Fawkescoin

A cryptocurrency without public-key cryptography

Joseph Bonneau[1] and Andrew Miller[2]

[1] Princeton University
[2] University of Maryland

Abstract. We present, Fawkescoin, a simple cryptocurrency using no public-key cryptography. Our proposal utilizes the distributed consensus mechanism of Bitcoin but for transactions replaces Bitcoin's ECDSA signatures with hash-based Guy Fawkes signatures. While this introduces a number of complexities, it demonstrates that a distributed cryptocurrency is in fact possible with only symmetric cryptographic operations with no dramatic loss of efficiency overall and several efficiency gains.

1 Introduction

Bitcoin [9] is a distributed, peer-to-peer digital currency system, which uses a public append-only ledger to maintain consensus about the ownership history of all coins in the system. The ledger is maintained by a community of mutually distrusting miners using economic incentives to maintain consensus. Bitcoin has demonstrated the possibility that an append-only ledger can be maintained in a decentralized manner. The consensus mechanism requires no public key cryptography, utilizing only hash computations.

However, coins in Bitcoin are controlled by public-key signatures. Technically, all bitcoins are controlled by a transaction script which indicates the conditions under which they may be transferred to another script. Most often, this script requires a signature from one or more designated public keys, so ownership of the key entails ownership of the bitcoins assigned to the transaction script. The only signature algorithm currently supported is Elliptic Curve DSA over the NIST P-256 curve.

We propose a new cryptocurrency scheme called Fawkescoin which shows that, surprisingly, it is possible to build a system with similar properties to Bitcoin using no asymmetric cryptography at all. The only cryptographic primitive required is a one-way, preimage-resistant[3] hash function with no length extension attacks. SHA-3 (Keccak) [2] is a candidate, or any Merkle-Damgård function with the length unambiguously prepended to the input [3]. Fawkescoin is extremely simple for clients, with coin ownership controlled by knowledge of a hash preimage instead of knowledge of a private key.

We could build a cryptocurrency without public-key operations in a very straightforward manner by adapting a variant of Lamport's one-time signature

[3] Collision resistance is a more challenging property, but this is not necessary here.

scheme [5], as Bitcoin keys need only be used to sign one message. However, even with compression techniques [8] these schemes result in signatures which, at thousands of bits, are considered unwieldy by the Bitcoin community.

Instead our construction builds on the Guy Fawkes signature protocol [1], demonstrated by Anderson et al. in 1998 to enable secure signatures using only a hash function and a secure timeline service. Observing that Bitcoin's block chain must serve as a secure timeline service for the currency to work, we can replace Bitcoin's ECDSA signatures with Guy Fawkes-style signatures and achieve an efficient digital currency with only symmetric cryptography.

2 Simplified Fawkescoin protocol

We assume the existence of a mechanism for maintaining global consensus on an append-only log, which is usually called the *block chain* in Bitcoin. The block chain imposes a partial ordering over transactions, which need not be a total ordering. Transactions may be published in batches, usually called *blocks*.

In Bitcoin, this is achieved (with some issues [4,10] and objections [6]) by a community of miners a solving a proof-of-work puzzle in exchange for newly minted coins and a longest-chain rule in effect to establish consensus, but other mechanisms would be suitable for our purposes. We assume, like in Bitcoin, that the log may occasionally fork but consensus will eventually be re-established. Temporary forks introduce a number of subtle issues to deal with, but we must tolerate forks to be able to build on top of Bitcoin's consensus model instead of an ideal append-only log with no forks.

2.1 Minting

To mint coins, one inserts a special transaction into the ledger:

$$\mathsf{Mint} : v; \mathbf{H}(X)$$

A value of v now belongs to the address $\mathbf{H}(X)$, where X is a random, secret value known only to the owner of the new address $\mathbf{H}(X)$ for a preimage-resistant hash function $\mathbf{H}$. The community must agree to rules about who has the right to mint coins and with what values; this is outside the scope of our technical definition. In Bitcoin, for example, miners may currently mint 25 new coins when they find a new block, with this value scheduled to decline gradually over time. A number of alternative cryptocurrencies have adopted different rules.

2.2 Transfer

To transfer coins from address $\mathbf{H}(X)$ to address $\mathbf{H}(Y)$, the owner publishes an initial transfer message:

$$\mathsf{Transfer} : \mathbf{H}\{X; \mathbf{H}(Y)\}$$

If this appears in the block chain in block i, this will serve as proof that whoever crafted this message knew the value X at block i, which is crucial for establishing that the coin was transferred by its proper owner. The owner of X must wait until a sufficient number of subsequent blocks are published to ensure that this message is permanently in the block chain before actually revealing X, so that this proof of knowledge of X at time i is not overwritten by a block chain fork.

Once this is achieved, the owner of X publishes a second message to finalize the transaction:

$$\mathsf{Finalize} : X; \mathbf{H}(Y)$$

This message allows anybody to verify the initial Transfer message in block i committed to a transfer from $\mathbf{H}(X)$ to $\mathbf{H}(Y)$ and knew the value of X at the time. Security rests on that fact that the Transfer message was published before X was public knowledge, so only the legitimate owner of the address $\mathbf{H}(X)$ could have inserted the Transfer message. Note that, because X is public once the Finalize message is sent, $\mathbf{H}(X)$ can never again be used as an address.

As with Bitcoin, the owner of $\mathbf{H}(Y)$ must wait an additional confirmation period before accepting that they control the coins to guard against double-spending, since the owner of $\mathbf{H}(X)$ may have published a different Transfer message that could be opened in the case of a fork. After a suitable confirmation period, the address $\mathbf{H}(Y)$ owns the value v, which can be transferred to a subsequent address $\mathbf{H}(Z)$ using the exact same protocol.

3 Complete Fawkescoin protocol

The simplified protocol is limited to transferring indivisible coins that forever retain their initial value from the time of minting. We can easily modify the messages to enable arbitrary splitting and merging of values. We also want to include block index numbers in messages to avoid searching the entire block chain during transaction verification. The Mint transaction needs no changes.

The Transfer messages now contains a list of input and output addresses:

$$\mathsf{Transfer} : \mathbf{H}\{[(X_0, i_0), (X_1, i_1)...], [(\mathbf{H}(Y_0), v_0), (\mathbf{H}(Y_1), v_1), ...]\}$$

The input addresses X_j must all be known to craft this transaction. The indices i_j indicate in which block each X_j's receipt of funds was finalized. Each output address $\mathbf{H}(Y_k)$ receives some value v_k, with the obvious constraint that the total of the inputs is greater than or equal to the total of the outputs.

The Finalize message is now simply:

$$\mathsf{Finalize} : i; [(X_0, i_0), (X_1, i_1)...], [(\mathbf{H}(Y_0), v_0), (\mathbf{H}(Y_1), v_1), ...]$$

with all of the information from the Transfer message repeated and, for efficiency, the block i in which the Transfer message appeared.

As with Bitcoin, it is now possible to implement a transaction that transfers some holdings to a new address $\mathbf{H}(Y)$ and keep the "change" at a new address. Bitcoin clients already create fresh addresses for change to increase anonymity, though with Fawkescoin the change address *must* be new to ensure security.

4 Preventing race-condition theft and DoS by miners

Without further mechanisms, the owner of a coin is vulnerable to a race-condition after broadcasting their Finalize message. Anybody receiving this message (including the miners, who must see it in the pending transaction pool) could observe the value X and attempt to block the publication of the Finalize message by quickly crafting and publishing their own Transfer and Finalize messages sending the value held by X to their own address Z.

4.1 Earliest Transfer wins

One mitigation is an "earliest Transfer wins" rule, which specifies that if two Finalize messages are published for the same X, then whichever one corresponds to an earlier Transfer message wins. This ensures the legitimate owner can always control transfer of the coin, since they can always publish a valid Transfer message before anybody else.

However, this introduces a double-spending attack. The recipient of a coin can never trust that their ownership is beyond dispute, because the legitimate owner may have published an earlier Transfer message that remains latent in the block chain. To prevent this we must establish a maximum time-limit Δ beyond which a Transfer message is considered invalid if not matched by a Finalize transaction. A recipient can then be confident in their ownership of a coin if no other party tries to claim it after Δ have been passed since the Transfer message transferring the coin to them.

4.2 Optimization via transaction tagging

The Δ block waiting period for transaction confirmation can be removed by tagging each Transfer message in a way that unambiguously ties the message to X, so one can safely determine that there are no earlier valid Transfer messages in the block chain that might correspond to X. To do so requires appending to each Transfer message a tag $\mathbf{H}'(X)$ computed using a tweaked hash function. This portion of the Transfer message will be identical for any valid Transfer message involving X but reveals no information about X.

4.3 Preventing DoS with Guy Fawkes multi-use signatures

The anti-theft mechanisms in turn introduces a potential denial-of-service attack: if a coin owner relays a valid Finalize transaction but miners can prevent it from appearing in the block chain within Δ blocks, then they can use the revealed secret value X to publish their own Transfer and Finalize messages to claim the coin which can't be overridden by the earlier Transfer. We would like a rule then that if a Transfer message isn't matched within Δ blocks, the corresponding value X can no longer be used to redeem the coin. Now miners can't steal a coin by delaying the Finalize message past the cutoff of Δ blocks, but they can cause the coin to be unspendable.

This can be prevented by using a modified Guy Fawkes algorithm which enables multiple signatures per public address, giving the legitimate coin owner multiple chances to overcome a denial-of-service attack. There are two potential ways to implement this. One is by publishing an iterated hash $\mathbf{H}^n(X)$ as the public address, with each preimage $\mathbf{H}^i(X)$ for $1 \leq i < n$ serving as a secret which can be revealed. The second is by using the root of a Merkle tree $\mathbf{H}^{\text{Merkle}}(X)$ with n leaves, each of which is a preimage that can be revealed. Either allows n signatures from a single public address. The first scheme has signatures of size $\Theta(1)$ that require $\Theta(n)$ hash computations to verify; the second scheme produces signatures of size $\Theta(\lg n)$ which require $\Theta(\lg n)$ hash computations to verify. We expect in practice the first scheme will be preferable.

With either construction, we can provide reasonable guarantees against extended denial of service as in the worst case that the legitimate owner can't publish anything for Δ blocks they will have n additional attempts to spend their coin. We also must extend the tag to include the index of the preimage to be revealed, so the transaction tag would be $\mathbf{H}'(X||i)$ if the i^{th} preimage must be revealed.

4.4 Choosing an expiry delay

With a first-Transfer wins rule and tagging, we might recommend a Transfer expiry delay of 12 blocks. This lets a client publish a Finalize message 6 blocks after the Transfer message is published, allowing the standard confirmation period of 6 blocks to be confident the Transfer message is permanently in the block chain, and still leaving 6 blocks to get a Finalize message placed in the block chain in case of malicious miners. In the common case this will add no apparent latency to the recipient(s) in the transaction as they will wait an additional 6 blocks' confirmation period to accept the transaction anyways, at which point they can accept that they have received funds.

This could be established as a global constant, or better yet embedded into X itself. For example, we might reserve by convention the high-order 16 bits of a 256-bit X value to represent Δ. This value will then be publicly known as soon as X is published.

5 Comparison to Bitcoin

We claim that Fawkescoin replicates the core functionality of Bitcoin as it is used today, with no public-key cryptography. Some observations:

5.1 Cryptographic security

Bitcoin's security inherently depends on the security of the hash function used in its signature scheme. Therefore Fawkescoin is strictly more secure from a cryptographic standpoint as it has no reliance on the security of elliptic curve cryptography. This eliminates the risk of a catastrophic algorithmic break of

discrete log on the curve P-256 or rapid advances in quantum computing. It also reduces the risk of implementation flaws, as it is considered far easier to securely implement a hash function than asymmetric cryptographic primitives.

Fawkescoin also significantly reduces the risk of subtle entropy failures. Stealing funds from Fawkescoin requires finding a second preimage for a hash output of a random input X, the difficulty of which degrades gracefully as the strength of the random number generator X degrades. With elliptic curves, there are a number of subtle flaws arising from biased random numbers, particularly if a key is used repeatedly with non-random nonces. In practice, the risk is much lower in Bitcoin however if each signing key is only used once which is common practice.

5.2 Forking security

Fawkescoin has much worse behavior than Bitcoin in the case of a long fork of the block chain. In Bitcoin, an attacker capable of producing of a long fork (i.e., an attacker who temporarily wields a large amount of hash power) may perform effective double-spending attacks; however, the attacker can only double-spend their own coins which they actively relay transactions for in the blocks which are overwritten by the fork. In Fawkescoin, however, after a fork the attacker may steal the value of any transaction whose Transfer and Finalize message both appear in the overwritten fork.

5.3 One-time addresses

In Bitcoin it is often recommended (and implemented in most clients) that fresh addresses be used in every transaction for security and anonymity reasons. In Fawkescoin, this is mandatory as addresses can only be used securely once. This can be relaxed by using multi-use Guy Fawkes signatures, as proposed in Section 4.3. Other workarounds exist: for example, a business can publish a large number of addresses offline, or can transfer a new address to clients on demand. It would be possible to modify Fawkescoin to allow multiple transfers to an address $\mathbf{H}(X)$, but these would all need to be spent at once at which point $\mathbf{H}(X)$ would need to be retired.

Ostensibly, storing one private key may be more compact than a large number of secret values in Fawkescoin. However, Fawkescoin clients can avoid this by storing a single master-secret and deriving one-time addresses from it deterministically using a PRNG.

5.4 Efficiency

Fawkescoin has a number of small efficiency advantages, for example, relieving clients of the need to perform public-key operations. This might be of benefit on highly constrained devices, though it's unclear if this confers any practical advantage, as in either scheme a device must be able to contact the network

and verify the integrity of the block chain. The block chain would also be more compact in Fawkescoin due to the smaller sizes involved and lack of signatures, though by less than a factor of two; we can implement Fawkescoin using a secure hash function with 128 bits of output (since collision attacks are not a concern). A major advantage is complete verification of the block chain; while this process requires millions of signature verifications in Bitcoin, they are replaced by millions of hash computations in Fawkescoin.

A major disadvantage for Fawkescoin is the need for 2 confirmation periods during any transfer compared to 1 for Bitcoin, effectively doubling the latency of the system.

5.5 Transaction fees, priority, and anti-DoS countermeasures

Transaction fees can be realized in Fawkescoin exactly as they are in Bitcoin by having the total output value of a transaction be less than the total input value and allowing miners to claim the difference. However, this presents some difficulty since two messages are necessary. It would be possible for the transaction to include fees for both miners (the one who mines the block containing the Transfer message and the one who mines the block containing the Finalize message). However, until the Finalize message is published, there is no way to tell a valid Transfer message from an invalid one. At the moment, Bitcoin prevents spam transactions in two ways. The first is by transaction fees, and the second is by transaction priority based on age. Although fees are optional, the standard miner policy is to require fees for transactions without sufficient age/priority. This means the Transfer message carries no priority or fee, so it would not be relayed due to existing anti-DoS measures. There are several potential approaches:

Out-of-band payments In order to post a transaction, a user must make some arrangement out of band with a miner, who includes the Transfer message based on some form of trust that the Transfer message contains a commitment to an actual fee.

Split transaction fees Transaction fees might be split evenly between the miner including the Transfer message in a block and the miner including the Finalize message in a block. This incentivizes miners to include Transfer messages on the hope that they will eventually lead to fees if a Finalize message is published. The transaction owner still needs to convince a miner to include the transaction even though it is not evident that the transaction will be finalized. One approach is a proof in zero knowledge that the transfer message has a preimage paying the miner a useful amount. This negates some of the performance benefits of Fawkescoin, but would leave block chain validation very efficient.

Merkle tree of Transfer messages Finally, miners might place all Transfer messages in a Merkle tree and only include the root in their block. The Finalize

message would need to include a proof-of-inclusion for the Transfer message. This reduces the need to limit spam, since spam Transfer messages won't increase the size of each block. However, Finalize messages will grow as the proof-of-inclusion will be logarithmic in the number of Transfer messages included in a block. This approach also removes the possibility of the tagging optimization.

5.6 Multi-sig transactions

Transactions with multiple possible input addresses controlled by untrusting parties are possible (though not widely used) in Bitcoin. For example, the CoinJoin protocol [7] uses them to mix funds for anonymity purposes. There doesn't appear to be a simple way to achieve this in Fawkescoin, as any party finalizing a transaction risks leaking their private key if their counter-party doesn't publish a Finalize message.

5.7 Scripting functionality

The scripting functionality in Bitcoin is not included in this simple presentation of Fawkescoin. In principal, arbitrary scripts could be included in Fawkescoin transfers just as in Bitcoin. The main difficulty is that Fawkescoin transactions are inherently a one-shot proposition. If a script fails for whatever reason during verification, it effectively destroys the value associated with any input addresses as they will have their secret revealed.

6 Conclusion

Bitcoin itself is something of a curiosity from an academic standpoint in that it was discovered decades after the requisite cryptographic primitives were available. Our work shows that it was in fact possible even before the discovery of public-key cryptography. The subtleties with double-spending and spam transactions support that a public-key based approach is preferable given the efficiency of elliptic-curve operations on modern hardware and their strong security track record. However, Guy Fawkes signatures could be implemented as an alternative option within a cryptocurrency such as Bitcoin. This is not a simple addition which can be trivially bolted on to Bitcoin's existing scripting language given the required validation rules introduced in this paper to prevent double-spending and spam. However, it might be worthwhile as an alternative allowing very simple clients to participate or as a hedge against a catastrophic break of the discrete log problem in elliptic-curve groups.

References

1. Anderson, R., Bergadano, F., Crispo, B., Lee, J.H., Manifavas, C., Needham, R.: A new family of authentication protocols. ACM SIGOPS Operating Systems Review 32(4), 9–20 (1998)
2. Bertoni, G., Daemen, J., Peeters, M., Van Assche, G.: Keccak sponge function family main document. Submission to NIST (Round 2) 3 (2009)
3. Coron, J.S., Dodis, Y., Malinaud, C., Puniya, P.: Merkle-Damgård revisited: How to construct a hash function. In: Advances in Cryptology–CRYPTO 2005. pp. 430–448. Springer (2005)
4. Eyal, I., Sirer, E.G.: Majority is not enough: Bitcoin mining is vulnerable. Financial Cryptography (2014)
5. Lamport, L.: Constructing digital signatures from a one-way function. Tech. rep., Technical Report CSL-98, SRI International Palo Alto (1979)
6. Laurie, B.: Decentralised currencies are probably impossible but let's at least make them efficient (2011)
7. Maxwell, G.: CoinJoin: Bitcoin privacy for the real world. `https://bitcointalk.org/index.php?topic=279249.0` (August 2013)
8. Merkle, R.C.: A certified digital signature. In: Advances in Cryptology. pp. 218–238. Springer (1990)
9. Nakamoto, S.: Bitcoin: A peer-to-peer electionic cash system (2008)
10. Sompolinsky, Y., Zohar, A.: Accelerating Bitcoin's Transaction Processing. Fast Money Grows on Trees, Not Chains. Cryptology ePrint Archive, Report 2013/881 (2013), `http://eprint.iacr.org/`