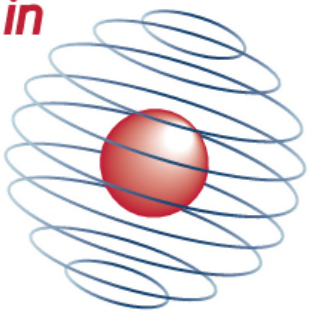
CENTRE *for* DOCTORAL TRAINING *in*
CYBER SECURITY



CDT Technical Paper

27/15

An Evaluation of the Effects of Broken Cryptographic Primitives on Bitcoin

Ilias Giechaskiel

An Evaluation of the Effects of Broken Cryptographic Primitives on Bitcoin

Ilias Giechaskiel*

CDT in Cyber Security, University of Oxford, Oxford, UK
ilias.giechaskiel@cybersecurity.ox.ac.uk

Abstract. The Bitcoin cryptocurrency relies heavily on a variety of cryptographic functions and operations, which are currently assumed to be secure, but will inevitably be broken in the future. As Bitcoin tries to compete against traditional currencies, it remains to be seen how the Bitcoin protocol will need to change in response to weakened cryptography. To this end, this study systematically evaluates the effects of broken cryptographic primitives on the operation of the Bitcoin network, and the changes to the Bitcoin protocol that will be necessary in response. We conclude that a broken hash function only requires switching over to a new hash function, without the need to re-write the blockchain, and is well serviced by the "checkpoint" mechanisms already built into Bitcoin. However, a vulnerability of the signature scheme cannot be dealt with in the same manner without side-effects, as it may lead to lost or stolen coins, even if the process is gradual and is conducted before the cryptographic primitive is broken. We conclude that solving this problem either requires some degree of centralization, or the use of Zero-Knowledge Proofs along or on top of Bitcoin.

1 Introduction

Ever since Bitcoin was introduced in 2008 by Nakamoto [31] as a "peer-to-peer electronic cash system", it quickly gained traction among enthusiasts, and its value against the dollar shot from a mere 5 cents per BTC in 2010 to over $1,100/BTC for a brief period of time in 2014 [41]. As we discuss in our overview of the Bitcoin system (Section 2), Bitcoin makes heavy use of cryptographic primitives (detailed in Section 3), but in evaluating Bitcoin, existing work (Section 2.4) assumes that these cryptographic primitives provide strong guarantees. As a result, the literature instead focuses on issues such as volatility [41], mining [18,19,28], double spending [25], and anonymity/privacy attacks [34,3,8]. Although it is highly unlikely that the cryptographic primitives will get severely broken in the foreseeable future, they will undoubtedly get weaker over time (for instance through faster-than-brute-force collisions or pre-images), so if Bitcoin wants to establish itself as a long-term currency, questions of how to switch

* I would like to thank my supervisor Cas Cremers for his helpful guidance and comments throughout this project. My DPhil is supported by the Clarendon Fund, the EPSRC, Kellogg College, and the CDT in Cyber Security.

over to stronger primitives will need to be answered. Though there are already some high-level plans in place [2], they essentially consist of switching to a new, stronger primitive, but without details of how these transitions would occur. It is especially important to identify any potential problems with any modifications early on, because such changes will necessitate a "hard fork", that could cause the community to split, as happened recently during the block size debate [22].

To this end, our contributions are as follows:

1. We systematically look at the cryptographic primitives used in Bitcoin (Section 3) and how they interact on a theoretical level (Appendix A).
2. We identify how different types of attacks on individual primitives impact the various parts of the Bitcoin ecosystem (Section 4).
3. We examine mechanisms for migrating to new primitives (Section 5).
4. We show that switching hash functions is sufficient for first and second preimage collision attacks, and is easily serviced by the existing checkpoint mechanisms (Section 5.1).
5. We identify problems with updating to a different signature scheme, where straightforward solutions would either require some form of centralization or result in lost/stolen coins (Section 5.2).
6. We propose a Zero-Knowledge Proof solution to the problem of changing the signature scheme in a distributed fashion (Section 5.3).
7. Overall, we highlight the importance of studying Bitcoin's future under reduced cryptographic guarantees.

2 Bitcoin Background and Related Work

Bitcoin was introduced as a "peer-to-peer electronic cash system" [31]: a way to make (online) payments without a trusted third-party, such as a bank. In this section, we summarize the main components of the Bitcoin system, though for a more complete introduction, please consult the original Bitcoin paper [31], the surveys [10,37], or other related work (Section 2.4).

In summary, transferring Bitcoin coins is achieved through transactions (Section 2.1), where cryptographic signatures serve as proof of ownership.[1] Mining (Section 2.2) is the component which controls the rate by which new coins get generated, and which determines what transactions are valid, as a means of solving the double-spending problem in a distributed fashion. Finally, the peer-to-peer (P2P) network (Section 2.3) is the underlying component that allows communications among participating parties.

2.1 Transactions

A *transaction* in Bitcoin transfers money ("Bitcoins") from some "accounts" into different accounts.[2] A transaction, when looking at the lower-level details,

[1] The more technical cryptographic details will be explained in Section 3.

[2] As will become clear, the term account is not accurate, but it suffices for a high-level understanding of transactions.

contains an array of inputs and outputs, and the double SHA-256 hash (Section 3) of the transaction is what uniquely identifies a given transaction. Each output contains an integer value representing the amount of *satoshis* (equal to 10^{-8} bitcoins or BTCs) as well as a *script* (Section 2.1.1) that details the (cryptographic) conditions under which this output can be redeemed. Inputs are references to previous transactions (through their unique hash) and contain the scripts necessary to prove authorization for spending these outputs. Transactions can also be "time locked", to ensure that they do not get completed until after a specific point (or block) in time.

For transactions, the sum of the output values is no more than the sum of the input values, with the exception of special "coinbase transactions", which are generated by miners when creating new blocks (Section 2.2).

2.1.1 Scripts The scripts that are used in transactions are a series of mathematical operations/computer instructions (*opcodes*) that get executed like normal computer programs. However, Bitcoin's *scripting language* is stack-based, and without loops, so it is a simple language that is not very expressive, and certainly not Turing-complete. Though there is a variety of opcodes for string, arithmetic, and cryptographic operations, as well as for representing data, controlling the program flow and the stack, many of them are disabled for security purposes. Specifically, some of the opcodes (such as hashing or checking digital signatures) are computationally intensive, so allowing an unbounded number of them, or even "strange" combinations can be used to attack individual nodes (Section 2.3), by having them waste computational cycles, resulting in a Denial-of-Service (DoS) attack. As a result, most nodes only accept and relay a certain set of *standard scripts*, though individual nodes do not need to use this default policy, and as a matter of fact there are some mining pools (Section 2.2) that accept non-standard transactions but at the cost of higher fees.

Scripts are used in transaction outputs as *locking* scripts and in transaction inputs as *unlocking* scripts. Locking scripts specify the conditions under which an output can be spent, while unlocking scripts show that the conditions (usually in the form of a cryptographic signature) to spend an output are satisfied. As explained in great detail in [5], there are five standard (locking) scripts, shown here from most popular to least popular:[3]

- **Pay-to-Public-Key-Hash (P2PKH)**. The unlocking script much provide a public key which hashes to the given value, and must then sign the transaction under this key.
- **Public-Key**. The unlocking script must sign the transaction under the provided key.
- **Pay-to-Script Hash (P2SH)**. This script is the hash under RIPEMD-160 and SHA-256 (Section 3) of a non-P2SH standard transaction. The unlocking script then provides the full script (which hashes to this value) as well as

[3] Based on a 2014 analysis [33], at the time of which P2PKH transactions accounted for around 99% of all transactions.

any necessary signatures. This script is typically used to shorten the length of multi-signature transactions.
- **Multi-Signature**. An M-of-N ($N \leq 15$) multi-signature scheme provides N public keys, and requires M signatures in the unlocking script.
- **Data Output (OP_RETURN)**. The output cannot be redeemed, but can be used to store up to 40 arbitrary bytes in the blockchain for a fee.

2.2 Mining

Transactions are combined into *blocks* that get published in a global append-only transaction log, called the *blockchain*, through a process called *mining*. Miners are required to solve computational puzzles (often referred to as *proof-of-work*), in which they need to find a nonce value that causes the block's double SHA-256 hash to start with d consecutive zero bits. Due to the difficulty of this puzzle, miners often form mining *pools* in which the work is divided among members, and fees are split proportionally to the computational power each miner brings to the pool. The difficulty value d is adjusted such that on average one block gets mined every 10 minutes. Miners are incentivized to mine because they receive the leftover fees of transactions (equal to the difference of inputs minus outputs of all transactions) as well as a fixed block creation reward. This reward is currently equal to 25 BTC, but halves approximately every 4 years, to reach a maximum of 21 million total BTC in existence by 2140 [5], after which point miners only incentive will be the transaction fees.

It is important to note that it possible for the blockchain to *fork* into separate branches, where different miners have added different blocks to the top of the chain. Honest miners always work on the longest chains and choose randomly when the branches are of equal length, but should the blocks contain conflicting transactions, this can lead to *double-spending*. As a result, multiple confirmation blocks (typically 6) are needed to ensure that a transaction will not be "removed" from the blockchain (through forking), while hard-coded checkpoints in the client are used to prevent deeper forks.

Note that if a single miner (or pool) is in control of a high enough percentage (more than 50% [28], but possibly as little as 33% [19]) of the total computational power, then this miner will be able to reap a reward that is higher than his share, and possibly destabilize the system. This is because the majority miner is essentially able to ignore blocks found by other miners and keep building his own chain which will become the longest in expectation.

2.3 Network

The last key component of Bitcoin is its Peer-to-Peer (P2P) network that allows it to operate in a distributed fashion. Nodes broadcast transactions and blocks to other nodes, which they discover through hardcoded directory servers. These messages are then relayed to other nodes in order to reach the entire network through *flooding*, but in order to prevent DoS attacks only transactions and blocks that pass stricter-than-necessary validations are relayed. For instance,

only transactions containing standard scripts (Section 2.1.1) are relayed, and only if their fees are higher than a given threshold. It is due to the latency associated with this type of broadcast (and to reduce the possibility of forks) that blocks only get generated every 10 minutes on average.

2.4 Related Work

The surveys [10,37] provide great explanations of the technical aspects, challenges, alternate cryptocurrencies (altcoins), and open research areas surrounding Bitcoin. [28,18,19] look at the economics of Bitcoin mining under the presence of adversaries, while [25] considers double-spending attacks on fast payments in Bitcoin. [34,3] analyze the anonymity and privacy provided by Bitcoin by analyzing the published transactions, while [8] deanonymizes Bitcoin clients by focusing on the P2P network instead. [35] provides a more general analysis of the Bitcoin transaction graph, while [12] examines the use of Bitcoin in the (now defunct) black market Silk Road. There are also a lot of protocols designed either on top or extending Bitcoin, for instance to incentivize correct computations [29], to design fair protocols [7], or for secure multiparty computations [4]. Finally, a survey of the usability of Bitcoin key management [16] and a formal analysis of the Bitcoin backbone protocol [20] are perhaps closest in spirit to our work.

3 Cryptographic Primitives

In this section we look at the cryptographic primitives used in Bitcoin. For this section, we researched the Bitcoin Core source code [1] directly, which is the de-facto standard and implementation for Bitcoin, in order to ensure that we account for all usages of a given primitive. Section 3.1 looks at the hashes used in the main protocol, Section 3.2 at the signature scheme, while Section 3.3 looks at other cryptographic primitives used with Bitcoin.

3.1 Hash Functions

Bitcoin relies on the SHA-256 [32] and the RIPEMD-160 [15] hash functions. However, these hash functions are never used individually, but are always combined, either as a double SHA-256 (also called Hash256, or SHA-256d), or the nested RIPEMD-160 hash of SHA-256 (simply called Hash160). As we show in Appendix A, a collision for a nested hash is a collision for one of the two hashes, while a pre-image attack on the nested hash results at either a pre-image attack on the inner hash, or a collision attack on the outer hash. As a result, it is sufficient to focus on the hash combinations, rather than the individual SHA-256 and RIPEMD-160 functions.

Hash160 is found quite frequently in the blockchain transactions, as it is the hash used in the Pay-to-Public-Key-Hash (P2PKH), and in the Pay-to-Script-Hash (P2SH) scripts. Thus, a pre-image of this hash represents either a public-key (see Section 3.2), or a script, which typically (but not always) contains multi-signature public keys.

Hash256 is also used a lot in the blockchain, but in a different manner. First of all, it is the hash used for mining, so that miners need to find a nonce such that the SHA-256d hash of a block begins with d consecutive zeros. Moreover, it is used to hash transactions within a block into a *Merkle Tree*, a structure which summarizes the transactions present within a block. Because Merkle Trees allow efficient verification of whether a given transaction is present in a given block, it is used extensively in "light-weight" clients that only perform *Simplified Payment Verification (SPV)* and thus do not download full blocks [5]. Finally, it is also the hash of a transaction that is signed with a user's private key (Section 3.2).

3.2 Signature Scheme

Bitcoin uses the Elliptic Curve Digital Signature Algorithm (ECDSA) with the `secp256k1` [36] parameters defined by the Standards for Efficient Cryptography group. It wraps around OpenSSL's implementation, and only signs the SHA-256d hash of transactions. It is important to note that there are some public keys hard-coded into the clients that can be used by core developers (who have the corresponding private keys) to sign *Alert* messages to broadcast critical information to the network. It is also often mentioned that one should not re-use addresses from a privacy perspective [5], but as we see in Section 4, this notion is furthered by technical considerations as well.

3.3 Other Cryptographic Primitives

Though the two hash functions and the signature scheme are the only primitives used in the main Bitcoin system, it is worth mentioning that the implementation [1] makes use of cryptography in some other locations as well. X.509 Public Key Infrastructure (PKI) with SHA-1 or SHA-256 can be used for some merchant payments, while Bitcoin wallets are stored locally using the Advanced Encryption Standard (AES), and HMAC-SHA512 is used for key derivation in "hierarchical deterministic wallets". Finally, even though only standard scripts (Section 2.1.1) are accepted and relayed by most nodes, there are opcodes for SHA-1, SHA-256, RIPEMD-160, Hash160, and Hash256, as well as for checking one or more signatures.

4 Assuming Primitives are Broken

In this section we explain how broken cryptographic primitives affect the various parts of Bitcoin. Given that these functions have undergone considerable public scrutiny, it is highly unlikely that they will fail completely. Instead, the complexity of attacks (primarily collisions) may decrease to less-than-brute-force, and computational power will increase. For example, after MD5 was introduced in 1992, the first collision was found in 2004 and took an hour in a cluster [39], and took less than a minute on a laptop only 2 years later [27]. Similarly, for SHA-1 (introduced in 1995), the first attack required 2^{69} operations [38] (compared

to 2^{80} for brute-force), which was reduced to 2^{63} soon thereafter [13]. Even if such attacks are years away, it is still important to study the effects of broken primitives on Bitcoin, because Bitcoin wants to be a long-term currency, and we simply don't know how attacks on primitives would change 50 years from now.

Moreover, sometimes a flawed implementation, or bad randomness can have the same effect as a broken primitive. For instance, the original Bitcoin implementation for the Merkle Tree used in Bitcoin was vulnerable to CVE-2012-2459. This could allow attackers to cause a DoS attack, by modifying a block without changing the Merkle root hash.[4] Moreover, reusing the same random number for two ECDSA signatures can be used to recover the private key,[5] and nonces can also be recovered through side-channel attacks [40]. This tells us that addresses should be not reused, not just for privacy, but also for technical reasons, and also that there are cases where our assumption of broken primitives might not be as strong as it appears. Of course, fixing an implementation bug (and not a specification one) does not require a hard fork (and may not even require a soft fork), but hopefully indicates that this topic is worthy of study (since not "all bets are off"), and not a mere academic curiosity.

Because there is a well-defined relationship between how individual hashes and their combinations break,[6] it suffices to look at hash combinations, rather than the individual functions. Though some of our discussion stems from the Bitcoin contingency plans [2], we went through a more systematic approach to assess the impact of a given cryptographic function. In our analysis we considered how the primitive affects the Bitcoin ecosystem across multiple dimensions. For instance, we thought about modifications on local (e.g. wallet) vs. global (e.g. blockchain) level, the role of the attacker (e.g. simple client vs. miner), as well as the "way" in which a given primitive might be broken (e.g. first vs. second pre-image collision). In order to keep the problem more tractable, we assume that any collisions and pre-images that can be created are meaningful, and do not discuss attacks that just speed up the time of computation in order to gain majority of the network. We summarize our findings here.

4.1 Hash160

A first pre-image attack (i.e. one that given y returns x such that $h(x) = y$) exposes the public key of a P2PKH script and the inner script of a P2SH script.[7] Though there is some impact on privacy/anonymity, this is minimal, as the intent of this hash is not to protect data, but to shorten the input's length while protecting from collisions. Moreover, the protected coins are safe, because there is no way to get the EC private key from the public key.

[4] See `https://bitcointalk.org/?topic=102395` for details.

[5] `http://www.nilsschneider.net/2013/01/28/recovering-bitcoin-private-keys.html` contains a discussion.

[6] A more formal explanation is given in Appendix A.

[7] More concretely, it exposes *a* public key/script that hashes to the given value. It is not necessary that it is the original value, but as mentioned we only focus on meaningful attacks.

Similarly, being able to create a collision (i.e. find $x_1 \neq x_2$ such that $h(x_1) = h(x_2)$) does not have particularly exciting consequences. Specifically, and even if the collision is meaningful in the context of Bitcoin, you merely have two public keys (or scripts), for which you do not have the corresponding private keys. That said, assuming you have the private keys as well (and perhaps just one suffices), you can create a plausible deniability scenario, in which you ask for payment to x_1, but claim payment through x_2. Because only the hash is revealed at time of payment (through the script), and because $h(x_1) = h(x_2)$, you can claim that somebody else stole your coins, and rely on the "kindness" of the communicating party to transfer the funds to a different address.

4.2 Hash256

Overall, despite the fact the attacks on Hash160 do not result on anything groundbreaking, if Hash160 is broken due to SHA-256, then so will Hash256, but with more interesting results. The main effect of a first pre-image attack is that it trivially results in faster mining, with all of the associated side-effects of increased control over the mining power (e.g. double spending).

Collisions are much more dangerous. Given a (meaningful) collision, you can ask for a signature on an innocuous transaction (e.g. pay 1$ to address X), but actually transmit a malicious transaction (e.g. pay 1,000,000$ to address Y), and the transaction will go through correctly, as the signature scheme operates on the hash of the transaction. Moreover, this same technique can be used to prevent a legitimate transaction from going through, or even splitting the network due to different transactions or even entire Merkle trees with the same hash. It can also be used to fool the signature cache present in clients, and can possibly be used to exploit the alert mechanism (Section 3.2), but such exploitation seems unlikely and would heavily depend on the circumstances.

4.3 ECDSA

A main vulnerability with the signature scheme would be to be able sign any message given just the public key (equivalent to recovering the private key from the public key, possibly also given access to some innocuous signatures). It would be trivial to then steal or destroy keys assuming the public key is known, but this is *not* necessarily always the case. Specifically, if the key has not been re-used and is not used in a multi-signature scheme (which does not hash keys), it is protected with Hash160. This means that funds are safe for the time being, though the security hinges on the fact that the public key remains secret! As a result, the key cannot be used to make transactions before moving to a new scheme (Section 5), because to make transactions, one needs to reveal the keys, which can then be used to prevent transactions or split the network by submitting different signatures (i.e. attempting to double spend somebody else's coin).

A somewhat less radical vulnerability would be a malleability attack, where given the signature $s(x)$ of x, one can determine the signature of $f(x)$ as $s\left(f\left(x\right)\right) =$

$g(s(x))$ for some functions f, g. The result is similar in that one can ask for a signature of an innocent transaction but submit the modification.

In either of the two scenarios presented, the alert system is compromised, and it becomes clear that address reuse brings more than just anonymity concerns.

4.4 Others

As mentioned in Section 3.3, there are other primitives that are used in Bitcoin, but do not appear in the blockchain. One example are the various cryptographic opcodes, but since standard transactions restrict their use, they are not of great importance. The problems of storing keys and wallets are interesting, but not within scope for this work, especially because access needs to occur through an external avenue (e.g. local malware, or website compromise). The one exception is hierarchical deterministic wallets, where if SHA-512 is broken, it would be possible to derive the master key (and consequently all derived private keys), as well as link individual addresses/transactions, thus having an impact on anonymity.

5 Migration Plans

The Bitcoin community has created a "wiki" which among other things describes some contingency plans "for the first few days after a catastrophic failure of the network in order to save time should any such failure actually occur" [2]. Though the focus is on "getting the word out" and "coordination", the wiki describes some scenarios in which these plans will be necessary, and those include broken SHA-256 and ECDSA algorithms. It is important to note the plans themselves seem to only be high-level, and not completely well thought-out, since the wiki states that "once the plans themselves are well-accepted, code implementing the plans can be written and tested in case the code is ever required" [2].[8]

As a result, in this section we take a more systematic approach to explain the problems with these high-level plans for the cases when a cryptographic primitive is already broken or will be broken in the near future. Specifically, we discuss the mechanisms for migrating hashes in Section 5.1, and find that the transition should be smooth. However, we find in Section 5.2 that using "some other signing algorithm" and "automatically send[ing] all old transactions somewhere else using the new algorithm" [2] is too abstract to cover all the scenarios. Specifically, we find that trying to interpret this suggestion in a straightforward manner requires either some degree of centralization, or will lead to some lost/stolen coins. As a result, we propose a Zero-Knowledge Proof (ZKP) approach in Section 5.3.

5.1 Switching Hashes

In Section 5.1.1 we summarize the contingency plans in case of a broken SHA-256, and perform our analysis in Section 5.1.2.

[8] The plans for the broken crypto algorithms suggest that "code for all of this should be prepared" [2], but to the best of our knowledge, no such mechanism has been built into Bitcoin.

5.1.1 The Plans The wiki describes plans in case of a "severe, 0-day failure of SHA-256", where "first/second preimage resistance or collision resistance can be defeated with only a few days of work" [2]:

- Users will be notified to shut down their clients. Note that the attacker may be able to send valid alerts, which could disrupt notification efforts.
- OP_CHECKSIG will be changed to use some other hash outside of old blocks.
- All addresses in the version-1 chain that have a known public key and at least one unspent output will have their public keys hardcoded into the client. When a version-2 transaction spends one of these version-1 outputs, the hardcoded public key will be used instead of the hash.
- The version-1 chain will be securely hashed into a hash tree. At least the root of the version-1 tree will be hardcoded into the client.
- All hashing Bitcoin does will use the new hashing algorithm.

5.1.2 The Analysis First of all, it is worth mentioning that the plans above only discuss SHA-256, and ignore RIPEMD-160. As we saw in Section 4.1, this is not problematic, as a broken Hash160 does not result in interesting attacks.

Shutting down the clients to not accept new transactions is perfectly reasonable, since collisions can be created. Moreover, from Section 4.2, it is clear that even if one can get pre-images, there are no issues with historical blocks or transactions. However, in order to maintain the previous transactions, one needs to snapshot the blockchain prior to any new transactions, even under the new hash, and Bitcoin already provides a mechanism for checkpoint lock-ins.[9] In principle, an entire new tree is not necessary to ensure security, but it is necessary for efficiency (in verifying the checkpoint and intermediate transactions), so the entire data structure (such as a Merkle tree) needs to use the new hash.

The plans also suggest hard-coding known public keys with unspent outputs. Though this might be prudent, it is unclear what additional complexity this will introduce, and by our discussion it shouldn't be necessary: given just an address hash, even if you get a public key, deriving the private key should be infeasible assuming the ECC algorithm is still strong. This of course assumes that the public key was not chosen with bad randomness, but in that case hardcoding the key would not have helped anyways.

Finally, note that any change to a new hash algorithm clearly necessitates a hard fork. Though one could try to switch the inner hash in order to only result in a soft fork, this would completely defeat the purpose as the combined hash would still be vulnerable, and would introduce considerable additional complexity when further chaining of hashes is necessary (e.g. for the Merkle tree).

Overall, we see that the plans for a broken SHA-256 hash are sufficient, and the existing code for checkpoints and Merkle trees should make the transition relatively painless.

[9] See `https://github.com/bitcoin/bitcoin/blob/master/src/chainparams.cpp`

5.2 Interpreting the Signature Scheme Upgrades

In Section 5.2.1 we summarize the contingency plans for a broken ECDSA algorithm, and try to interpret them in Section 5.2.2.

5.2.1 The Plans The wiki describes the response for the scenario where"an attacker can sign for a public key that he does not own the private key for in only a few days of work" [2]:

- If the attacker can't get the private key from the public key easily and a stronger algorithm that can use ECDSA keys is available:
  - Switch to the stronger algorithm.
  - Get users to update. Alerts will be compromised.
- Otherwise:
  - OP_CHECKSIG should use some other signing algorithm.
  - As soon as the new version of Bitcoin is run, it should automatically send all old transactions somewhere else using the new algorithm.
  - Get users to update immediately. Alerts will be compromised.

5.2.2 The Interpretation In either of the described cases, it is clear that users will need to update to a new algorithm immediately, and will not be able to trust alerts. "If the attacker can't get the private key from the public key easily and a stronger algorithm that can use ECDSA keys is available", it is indeed trivial to make changes, because users do not need to think about how to protect their keys, or how to switch to new ones, and instead just need to update their software.

If this is not the case, however, things are not very clear, and the proposal of "automatically send[ing] all old transactions somewhere else using the new algorithm" is very vague. There are two scenarios to consider: (1) having to migrate to new keys and (2) having to protect the public key because an attacker could use it to derive the private key. In this section, we try to interpret how these two scenarios would be dealt with. We suggest what we think would be reasonably straightforward approaches, and discuss their drawbacks.

In the case where Bitcoin needs to migrate to a new algorithm, and there is some transition time available (because the algorithm is not broken yet), one possibility is to introduce a migration script (in the sense of the standard scripts in Section 2.1.1) to switch keys. This could be done in a soft-fork manner (e.g. by embedding data in an OP_RETURN transaction), but if this period is not long enough, coins may be permanently lost. This is not entirely different from having to change bank notes, or migrating to a new currency in real economies, where there is only a set period to exchange the older currency for the new one, but it does raise some concerns with users who are not frequently present or online. Moreover, in centralized scenarios with banks, much of this process is automated, and the fact that in Bitcoin, there is no external way of contacting owners or proving ownership in any other way, this proposal might be met with some resistance from the Bitcoin community.

Assuming, however, that there is a 0-day vulnerability, so that there is no transition period, it is unclear how to transition to a new algorithm. First of all, if the public key is revealed in the blockchain, and one can get private keys from public keys, then there is no mechanism internal to Bitcoin that can tie the owner of an old address to a new address, as addresses are so closely linked to identity. As a result, coins in these addresses may be permanently stolen. However, there may be some out-of-band mechanisms for well-known addresses that can verify their identity (e.g. pools, online wallets, etc.) that could then have their new keys hard-coded into the updated client.

Finally, for addresses that are still protected under an unbroken hash, migrating to a new key would be easy if there is a trusted central authority (e.g. the core devs). You simply send them (outside of Bitcoin, so over email for instance) the hash in the blockchain, together with the old key that hashes to the given value, and the new key and these get hardcoded into the new client. However, this is problematic for two reasons: first of all, it goes against Bitcoin's decentralized philosophy, and second it suffers from the same issue of having to do the migration over a certain period of time, so that the keys can be hardcoded into the new client. As a matter of fact, transition will be harder because either this migration is only done once (in which case coins are lost), or migration is done multiple times so that the downtime is not excessive (in which case coins will be temporarily unavailable, and it is not certain that all nodes will update multiple times). There is, of course, the possibility of duplicating the alert mechanism to disseminate the new keys as they come along, but once more this centralization leaves something to be desired.

Overall, we see that if the private key cannot be derived from the public key and if there is an algorithm that can still use the same keys, then the transition is easy. If there is no such algorithm, but the private key is still safe, transitioning can still happen relatively easily, but the transitory period may result in lost (not stolen) coins. Finally, if the private key can be derived from the public key, straightforward approaches require some degree of centralization or some out-of-band mechanism to deal with Bitcoin's lack of identities.

5.3 A Zero-Knowledge Proof Solution

As discussed above, if adversaries can get the private key from the public key, the easiest way to switch keys is through a trusted third party. The problem, of course, is that this trust does not necessarily exist in a distributed scenario, so revealing the old public key should be avoided if possible. In this section, we propose and discuss two mechanisms: a commitment scheme that partially works, and a more complete Zero-Knowledge Proof (ZKP) solution. In short, the commitment scheme works by combining the hash of the existing key (which is already contained in the blockchain) and the new key, adding them to the blockchain, and after a few confirmation blocks revealing the old key. Though this does have the effect of tying the old and new key together, the revelation of the previous key results in some possible problems similar to double spending attacks. The ZKP approach we propose still ties the old key with the new one in

the same fashion, but does not actually reveal the key at the end, so it does not suffer from this drawback. In the discussion that follows, s refers to some secret (which in our case is the previous public key), h is a secure hash function, and p is the new public key we want to switch to.

At a high-level, the solution works as follows: the user asks to migrate the hash to the new public key, and then he is assigned a random miner to communicate with. They run a ZKP protocol, and the miner then includes the transcript in the blockchain via a transaction. This is summarized in Figure 1.

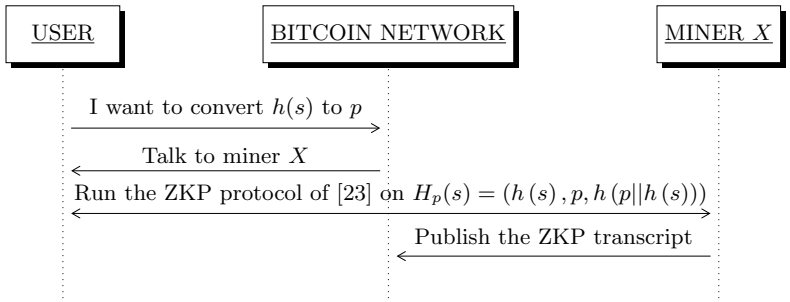


Fig. 1. Applying the ZKP protocol to Bitcoin

However, some of the finer details require a bit more motivation, so we start with the commitment scheme, which uses a two-pass approach that relies on revealing the key at the end. In this scheme, in round 1, add $M_1 = (h(s), p, h(s||h(s)||p))$ to the blockchain (e.g. via OP_RETURN), and in round 2, after M_1 has been in the blockchain for at least X blocks (where $X = 6$ in the standard implementation) reveal $M_2 = (s, h(s), p)$. Certainly, this revelation ties the old key s to the new key p, but assumes that one can efficiently verify that M_1 is the first message to be added in the blockchain that contains $h(s)$ and for which there is a corresponding M_2. In essence, there are two problems: (1) there is nothing stopping someone from creating spurious M_1 messages, with a well-known $h(s)$ part, but for which the second part of the message does not correspond to a valid hash, and (2) because the revelation contains the secret s, one needs to ensure that this transaction makes it to the blockchain, before someone recreates the same messages but with a different public key p'.

Though it is possible that by using extra data structures one can solve this problem, it is worth taking a step back and abstracting away from Bitcoin. We see that a better way of phrasing the problem of switching over to a new key is the following: given $h(s)$ for some secret s and hash function h, how can you reveal that you know s, without revealing s, *and* also tie your revelation (proof) to a new value p? Incorporating p is the easy part informed by the commitment scheme: it is equivalent to just asking for the pre-image proof for $H_p(s) = (h(s), p, h(p||h(s)))$. Note that $H_p(s)$ can be computed just from p

(which is given in plaintext) and $h(s)$ (which exists in the blockchain), so it does not require revealing the secret s.

The key idea is to use the garbled circuit protocol introduced in [23] in order to achieve a Zero-Knowledge Proof for knowing s such that $y = H_p(s)$. It would be very hard to explain both the protocol and the theory behind the protocol within the confines of this report, so we briefly summarize the main points, and turn to how to apply this protocol in the context of Bitcoin.

One of the building blocks is the 1-out-of-2 *oblivious transfer* protocol, where, informally, the sender has two messages m_0, m_1, and the receiver a bit b. The receiver wants to know m_b, without revealing b, and the sender wants to reveal m_b without revealing m_{1-b}. Such a protocol was introduced by Even, Goldreich, and Lempel in 1985 [17], and can be used with any public-key cryptosystem. The second building block is a *garbled circuit/scheme*, which although introduced by Yao, was only recently formalized by Bellare, Hoang, and Rogaway [6]. One of the ideas behind garbled circuits is that a function can be represented as a sequence of wires and logic gates ("circuit"), and that the circuit can be obfuscated by "encrypting" the wires. Specifically for a wire w_i there are two keys, K_i^0 and K_i^1 corresponding to the values $w_i = 0$ and $w_i = 1$. In order to compute the result at the end on input $w = (w_0, \ldots, w_n)$, one needs exactly one of K_i^0, K_i^1 for all i, so by using oblivious transfer, the sender (circuit creator) will only reveal one of them, while the receiver (circuit evaluator) will not reveal which one was chosen (and hence the value w_i will remain secret). This is the key idea behind the protocol in [23], which allows this proof to be Zero-Knowledge (first formally defined in [21]): the verifier is (probabilistically) convinced that the prover knows the secret value, but learns nothing more than what he would know by simulating the interaction himself.

One of the problems with this protocol (in terms of applying it to Bitcoin) is that it is interactive and that it relies on private randomness.[10] Though splitting the interaction across multiple blocks is conceivable, it would take too long to complete. Instead, executing the protocol externally (e.g. with a miner directly) and then publishing the full transcript (with the verifier's but not the prover's randomness) would be ideal. Publishing the full transcript is necessary for someone to verify the *run* of the protocol, because by the very definition of ZKPs, the verifier would not be convince anyone else independently. The only problem that still needs to be solved is how to choose the randomness so that the prover cannot cheat (e.g. by knowing the randomness beforehand) and so that it verifiable within Bitcoin. The latter is straightforward: the verifier randomness needs to be revealed as part of the message at the end. As for proving that the verifier and the prover are not colluding, it suffices to ensure that the verifier is chosen randomly in proportion to computational power, so that probabilistically, the prover is not able to collude with the verifier assuming he does not control a large part of the mining power. And, of course, to ensure *soundness*

[10] Note that in general, it is possible to do non-interactive Zero-Knowledge Proofs [9] and use public randomness [11], but this usually requires reductions to NP, and is thus inefficient, while the given protocol is efficient in practice.

(i.e. to ensure that cheating provers cannot convince others with high probability), multiple runs with different verifiers will need to be completed, though the actual number will depend on concrete probability requirements set by the Bitcoin developers.

Overall, we see that a commitment scheme goes in the right direction of solving the key migration problem, but requires better data structures to ensure that the reveal phase cannot be exploited, while the ZKP solution requires some interactions outside of the Bitcoin protocol, but which are still distributed in nature, and which are published on the blockchain as the proof, as opposed to being hardcoded in the client.

6 Conclusions

In conclusion, we systematically looked at the cryptographic primitives used in the Bitcoin cryptocurrency, and identified the types of attacks that will be possible, should these primitives be broken. We also looked at the contingency plans for such scenarios, and concluded that even though switching hashes is easily accomplished through the existing checkpoint mechanism, the plans for changing signature scheme are underspecified. Moreover, the straightforward solutions either require centralization or can result in lost/stolen coins, so to that end, we proposed a Zero-Knowledge Proof and a commitment scheme solution to the problem of changing the signature scheme in a distributed fashion. These solutions are particularly interesting, because they allow moving to a new scheme even *after* the old one is broken, assuming that the key is protected by a hash that is still strong,[11] illustrating further that addresses should not be reused.

Overall, we believe this to be an important topic of research, because if Bitcoin wants to remain a long-term currency, users need to put trust and faith into the system that their coins will still be there even as the crypto gets inevitably broken in the future. Future work should thus focus on understanding how Bitcoin addresses and scripts are currently being used, so that there is a better comprehension of the amount of addresses and coins that would be vulnerable. Moreover, it would be interesting to see how quickly the Bitcoin system should be updated so that the currency is not devalued, and overall ensure that the high-level contingency plans are reflected within the Bitcoin code.

References

1. Bitcoin source code. `github.com/bitcoin/bitcoin`. Accessed: 2015-09-15.
2. Bitcoin Wiki: Contingency plans. `https://en.bitcoin.it/wiki/Contingency_plans`. Accessed: 2015-09-15.

[11] To this end, the multi-signature payment scheme should move towards concealing public keys under hashes (e.g. through the P2SH directly), instead of revealing the public key directly as is currently the case.

3. Androulaki, E., Karame, G., Roeschlin, M., Scherer, T., and Capkun, S. Evaluating user privacy in Bitcoin. In *Financial Cryptography and Data Security*. 2013.
4. Andrychowicz, M., Dziembowski, S., Malinowski, D., and Mazurek, L. Secure multiparty computations on Bitcoin. In *IEEE Symposium on Security and Privacy (SP)* (2014).
5. Antonopoulos, A. M. *Mastering Bitcoin: Unlocking Digital Crypto-Currencies*, 1st ed. O'Reilly Media, Inc., 2014.
6. Bellare, M., Hoang, V. T., and Rogaway, P. Foundations of garbled circuits. In *CCS* (2012).
7. Bentov, I., and Kumaresan, R. How to use Bitcoin to design fair protocols. In *CRYPTO*. 2014.
8. Biryukov, A., Khovratovich, D., and Pustogarov, I. Deanonymisation of clients in Bitcoin p2p network. In *CCS* (2014).
9. Blum, M., Feldman, P., and Micali, S. Non-interactive zero-knowledge and its applications. In *STOC* (1988).
10. Bonneau, J., Miller, A., Clark, J., Narayanan, A., Kroll, J., and Felten, E. SoK: Research perspectives and challenges for Bitcoin and cryptocurrencies. In *IEEE Symposium on Security and Privacy (SP)* (2015).
11. Canetti, R., Lin, H., and Paneth, O. Public-coin concurrent zero-knowledge in the global hash model. In *Theory of Cryptography*. 2013.
12. Christin, N. Traveling the Silk Road: A measurement analysis of a large anonymous online marketplace. In *WWW* (2013).
13. Cochran, M. Notes on the wang et al. 2^{63} sha-1 differential path. Cryptology ePrint Archive, Report 2007/474, 2007. `http://eprint.iacr.org/2007/474`.
14. Coron, J.-S., Dodis, Y., Malinaud, C., and Puniya, P. Merkle-damgrd revisited: How to construct a hash function. In *CRYPTO*. 2005.
15. Dobbertin, H., Bosselaers, A., and Preneel, B. RIPEMD-160: A strengthened version of RIPEMD. In *Fast Software Encryption*. 1996.
16. Eskandari, S., Barrera, D., Stobert, E., and Clark, J. A First Look at the Usability of Bitcoin Key Management. In *NDSS Workshop on Usable Security (USEC)* (2015).
17. Even, S., Goldreich, O., and Lempel, A. A randomized protocol for signing contracts. *Commun. ACM 28*, 6 (June 1985).
18. Eyal, I. The miner's dilemma. *CoRR abs/1411.7099* (2014).
19. Eyal, I., and Sirer, E. Majority is not enough: Bitcoin mining is vulnerable. In *Financial Cryptography and Data Security*. 2014.
20. Garay, J. A., Kiayias, A., and Leonardos, N. The Bitcoin backbone protocol: Analysis and applications. In *EUROCRYPT* (2015).
21. Goldwasser, S., Micali, S., and Rackoff, C. The knowledge complexity of interactive proof systems. *SIAM J. Comput. 18*, 1 (Feb. 1989).
22. Hearn, M. Why is Bitcoin forking? A tale of differing visions. `https://medium.com/faith-and-future/why-is-bitcoin-forking-d647312d22c1`, August 15, 2015. Accessed: 2015-09-15.
23. Jawurek, M., Kerschbaum, F., and Orlandi, C. Zero-knowledge using garbled circuits: How to prove non-algebraic statements efficiently. In *CCS* (2013).
24. Joux, A. Multicollisions in iterated hash functions. application to cascaded constructions. In *CRYPTO* (2004).
25. Karame, G. O., Androulaki, E., and Capkun, S. Double-spending fast payments in Bitcoin. In *CCS* (2012).

26. Katz, J., and Lindell, Y. *Introduction to Modern Cryptography*. Chapman & Hall/CRC, 2007.
27. Klima, V. Tunnels in hash functions: Md5 collisions within a minute. Cryptology ePrint Archive, Report 2006/105, 2006. http://eprint.iacr.org/2006/105.
28. Kroll, J. A., Davey, I. C., and Felten, E. W. The economics of Bitcoin mining, or Bitcoin in the presence of adversaries. In *Workshop on the Economics of Information Security* (2013).
29. Kumaresan, R., and Bentov, I. How to use Bitcoin to incentivize correct computations. In *CCS* (2014).
30. Leurent, G., and Wang, L. The sum can be weaker than each part. In *EUROCRYPT*. 2015.
31. Nakamoto, S. Bitcoin: A peer-to-peer electronic cash system, http://bitcoin.org/bitcoin.pdf, 2008.
32. National Institute of Standards and Technology. *FIPS PUB 180-4: Secure Hash Standard*. Mar. 2012.
33. QuantaBytes. A survey of Bitcoin transaction types. http://www.quantabytes.com/articles/a-survey-of-bitcoin-transaction-types, April 13, 2014. Accessed: 2015-09-15.
34. Reid, F., and Harrigan, M. An analysis of anonymity in the Bitcoin system. In *Security and Privacy in Social Networks*. 2013.
35. Ron, D., and Shamir, A. Quantitative analysis of the full Bitcoin transaction graph. In *Financial Cryptography and Data Security*. 2013.
36. Standards for Efficient Cryptography. Sec 2: Recommended elliptic curve domain parameters version 2.0. http://www.secg.org/sec2-v2.pdf, 2010.
37. Tschorsch, F., and Scheuermann, B. Bitcoin and beyond: A technical survey on decentralized digital currencies. Cryptology ePrint Archive, Report 2015/464, 2015. https://eprint.iacr.org/2015/464.
38. Wang, X., Yin, Y. L., and Yu, H. Finding collisions in the full sha-1. In *CRYPTO* (2005).
39. Wang, X., and Yu, H. How to break md5 and other hash functions. In *EUROCRYPT* (2005).
40. Yarom, Y., and Benger, N. Recovering OpenSSL ECDSA nonces using the FLUSH+RELOAD cache side-channel attack. Cryptology ePrint Archive, Report 2014/140, 2014. https://eprint.iacr.org/2014/140.
41. Yermack, D. Is Bitcoin a real currency? An economic appraisal. Tech. rep., National Bureau of Economic Research, 2013.

A Nested Hashes

Though it is not entirely clear why Bitcoin always uses two nested hashes, people suspect it may be to prevent "length extension attacks" that exist in Merkle-Damgård-based hash functions such as SHA-256 and RIPEMD-160 [14]. Indeed, even if hash function H is vulnerable to a length extension attack such that $H(x||m) = f(H(x), m)$ for all m and function f, one cannot use (just) this relation to derive $H(H(x||m))$ from $H(H(x))$. However, as we see in this section, nested hashes does bring some possibly unexpected consequences.

Here, we consider a hash function as a one-way function which is not based on hardness assumptions, i.e. for which we do not have a proof that it is hard

to invert.[12] Moreover, in order to have proofs on asymptotic complexity, we typically work on families of functions, but we do not make this explicit in the proofs below.We prove in Lemma 1 that a pre-image attack on the nested hash results at either a pre-image attack on the inner hash, or a collision attack on the outer hash and in Lemma 2 that a collision for a nested hash is a collision for one of the two hashes.

Lemma 1 (Preimage Attack). *Let h_1, h_2 be hash functions with $h_1 : \{0,1\}^n \to \{0,1\}^{l_1(n)}$ and $h_2 : \{0,1\}^{l_1(n)} \to \{0,1\}^{l_2(n)}$, and define $h : \{0,1\}^n \to \{0,1\}^{l_2(n)}$ by $h = h_2 \circ h_1$. Then, if there is an efficient (for some definition) first pre-image algorithm for h, there is an efficient algorithm that given $y = h_1(x)$ returns x' such that either $h_1(x') = y$ or $h_2(h_1(x')) = h_2(y)$.*

Proof. Let A be the probabilistic algorithm running time in time $f(n)$ (not necessarily polynomial, but "efficient"), such that

$$\Pr_{X \xleftarrow{R} \{0,1\}^n} \left[A\left(h\left(X\right), 1^n\right) \in h^{-1}\left(h\left(X\right)\right)\right] = p(n)$$

where $p(n)$ is not negligible. We construct algorithm A' as follows: on input $y = h_1(x)$, A' calculates $z = h_2(y)$ and runs A on z. With probability $p(n)$, A returns some x' such that $h(x') = z$. If $h_1(x') = y$, then this is a preimage for y under h_1, else $h_2\left(h_1\left(x'\right)\right) = h_2(y)$ for a collision under h_2. The probability of success (under the same distribution $X \xleftarrow{R} \{0,1\}^n$) for A' thus remains the same, while its runtime is only increased by the calculation of the h_2 function a constant number of times, which by definition is polynomial.

Remark 1. Assuming that for every $y \in \{0,1\}^{l_1(n)}$ there exists $x \in \{0,1\}^n$ such that $y = h_1(x)$, the above algorithm can be turned into an algorithm for finding pre-images for h_2 as well. Indeed, given $z = h_2(y) = h(x)$, running A on z would return x' such that $z = h(x') = h_2\left(h_1\left(x'\right)\right)$, so $h_1(x')$ is a pre-image for z.

However, we cannot have the same sense of efficiency if we do not restrict ourselves to polynomial algorithms. Specifically, even if $l_1(n) = \frac{n}{\alpha}$ for some constant $\alpha > 1$, if A is exponential (but better than brute-force) in n, this does not translate to being efficient in $l_1(n)$ for the same definition of efficiency. Moreover, we can also not keep the same definition of high probability, since A runs over $\{0,1\}^n$, while we want our modified algorithm to run over $\{0,1\}^{l_1(n)}$. This would require a stronger sense of uniform distribution of the outputs of h_1, but because h_1 is deterministic this is harder to define.

Note that having a pre-image under h_1 of any $y \in \{0,1\}^{l_1(n)}$ may be a desirable (and even expected) property of a hash function, but it is not necessary. Suppose, for instance that $h : \{0,1\}^n \to \{0,1\}^m$ is a hash function. Then

$$h'(x) = \begin{cases} 1 & \text{if } h(x) = 0 \\ h(x) & \text{otherwise} \end{cases}$$

[12] For definitions of one-way functions, probabilistic polynomial-time algorithms (PPTs), negligible functions, and collisions please refer to cryptographic textbooks, such as [26].

is also easily seen to be a hash function from $\{0,1\}^n$ to $\{0,1\}^m$ that nonetheless completely misses 0.

Lemma 2 (Collision Attack). *Let h_1, h_2 be hash functions with $h_1 : \{0,1\}^n \to \{0,1\}^{l_1(n)}$ and $h_2 : \{0,1\}^{l_1(n)} \to \{0,1\}^{l_2(n)}$, and define $h : \{0,1\}^n \to \{0,1\}^{l_2(n)}$ by $h = h_2 \circ h_1$. Then, a collision (x_1, x_2) for h is either a collision for h_1 or $(h_1(x_1), h_1(x_2))$ is a collision for h_2.*

Proof. Let (x_1, x_2) such that $x_1 \neq x_2$ and $h(x_1) = h(x_2)$. Let $y_1 = h_1(x_1)$ and $y_2 = h_1(x_2)$. Then either $y_1 = y_2$, so (x_1, x_2) are a collision for h_1, or $y_1 \neq y_2$ with $h_2(y_1) = h(x_1) = h(x_2) = h_2(y_2)$, so (y_1, y_2) are a collision for h_2.

Though the above results are sufficient for the purposes of our report, they are not complete in the sense that they do quantify how many bits of security the composition provides. Future work could thus be inspired by Joux's work on multicollisions (i.e. for the concatenation combiner) [24] and Leurent and Wang's work on the XOR combiner [30].