $\mathsf{xsk}_i = (\mathsf{sk}_i, c_i)$ the following way:

- Let H be collision resistant with output length 512
- Let H_L and H_R be the 256 first and 256 last bits of H respectively.
- Set $c_i = \mathsf{H}_R(c, \mathsf{pk}, i)$ and $\mathsf{sk}_i = \mathsf{H}_L(c, \mathsf{pk}, i) + \mathsf{sk} \mod N$ where $N = |\mathsf{secp256k1}|$.
- Set $\mathsf{pk}_i = \mathsf{PK}^{\mathsf{EC}}(\mathsf{sk}_i)$.

In the hardened version, sk is used instead of pk in the third step.

Differently stated, we define $\mathsf{H}^h(c,x,i) = (\mathsf{H}_L(c,x,i)+x, \mathsf{H}_R(c,x,i))$ and $\mathsf{H}^{nh}(c,x,i) = (\mathsf{H}_L(c, PK^{EC}(x), i) + x, \mathsf{H}_R(c, PK^{EC}(x), i))$ and get that $\mathsf{xsk}_i = \mathsf{H}^h(c,x,i)$ in the hardened case and $\mathsf{xsk}_i = \mathsf{H}^{nh}(c,x,i)$ in the non-hardened case. For each derivation step $s = (i,p)$ we define $\mathsf{H}^s(x) = \mathsf{H}^p(x,i)$. Finally, for each derivation path P we define H^P recursively as $\mathsf{H}^\emptyset(x) = x$ and $\mathsf{H}^{P,s}(x) = \mathsf{H}^s(\mathsf{H}^P(x))$.

Note that given $\mathsf{xpk} = (c, \mathsf{pk})$ one can compute $\mathsf{pk}_i^{nh} = \mathsf{PK}^{\mathsf{EC}}(\mathsf{H}_L(c,\mathsf{pk},i) + \mathsf{sk}) = \mathsf{PK}^{\mathsf{EC}}(\mathsf{H}_L(c,\mathsf{pk},i)) + \mathsf{PK}^{\mathsf{EC}}(\mathsf{sk}) = \mathsf{PK}^{\mathsf{EC}}(\mathsf{H}_L(c,\mathsf{pk},i)) + \mathsf{pk}$ (where we used the fact that $\mathsf{PK}^{\mathsf{EC}}$ is a group homomorphism). Also note that given sk_i^{nh} and $\mathsf{xpk} = (c,\mathsf{pk})$ one can compute $\mathsf{sk} = \mathsf{sk}_i - \mathsf{H}_L(c,\mathsf{pk},i)$, so it is not secure to provide an extended public-key along with a non-hardened non-extended child secret-key.

In practice, this construction is instantiated with $\mathsf{SHA\text{-}512}$ acting as H.

We will need the following:

Proposition 2. *If* H *is a random oracle and* s *is a derivation step then* H^s *is a random oracle.*

Proof. For any function H whose output is 512 bit let H_L (resp. H_R) be the function that only output the first (resp. last) 256 bits of H. Let $s = (i,p)$.

By assumption H is random, which implies that H_L is random. Since $\mathsf{PK}^{\mathsf{EC}}$ is injective, it follows that $(c,x,i) \mapsto \mathsf{H}_L(c, \mathsf{PK}^{\mathsf{EC}}(x), i) = \mathsf{H}_R^{nh}$ is also random. Finally, we note that if f is uniformly random and g is fixed then $f+g$ is uniformly random, from which it follows that H_R^{nh} is also random. It follows that H^{nh} is random as needed. The proof for H^h is similar.

We now note that $\mathsf{H}^{(i,p)}$ is simply $\mathsf{H}^p(\cdot, i)$, but the result of fixing a part of the input of a random function is also a random function (with a smaller domain), which completes the proof. □

From this follows by induction:

Corollary 1. *If* H *is a random oracle and* P *is a non-empty derivation path then* H^P *is collision-resistant.*

Finally we define the function $\mathsf{Der}(\mathsf{msk}, P) = \mathsf{H}^P(\mathsf{msk})$. The function Der is not collision-resistant: if P_1 and P_2 are two non-empty derivation paths, and $P = P_1 \| P_2$ is their concatenation, then it holds for any msk that $\mathsf{Der}(\mathsf{msk}, P) = \mathsf{Der}(\mathsf{Der}(\mathsf{msk}, P_1), P_2)$. However, these are the only forms of collision a bounded adversary could feasibly produce. This property will be useful to us in Section 3.4, so we define and prove it formally.

Definition 2. *Let* $f(x,y)$ *be a two variable function. A point* (x', y') *is an* f-suffix *of* (x,y) *if there exists* y'' *such that* $y = y'' \| y'$ *and* $x' = f(x, y'')$.

A pair of inputs $(x,y), (x',y')$ *is a* non-suffix collision *if* $f(x,y) = f(x',y')$ *but neither of the input is an* f*-suffix of the other.*

The function f *is* collision resistant up to suffixes *if it is infeasible to find a non-suffix collision.*

Proposition 3. *If* H *is a random oracle then* Der *is collision resistant up to suffixes.*

Proof. Let $\mathcal{A}$ be a QPT adversary which outputs a non-suffix collision $(\mathsf{msk}, P), (\mathsf{msk}', P')$ with probability ε. We prove that we can recover from $(\mathsf{msk}, P), (\mathsf{msk}', P')$ a collision in a function known to be a random-oracle, and in particular collision-resistant. From this will follow that $\varepsilon = \mathsf{negl}(\lambda)$, completing the proof.

In most cases, we will find a collision in H^Q for some non-empty path Q, which is a random oracle by Corollary 1. In the remaining case we will find a collision in H_L, which is a random oracle by the hypothesis that H is a random oracle.

If $P = P'$ we get that $\mathsf{msk} \neq \mathsf{msk}'$ is a collision for H^P. Assume that $P \neq P'$.

If there is a P'' such that $P = P''\|P'$, then by the assumption that $(\mathsf{msk}, P), (\mathsf{msk}', P')$ is non-suffix we have that $\mathsf{msk}' \neq \mathsf{H}^{P''}(\mathsf{msk})$. However it does hold that $\mathsf{H}^{P'}(\mathsf{msk}') = \mathsf{H}^{P'}(\mathsf{H}^{P''}(\mathsf{msk}))$, so we found a collision in $\mathsf{H}^{P'}$.

The remaining case is that neither P, P' is a suffix of the other. Let S be the longest shared suffix of P, P'. Let Q, Q' be paths such that, $P = Q\|S$ and $P' = Q'\|S$ (note that both Q and Q' are necessarily non-empty). If $\mathsf{H}^Q(\mathsf{msk}) \neq \mathsf{H}^{Q'}(\mathsf{msk}')$ then these points constitute a collision of H^S. Otherwise, let s and s' be the last steps in Q and Q' respectively, and let $\tilde{Q}$ and $\tilde{Q}'$ be Q and Q' with the last step removed. By the maximality of S we have that $s \neq s'$. However, we have that $\mathsf{H}^s(\mathsf{H}^{\tilde{Q}}(\mathsf{msk})) = \mathsf{H}^Q(\mathsf{msk}) = \mathsf{H}^{Q'}(\mathsf{msk}') = \mathsf{H}^{s'}(\mathsf{H}^{\tilde{Q}'}(\mathsf{msk}'))$. Let $s = (i, p)$ and $s' = (i', p')$. If $p = p'$ we get from $s \neq s'$ that $i \neq i'$, and since $\mathsf{H}^p(\mathsf{H}^{\tilde{Q}}(\mathsf{msk}), i) = \mathsf{H}^p(\mathsf{H}^{\tilde{Q}'}(\mathsf{msk}'), i')$ we found a collision in H^p. If $p \neq p'$ assume without loss that $p = h$ and $p' = nh$, and also set $\mathsf{H}^Q(\mathsf{msk}) = \mathsf{H}^{Q'}(\mathsf{msk}') = (\mathsf{sk}, c)$, then we have that $H_R(c, \mathsf{sk}, i) = H_R(c, \mathsf{PK}^{\mathsf{EC}}(\mathsf{sk}), i')$. However, recall that in [Wui13] secret-keys are encoded as 32 byte strings whereas public-keys are encoded as 33 byte strings (see Section 2.1.2), whereby $(c, \mathsf{sk}, i) \neq (c, \mathsf{PK}^{\mathsf{EC}}(\mathsf{sk}), i')$ so we found a collision in H_R. □

So far we have considered H to have fixed output length of 512 bits. The next property we require is stated more naturally when we think of H_ℓ as having output length 2ℓ and of ${\mathsf{PK}^{\mathsf{EC}}}_\ell$ as an arbitrary efficiently computable injection from $\{0,1\}^\ell$ to $\{0,1\}^{\ell+c}$ for some $c > 0$ (when instantiated with $\mathsf{H} = \mathsf{SHA\text{-}512}$ and $\mathsf{PK}^{\mathsf{EC}}$ specified in BIP-39 we have that $\ell = 256$ and $c = 8$). For a string $x \in \{0,1\}^{2\ell}$ we use x_L and x_R to denote its first and last ℓ bits respectively.

Proposition 4. *Let $\mathcal{A}$ be a QPT adversary with oracle access to H_ℓ that gets as input $x \in \{0,1\}^{2\ell}$ and outputs a path Q and a string x' of length at least ℓ. If H_ℓ is a random oracle, then*

$$\mathbb{P}[\mathsf{H}_\ell(x) = \mathsf{H}_\ell^Q(x') \mid x \leftarrow \{0,1\}^{2\ell}, (x', Q) \leftarrow \mathcal{A}(x)] = \mathsf{negl}(\ell).$$

Proof. We show that this is infeasible for Q which only contains a single hardened derivation step. The proof is identical for a non-hardened derivation step and easily extends to a general Q by induction, which is omitted.

For the sake of readability let H denote H_ℓ. Suppose an adversary finds i such that $\mathsf{H}(x) = \mathsf{H}^{(i,h)}(x')$. Then in particular $\mathsf{H}_R(x) = \mathsf{H}_R(x', i)$. Since H_R is a random oracle and is thus collision-resistant, it follows that with overwhelming probability $x = (x', i)$. Thus, to have an equality, we must have that $\mathsf{H}_L(x) = \mathsf{H}_L(x) + x_L$, implying that $x_L = 0$. However, since x is uniformly random, this only happens with probability $2^{-\ell} = \mathsf{negl}(\ell)$. □

2.8 Picnic Signatures

The principal tool we use for signature lifting is the *Picnic* signature scheme of Chase et al. [CDG+17]. The Picnic scheme signature scheme can be instantiated using *any* post-quantum one-way function f to obtain a signature scheme which is post-quantum $\mathsf{EUF\text{-}CMA}$ secure in the QROM (recall Section 2.2). In the obtained scheme, a secret-key is a random point x in the domain of f, and the corresponding public key is $f(x)$.

The Picnic scheme, instantiated with a particular block cipher called *LowMC*, was submitted to NIST for standardization. Their designed prevailed the first two rounds of the competition. It was decided that Picnic will not proceed to the third round due to the novelty of techniques it applies compared with other candidates, however, it was decided to retain Picnic as an alternative candidate [AASA+20]. This means that, while Picnic was not chosen to be NIST standardized, it successfully withstood heavy scrutiny.

3 Signature Lifting

Chase et al. [CDG+17] introduce a signature scheme called *Picnic*, that could be instantiated using *any* post-quantum one-way function f to obtain a signature scheme which is post-quantum $\mathsf{EUF\text{-}CMA}$ secure in the QROM (recall Section 2.2). In the obtained scheme, a secret-key is a random point x in the domain of f, and the corresponding public key is $f(x)$.

In Section 1.2 we gave an overview of how this might be useful to protect procrastinators. The purpose of the current section is to provide explicit constructions and a formal treatment of their security properties.

3.1 Lifting Signature Schemes

Recall that a *correct signature scheme* is a tuple of three polynomial time procedures $\mathsf{DS} = (\mathsf{KeyGen}, \mathsf{Sign}, \mathsf{Ver})$ such that if $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{KeyGen}$ then it holds for any m that if $\sigma \leftarrow \mathsf{Sign}_{\mathsf{sk}}(m)$ then $\mathsf{Ver}_{\mathsf{pk}}(m, \sigma)$ accepts.

Definition 3. *Let* $\mathsf{DS} = (\mathsf{KeyGen}, \mathsf{Sign}, \mathsf{Ver})$ *be a correct digital signature scheme. A* lifting *of* DS *is two procedures* $\widetilde{\mathsf{Sign}}, \widetilde{\mathsf{Ver}}$ *such that* $(\mathsf{KeyGen}, \widetilde{\mathsf{Sign}}, \widetilde{\mathsf{Ver}})$ *is a correct digital signature scheme.*

In Section 2.1 we described the notion of $\mathsf{EUF\text{-}CMA}$ security, we extend this notion in a way appropriate for liftings.

Definition 4. *Let* $\mathsf{DS} = (\mathsf{KeyGen}, \mathsf{Sign}, \mathsf{Ver})$ *and let* $(\widetilde{\mathsf{Sign}}, \widetilde{\mathsf{Ver}})$ *be a lifting, we define the* $\mathsf{EUF\text{-}LCMA}$ *security game as follows:*

- *The challenger* $\mathcal{C}$ *samples* $(\mathsf{sk}, \mathsf{pk}) \leftarrow \mathsf{KeyGen}(1^\lambda)$ *and gives* pk *and* 1^λ *as input to the adversary* $\mathcal{A}$
- $\mathcal{A}$ *is allowed to make* classical *queries both the oracles* $\mathsf{Sign}_{\mathsf{sk}}$ *and* $\widetilde{\mathsf{Sign}}_{\mathsf{sk}}$
- $\mathcal{A}$ *outputs a tuple* (m, σ)
- $\mathcal{A}$ *wins the game if* $\widetilde{\mathsf{Sign}}_{\mathsf{sk}}$ *was never queried on* m*, and* $\widetilde{\mathsf{Ver}}_{\mathsf{pk}}(m, \sigma)$ *accepts.*

Note that if we remove the access to the oracle $\mathsf{Sign}_{\mathsf{sk}}$ *we recover the post-quantum* $\mathsf{EUF\text{-}CMA}$ *game (see Section 2.1.1) for the scheme* $(\mathsf{KeyGen}, \widetilde{\mathsf{Sign}}, \widetilde{\mathsf{Ver}})$*.*

The lifting is a post-quantum (strong) lifting *if the winning probability of* $\mathcal{A}$ *in the* $\mathsf{EUF\text{-}CMA}$ *game (*$\mathsf{EUF\text{-}LCMA}$ *game) is* $\mathsf{negl}(\lambda)$ *for any QPT* $\mathcal{A}$*.*

3.2 Key-lifted Signature Schemes

Let $(\mathsf{KeyGen}, \mathsf{Sign}, \mathsf{Ver})$ be the ECDSA (or Schnorr) digital signature scheme over the curve $\mathsf{secp256k1}$, let H be a random oracle, and define the following modified scheme:

- $\mathsf{KeyGen}'(1^\lambda)$: samples $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{KeyGen}(1^\lambda)$, and outputs $(\mathsf{sk}, \mathsf{pk}' = \mathsf{H}(\mathsf{pk}))$.
- $\mathsf{Sign}'_{\mathsf{sk}}(m)$: outputs $\sigma' = (\sigma, \mathsf{pk})$ where $\sigma \leftarrow \mathsf{Sign}_{\mathsf{sk}}(m)$ (recall that $\mathsf{pk} = \mathsf{PK}^{\mathsf{EC}}(\mathsf{sk}')$, see Section 2.1.2).
- $\mathsf{Ver}'_{\mathsf{pk}'}(m, \sigma')$: interprets σ' as (σ, pk), accepts iff $\mathsf{H}(\mathsf{pk}) = \mathsf{pk}'$ and $\mathsf{Ver}_{\mathsf{pk}}(m, \sigma)$ accepts.

This modification reflects how ECDSA is used in Bitcoin, where the user typically posts a $\mathsf{SHA\text{-}256}$ hash of their public-key, and only posts the public-key when signing a message.

We now define a lifting of $(\mathsf{KeyGen}, \mathsf{Sign}, \mathsf{Ver})$:

- $\widetilde{\mathsf{Sign}}_{\mathsf{sk}} \equiv \mathit{Picnic}(\mathsf{H}).\mathsf{Sign}_{\mathsf{PK}^{\mathsf{EC}}(\mathsf{sk})}$
- $\widetilde{\mathsf{Ver}} \equiv \mathit{Picnic}(\mathsf{H}).\mathsf{Ver}$

It is straightforward to check that $(\widetilde{\mathsf{Sign}}, \widetilde{\mathsf{Ver}})$ is a lifting of $(\mathsf{KeyGen}', \mathsf{Sign}', \mathsf{Ver}')$. We call this scheme the *key-lifted scheme.*

Proposition 5. *If* H *is modeled as a random-oracle, then* $(\widetilde{\mathsf{Sign}}, \widetilde{\mathsf{Ver}})$ *is a post-quantum lifting (see Definition 4).*

Proof. Let $\mathcal{A}$ be an $\mathsf{EUF\text{-}CMA}$ adversary for $(\mathsf{KeyGen}', \widetilde{\mathsf{Sign}}, \widetilde{\mathsf{Ver}})$ which wins with probability ε, we construct an $\mathsf{EUF\text{-}CMA}$ adversary $\mathcal{A}_p$ for $\mathit{Picnic}(H)$ which wins with probability ε.

$\mathcal{A}_p$ uses the pk she received from $\mathcal{C}$ as input to $\mathcal{A}$, and responds to oracle queries on m by querying her own oracle. Since $\mathsf{PK}^{\mathsf{EC}}$ is a bijection and sk distributes uniformly it follows that $\mathsf{PK}^{\mathsf{EC}}(\mathsf{sk})$ also distributes uniformly. Hence, the view of $\mathcal{A}$ distributes identically in the simulation and in the real $\mathsf{EUF\text{-}CMA}$ game. Hence, with probability ε the output (σ, m) wins the $\mathsf{EUF\text{-}CMA}$ game for $\mathit{Picnic}(H)$. But since $\mathcal{A}$ and $\mathcal{A}_p$ make the same queries, (σ, m) is winning output for $\mathcal{A}$ iff it is a winning output for $\mathcal{A}_p$.

We now invoke [CLQ20, Theorem 4], which asserts that a random oracle is post-quantum one-way (even against a non-uniform QPT adversary with a quantum advice), and [CDG+17, Corollary 5.1], which asserts that $\mathit{Picnic}(\mathsf{H})$ is post-quantum $\mathsf{EUF\text{-}CMA}$ secure whenever f is post-quantum one-way, to conclude that $\varepsilon = \mathsf{negl}(\lambda)$. □

We note that $(\widetilde{\mathsf{Sign}}, \widetilde{\mathsf{Ver}})$ is *not* a strong lifting, since the output of $\mathsf{Sign}'_{\mathsf{sk}}$ (on any message) contains a copy of pk, from which a quantum adversary can calculate sk.

3.3 Preimage Extractability from Picnic Signatures

In the next section, we introduce *seed-lifted* schemes. As we explain therein, the seed-lifted scheme is constructed such that an adversary with access to valid signatures with respect to some keys can't feasibly create a signature that is valid with respect to a *different* (but related) public-key. This type of security can not be implied by the $\mathsf{EUF\text{-}CMA}$ security of *Picnic* alone, as such security only prohibits creating signed documents verifiable by *the same* public-key.

In order to overcome this, we want to exploit the relations between the two public-keys – the one given to the adversary, and the one with respect to which their output passes verification – to produce a collision in a function known to be collision resistant. However, in order to do that, we need to compute the *secret*-key corresponding to the public-key produced by the adversary.

Roughly speaking, the property we need is that if it is feasible to produce a signature verifiable by some public-key, then it is also feasible to produce the corresponding secret-key.

To see this holds for the scheme $\mathit{Picnic}(f)$, we note that in this scheme a signature is an accepting transcript of a slightly modified version of the Unruh transform [Unr15], applied to particular Σ-protocol used as an argument of knowledge for a preimage of f used to instantiate the scheme[7]. The purpose of the modification is two-fold: both to incorporate the message m into the transcript, and to slightly generalize Unruh's transform, whose original formulation does not apply to the particular Σ-protocol used in [CDG+17].

In particular, part of the proof of [CDG+17, Corollary 5.1] extends [Unr15, Theorem 18], proving that for any post-quantum one-way function f, a valid $\mathit{Picnic}(f)$ signature (on any message) is an *argument of knowledge* whose error is negligible in the length of the input.

We summarize this discussion in the following:

Proposition 6 (Picnic extractability). *Let f be a post-quantum one-way function with input length λ. Assume there exists a QPT adversary $\mathcal{A}$ that, on a uniformly random input $\mathsf{pk} \leftarrow \{0,1\}^\lambda$, outputs with probability ε a signed document (m, σ) such that $\mathit{Picnic}(f).\mathsf{Ver}_{\mathsf{pk}}(m, \sigma)$ accepts. Then there exists a QPT* extractor *$\mathcal{E}$ that, on a uniformly random input $\mathsf{pk} \leftarrow \{0,1\}^\lambda$, outputs sk such that $f(\mathsf{sk}) = \mathsf{pk}$ with probability $\varepsilon - \mathsf{negl}(\lambda)$.*

3.4 Seed-lifted Signature Schemes

We now describe how to use the *Picnic* scheme to allow securely spending a pre-quantum UTXO even if its public-key has leaked, provided that the public key was generated by an HD wallet (see Section 2.7).

Like in key-lifting, we modify the ECDSA scheme to reflect how such addresses are used in practice. Unlike key-lifting, here the *public*-key remains the same, whereas we modify the *secret*-key. Instead of using

[7] For an overview of Σ-protocols and how they are used to construct signature schemes, we refer the reader to [Kat10, Chapter 8]

the ECDSA secret-key sk, we use the seed s and password pw used to generate the master public-key (see Section 2.6) and the derivation path P corresponding to pk as a secret-key. Since sk can be easily calculated from (s, pw, P), defining the modified scheme is straightforward:

- $\mathsf{KeyGen}(1^\lambda)$ samples $\mathsf{sk} = (s, pw, P)$ where $s \leftarrow \{0,1\}^\lambda$, and pw and P are chosen arbitrarily such that P is independent from (s, pw), sets $\mathsf{pk} = \mathsf{PK}^{\mathsf{EC}}(\mathsf{Der}(\mathsf{KDF}(s, pw), P)_L)$, outputs $(\mathsf{sk}, \mathsf{pk})$.
- $\mathsf{Sign}_{(s,pw,P)}(m)$ outputs $\sigma \leftarrow \mathsf{ECDSA.Sign}_{\mathsf{sk}'}(m)$ where $\mathsf{sk}' = \mathsf{Der}(\mathsf{KDF}(s, pw), P)_L$.
- $\mathsf{Ver} \equiv \mathsf{ECDSA.Ver}$

Where $\mathsf{KDF} = \mathsf{H}^{2048}$ as in BIP-32 (see Section 2.6), and Der is the function defined in Section 2.7.

We now define a lifting for this scheme. The lifting is constructed in such a way that its security assures that the adversary is unable to forge a signature verifiable by *any* key in the wallet, not just the key used by the honest user. Set $\mathsf{KDF}^{pre} = \mathsf{H}^{2047}$ and $\mathsf{KDF}^{pq} = H$.

Definition 5. *The* seed-lifting *of the scheme above is:*

- $\widetilde{\mathsf{Sign}}_{(s,pw,P)}$ *outputs* $(\sigma, \mathsf{msk}, P)$ *where* $\sigma \leftarrow Picnic(\mathsf{KDF}^{pq}).\mathsf{Sign}_{\mathsf{KDF}^{pre}(s,pw)}(m, P)$ *and* $\mathsf{msk} = \mathsf{KDF}(s, pw)$
- $\widetilde{\mathsf{Ver}}_{\mathsf{pk}}(m, (\sigma, \mathsf{msk}, P))$ *accepts if* $\mathsf{Der}(\mathsf{msk}, P) = \mathsf{pk}$ *and* $Picnic(\mathsf{KDF}^{pq}).\mathsf{Ver}_{\mathsf{msk}}((m, P), \sigma)$ *accepts.*

Theorem 1. *If* H *is modeled as a random oracle, then the seed-lifting is a strong lifting.*

We prove this in two steps: we show that the seed-lifting is a post-quantum lifting, and then infer from that that it is actually a strong lifting. The first step requires most of the effort, so we start with the shorter and more straightforward second step.

Proposition 7. *If the seed-lifting is* $\mathsf{EUF\text{-}CMA}$ *secure, then it is* $\mathsf{EUF\text{-}LCMA}$ *secure.*

Proof. The idea is that all signatures provided by the seed-lifted schemes contain information that could be used to perfectly simulate the signature oracle of the modified scheme.

Let $\mathcal{A}$ win the $\mathsf{EUF\text{-}LCMA}$ game with probability ε, and consider the adversary $\mathcal{A}'$ for the $\mathsf{EUF\text{-}CMA}$ game which operates as follows:

- $\mathcal{A}'$ initiates the simulation of $\mathcal{A}$ by providing her the pk she obtained from the challenger.
- $\mathcal{A}'$ queries the signature oracle (recall that since $\mathcal{A}'$ is an $\mathsf{EUF\text{-}CMA}$ adversary, she only has access to the signature oracle of the lifting) on a uniformly random $m_0 \leftarrow \{0,1\}^\lambda$ to obtain $(\sigma, \mathsf{msk}, P)$.
- $\mathcal{A}'$ resumes simulation of $\mathcal{A}$, using her own oracle access to answer queries on $\widetilde{\mathsf{Sign}}_{\mathsf{sk}}$, and responding to queries of the form $\mathsf{Sign}_{\mathsf{sk}}(m)$ with $\sigma \leftarrow \mathsf{ECDSA.Sign}_{\mathsf{Der}(\mathsf{msk},P)}(m)$.
- Outputs the output of $\mathcal{A}$

Note that the view of $\mathcal{A}$ in the real and simulated games is identical.

By hypothesis, $\mathcal{A}$ outputs (σ, m) such that $\mathsf{Ver}_{\mathsf{pk}}(m, \sigma)$ accepts and $\mathcal{A}$ never made a query on m is ε. In this event, $\mathcal{A}'$ wins the game unless $m = m_0$. However, since m_0 was chosen uniformly, and the view of $\mathcal{A}$ is independent of m_0, it follows that $m \neq m_0$ with overwhelming probability. Hence $\mathcal{A}_p$ wins the $\mathsf{EUF\text{-}CMA}$ game with probability $\varepsilon - \mathsf{negl}(\lambda)$. But by hypothesis the seed lifting is post-quantum, so it follows that $\varepsilon = \mathsf{negl}(\lambda)$, whereby the lifting is strong. □

The first step is the following statement:

Proposition 8. *If* H *is modeled as a random oracle, the seed-lifting is* $\mathsf{EUF\text{-}CMA}$ *secure.*

Proof. We prove this by a sequence of hybrids.

We start with $H_0(\mathcal{A}, 1^\lambda)$ is just a restatement of the $\mathsf{EUF\text{-}CMA}$ game, explicated for the seed-lifting:

1. $\mathcal{C}$ samples $(\mathsf{sk}, \mathsf{pk}) \leftarrow \mathsf{KeyGen}(1^\lambda)$ where $\mathsf{sk} = (s, pw, P)$
2. Set $\mathsf{sk}_p = \mathsf{KDF}^{pre}(s, pw)$

3. Set $\mathsf{msk} = \mathsf{KDF}^{pq}(\mathsf{sk}_p)$, note that by definition $\mathsf{pk} = \mathsf{Der}(\mathsf{msk}, P)$
4. $\mathcal{C}$ send pk to $\mathcal{A}$
5. Let q be the number of queries made by $\mathcal{A}$, for $i = 1, \dots, q$:
   (a) $\mathcal{A}$ sends m_i to $\mathcal{C}$
   (b) $\mathcal{C}$ computes $\sigma \leftarrow \mathit{Picnic}(\mathsf{KDF}^{pq}).\mathsf{Sign}_{\mathsf{sk}_p}(m, P)$
   (c) $\mathcal{C}$ sends $(\sigma, \mathsf{msk}, P)$ to $\mathcal{A}$
6. $\mathcal{A}$ sends $(m, (\sigma, \mathsf{msk}', P'))$ to $\mathcal{C}$
7. if $\exists i : m_i = m$, output 0
8. if $\mathsf{Der}(\mathsf{msk}', P') \neq \mathsf{pk}$, output 0
9. output 1 if $\mathit{Picnic}(\mathsf{KDF}^{pq}).\mathsf{Ver}_{\mathsf{msk}'}((m, P'), \sigma)$ accepts, 0 otherwise.

In the next hybrid $H_1(\mathcal{A}, 1^\lambda)$ we choose sk_p uniformly at random:

2. ~~Set $\mathsf{sk}_p = \mathsf{KDF}^{pre}(s, pw)$~~
   $\mathcal{C}$ samples $\mathsf{sk}_p \leftarrow \{0,1\}^\ell$ (where ℓ is the secret-key length)

Claim. For any QPT $\mathcal{A}$ it holds that $|\mathbb{P}[H_0(\mathcal{A}, 1^\lambda) = 1] - \mathbb{P}[H_1(\mathcal{A}, 1^\lambda) = 1]| = \mathsf{negl}(\lambda)$.

Proof. Consider the distributions $\mathbb{D}_0$ and $\mathbb{D}_1$ where $\mathbb{D}_0$ is uniform on $\{0,1\}^\ell$ and $\mathbb{D}_1$ is sampled by sampling $(\mathsf{sk} = (s, pw, p), \mathsf{pk}) \leftarrow \mathsf{KeyGen}(1^\lambda)$ and outputting $\mathsf{KDF}^{pre}(s, pw)$.

Recall that $\mathsf{KDF}^{pre} = \mathsf{H}^{2047}$. That H is uniformly random does not imply that KDF^{pre} is uniformly random. However, [BDD+17] prove that the distribution of iterating a random function (with an exponentially large domain) a constant number of times is computationally indistinguishable from uniformly sampling a function. Combined with the fact that $\gamma_{(s,pw)} = \mathsf{negl}(\lambda)$ (recall Definition 1) we get that if from Proposition 1 that $|\mathbb{P}[\mathcal{A}_D^{\mathbb{D}_0}(1^\lambda) = 1] - \mathbb{P}[\mathcal{A}_D^{\mathbb{D}_0}(1^\lambda) = 1| = \mathsf{negl}(\lambda)$, where $\mathcal{A}_D^{\mathbb{D}}$ is any QPT procedure that is given oracle access to a distribution $\mathbb{D}$.

Now consider the following procedure $\mathcal{A}_D^{\mathbb{D}}(1^\lambda)$:

- $\mathcal{A}_D$ samples $\mathsf{sk}_p \leftarrow \mathbb{D}$
- $\mathcal{A}_D$ samples $(\mathsf{sk}, \mathsf{pk}) \leftarrow \mathsf{KeyGen}(1^\lambda)$, she discards all data but P
- $\mathcal{A}_D$ computes $\mathsf{msk} = \mathsf{KDF}^{pq}(\mathsf{sk}_p)$ and $\mathsf{pk} = \mathsf{Der}(\mathsf{msk}, P)$
- $\mathcal{A}_D$ simulates the $\mathsf{EUF\text{-}CMA}$ game with $\mathcal{A}$ by exactly emulating $\mathcal{C}$, and outputs the output of $\mathcal{C}$

Note that the view of $\mathcal{A}$ in the simulation carried by $\mathcal{A}_D^{\mathbb{D}_b}(1^\lambda)$ is identical to the view in the hybrid $H_b(\mathcal{A}, 1^\lambda)$. Hence $\mathbb{P}[H_b(\mathcal{A}, 1^\lambda) = 1] = \mathbb{P}[\mathcal{A}_D^{\mathbb{D}_b}(1^\lambda) = 1]$ and it follows that $|\mathbb{P}[H_0(\mathcal{A}, 1^\lambda) = 1] - \mathbb{P}[H_1(\mathcal{A}, 1^\lambda) = 1]| = \mathsf{negl}(\lambda)$. □

In the hybrid H_2 we add a new failure condition that trivially never holds, it will be useful for the following hybrids:

7. ...
8. if $\mathsf{Der}(\mathsf{msk}', P') \neq \mathsf{pk}$, output 0
9. Set $E = \mathsf{True}$
10. if $E = \mathsf{False}$, output 0
11. output 1 if $\mathit{Picnic}(\mathsf{KDF}^{pq}).\mathsf{Ver}_{\mathsf{msk}'}((m, P'), \sigma)$ accepts, 0 otherwise.

Since E is always true, it follows that H_1 and H_2 execute identically, proving:

Claim. For any QPT $\mathcal{A}$ it holds that $\mathbb{P}[H_1(\mathcal{A}, 1^\lambda) = 1] = \mathbb{P}[H_2(\mathcal{A}, 1^\lambda) = 1]$.

In the hybrid H_3 we decompose E into a conjunction of several events $E_1, \dots, E_4$:

9. ~~Set $E = \mathsf{True}$~~
   Set $E = E_1 \vee E_2 \vee E_3 \vee E_4$ where :

   (a) $E_1 = \mathsf{True}$ if $(\mathsf{msk}, P) \neq (\mathsf{msk}', P')$ and neither is a Der-suffix of the other
   (b) $E_2 = \mathsf{True}$ if (msk, P) is a Der-suffix of (msk', P')
   (c) $E_3 = \mathsf{True}$ if (msk', P') is a Der-suffix of (msk, P)
   (d) $E_4 = \mathsf{True}$ if $(\mathsf{msk}, P) = (\mathsf{msk}', P')$
10. if $E = \mathsf{False}$, output 0

Claim. For any QPT $\mathcal{A}$ it holds that $\mathbb{P}[H_2(\mathcal{A}, 1^\lambda) = 1] = \mathbb{P}[H_3(\mathcal{A}, 1^\lambda) = 1]$.

Proof. It suffices to show that in H_3 it holds with certainty that $E = \mathsf{True}$.

Consider the strings (msk, P), (msk', P'). If they are equal, then $E_4 = \mathsf{True}$. If $\mathsf{Der}(\mathsf{msk}, P) \neq \mathsf{Der}(\mathsf{msk}', P')$ then neither can be a suffix of the other (since a suffix is a type of collision), so $E_1 = \mathsf{True}$. If $\mathsf{Der}(\mathsf{msk}, P) = \mathsf{Der}(\mathsf{msk}', P')$ then they constitute a collision in Der. If it is a suffix collision, then either $E_2 = \mathsf{True}$ or $E_3 = \mathsf{True}$ (depending on which of the strings is a suffix of the other). If it is not a suffix collision, then $E_1 = \mathsf{True}$. □

In the hybrid H_4 we set E_1 to false:

9. Set $E = E_1 \vee E_2 \vee E_3 \vee E_4$ where:
   (a) ~~$E_1 = \mathsf{True}$ if $(\mathsf{msk}, P) \neq (\mathsf{msk}', P')$ and neither is a Der-suffix of the other~~
   $E_1 = \mathsf{False}$
   (b) $E_2 = \mathsf{True}$ if (msk, P) is a Der-suffix of (msk', P')
   (c) $E_3 = \mathsf{True}$ if (msk', P') is a Der-suffix of (msk, P)
   (d) $E_4 = \mathsf{True}$ if $(\mathsf{msk}, P) = (\mathsf{msk}', P')$
10. if $E = \mathsf{False}$, output 0

Claim. For any QPT $\mathcal{A}$ it holds that $|\mathbb{P}[H_3(\mathcal{A}, 1^\lambda) = 1] - \mathbb{P}[H_4(\mathcal{A}, 1^\lambda) = 1]| = \mathsf{negl}(\lambda)$.

Proof. We note that H_3 conditioned on $E_1 = \mathsf{False}$ is identical to H_4. Hence, $|\mathbb{P}[H_3(\mathcal{A}, 1^\lambda) = 1] - \mathbb{P}[H_4(\mathcal{A}, 1^\lambda) = 1]|$ is at most the probability that $E_1 = \mathsf{True}$ in H_3.

Consider a QPT adversary $\mathcal{A}_c$ that perfectly simulates $H_3(\mathcal{A}, 1^\lambda)$ until receiving an output. $\mathcal{A}_c$ then outputs $((\mathsf{msk}, P), (\mathsf{msk}', P'))$ where (msk', P') was taken from $\mathcal{A}$'s output. The event that $((\mathsf{msk}, P), (\mathsf{msk}', P'))$ is a non-suffix collision in Der is exactly the event that $E_1 = \mathsf{True}$. Proposition 3 asserts this event has negligible probability. □

In the hybrid H_5 we set E_2 to false:

9. Set $E = E_1 \vee E_2 \vee E_3 \vee E_4$ where:
   (a) $E_1 = \mathsf{False}$
   (b) ~~$E_2 = \mathsf{True}$ if (msk, P) is a Der-suffix of (msk', P')~~
   $E_2 = \mathsf{False}$
   (c) $E_3 = \mathsf{True}$ if (msk', P') is a Der-suffix of (msk, P)
   (d) $E_4 = \mathsf{True}$ if $(\mathsf{msk}, P) = (\mathsf{msk}', P')$
10. if $E = \mathsf{False}$, output 0

Claim. For any QPT $\mathcal{A}$ it holds that $|\mathbb{P}[H_4(\mathcal{A}, 1^\lambda) = 1] - \mathbb{P}[H_5(\mathcal{A}, 1^\lambda) = 1]| = \mathsf{negl}(\lambda)$.

Proof. Like in the previous claim, we have that $|\mathbb{P}[H_4(\mathcal{A}, 1^\lambda) = 1] - \mathbb{P}[H_5(\mathcal{A}, 1^\lambda) = 1]|$ is bound by the probability that $E_2 = \mathsf{True}$ in H_4.

Consider the following adversary $\mathcal{A}_s$ that given a uniformly random sk_p does the following:

– calculates $\mathsf{msk} = \mathsf{KDF}^{pq}(\mathsf{sk}_p)$
– uses KeyGen to sample a path P
– calculates $\mathsf{pk} = \mathsf{Der}(\mathsf{msk}, P)$

- simulates $\mathcal{A}$ using pk as input and sk_p, P to answer oracle calls
- recovers (msk', P') from the output of $\mathcal{A}$
- if (msk, P) is a Der-suffix of (msk', P'), let Q such be a path such that $P' = Q \| P$, and output (msk', Q), otherwise output $\perp$.

Note that the event that $E_2 = \mathsf{True}$ in H_4 is exactly the event that $\mathcal{A}_s$ didn't output $\perp$. In this case, the output (msk', Q) if $\mathcal{A}$ satisfies that $\mathsf{Der}(\mathsf{msk}', Q) = \mathsf{KDF}^{pq}(\mathsf{sk}_p)$. Recalling that sk_p is uniformly random, $\mathsf{KDF}^{pq} = \mathsf{H}$ and $\mathsf{Der}(\mathsf{msk}', Q) = \mathsf{H}^Q(\mathsf{msk}')$, it follows from Proposition 4 that this could only happen with negligible probability. □

In the hybrid H_6 we set E_3 to false:

9. Set $E = E_1 \vee E_2 \vee E_3 \vee E_4$ where:
   (a) $E_1 = \mathsf{False}$
   (b) $E_2 = \mathsf{False}$
   (c) ~~$E_3 = \mathsf{True}$ if (msk', P') is a Der-suffix of (msk, P)~~
   $E_3 = \mathsf{False}$
   (d) $E_4 = \mathsf{True}$ if $(\mathsf{msk}, P) = (\mathsf{msk}', P')$
10. if $E = \mathsf{False}$, output 0

Claim. For any QPT $\mathcal{A}$ it holds that $|\mathbb{P}[H_5(\mathcal{A}, 1^\lambda) = 1] - \mathbb{P}[H_6(\mathcal{A}, 1^\lambda) = 1]| = \mathsf{negl}(\lambda)$.

Proof. Keeping the same line of reasoning, it suffices to show that the event that $E_3 = \mathsf{True}$ in H_5 has negligible probability.

The event E_3 is the event that $\mathcal{A}$ managed to produce an output $((\sigma, \mathsf{msk}', P'), m)$ with the following properties:

- (msk', P') is a Der-suffix of (msk, P). That is, there is some non-empty path Q such that $P = Q \| P'$ and $\mathsf{msk}' = \mathsf{Der}(\mathsf{msk}, Q)$.
- $\mathit{Picnic}(\mathsf{KDF}^{pq}).\mathsf{Ver}_{\mathsf{msk}'}((m, P'), \sigma)$ accepts.
- $\mathcal{A}$ has never made a signature query on m.
- $\mathsf{Der}(\mathsf{msk}', P') = \mathsf{pk}$.

Say this happens with probability ε. Since P is sampled independently of sk_p, there has to be a particular path $\tilde{P}$ such that the probability of E conditioned on $P = \tilde{P}$ is at least ε.

Let $q + 1$ be the length of P (recall that P must have length at least two for the event E_3 to hold), for any $t = 1, \ldots, q$ let P_t be the suffix of $\tilde{P}$ of length t. Then there must exist some $\tilde{t}$ such that $P' = P_{\tilde{t}}$ with probability at least ε / q.

Let $\mathcal{A}_z$ be the procedure that on input sk_p does the following:

- Calculates $\mathsf{msk} = \mathsf{H}(\mathsf{sk}_p)$ and $\mathsf{pk} = \mathsf{PK}^{\mathsf{EC}}(\mathsf{Der}(\mathsf{msk}, \tilde{P})_L)$ and gives pk as input to $\mathcal{A}$
- Simulates the hybrid H_5 responding to signature queries with $(\sigma, \mathsf{msk}, \tilde{P})$ where $\sigma \leftarrow \mathit{Picnic}(\mathsf{H}).\mathsf{Sign}_{\mathsf{sk}_p}(m, \tilde{P})$
- Resumes operations until obtaining output $((\sigma, \mathsf{msk}', P'), m)$.
- Outputs σ if all the following hold: $P' = P_{\tilde{t}}$, $\mathsf{msk}' = \mathsf{Der}(\mathsf{msk}, Q)$ where $\tilde{P} = Q \| P'$, m was never the input to a query and $\mathit{Picnic}(H).\mathsf{Ver}_{\mathsf{msk}'}((m, P), \sigma)$ accepts. Otherwise, output $\perp$.

By design. If sk_p is uniformly random, then the probability that $\mathcal{A}_z$ doesn't output $\perp$ is at least ε / q. In this case, $\mathcal{A}_z$ managed to output a signature that passes the $\mathit{Picnic}(\mathsf{H})$ verification with respect to the public-key msk'.

It follows from Proposition 6 that there exists an extractor $\mathcal{E}$ which outputs an H-preimage x of msk' with probability $\varepsilon / q - \mathsf{negl}(\lambda)$. We thus have that $\mathsf{H}(x) = \mathsf{H}^Q(\mathsf{msk})$. By Proposition 4 this can only happen with negligible probability. We get that $\varepsilon / q - \mathsf{negl}(\lambda)$ is negligible and so ε is also negligible. □

In the last hybrid H_7 we set $E_4 = \mathsf{False}$:

9. Set $E = E_1 \vee E_2 \vee E_3 \vee E_4$ where:
   (a) $E_1 = \mathsf{False}$
   (b) $E_2 = \mathsf{False}$
   (c) $E_3 = \mathsf{False}$
   (d) ~~$E_4 = \mathsf{True}$ if $(\mathsf{msk}, P) = (\mathsf{msk}', P')$~~
   $E_4 = \mathsf{False}$
10. if $E = \mathsf{False}$, output 0

Claim. For any QPT $\mathcal{A}$ it holds that $|\mathbb{P}[H_5(\mathcal{A}, 1^\lambda) = 1] - \mathbb{P}[H_6(\mathcal{A}, 1^\lambda) = 1]| = \mathsf{negl}(\lambda)$.

Proof. Keeping the same line of reasoning, it suffices to show that the event that $E_4 = \mathsf{True}$ in H_6 has negligible probability.

Let ε be the probability that $E_4 = \mathsf{True}$ in H_6, let $\mathcal{A}_p$ be the following EUF-CMA adversary for $Picnic(\mathsf{KDF}^{pq})$:

- Let $(\mathsf{sk}_p, \mathsf{pk}_p)$ denote the keys generated by $\mathcal{C}$.
- After receiving pk_p, $\mathcal{A}_p$ samples $((s, pw, P), \mathsf{pk}) \leftarrow \mathsf{KeyGen}(1^\lambda)$ and discards all data but P (recall that P is sampled independently of (s, pw).
- $\mathcal{A}_p$ uses $\mathsf{pk} = \mathsf{Der}(\mathsf{pk}_p, P)$ as input to $\mathcal{A}$.
- $\mathcal{A}_p$ responds to a signature query on m by querying the $Picnic(\mathsf{KDF}^{pq}).\mathsf{Sign}_{\mathsf{sk}_p}$ oracle on (m, P) to obtain σ and outputting $(\sigma, \mathsf{pk}_p, P)$.
- $\mathcal{A}_p$ resumes the simulation until obtaining an output $((\sigma, \mathsf{msk}', P'), m)$. If $(\mathsf{pk}_p, P) = (\mathsf{msk}', P')$, m was never queried and $Picnic(\mathsf{KDF}^{pq}).\mathsf{Ver}_{\mathsf{pk}_p}(m, \sigma)$ accepts, $\mathcal{A}_p$ outputs m. Otherwise $\mathcal{A}_p$ outputs $\perp$.

Note that the view of $\mathcal{A}$ in the simulation and in H_5 is identical. Also note that $\mathcal{A}_p$ either wins the EUF-CMA game or outputs $\perp$, and the former happens exactly when $\mathcal{A}$ outputs a winning output. That is, the probability that $\mathcal{A}_p$ wins the game is exactly ε.

Now note that, since $\mathsf{KDF}^{pq} = \mathsf{H}$ is modeled as a random oracle, it follows from [CLQ20, Theorem 4] that it is post-quantum one-way. It then follows from [CDG+17, Corollar 5.1] that $Picnic(\mathsf{KDF}^{pq})$ is EUF-CMA secure. Since $\mathcal{A}_p$ is a QPT adversary that wins the EUF-CMA game for $Picnic(\mathsf{KDF}^{pq})$ with probability ε, it follows that $\varepsilon = \mathsf{negl}(\lambda)$.

□

By stringing all claims above together and applying the triangle inequality, we obtain that it holds for any QPT adversary $\mathcal{A}$ that $|\mathbb{P}[H_0(\mathcal{A}, 1^\lambda) = 1] - \mathbb{P}[H_7(\mathcal{A}, 1^\lambda) = 1]| = \mathsf{negl}(\lambda)$.

We conclude the proof by noting that in H_7 it always holds that $E = \mathsf{False}$, whereby, for any QPT adversary $\mathcal{A}$ we have that $\mathbb{P}[H_7(\mathcal{A}, \lambda) = 1] = 0$. It follows that $\mathbb{P}[H_0(\mathcal{A}, 1^\lambda) = 1] = \mathsf{negl}(\lambda)$ as needed.

□

3.4.1 Size of Lifted Signatures The scheme resulting from lifting a one-way function might admit prohibitively large signatures. The size of the signature is a consequence of the size of the circuit calculating f.

In [CDG+17], the authors isolate a one-way function particularly suitable for the task (namely, the encryption circuit of a block cipher called LowMC [ARS+15]). They use this cipher to construct the Fish and Picnic schemes. The Fish scheme has a signature length of about 120KB, but its security is only known in the ROM, whereas the signature sizes of Picnic are about 195 KB, and it is proven to be secure in the QROM.

A follow-up work [KZ20] optimizes the Picnic scheme *with respect to the same one-way function* to below 50KB.

In our setting, we are not free to choose the function. We always instantiate *Picnic* with either SHA-256 or SHA-512 depending on the context. In [CDG+17], the authors manage to apply their technique to SHA-256 to obtain signatures of size 618KB with 128 bits of security (the performance of the optimized version of

[KZ20] when instantiated with SHA-256 is not analyzed). The authors are not aware of empirical data for *Picnic* signature sizes when instantiated with SHA-512.

A further advantage of the [CDG+17,KZ20] constructions is that they could be instantiated with a wide range of parameters. The size of the signature could hence be decreased by reducing the number of bits of security. For example, in [CDG+17] it is noted that the size of the signature instantiated with SHA-256 and admitting 80 bits of security is 385KB long, which is an improvement by a factor of almost a half.

We are hopeful that the [KZ20] scheme affords significantly shorter signatures when instantiated with the functions we require, and further optimizations could further reduce signature sizes.

4 Detecting Quantum Adversaries

In this section, we introduce *quantum canaries* – puzzles designed to be intractable for classical computers but solvable for quantum computers whose scale is significantly smaller than required to compromise ECDSA signatures. A solution to the puzzle will then act as a heads-up for the network that the quantum era is near. To incentivize quantum entities to solve the puzzle, thus alerting the network to the presence and compromising the quantum loot, we propose awarding a monetary prize to the first solver. In Section 4.5, we provide a game theoretic analysis of two adversaries competing for the canary bounty and quantum loot.

Once the canaries are set up, they could be used to implement policies in consensus, such as "any transaction posted more than 10,000 blocks after the canary puzzle was solved which is not quantum-cautious is invalid".

The idea of using cryptographic canaries to award bounties for discovering exploits was first discussed in [Dra18]. We discuss in more depth the specific application of canaries to detect quantum adversaries. In particular, we analyze the behavior of two adversaries competing for the quantum loot and discuss the option to fund the bounty for the canary based on UTXOs that would be burned.

4.1 Properties of Good Canaries

We expect a quantum canary to satisfy the following properties:

- **Similarity** The task of killing the canary should be as similar as possible to the task of forging an ECDSA signature for a selected message given access to the public-key. If the challenge we choose for killing the canary is vastly different from breaking ECDSA, it becomes more plausible that a future optimization could reduce the scale required for one task over the other, leading to a scenario where killing the canary becomes nearly as hard or even harder than breaking ECDSA, allowing a future attacker to claim the bounty *and* the loot. There are several incomparable complexity metrics for "hardness" in quantum computing, the most studied being the the circuit depth (directly related to the time complexity) and the number of qubits (space complexity). Different problems offer different space-time trade-offs, and since it is impossible to predict which quantum resource will develop more quickly, it is desirable that killing the canary will have trade-offs similar to breaking ECDSA.
- **Incentive** There should be a clear incentive for quantum entities able to kill the canary to do so, rather than hiding their quantum capabilities until they mature enough to loot pre-quantum coins.
- **Security** The mechanism for posting the solution to the challenge should be secure against forking attacks. In particular, it should be infeasible for anyone listening to the mempool to claim the solution as their own.
- **Nothing Up My Sleeve** The procedure for generating the challenge should be publicly known. Any randomness used for generating the challenge should be sampled from a verifiably random source. All steps in the generating procedure should be justified, and arbitrary choices (that could conceal backdoor solutions) should be avoided.

4.2 Choosing the Puzzle

Since our ultimate concern is adversaries that are able to break ECDSA signatures over the secp256k1 curve, a natural contender for a puzzle is to forge an ECDSA signature over a curve with less security[8].

A necessary (but not sufficient) condition for this solution to have an untrusted setup is that we are able to generate a public-key without learning anything about the matching secret-key. Fortunately, ECDSA has the nice feature that the public-key is merely a random point on the elliptic curve. This leads to the following approach:

- Select an appropriate curve,
- sample a random point pk on the curve and a random string r,
- post (pk, r) to the blockchain,
- to kill the canary, post σ which passes ECDSA verification as a signature of r with respect to the chosen curve.

The missing components to fully specify the canary are the curve itself, and a method for generating a public-key which does not provide information about the corresponding secret-key.

It is desirable that the family of curves we choose from bear as much similarity as possible to secp256k1, or we risk the scenario previously described where an optimization is found to computing discrete logarithm on secp256k1 but not on the challenge curve (or vice versa).

The secp256k1 curve is a curve over a prime field whose order is also prime, given in Weierstrass form. Recent works [RNSL17,HJN+20] provide concrete cryptanalysis for computing discrete logarithms in such elliptic groups as a function of the order of the prime field above which they are defined.

The discussion above suggests that a good choice for a curve family might be the family of all Weierstrass curves over prime fields of prime order whose binary representation has length b, for some appropriately chosen b.

However, simply choosing an arbitrary curve of convenient parameters and counting our blessings is not a sensible approach. Appropriately choosing a secure elliptic curve is a very difficult task, and a carelessly chosen curve could easily exhibit classically feasible exploits (for an overview of selecting elliptic curves, see e.g. [BCLN16]). Unfortunately, our current application is unusual in the sense that we deliberately seek out curves whose bit security is *intermediate*, in the sense that it is lower than currently used curves but still high enough to withstand classical attacks. Virtually all available literature focuses on curves chosen to be as *strong* as possible.

Moreover, the secp256k1 curve was chosen partly due to its convenient parameterization, which allows for optimizing arithmetic operations over the curve group. This has no effect on the asymptotic computational cost of any algorithm computing the discrete logarithm, but has a significant impact on the constants. It is plausible that computing a discrete logarithm over secp256k1 might be more efficient than computing a discrete logarithm over a general curve with fewer bits of security due to these optimizations. The curve selection process must be aware of these nuances. It might be desirable to require that the canary curves also furnish a compact binary representation. We leave the task of choosing a concrete curve to further discussion and future work.

4.3 Funding the Incentive

The bounty could be funded from several sources: it could be raised from the community, freshly minted for that purpose, or “borrowed” from future inflation (e.g. allocating 5% of all future block rewards). Another funding source unique to our setting is the funds destined to burn. Recall that in our proposed solution (see Section 1.7), all UTXOs whose addresses were posted to the blockchain prior to 2013 will become forever

[8] Another natural contender is to skip the signing mechanism and simply require the solver to solve a logarithmic equation over a suitable elliptic curve. However, it is hard to argue that the difficulty of this task scales the same as creating an ECDSA signature. An ECDSA signature contains many steps besides solving the logarithmic equation, so implementing the entire signing mechanism is better in terms of similarity.

unspendable two months after the canary is killed. The bounty for killing the canary could be taken from these funds.

However, we do not recommend using the burned funds, as there is uncertainty regarding how much of these funds will remain by the time the quantum era starts. This uncertainty could deter adversaries from claiming the bounty, and encourage them to wait for the loot.

4.4 Adaptation to Taproot

So far we implicitly assumed that quantum loot is locked behind ECDSA signatures. However, as the newly deployed Taproot update [WNT20] gains adoption, more quantum loot is accumulated behind Schnorr signatures (see Section 2.4). Fortunately, the implementation of Schnorr signatures in Taproot is instantiated with the same secp256k1 curve used to instantiate the ECDSA scheme.

However, one might argue that Schnorr and ECDSA are not similar enough, and by specializing the puzzle to ECDSA we take the risk that a quantum adversary will be able to loot P2TR UTXOs before they would be able to collect the bounty.

Fortunately, in both the ECDSA and Schnorr schemes, the public-key is a uniformly random point on secp256k1. Hence, a wide adoption of secp256k1-instantiated Schnorr signatures (e.g. via Taproot) could be addressed by modifying the puzzle such that a valid solution is a signature which passes *either* ECDSA or Schnorr verification (w.r.t. to the sampled public-key).

4.5 Game Theoretic Analysis

In this section we argue that quantum capable entities, including dishonest ones, are incentivised to claim the bounty, even at the price of relinquishing the loot. To do so, we consider an idealized setting in which exactly two quantum adversaries exist. Furthermore, for each adversary, it is known (to both adversaries) exactly when they would be able to claim the loot and the bounty.

We consider two strategies: the *early strategy*, in which the adversary claims the bounty as soon as possible, and the *late strategy*, in which the adversary claims the bounty as soon as possible *without relinquishing the loot*. That is, if an adversary can claim the bounty at time t_b and the loot at time t_ℓ, then the early strategy is to try to claim the bounty at time t_b; whereas the late strategy is to try to claim the bounty at time $\max\{t_b, t_\ell - w\}$ and the loot at time t_ℓ, where w is the quantum adjustment period between the time the canary is killed and the time the quantum loot is burned (Of course, whether one party's attempt is successful depends on the other party's strategy).

Note that if $t_b > t_\ell - w$ then both strategies coincide, and the adversary becomes *degenerate*.

In Table 3 we analyze two non-degenerate adversaries with all their possible timelines. We use ◖ and ◗ (resp. ≪, ≫) to denote the points in time in which the first (resp. second) adversary is able to claim the bounty and loot, respectively. Note that by the design of the canary, ◖ always happens *before* ◗, since killing the canary is strictly easier than forging a standard ECDSA signature, which is required for claiming (part of) the loot.

Since the pay-offs are symmetric, we assume without loss of generality that the first (faster) player is faster, and can claim the bounty before the second (slower) player. We use ● (resp. ▼) to denote the point in time in which the adversary would claim the bounty if they follow the late strategy. That is, the time difference between ● and ◗ (resp. ▼ and ≫) is always w.

The matrices on the right column show the payoffs for each adversary in each possible timeline given the strategy of both adversaries. The rows of the matrices represent the strategy of the fast adversary, whereas the columns represent the strategy of the slow adversary. The headings "E" and "L" denote the early and late strategies, respectively. The utility of the bounty is denoted by b and the loot by ℓ. The pure Nash equilibria of the payoff matrices are underlined.

We find that in the first two scenarios, the equilibria strategies do not claim the loot.

The effect of decreasing w could be described as moving ● to the right (towards ◗) and, similarly, ▼ towards ≫, where both are moved the same distance without moving any of the other points. The solid

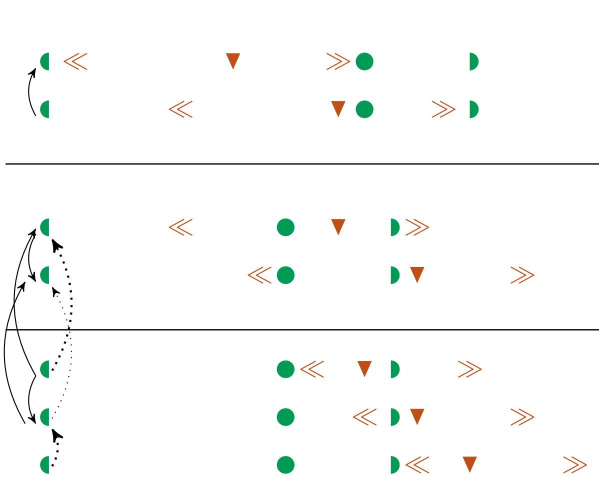
Timelines	Payoffs
	$\begin{array}{ccc} & E & L \\ \hline E & \underline{(b,0)} & \underline{(b,0)} \\ L & (0,b) & (0,b+\ell) \end{array}$
	$\begin{array}{ccc} & E & L \\ \hline E & \underline{(b,0)} & (b,0) \\ L & \underline{(0,b)} & (b+\ell,0) \end{array}$
	$\begin{array}{ccc} & E & L \\ \hline E & (b,0) & (b,0) \\ L & \underline{(b+\ell,0)} & \underline{(b+\ell,0)} \end{array}$

Table 3: Best viewed in color. An analysis of two non-degenerate dishonest quantum adversaries with respect to all possible timelines. The timelines are grouped into scenarios, where each scenario corresponds to a different payoff matrix. Each timeline advances in time from left to right. ◖ (resp. ≪) and ◗ (resp. ≫) represent when the faster (resp. slower) adversary is capable of claiming the bounty and the quantum loot respectively. ● (resp. ▼) represents the earliest point in which the faster (resp. slower) adversary can claim the bounty without forfeiting the loot, and is always at distance w from ◗ (resp. ≫). The payoff matrices list the outcome of the game for both players given their timeline and the strategies they chose. The headings of the matrix rows and columns represent the strategies chosen by both players, where "E" stands for the early strategy, and "L" stands for the late strategy. The rows (resp. columns) of the matrices represent the strategy chosen by the faster (resp. slower) adversary. Each entry of the matrix specifies the outcome for the faster and slower players, in that order. Pure Nash equilibria are denoted by an underline. Solid arrows describe how timelines transform as the waiting time decreases, and dotted arrows describe how timelines transform as the bounty increases.

arrows show how timelines change as w is decreased. We find that as w decreases, we converge into scenarios where Nash equilibria strategies do not claim the loot. The exception is the last timeline, which describes a scenario where one of the adversaries is so powerful that they can claim the loot before the other adversary could claim the bounty. In this scenario, the more powerful adversary will win both the loot and the bounty regardless of the waiting time. This motivates choosing the waiting time to be as small as possible.

Finally, we note that the effect of increasing the bounty could be modeled as follows: an increased bounty might cause a player to increase the investment in building a (small) quantum computer, in the scale needed to win the bounty. This can be viewed visually as moving ◖ and ≪ to the left (i.e., earlier periods in time). However, since the bounty is orders of magnitude smaller than the loot, it does not seem that increasing the bounty will make looting capable adversaries appear much sooner. The effect of earlier times in which the bounty could be claimed on the various scenarios is depicted by the dashed arrows. Note in particular that the last timeline also converges to timelines where the bounty is not taken. This motivates picking a fairly large bounty, so this dynamic would occur.

We do not provide a detailed analysis of the scenario where one (or both) adversaries are degenerate, as it does not provide much insight, and degenerate adversaries transform to non-degenerate ones as w decreases.

5 Quantum-Cautious Spending Methods

In this section, we propose several methods for quantum-cautious spending. Throughout this section, we discuss each method as if it is the only one being implemented. In Section 1.6, we discuss compatibility issues between different methods that should be addressed in any system which implements more than one method, and in Section 5.5 we specify how to securely combine FawkesCoin (Section 5.2) and Lifted FawkesCoin (Section 5.4). The way we propose to combine these methods is detailed in Section 1.7.

Existing quantum-cautious spending methods [BM14,SIZ+18,IKS19] are only applicable to UTXOs whose public-keys have not leaked and require the user to have access to post-quantum UTXOs (or to use a side payment to compensate the miner for including their transaction). Our methods manage to remove this requirement and extend the set of cautiously-spendable UTXOs to include derived UTXOs as well as lost UTXOs created after 2013.

5.1 Leaked UTXOs

So far, we have made a distinction between hashed and non-hashed UTXOs (which we further divided into several sets). In particular, some quantum-cautious spending methods treat hashed UTXOs differently than other UTXOs.

The problem with this is that the operation of any spending method should only depend on data available on the blockchain. However, it is impossible to read off the fact that a UTXO is hashed from the blockchain (as its public-key may have leaked in other ways).

We thus approximate the set of non-hashed UTXO by using the set of *leaked* UTXOs. A UTXO is considered *leaked* if its public-key appears anywhere *on the blockchain*. For the rest of the section, we use the term hashed UTXO to mean a UTXO which is not leaked.

To account for the gap between the definitions, we propose a *good Samaritan* mechanism which allows users to post public-keys not already on the blockchain to the mempool, and Samaritan miners to include them on the blockchain, thus transforming non-leaked UTXOs into leaked ones. The miners obtain no fees for including the public-keys. By allocating 1 Kilobyte for good Samaritan reports, each block could contain about 30 reported addresses. The increase in both block size and the computational overhead of verifying the block is negligible.

Note that the good Samaritan mechanism should only be available in the pre-quantum era, as a quantum adversary could listen for reported public-keys and attempt to steal them.

The purpose of the good Samaritan mechanism is twofold. First, it allows users to report public-keys they encountered outside the blockchain. Second, it allows users who lost their secret-key but still hold their public-key to report it, making their UTXO lost so they could spend it using permissive FawkesCoin when the quantum era arrives (see Section 5.2.2).

5.2 FawkesCoin

FawkesCoin [BM14] is a blockchain protocol that avoids public signature schemes altogether by employing the Fawkes signatures of [ABC+98]. The core idea is that in order to spend a UTXO the user commits to a certain transaction and reveals the transaction once some predetermined period of time has passed (we further discuss the considerations for choosing the waiting time in Section 5.2.5, where we propose a waiting time of 100 blocks). The security of the protocol follows from the observation that if the waiting time is chosen long enough, then after revealing the transaction, an attacker has a negligible chance to complete a commit-wait-reveal cycle before the honest user manages to include the revealed transaction in the blockchain.

In the original FawkesCoin design, the secret-key is a random string r, and the public address for spending to this key is $\mathsf{H}(r)$, where H is some agreed upon collision-resistant hash function. In order to spend a UTXO with address $\mathsf{H}(r)$ to another address $\mathsf{H}(s)$, the user posts $\mathsf{H}(r, \mathsf{H}(s))$ as a commitment, and r as a reveal. Given r and $\mathsf{H}(s)$, anyone can verify that $\mathsf{H}(r)$ and $\mathsf{H}(r, \mathsf{H}(s))$ evaluate correctly.

Bonneau and Miller note that their solution could be integrated into Bitcoin and that a UTXO could be spent by using a hash of a valid transaction as a commitment and the transaction itself as a reveal. This allows cautious spending of pre-quantum hashed UTXOs. They point out that this approach could mitigate "a catastrophic algorithmic break of discrete log on the curve P-256 or rapid advances in quantum computing."

The main obstacle to adopting FawkesCoin is in incentivizing miners to include commitment messages in the blocks in the first place. However, this issue could be completely circumvented if the user already has access to post-quantum UTXOs that they could use to pay the fees. In this section, we assume that it is the case.

While it is desirable that users could use the committed transaction to pay the miner fee, achieving this feature without allowing denial-of-service attacks proves far from trivial. In Section 5.4 we introduce *Lifted FakwesCoin*, a variation of FawkesCoin which allows paying the transaction fee out of the spent UTXO by using lifted signatures (see Section 3).

We present three modes of operations for FawkesCoin, each more permissive than the other. The modes are presented incrementally, each mode increasing the set of spendable UTXOs. The original FawkesCoin introduced in [BM14] is equivalent to the restrictive mode further restricted to only allow spending hashed UTXOs.

5.2.1 Restrictive FawkesCoin Restrictive FawkesCoin allows spending a hashed UTXO by creating a transaction spending it, posting the hash of the transaction as a commitment, and posting the transaction itself to reveal it. It also allows spending *any* UTXO by committing to a derivation seed instead.

Spending a Hashed UTXO*.* The protocol for spending a Hashed UTXO u is as follows:

1. (Spender creates transaction) The spender creates a transaction tx spending u, which includes a standard fee.
2. (Spender creates commitment) The spender creates a transaction having $\mathsf{H}(\mathsf{tx})$ as its payload, which is signed and pays fees using a post-quantum UTXO. We refer to this committing transaction as ctx.
3. (Spender posts commitment) The spender posts ctx to the mempool.
4. (Miners include commitment) The miners include ctx in their blocks, the fee for ctx goes to the miner who included it first, as usual.
5. (Spender waits) Once ctx is included in the blockchain, the spender waits for 100 blocks to be mined above it (see Section 5.2.5).
6. (Spender posts reveal) The spender posts tx to the mempool.
7. (Miners validate) The miners verify that:
   - $\mathsf{H}(\mathsf{tx})$ appears in the payload of a committing transaction ctx at least 100 blocks old, and
   - the UTXO was hashed when the commitment was posted: the public-key used to sign tx does not appear in the blockchain before ctx.

   If tx does not satisfy both conditions, it is considered invalid.
8. (Miners include reveal) The miners include tx in their block. The fee for tx goes to the miner who included it first, subject to the two conditions above.
9. (Receiver waits for confirmation) The receiver considers the transaction completed once tx accumulated six confirmations.

Spending a Derived UTXO*.* To spend a derived UTXO whose address is pk (or a hash thereof), we require the user to commit and reveal a parent extended secret-key xsk_{par} and a derivation path P such that $\mathsf{PK}^{\mathsf{EC}}(\mathsf{Der}(\mathsf{xsk}_{par}, P)) = \mathsf{pk}$ (see Section 2.1.2 and Section 2.7). We stress that once xsk_{par} is revealed, anyone could use FawkesCoin to spend any UTXO whose address has been derived from xsk_{par}. Hence, to maintain the safety of their funds, the user must commit to *all* UTXOs whose addresses were derived from xsk_{par} before they start revealing them.

The protocol for spending a derived UTXO u whose address corresponds to a public-key pk is as follows (the differences with the protocol for spending a hashed UTXO are underlined):

1. (Spender creates transaction) The spender creates a transaction tx spending u, which contains in its payload (xsk_{par}, P) such that $\mathsf{PK}^{\mathsf{EC}}(\mathsf{Der}(\mathsf{xsk}_{par}, P)) = \mathsf{pk}$ and includes a standard fee.
2. (Spender creates commitment) The spender creates a transaction having $\mathsf{H}(\mathsf{tx})$ as its payload, which is signed and pays fees using a post-quantum UTXO. We refer to this committing transaction as ctx.
3. (Spender posts commitment) The spender posts ctx to the mempool.
4. (Miners include commitment) The miners include ctx in their blocks, the fee for ctx goes to the miner who included it first, as usual.
5. (Spender waits) Once ctx is included in the blockchain, the spender waits for 100 blocks to be mined above it (see Section 5.2.5).
6. (Spender posts reveal) The spender posts tx to the mempool.
7. (Miners validate) The miners verify that:
   - $\mathsf{H}(\mathsf{tx})$ appears in the payload of a committing transaction ctx at least 100 blocks old, and
   - $\mathsf{PK}^{\mathsf{EC}}(\mathsf{Der}(\mathsf{xsk}_{par}, P)) = \mathsf{pk}$

   If tx does not satisfy both conditions, it is considered invalid.
8. (Miners include reveal) The miners include tx in their block. The fee for tx goes to the miner who included it first, subject to the two conditions above.
9. (Receiver waits for confirmation) The receiver considers the transaction completed once tx accumulated six confirmations.

Remark 4. Note that when spending a derived UTXO, the pre-quantum signature is not actually required for validation, as suffices to verify $\mathsf{PK}^{\mathsf{EC}}((\mathsf{xsk}_{par})_{(i,p)}) = \mathsf{pk}$. We include the signature in tx so that restrictive FawkesCoin could be implemented as a soft-fork (see p. 16). If FawkesCoin is implemented in a hard-fork, the signature could be removed to conserve space.

Hashed/Derived Confirmation Times. The confirmation time is the length of a single commit-wait-reveal cycle (which we propose setting to 100 blocks in Section 5.2.5), followed by waiting the current number of confirmation blocks once the reveal message is included (e.g., six blocks in Bitcoin).

Transaction Size Increase. The component of the transaction dominating its size is the signature of the post-quantum UTXO, which is about an order of magnitude larger than transaction spending a pre-quantum UTXO (see Section 2.5). However, a single post-quantum transaction can be used to spend several pre-quantum UTXOs (either by committing to several transaction, or by committing to a transaction spending several UTXOs), making the size increase additive.

Spendability Threshold. Since the fee is paid using the post-quantum UTXO, the only obstruction to spendability is if the UTXO is less valuable then the fees required to post it. Disregarding the post-quantum UTXO (see p. 9), the size of a FawkesCoin transaction is slightly larger than spending it usually (exactly by how much depends on whether the transaction spends a hashed or derived UTXO, and on whether we are in a soft- or hard-fork), whereby the spendability threshold stays the same up to a factor close to 1.

Requires a Post-Quantum UTXO*.* A post-quantum UTXO is required in order to pay the transaction fee.

Can be Implemented in Soft-Fork. Commitments are ordinary post-quantum transactions whereas reveals are ordinary pre-quantum transactions, whereby they would also be accepted by outdated nodes. However, a hard-fork implementation could conserve block space as the pre-quantum signature can be removed from derived UTXOs.

5.2.2 Unrestrictive FawkesCoin *Unrestrictive* mode extends the functionality of restrictive mode by providing a way to spend a naked UTXO. Naked UTXOs are spent like hashed UTXOs, except the user must leave a *deposit* as valuable as the UTXO they are trying to spend. After the transaction is revealed, it goes into a long *challenge period* during which any user can post a *fraud proof* by spending that same transaction in FawkesCoin *using the derivation seed*. If a fraud proof is posted, the revealed transaction is considered invalid, and the deposit goes to the address of the user who posted the fraud proof.

Spending Naked UTXO*s.* The protocol for spending a naked UTXO u is as follows:

1. (Spender creates transaction) The spender creates a transaction tx spending u, which also spends a post-quantum UTXO d called the *deposit*. The value of d must be at least the value of u plus the fees paid for including tx.
2. (Spender creates commitment) The spender creates a transaction having $H(\mathsf{tx})$ as its payload, which pays fees using a post-quantum UTXO. We refer to this committing transaction as ctx.
3. (Spender posts commitment) The spender posts ctx to the mempool.
4. (Miners include commitment) The miners include ctx in their blocks, the fee for ctx goes to the miner who included it first, as usual.
5. (Spender waits) Once ctx is included in the blockchain, the spender waits for 100 blocks to be mined above it (see Section 5.2.5).
6. (Spender posts reveal) The spender posts tx to the mempool.
7. (Miners validate) The miners verify that $\mathsf{H}(\mathsf{tx})$ appears in the payload of a committing transaction ctx at least 100 blocks old. If tx does not satisfy both conditions, it is considered invalid.
8. (Miners include reveal) The miners include tx in their block. The fee for tx goes to the miner who included it first, subject to the condition above.
9. (Challenge period start) The transaction tx enters a *challenge period* of one year.
10. (Owner can post fraud proof) During that period, any user holding the derivation seed for the address of u can post a *fraud proof*: a transaction fp spent using the FawkesCoin protocol for derived UTXOs spending the UTXO u. If a fraud proof is posted, the transaction tx is considered invalid. The miner is paid the fee they were supposed to for posting tx fro the deposit, and the rest of the deposit goes to same address fp is spent to.
11. (Receiver waits for confirmation) If no fraud proof was posted, the receiver consider the transaction tx completed six blocks after the challenge period is over.

Remark 5. For simplicity, we presented the method in a manner that requires a hard-fork: if a fraud proof is posted, then the deposit needs to be spent to a different address than the one specified in its output. This could be rectified by instead requiring that the deposit output is spent to an anyone-can-spend address, as explained in p. 16.

Naked Confirmation Times. The challenge period is very large to allow honest users ample time to notice if anyone attempted to steal their transactions and respond accordingly (see Section 5.2.6).

Offline Users Risked. Users holding naked derived UTXOs have to monitor the network for attempts to spend their money. In particular, if an adversary knows of a user that would not be online for a duration longer than a dispute period (say, if he's denied modern society and became a recluse, joined an Amish community, or is just forever trapped beyond the event horizon of a black hole), they can safely steal their naked derived UTXOs.

5.2.3 Permissive FawkesCoin

Permissive mode extends the functionality of unrestrictive mode by providing a way to spend *any* leaked UTXO without providing any proof of ownership. That is, we allow anyone to spend any leaked UTXO. However, to spend a leaked UTXO without providing a valid signature or a derivation seed, the spender must provide a deposit and wait for a lengthy challenge period, exactly like naked UTXOs are spent in unrestrictive mode. The deposit acts to deter adversaries from attempting to steal UTXOs, as they can not know whether the owner of the UTXO they are trying to steal can produce a fraud proof.

Spending Lost UTXO*.* The protocol for spending a lost UTXO u is as follows:

1. (Spender creates transaction) The spender creates a transaction tx spending u, which also spends a post-quantum UTXO d called the *deposit. The* UTXO *need not include a signature on* u. The value of d must be at least the value of u plus the fees paid for including tx.

2. (Spender creates commitment) The spender creates a transaction having $H(\mathsf{tx})$ as its payload, which pays fees using a post-quantum UTXO. We refer to this committing transaction as ctx.
3. (Spender posts commitment) The spender posts ctx to the mempool.
4. (Miners include commitment) The miners include ctx in their blocks, the fee for ctx goes to the miner who included it first, as usual.
5. (Spender waits) Once ctx is included in the blockchain, the spender waits for 100 blocks to be mined above it (see Section 5.2.5).
6. (Spender posts reveal) The spender posts tx to the mempool.
7. (Miners validate) The miners verify that H(tx) appears in the payload of a committing transaction ctx at least 100 blocks old. If tx does not satisfy both conditions, it is considered invalid.
8. (Miners include reveal) The miners include tx in their block. The fee for tx goes to the miner who included it first, subject to the condition.
9. (Challenge period start) The transaction tx enters a *challenge period* of one year.
10. (Owner can post fraud proof) During that period, any user holding the derivation seed for the address of u can post a *fraud proof*: a transaction fp spent using the FawkesCoin protocol for derived UTXOs spending the UTXO u. If a fraud proof is posted, the transaction tx is considered invalid. The miner is paid the fee they were supposed to for posting tx from d, and the remaining coin in d goes to the same address fp is spent to.
11. (Receiver waits for confirmation) If no fraud poof was posted, the receiver consider the transaction tx completed six blocks after the challenge period is over.

Remark 6. In the specification above we chose the value of the deposit d to be as high as the value of the spent UTXO u (plus the fees for including tx). This implies that an adversary has a negative expected profit from such an attack as long as they can't make an educated guess that a particular UTXO is lost with a probability higher than 1/2. It might be the case that such educated guesses are feasible, and if so the value of the deposit should be increased appropriately. Setting the value of the deposit to $\frac{p}{1-p}$ times the value of the spent output implies that attempting to steal a UTXO using permissive FawkesCoin has negative expected profit, unless knows that UTXO is lost with probability at least p. Increasing p makes such attacks less feasible at the cost of making spending lost UTXOs less affordable.

Lost Confirmation Times. Same as naked confirmation times.

Requires hard-fork. This method allows spending UTXOs without producing a signature, which can not be implemented in a soft-fork (see p. 16).

5.2.4 Quantum Loot

A UTXO is considered *quantum loot* if a quantum adversary is able to steal it without taking a meaningful risk. In restrictive mode, it is only possible to spend a UTXO if its public-key is hashed, or if the user has access to a derivation seed. Hence, in restrictive mode eliminates the quantum loot entirely.

One might argue that unrestrictive mode increases the quantum loot to include stealable UTXOs. However, such UTXOs are actually protected by the following dynamic: holders of derived UTXOs have an incentive to falsely declare them as lost, hoping to bait an adversary to try stealing them so they could claim the deposit. Furthermore, owners of naked UTXOs are incentivized to spend their UTXOs since they could not provide a fraud proof, whereas owners of leaked derived UTXOs have no such incentive.

The same argument applies for lost UTXOs in permissive mode.

5.2.5 The Wait Time

So far, we have yet to specify the length of a commit-wait-reveal cycle. That is, the number of blocks a user must wait after committing to a transaction before they can reveal it. Longer waiting times make the economy more secure but less usable.

The main consideration when choosing the wait time is *forking security*: the wait time should be longer than any plausible reorganization of the blockchain. Recall that the discussion in p. 16 points out that coinbase transactions require a considerably longer confirmation than a standard transaction. The justification for that is that if a reorganization reverts a coinbase transaction, then all transactions which spend money minted by that coinbase transaction become invalid too. By setting the cooldown time of a coinbase transaction high enough to guarantee that a coinbase transaction is *never* reverted, it is guaranteed that no reorganization can make a valid transaction invalid (as long as the owners of the UTXOs used as input therein do not attempt double spending them).

Our considerations are more similar to coinbase transactions. Once a reveal is posted, an attacker could use it to post competing commit and reveal messages. If the reorg is deep enough to revert the honest *commit* message, then it is possible that the adversarial commit and reveal will be included itself, making the honest transaction invalid.

Thus, taking a cue from Bitcoin, we propose a 100 blocks waiting time.

One might argue that 100 waiting blocks is too long as it slows down the spending time too much. It is also arguable that this concern becomes more pressing as it applies to many users (whereas coinbase cooldown only applies to miners). We have several responses to this objection:

- It is better to err on the side of caution. The consequence of choosing the wait time too short is that spending pre-quantum UTXOs becomes insecure, and quantum-cautious spending becomes *impossible*. This is arguably more detrimental to the economy than a predictable slowdown.
- The effects of quantum mining on fork rates and reorganization depths are still not yet understood. There is evidence that quantum mining increases the orphan rate [Sat20], and it could be the case that there are more consequences we are not yet aware of.
- Users who prefer short waiting times over small transaction sizes (that is, prefer paying faster over paying less) could use post-quantum signatures at confirmation speeds similar to today.
- It is possible to allow users to set their own waiting time when creating the UTXO. It is important to impose a *default* waiting time to prevent front-running attacks. However, there is no harm in allowing users to set a lower waiting time if they choose (conversely, users who feel that the 100 blocks waiting time is insufficient could specify higher waiting times).

5.2.6 Length of the Challenge Period We propose a challenge period of *one year*. We argue that the challenge period should be very long for two reasons:

- We want to prevent a situation where a UTXO was successfully stolen because the user did not notice an attempted steal in time or had no time to gather the resources required to post a proof of fraud. A year-long period would give users ample time to notice the attempt and make preparations.
- Any UTXO that could be spent in restrictive mode could still be spent in the other modes without any challenge period. Even if an adversary is trying to steal the UTXO, proving fraud is just done by regularly spending the UTXO. The only effect of allowing permissive mode on leaked derived UTXO is that a steal attempt might *force* a user to spend a UTXO earlier than they desired to, and in that case, a long period allows the user more flexibility choosing when to spend it.

5.3 Lifted Spending

In *lifted spending*, we simply replace the pre-quantum signature with a lifted version thereof (see Section 3), depending on the type of the UTXO.

Spending Hashed UTXOs. To spend a hashed UTXO, the user signs it with a key-lifted signature (see Section 3.2).

Spending Derived UTXOs. To spend a derived UTXO, the user signs it with a seed-lifted signature (see Section 3.4).

Remark 7. We stress that the security of lifted spending relies on the policy that key-lifted signatures could only be used to spend hashed UTXOs, and leaked UTXOs must be spent using seed-lifted signatures. This follows as quantum adversaries exposed to the public-key can compute the secret-key, which they could then use to create valid key-lifted signatures (see Section 3.2).

Hashed/Derived Confirmation Times. Besides using a different signature, the spending procedure is the same as in the current state, and so it has the same confirmation times.

Transaction Size Increase. As we discuss in Section 3.4.1, currently the size of a lifted signature is of the order of hundreds of kilobytes.

High Spendability Threshold. Since the UTXO is spent directly, the only barrier to spendability is that it is valuable enough to cover the cost of a transaction fee. Since lifted signatures are several orders of magnitude larger than pre-quantum signatures (see Section 3.4.1), the spendability threshold for using lifted signatures also grows several orders of magnitude larger.

Works Without Post-Quantum UTXO. The fee for spending coins is taken from the spent UTXO exactly like it is currently, so no additional coin is required.

Requires a hard-fork. Recall that in ECDSA the public-key is recoverable from a signature, whereby it is insecure to include an ECDSA signature in the transaction[9]. This means that valid lifted spending transactions can not be shaped as valid ECDSA transactions. Hence a hard-fork is required.

5.4 Lifted FawkesCoin

The *Lifted FawkesCoin* method combines FawkesCoin (see Section 5.2) and signature lifting (see Section 3) to allow spending hashed and derived UTXOs without using a side payment or a post-quantum UTXO.

As briefly touched upon in Section 5.2, a major difficulty in FawkesCoin is dealing with *denial-of-service attacks*. Namely, we need to prevent adversaries from cheaply spamming the blockchain. However, we need to allow users to post commitments, and at the time of commitment, it is not known what the commitment is for and if it will ever be revealed. Since the user is only charged a fee during the reveal, this opens up attacks where fake commitments are posted, or commitments for the same UTXO are posted several times, wasting valuable blockchain space for free.

In Section 5.2, we overcame this issue by requiring the user to pay *at commitment time* by using a post-quantum UTXO. However, it is desirable that users without access to post-quantum UTXOs could quantum-cautiously spend their money.

The authors of [BM14] propose two approaches to the denial-of-service problem, but both fall short of solving the problem:

- **Zero-Knowledge** The authors suggest that the user provides a ZK proof that the UTXO is valid and pays a "useful amount" of fees. However, as long as the transaction has not been revealed, there is nothing forcing the user to finalize the transaction or preventing them from committing to the same UTXO several times.
- **Merkle Trees** The authors suggest that the miner arranges all commitments in a Merkle tree and only posts the root, requiring users to post Merkle proofs in their reveal message. In this scenario, a spammer can not increase the size of the Merkle root, but by spamming the miner, they could increase the size of a Merkle *proof*.

In the current method, we propose to avoid denial-of-service attacks by using signature lifting (see Section 3). The idea is to limit the time the user has to post a commitment. If a user fails to post a commitment within this time, the miner can post the signature as proof and claim the coin the user tried to spend as their own. To prevent users from committing to the same UTXO on several blocks, we require the user to reveal in advance the UTXO u they are spending, and the miner has to post u to the blockchain along the commitment. No other commitments for that UTXO will be accepted as long as the commitment does not

[9] In contrast, in some implementations of the Schnorr signature scheme, including the one used in Bitcoin, recovering the public-key from a valid signature requires inverting a hash function and is thus considered infeasible, even for quantum adversaries.

expire. To prevent miners from abusing this power to censor UTXOs, if the user failed to reveal a transaction spending u, the miner is *required* to post a proof of ownership. If a commitment to a UTXO was posted, but neither a reveal nor a proof of ownership was posted before the UTXO expired, we assume the miner delayed the UTXO and penalize them to compensate the owner.

In more detail, this is the Lifted FaweksCoin protocol for spending a UTXO u:

1. (Spender creates transaction) The spender prepares a transaction tx spending u and paying fees as usual.
2. (Spender creates commitment) The spender prepares a triplet $(\mathsf{H}(\mathsf{tx}), \sigma, u, \alpha)$, where α is the amount of fees paid by tx, and σ is a lifted signature on $(\mathsf{H}(\mathsf{tx}), \alpha)$ which acts as a *proof of ownership* of u. The type of the signature σ depends on the type of UTXO being spent, as explained below.
3. (Spender posts commitment) The spender posts $(\mathsf{H}(\mathsf{tx}), \alpha)$ to the mempool
4. (Miners validate commitment) If $\mathsf{H}(u)$ appears in a previous commitment that has not yet expired, the transaction is invalid.
5. (Miners include commitment) The miner posts $(\mathsf{H}(\mathsf{tx}), H(u), \alpha)$ to the blockchain. The miner does not get a fee at this point, and the source of the fee is not yet determined, but an honest miner is guaranteed they will either get the fee or will have a chance to post σ in order to claim all the coins in u.
6. (Spender waits) After $(\mathsf{H}(\mathsf{tx}))$ is posted to the blockchain, the user waits for 100 blocks.
7. (Spender posts reveal) The user must post tx to the mempool soon enough so it will be included within the following 100 blocks. tx is considered invalid if it does not include a fee of α. The fee paid for including tx is split equally between the miner who included tx and the miner who included $(H(\mathsf{tx}))$.
8. (Receiver waits for confirmation) The receiver considers the transaction tx completed six blocks after the reveal was posted.
9. (Miner posts proof of ownership) If the user fails to have tx included to the blockchain within 100 blocks, the miner must post the proof of ownership σ to the blockchain within 100 blocks. If the miner posts the proof of ownership, they are given the entire value of u,
10. (Compensation of delayed UTXOs) If the miner fails to post the proof of ownership in time, they pay 3.35% of the value of u to the address of u.

Spending Hashed UTXO*s.* To spend a hashed UTXO, the user follows the protocol above using a key-lifted signature (see Section 3.2).

Spending Derived UTXO*s.* To spend a hashed UTXO, the user follows the protocol above using a seed-lifted signature (see Section 3.4), and includes msk, P in the payload of tx. It must hold that $\mathsf{PK}^{\mathsf{EC}}(\mathsf{Der}(\mathsf{msk}, P)) = \mathsf{pk}$ for the revealing transaction to be considered valid.

Hashed/Derived Confirmation Times. The confirmation times are the same as in regular FawkesCoin.

Transaction Size Increase. The transaction size is the same size of a regular *pre*-quantum transaction plus 64 bytes for commitment and the hash of the UTXO.

Spendability Threshold. While transaction sizes are small, they should be valuable enough to cover the cost of posting the proof of ownership. Including a less valuable UTXOs imposes a risk on the miner, so it will probably not be included. We further discuss the size of lifted signatures in Section 3.4.1.

Works Without Post-Quantum UTXO*.* The protocol makes it secure to pay the fees from the original UTXO, so no additional coin is required.

Limited Delay Attacks. A miner can delay spending an arbitrary UTXO by posting its hash alongside a fake commitment. However, once the transaction is expired, they will compensate the user.

Requires a Hard-Fork. In case the user fails to reveal the transaction in time, the miner is awarded the coins in the UTXO even though they do not have access to a valid signature. This requires nodes to consider a transaction valid even though it does not contain a signature for the UTXO it is trying to spend. As we explain in p. 16, this requires a hard-fork.

Offline Users Not Risked. Derived UTXO are only spendable using the derivation seed, so naked and lost UTXOs can not be spent using this method.

5.4.1 Delay Attack Fines The fine for delay attacks should reflect the damage done to the user. However, it should not be chosen too large, as it would make it hard for miners to include commitments on highly valuable UTXOs. We propose setting the fine to be the equivalent of an *annual* 100% interest, paid over the period the transaction was delayed.

Note that if rotation is employed with the periods we proposed in Section 1.7, then a delay attack actually prevents the user from spending their UTXO in the current Lifted FawkesCoin epoch, forcing them to wait as much as 2,500 blocks. Hence, we propose that the interest should be calculated as an annual interest of 100% accrued over a period of 25,000 minutes, which is 3.35% of the value of the UTXO.

The miner should be able to cover delay fines for all UTXOs whose hash is used in a Lifted FawkesCoin commitment. Let the *guaranteed coinbase value* be the value of the block reward and all transactions which are not Lifted FawkesCoin commitments (this is a lower bound on the block reward in case the miner is honest. Obviously, the final block reward could be lower if the miner performs delay attacks). If the guaranteed coinbase value is lower than the sum of all required deposits, then the miner needs to include an additional transaction covering the difference.

5.4.2 Obtaining Commitment Fees When processing a block, it is impossible to know how much fee the miner accrued for including Lifted FawkesCoin commitment. The most straightforward solution is, once the transaction is revealed, to create a UTXO spending the fee to an address specified in the block which contains the commitment. The problem with this solution is that it creates many UTXOs whose value is comparable to the cost of spending them, thereby increasing the amount of dust (see p. 17).

We propose to overcome this by delaying the coinbase processing. The Lifted FawkesCoin protocol guarantees that the exact amount of fees accrued by the miner is resolved after the expiry period is over (which amounts to 300 blocks using the 100 blocks waiting time we recommend in Section 5.2.5). Hence, processing the coinbase transaction of the block could be delayed for this length of time. This could be implemented by having the coinbase transaction of each block pay to the block that appears three waiting periods before it, if that block contains Lifted FawkesCoin commitments. The Kaspa cryptocurrency resolves a similar situation (where the final transaction fees are not known at the time the blocks are created) by implementing such an approach (see relevant documentation). Note that coinbase cooldown *starts* to take effect only after the expiry period is *finished*.

A disadvantage of this approach is that the elongated wait times hold for the entire coinbase transaction, including the block reward and the fees for transactions not made using Lifted FawkesCoin. This could be resolved by allowing the coinbase transaction to include the block reward and all fees except the Lifted FawkesCoin fees, and having each block reward Lifted FawkesCoin fees to the block appearing three wait periods before it (if there are any).

5.4.3 Increasing Lifted FawkesCoin's Throughput In the current design, a miner only has a limited time to post a fraud-proof. This has the effect that the number of commitments risk-averse miners will agree to include is, at most, the number of fraud-proofs the network can support. Otherwise, they take the risk that they would have to post more fraud-proofs than possible, eventually causing them to pay delay fines albeit being honest.

Given that fraud proofs are much larger than commitments, this greatly decreases the commitment throughput. For example, if a block can only contain one fraud-proof, then no more than 300 commitments will be included during a 500 blocks long FawkesCoin epoch.

Our approach to increasing the throughput is by extending the expiry time of Lifted FawkesCoin commitments in scenarios where many fraud-proofs are posted. If at the end of the Lifted FawkesCoin epoch (see Section 5.5) sufficiently many fraud-proofs were posted, a new Lifted FawkesCoin epoch starts immediately, during which fraud-proofs for commitments from previous epochs are allowed. We refer to this new epoch as an *extension*.

In more detail, we propose the following:

- Currently, if the spender failed to post a reveal, the miner has a period of 100 blocks to post a fraud-proof. We modify this rule such that the miner is allowed to post the fraud-proof at any time up to the *end of the epoch*. However, we still do not allow commitments to be posted within the last 300 blocks of the epoch, to allow miners a period of *at least* 100 blocks to post a proof.
- Say that 100 blocks can contain up to k fraud-proofs. At the end of the epoch:
  - If more than $k/2$ fraud-proofs were posted in the last 100 blocks, then:
    - Do not pay the miner's deposits to the users,
    - Start a new Lifted FawkesCoin epoch,
    - During this epoch, allow miners to post fraud-proofs for any unrevealed commitment that has not yet expired (even if the commitment was included in a previous epoch).
  - Else:
    - For each non-expired unsettled commitment, pay the fine left for that commitment to the address of the UTXO, and consider the commitment expired.
    - Do not initiate a new Lifted FawkesCoin epoch, but rather allow the epoch rotation to continue.

More generally, one can set the threshold to trigger an extension at kp fraud-proofs for any $0 < p < 1$. However, we now discuss possible attacks on this mechanism and conclude that $p = 1/2$ is a natural choice (or more precisely, that p should be *at most* $1/2$, but should not be set too low either).

There are two ways to abuse the extension mechanism: either by *forcing* or by *denying* an extension.

In order to force an extension, an adversary must post kp fraud proofs. The consequence of such an attack is threefold:

- It prevents users from using non-lifted FawkesCoin and in particular prevents spending lost UTXOs.
- Inability to use non-lifted FawkesCoin also makes it impossible to spend pre-quantum UTXOs whose value is below the cost of posting a fraud-proof.
- It extends the length of ongoing delay attacks without increasing the delay attack fine eventually paid to the owners of the attacked UTXO.

Note that during the attack, users can still use Lifted FawkesCoin and spend post-quantum UTXOs.

The adverse effects of this attack are disruptive but do not last once the attack is over, and maintaining the attack is very expansive: note that the total size of the maximal amount of fraud-proofs a block can contain is at least half a block, hence, the cost of posting kp fraud-proofs is at least as high as the fee for consuming a space equivalent to $50p$ blocks (moreover, maintaining such an attack for an extended period wastes a lot of space, plausibly increasing the cost of posting fraud-proofs). In particular, if $p = 1/2$, the cost of forcing a *each* extension is at least as high as the cost of block space equivalent to 25 blocks. Setting p too low might make such attacks affordable. We point out that the cost of an extension forcing attack could also be increased by increasing the length of the lifted FawkesCoin epoch, whereby increasing k.

An extension *denying* attack is more dangerous since it may be *profitable*. On the other hand, it can only be carried by a miner with a fraction of $q > 1 - p$ of the total hash rate. Such an adversarial miner can deny an extension by simply refusing to include fraud-proofs in her blocks. The adversarial miner can post many commitments to the mempool, so that they would be included by honest miners, and then deny the extension. By the end of the epoch, at most $(1-q)k$ of the corresponding fraud proofs will have been posted, and the adversarial miner would gain the delay attack fines for the remaining commitments.

Hence, we strongly recommend setting $p \leq 1/2$. This assures that such an attack can be only carried by $> 50\%$ attackers (recall that such powerful attackers can already severely damage the network in many other ways, known as 51% attacks).

5.5 Combining FawkesCoin and Lifted FawkesCoin

As briefly touched upon in Section 1.6, allowing FawkesCoin and Lifted FawkesCoin to operate in tandem compromises their security. To overcome this problem, we suggest segregating the operation of these solutions into epochs, where we expect Lifted FawkesCoin users to complete a commit-wait-reveal cycle within an epoch.

However, this still allows a vulnerability when spending derived transactions using Lifted FawkesCoin, since such a spend requires exposing msk, which could then be used to spend any other transaction created by the same wallet using non-Lifted FawkesCoin.

The most immediate solution to that is to modify non-Lifted FawkesCoin such that spending a derived UTXO requires committing and revealing the *seed*. But this has the unfortunate implication that a user must commit to their entire wallet before they could reveal it.

To avoid this issue, we keep a *registry* of known keys in the hierarchy. Let xsk_{par} be a key used to spend a derived transaction (either as a parent key in non-Lifted FawkesCoin or as msk in Lifted FawkesCoin). It is considered invalid to use any key derivable from xsk_{par} to spend a derived UTXO in non-Lifted FawkesCoin.

The problem is that it is impossible to exhaust the entire space of derivable keys. Typically, hardware wallets use a very limited space of addresses. We can thus agree on a reasonably small set of *regular* derivation paths $\mathcal{D}$ assuming that most derived addresses of the blockchain are of the form msk_P where $P \in \mathcal{D}$ and msk is a master secret-key of an existing HD wallet.

To account for users who might have a UTXO whose derivation path is not in $\mathcal{D}$, we can allow users to post messages of the form $(H(\mathsf{xsk}), P_1, \ldots, P_k)$ to the blockchain. Once the key xsk is revealed, the paths $P_1, \ldots, P_k$ will be checked along with the regular derivation paths. Let $\mathcal{D}_{\mathsf{xsk}}$ contain the set $\mathcal{D}$ along with any irregular path which appeared alongside $H(\mathsf{xsk})$ on the blockchain, and let $\mathcal{K}_{\mathsf{xsk}}$ contain all keys of the form $\mathsf{xsk}_{P'}$ where P' is a prefix of some $P \in \mathcal{D}_{\mathsf{xsk}}$. The registry will store a lost of tuples of the form $(\mathcal{K}_{\mathsf{xsk}}, b_{\mathsf{xsk}})$ where b_{xsk} is the height of the block where xsk was included. Any FawkesCoin transaction spending a derived UTXO using the key xsk_{par} will be considered invalid if there is some xsk such that $\mathsf{xsk}_{par} \in \mathcal{K}_{\mathsf{xsk}}$ and the transaction was *committed* in a block whose height is at least b_{xsk}.

In addition, any UTXO whose pk is in $\mathcal{K}_{\mathsf{xsk}}$ for some xsk is considered leaked, and cannot be spent as if it is hashed.

6 Open Questions and Further Research

- **Reducing the size of lifted signatures** The most pressing issue is the size of lifted signatures, discussed in Section 3.4.1. The size of the signature directly determines the spendability threshold one using Lifted FawkesCoin. Estimating and optimizing the sizes of the signatures we use is an important next step toward implementing our solution.
- **Seed-lifting without exposing msk** The seed-lifted scheme we present in Section 3.4 requires exposing the master secret-key of the HD wallet. While this requirement does not affect the security of our solution, it does cause some inconveniences, so devising a method for cautiously spending derived UTXOs without exposing msk might be desirable. There are two main drawbacks to exposing msk. First, exposing msk implies that once a single commitment has been revealed, the entire wallet must be spent using lifted FawkesCoin (see Section 5.5). Another concern is that HD wallets use the same seed to derive keys to many different cryptocurrencies, so carelessly exposing msk on one currency might compromise UTXOs from different currencies whose address was derived from the same seed.

  The most direct approach to spend a derived UTXO with derivation path P is to instantiate *Picnic* with the function $\mathsf{Der}(\mathsf{KDF}(\cdot), P)$. However, this requires *Picnic* to be secure when using the same secret-key in different instantiation. It is unclear whether *Picnic* is secure against such attacks, though it is very easy to modify *Picnic* such that it remains $\mathsf{EUF\text{-}CMA}$ secure, but becomes completely broken against such attacks (e.g. by modifying the signature to contain the ith bit of the secret-key, where (i, s) is the first step in P). Hence, making this approach secure requires introducing a formal security notion that prohibits such attacks, and proving that *Picnic* (or some modification thereof) satisfies this stronger form of security. We point out that another drawback of this approach is that it greatly increases signature sizes.

  We leave the problem of removing the need to expose msk, either following the approach above or by coming up with a different solution, to future research.

- **Instantiating the canary** The discussion in Section 4.2 suggests that the canary puzzle should be forging a signature of an ECDSA scheme instantiated with an elliptic curve similar to secp256k1. However, it still remains to choose a particular curve, and provide a way to sample a nothing-up-my-sleeve public-key for that curve.
- **Further analysis of canary adversaries** Our game theoretic analysis in Section 4.5 assumes the parties have perfect information about their advesary's capabilities. One step towards a more realistic model is to consider the game theory of canaries in the setting of imperfect information.

References

AASA+20. G. Alagic, J. Alperin-Sheriff, D. Apon, D. Cooper, Q. Dang, J. Kelsey, Y.-K. Liu, C. Miller, D. Moody, R. Peralta, et al. Status report on the second round of the NIST post-quantum cryptography standardization process. *US Department of Commerce, NIST*, 2, 2020.

ABC+98. R. J. Anderson, F. Bergadano, B. Crispo, J. L. andnumber Charalampos Manifavas, and R. M. Needham. A New Family of Authentication Protocols. *ACM SIGOPS Oper. Syst. Rev.*, 32(4):9–20, 1998.

ABL+18. D. Aggarwal, G. Brennen, T. Lee, M. Santha, and M. Tomamichel. Quantum Attacks on Bitcoin and How to Protect Against Them. *Ledger*, 3, Oct 2018.

ARS+15. M. R. Albrecht, C. Rechberger, T. Schneider, T. Tiessen, and M. Zohner. Ciphers for MPC and FHE. In E. Oswald and M. Fischlin, editors, *Advances in Cryptology - EUROCRYPT 2015 - 34th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Sofia, Bulgaria, April 26-30, 2015, Proceedings, Part I*, volume 9056 of *Lecture Notes in Computer Science*, pages 430–454. Springer, 2015, Cryptology ePrint Archive: Report 2016/687.

Bar21. B. Barak. An Intensive Introduction to Cryptography. https://intensecrypto.org/public/index.html, 2021. Accessed: 2022-02-23.

BCLN16. J. W. Bos, C. Costello, P. Longa, and M. Naehrig. Selecting elliptic curves for cryptography: an efficiency and security analysis. *J. Cryptogr. Eng.*, 6(4):259–286, 2016, Cryptology ePrint Archive: Report 2014/130.

BDD+17. R. Bhaumik, N. Datta, A. Dutta, N. Mouha, and M. Nandi. The Iterated Random Function Problem. In T. Takagi and T. Peyrin, editors, *Advances in Cryptology - ASIACRYPT 2017 - 23rd International Conference on the Theory and Applications of Cryptology and Information Security, Hong Kong, China, December 3-7, 2017, Proceedings, Part II*, volume 10625 of *Lecture Notes in Computer Science*, pages 667–697. Springer, 2017, Cryptology ePrint Archive: Report 2017/892.

BDF+11. D. Boneh, Ö. Dagdelen, M. Fischlin, A. Lehmann, C. Schaffner, and M. Zhandry. Random Oracles in a Quantum World. In D. H. Lee and X. Wang, editors, *Advances in Cryptology - ASIACRYPT 2011 - 17th International Conference on the Theory and Application of Cryptology and Information Security, Seoul, South Korea, December 4-8, 2011. Proceedings*, volume 7073 of *Lecture Notes in Computer Science*, pages 41–69. Springer, 2011, Cryptology ePrint Archive: Report 2010/428.

BDTJ18. L. Breidenbach, P. Daian, F. Tramèr, and A. Juels. Enter the Hydra: Towards Principled Bug Bounties and Exploit-Resistant Smart Contracts. In W. Enck and A. P. Felt, editors, *27th USENIX Security Symposium, USENIX Security 2018, Baltimore, MD, USA, August 15-17, 2018*, pages 1335–1352. USENIX Association, 2018, Cryptology ePrint Archive: Report 2017/1090.

BHK+19. D. J. Bernstein, A. Hülsing, S. Kölbl, R. Niederhagen, J. Rijneveld, and P. Schwabe. The SPHINCS$^+$ Signature Framework. In L. Cavallaro, J. Kinder, X. Wang, and J. Katz, editors, *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security, CCS 2019, London, UK, November 11-15, 2019*, pages 2129–2146. ACM, 2019, Cryptology ePrint Archive: Report 2019/1086.

Bit22a. Bitcoin Core. Secp256k1. https://en.bitcoin.it/wiki/Secp256k1, 2022. Accessed: 2022-10-20.

Bit22b. BitMEX Research. Bitcoin Address Re-use Statistics. https://blog.bitmex.com/bitcoin-address-re-use-statistics/, 2022. Accessed: 2022-05-30.

BL95. D. Boneh and R. J. Lipton. Quantum Cryptanalysis of Hidden Linear Functions (Extended Abstract). In D. Coppersmith, editor, *Advances in Cryptology - CRYPTO '95, 15th Annual International Cryptology Conference, Santa Barbara, California, USA, August 27-31, 1995, Proceedings*, volume 963 of *Lecture Notes in Computer Science*, pages 424–437. Springer, 1995.

Blo22. Blockchain.com. Unspent Transaction Outputs Chart. https://www.blockchain.com/charts/utxo-count, 2022. Accessed: 2022-09-10.

BM14. J. Bonneau and A. Miller. Fawkescoin - A Cryptocurrency Without Public-Key Cryptography. In B. Christianson, J. A. Malcolm, V. Matyás, P. Svenda, F. Stajano, and J. Anderson, editors, *Security Protocols XXII - 22nd International Workshop Cambridge, UK, March 19-21, 2014 Revised Selected Papers*, volume 8809 of *Lecture Notes in Computer Science*, pages 350–358. Springer, 2014.

BS20. D. Boneh and V. Shoup. A Graduate Course in Applied Cryptography (Version 0.5). http://toc.cryptobook.us/, 2020. Accessed: 2022-12-07.

But13. V. Buterin. Bitcoin Is Not Quantum-Safe, And How We Can Fix It When Needed. https://bitcoinmagazine.com/articles/bitcoin-is-not-quantum-safe-and-how-we-can-fix-1375242150,http://www.webcitation.org/6wDiIPU3l, 2013.

CDG+17. M. Chase, D. Derler, S. Goldfeder, C. Orlandi, S. Ramacher, C. Rechberger, D. Slamanig, and G. Zaverucha. Post-Quantum Zero-Knowledge and Signatures from Symmetric-Key Primitives. In B. Thuraisingham, D. Evans, T. Malkin, and D. Xu, editors, *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security, CCS 2017, Dallas, TX, USA, October 30 - November 03, 2017*, pages 1825–1842. ACM, 2017, Cryptology ePrint Archive: Report 2017/279.

CLQ20. K. Chung, T. Liao, and L. Qian. Lower Bounds for Function Inversion with Quantum Advice. In Y. T. Kalai, A. D. Smith, and D. Wichs, editors, *1st Conference on Information-Theoretic Cryptography, ITC 2020, June 17-19, 2020, Boston, MA, USA*, volume 163 of *LIPIcs*, pages 8:1–8:15. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020, arXiv: 1911.09176.

Cry22. CryptoQuant. Bitcoin: UTXO Age Bands. https://cryptoquant.com/asset/btc/chart/network-indicator/utxo-age-bands, 2022. Accessed: 2022-10-03.

CS20. A. Coladangelo and O. Sattath. A Quantum Money Solution to the Blockchain Scalability Problem. *Quantum*, 4:297, 2020, arXiv: 2002.11998.

Del22. Deloitte. Quantum computers and the Bitcoin blockchain. https://www2.deloitte.com/nl/nl/pages/innovatie/artikelen/quantum-computers-and-the-bitcoin-blockchain.html, 2022. Accessed: 2022-05-30.

DKL+18. L. Ducas, E. Kiltz, T. Lepoint, V. Lyubashevsky, P. Schwabe, G. Seiler, and D. Stehlé. CRYSTALS-Dilithium: A Lattice-Based Digital Signature Scheme. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2018(1):238–268, 2018.

Dra18. J. Drake. Quantum computers and the Bitcoin blockchain. A forum post in https://ethresear.ch/t/cryptographic-canaries-and-backups/1235, 2018. Accessed: 2022-09-21.

Eth22. Ethereum Foundation. Optimistic Rollups. https://ethereum.org/en/developers/docs/scaling/optimistic-rollups/, 2022. Accessed: 2022-10-03.

FHK+18. P.-A. Fouque, J. Hoffstein, P. Kirchner, V. Lyubashevsky, T. Pornin, T. Prest, T. Ricosset, G. Seiler, W. Whyte, and Z. Zhang. Falcon: Fast-Fourier lattice-based compact signatures over NTRU. *Submission to the NIST's post-quantum cryptography standardization process*, 36(5), 2018.

Gol04. O. Goldreich. *The Foundations of Cryptography - Vol. 2, Basic Applications*. Cambridge University Press, 2004.

HJN+20. T. Häner, S. Jaques, M. Naehrig, M. Roetteler, and M. Soeken. Improved Quantum Circuits for Elliptic Curve Discrete Logarithms. In J. Ding and J. Tillich, editors, *Post-Quantum Cryptography - 11th International Conference, PQCrypto 2020, Paris, France, April 15-17, 2020, Proceedings*, volume 12100 of *Lecture Notes in Computer Science*, pages 425–444. Springer, 2020, arXiv: 2001.09580.

IKK20. D. I. Ilie, K. Karantias, and W. J. Knottenbelt. Bitcoin Crypto–Bounties for Quantum Capable Adversaries. Cryptology ePrint Archive, Paper 2020/186, 2020. https://eprint.iacr.org/2020/186.

IKS19. D. I. Ilie, W. J. Knottenbelt, and I. D. Stewart. Committing to Quantum Resistance, Better: A Speed-and-Risk-Configurable Defence for Bitcoin Against a Fast Quantum Computing Attack. In P. M. Pardalos, I. S. Kotsireas, Y. Guo, and W. J. Knottenbelt, editors, *Mathematical Research for Blockchain Economy, 1st International Conference, MARBLE 2019, Santorini, Greece, May 6-9, 2019*, pages 117–132. Springer, 2019.

Kat10. J. Katz. *Digital Signatures*. Springer, 2010.

KL14. J. Katz and Y. Lindell. *Introduction to Modern Cryptography, Second Edition*. CRC Press, 2014.

KZ20. D. Kales and G. Zaverucha. Improving the performance of the picnic signature scheme. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, pages 154–188, 2020.

LK21. H. T. Larasati and H. Kim. Quantum Cryptanalysis Landscape of Shor's Algorithm for Elliptic Curve Discrete Logarithm Problem. In H. Kim, editor, *Information Security Applications - 22nd International Conference, WISA 2021, Jeju Island, South Korea, August 11-13, 2021, Revised Selected Papers*, volume 13009 of *Lecture Notes in Computer Science*, pages 91–104. Springer, 2021.

LRS19. T. Lee, M. Ray, and M. Santha. Strategies for Quantum Races. In A. Blum, editor, *10th Innovations in Theoretical Computer Science Conference, ITCS 2019*, volume 124 of *LIPIcs*, pages 51:1–51:21. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2019, arXiv: `1809.03671`.

NBF+16. A. Narayanan, J. Bonneau, E. W. Felten, A. Miller, and S. Goldfeder. *Bitcoin and Cryptocurrency Technologies - A Comprehensive Introduction*. Princeton University Press, 2016.

PRVB13. M. Palatinus, P. Rusnak, A. Voisine, and S. Bowe. Mnemonic code for generating deterministic keys. Bitcoin Improvement Proposal (BIP) 39 `https://github.com/bitcoin/bips/blob/master/bip-0039`, 2013.

RMD+21. P. P. Rohde, V. Mohan, S. Davidson, C. Berg, D. Allen, G. K. Brennen, and J. Potts. Quantum crypto-economics: Blockchain prediction markets for the evolution of quantum technology, 2021, arXiv: `2102.00659`.

RNSL17. M. Roetteler, M. Naehrig, K. M. Svore, and K. E. Lauter. Quantum Resource Estimates for Computing Elliptic Curve Discrete Logarithms. In *Advances in Cryptology - ASIACRYPT 2017 - 23rd International Conference on the Theory and Applications of Cryptology and Information Security, Hong Kong, China, December 3-7, 2017, Proceedings, Part II*, pages 241–270, 2017, arXiv: `1706.06752`.

RWC+21. M. Raavi, S. Wuthier, P. Chandramouli, Y. Balytskyi, X. Zhou, and S. Chang. Security Comparisons and Performance Analyses of Post-quantum Signature Algorithms. In K. Sako and N. O. Tippenhauer, editors, *Applied Cryptography and Network Security - 19th International Conference, ACNS 2021, Kamakura, Japan, June 21-24, 2021, Proceedings, Part II*, volume 12727 of *Lecture Notes in Computer Science*, pages 424–447. Springer, 2021.

Sat20. O. Sattath. On the insecurity of quantum Bitcoin mining. *Int. J. Inf. Sec.*, 19(3):291–302, 2020, arXiv: `1804.08118`.

Sho94. P. W. Shor. Algorithms for Quantum Computation: Discrete Logarithms and Factoring. In *35th Annual Symposium on Foundations of Computer Science, Santa Fe, New Mexico, USA, 20-22 November 1994*, pages 124–134. IEEE Computer Society, 1994, arXiv: `quant-ph/9508027`.

SIZ+18. I. Stewart, D. Ilie, A. Zamyatin, S. Werner, M. F. Torshizi, and W. J. Knottenbelt. Committing to Quantum Resistance: A Slow Defence for Bitcoin against a Fast Quantum Computing Attack. Cryptology ePrint Archive, Paper 2018/213, 2018. `https://eprint.iacr.org/2018/213`.

TC22. G. Tamvada and S. Celi. Deep dive into a post-quantum signature scheme. `https://blog.cloudflare.com/post-quantum-signatures/`, 2022. Accessed: 2022-05-22.

Tod13. P. Todd. REWARD offered for hash collisions for SHA1, SHA256, RIPEMD160 and other (Bitcoin Talk Forum). `https://en.bitcoin.it/wiki/Secp256k1`, 2013. Accessed: 2022-12-6.

Tra22. Transactionfee.info. Output Types by Count. `https://transactionfee.info/charts/output-type-distribution-count/`, 2022. Accessed: 2022-10-26.

Unr15. D. Unruh. Non-Interactive Zero-Knowledge Proofs in the Quantum Random Oracle Model. In E. Oswald and M. Fischlin, editors, *Advances in Cryptology - EUROCRYPT 2015 - 34th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Sofia, Bulgaria, April 26-30, 2015, Proceedings, Part II*, volume 9057 of *Lecture Notes in Computer Science*, pages 755–784. Springer, 2015, Cryptology ePrint Archive: `Report 2014/587`.

Wes21. B. Westerbaan. Sizing Up Post-Quantum Signatures. `https://blog.cloudflare.com/sizing-up-post-quantum-signatures/`, 2021. Accessed: 2022-05-22.

Wig22. T. Wiggers. Making protocols post-quantum. `https://blog.cloudflare.com/making-protocols-post-quantum/`, 2022. Accessed: 2022-05-22.

WNR20. P. Wuille, J. Nick, and T. Ruffing. Schnorr Signatures for secp256k1. Bitcoin Improvement Proposal (BIP) 340 `https://github.com/bitcoin/bips/blob/master/bip-0340.mediawiki`, 2020.

WNT20. P. Wuille, J. Nick, and A. Towns. Taproot: SegWit version 1 spending rules. Bitcoin Improvement Proposal (BIP) 341 `https://github.com/bitcoin/bips/blob/master/bip-0341.mediawiki`, 2020.

Wui13. P. Wuille. Hierarchical deterministic wallets. Bitcoin Improvement Proposal (BIP) 32 `https://github.com/bitcoin/bips/blob/master/bip-0032`, 2013.

A Merkle–Damgård as a Random Oracle

In Section 2.2, we stated that throughout this paper we replace the SHA-256 and SHA-512 hashes with a random oracle H. More generally, these hashes are constructed by applying a technique called the *Merkle–Damgård transform*, which transforms a collision-resistant compressing function f (e.g., $f : \{0,1\}^{2n} \to \{0,1\}^n$) to a collision-resistant function H_f with a variable input length.

The purpose of this appendix is to isolate the properties we require of H, and argue that they also hold for H_f, when f is modeled as a random oracle.

The "bad" property of H_f is that, given the hash $\mathsf{H}_f(m)$ and oracle access to f, an adversary can easily compute $\mathsf{H}_f(m\|m')$ for a large variety of strings m' (note that the adversary need not know m for that). This is called a *length extension attack*. Using this property, one can easily construct cryptographic schemes which are secure in the random oracle model, but become broken when the oracle is replaced with H_f (see e.g. the "prepend the key" MAC construction in [BS20, Section 8.7]).

However, in our analysis, we only need two properties of the random oracle. The first property is that it is collision-resistant. This is known to hold (and indeed, is the original motivation) for the Merkle–Damgård transform (see [BS20, Theorem 8.3]).

The second property we need is an analogue of Proposition 1. That is, that the output of a "sufficiently random" input is indistinguishable from uniform. We only use this assumption in two cases: one is for key derivation from a seed (see Section 2.6) where it is known that the input starts with sufficiently many random bits. The other is applying a hash to a string sampled from a distribution known to be indistinguishable from uniform (typically generated in the first case). Hence, it suffices to prove the following:

Proposition 9. *Let $f : \{0,1\}^{2n} \to \{0,1\}^n$ be a random function. Let $\mathcal{S}$ be a distribution of strings of length at least n such that the distribution on the first n bits is computationally indistinguishable from uniform, then $\mathsf{H}_f(\mathcal{S})$ is computationally indistinguishable from uniform for an adversary with oracle access to f.*

Proof. We first observe that if a distribution $\mathcal{T}$ on strings of length n is indistinguishable from uniform then $\gamma_{\mathcal{T}} = \mathsf{negl}(n)$: say that $\mathbb{P}_{\mathcal{T}}(x) = \alpha$, and consider an adversary $\mathcal{A}$ that samples x' and outputs 1 iff $x' = x$. Clearly $\mathcal{A}$ is polynomial, and we have $\mathbb{P}[\mathcal{A}^{\mathcal{T}}() = 1] = 1/\alpha$ and $\mathbb{P}[\mathcal{A}^{\mathcal{U}}() = 1] = 2^{-n}$. By hypothesis we have that $|1/\alpha - 2^{-n}| = \mathsf{negl}(n)$. It follows that $\mathbb{P}_{\mathcal{T}}(x) = \mathsf{negl}(n)$ for all strings x, hence $\gamma_{\mathcal{T}} = \mathsf{negl}(n)$.

We also observe that if $\mathcal{T} = (\mathcal{T}_1, \mathcal{T}_2)$, then $\gamma_{\mathcal{T}} \leq \gamma_{\mathcal{T}_1}$: let (x_1, x_2) that maximizes $\mathbb{P}_{\mathcal{T}}[(x_1, x_2)]$, then $\gamma_{\mathcal{T}_1} \geq \mathbb{P}_{\mathcal{T}_1}[x_1] = \sum_{x_2'} \mathbb{P}_{\mathcal{T}}[(x_1, x_2')] \geq \mathbb{P}_{\mathcal{T}}[(x_1, x_2)] = \gamma_{\mathcal{T}}$.

Recall that the first step of applying Merkle-Damgård to an input x is to create a padded $\hat{x}$ whose length is a multiple of n (the specifics of the padding are crucial for collision resistance, but irrelevant to the current proof), and writing $\hat{x} = x_1\| \dots \|x_k$ where each x_i is a string of length n. Crucially, if $|x| \geq n$ then x_1 is exactly the first n bits of x.

We then compute $y_1 = f(0^n, x_1)$ and $y_j = f(y_{j-1}, x_j)$ for $j = 2, \dots, k$, until finally we obtain $\mathsf{H}_f(x) := y_k$.

By hypothesis if $x \leftarrow \mathcal{S}$ then $\gamma_{x_1} = \mathsf{negl}(n)$. From the observation above, we get that $\gamma_{(0,x_1)} = \mathsf{negl}(n)$. Since f is random it follows from Proposition 1 that $y_1 = f(0^n, x_1)$ is indistinguishable from uniform. Hence $\gamma_{y_1} = \mathsf{negl}(n)$, so we can repeat the argument to get that $\gamma_{(y_1,x_2)} = \mathsf{negl}(n)$ and y_2 is indistinguishable from random. We repeat the process k times to obtain that $y_k = \mathsf{H}_f(x)$ is also indistinguishable from uniform. □

For completeness, we point out the places in the paper where we use the assumption that H is modeled as a random oracle, and observe that we only use the two properties above:

- In Section 2.6 we argue that 2048 successive applications of SHA-512 constitute a PBKDF. By successive applications of Proposition 9 we get that $\mathsf{SHA\text{-}512}^{2048}$ is indeed a PBKDF, as long as we assume that the first n bits of the input are indistinguishable from uniform. In practice, the key-generation procedure of HD wallets uses an input that starts with a long uniform string, so this assumption is justified.
- Throughout Section 2.7 we prove that if H is a random oracle, then several functions derived from H have some collision resistance properties. These arguments are readily adaptable to the weaker assumption that H is collision-resistant.
- In Section 3 we assume that H is modeled as a random oracle. We only use this fact directly to argue that *Picnic*(H) is secure, which only requires H is one-way. The fact that H is one-way already follows from the assumption that it is collision-resistant. All other appeals to the randomness of H are to apply the statements of Section 2.6 and Section 2.7.