# When The “Crypto” in Cryptocurrencies Breaks: Bitcoin Security Under Broken Primitives

Ilias Giechaskiel, Cas Cremers, Kasper Rasmussen
University of Oxford, Oxford, UK
{ilias.giechaskiel, cas.cremers, kasper.rasmussen}@cs.ox.ac.uk

**Abstract**—Digital currencies such as Bitcoin rely on cryptographic primitives to operate. However, past experience shows that cryptographic primitives do not last forever: increased computational power and advanced cryptanalysis cause primitives to break, and motivate the development of new ones. It is therefore crucial for maintaining trust in a cryptocurrency to anticipate such breakage. We present the first systematic analysis of the effect of broken primitives on Bitcoin. We analyze the ways in which Bitcoin’s core cryptographic building blocks can break, and the subsequent effect on the main Bitcoin security guarantees. Our analysis reveals a wide range of possible effects depending on the primitive and type of breakage, ranging from minor privacy violations to a complete breakdown of the currency. Our results lead to several suggestions for the Bitcoin migration plans, and insights for other cryptocurrencies in case of broken or weakened cryptographic primitives.

## 1 Introduction

CRYPTOCURRENCIES such as Bitcoin rely on cryptographic primitives for their guarantees and correct operation. Such primitives typically get weakened over time, due to progress in cryptanalysis and advances in the computational power of the attackers. It is therefore prudent to expect that, in time, the cryptographic primitives used by Bitcoin will be partially, if not completely, broken.

In anticipation of such breakage, the Bitcoin community has created a wiki page that contains draft contingency plans [1]. However, these plans are informal and incomplete: no adequate transition mechanism has been built into Bitcoin, and no plans for weakened primitives have been considered. Primitives rarely break abruptly: for hash functions, it is common that first a single collision is found. This is then generalized to multiple collisions, and only later do arbitrary collisions become feasible to compute. In parallel, the complexity of attacks decreases to less-than-brute-force, and computational power of attackers increases.

Even if such attacks are years away from being practical, it is crucial to anticipate the impact of broken primitives so that appropriate contingency plans can be put in place. Our work contributes towards filling this gap. We provide the first systematic analysis of the impact of broken primitives on Bitcoin.[1] By analyzing the failure of primitive properties, both in isolation and in combination, we describe the range of consequences different breaks have, and pinpoint their exact cause.

We show that a break in RIPEMD160 can allow an attacker to repudiate payments. SHA256 collisions and second pre-image attacks, as well as selective forgery of ECDSA signatures, all allow an adversary to steal or destroy coins. Finally, SHA256 pre-image attacks have the most severe consequences, as they allow an attacker to take complete control over the Bitcoin system, but only through exploiting the flexibility of the coinbase transaction.

Our investigations raise concerns about the currently-specified migration plans for Bitcoin, being overly conservative in some respects, while inadequate in others. To that end, we make suggestions regarding future iterations of Bitcoin in response to entirely broken and partially weakened primitives, and relate Bitcoin’s security model to that of other currencies.

## 2 Background

In this section, we give a description of Bitcoin, the popular peer-to-peer (P2P) cryptocurrency introduced in 2008 by Satoshi Nakamoto [2]. Fig. 1 shows a high-level view of the main component of Bitcoin—the blockchain—which will guide this section. The blockchain is a public log of all Bitcoin transactions that have occurred, combined together in components called blocks. Transactions use a scripting language which determines the owners of coins (Section 2.1), and it is up to “miners” to verify that only valid transactions occur. To ensure that nobody can change or remove past transactions, miners have to solve a hard computational puzzle, known as a Proof-of-Work (Section 2.2). The final component of Bitcoin is its underlying P2P network which enables distributed communication (Section 2.3).

### 2.1 Transactions and Scripts

Bitcoin is an electronic cash system [2], so *transactions* to transfer coins between users are central to its structure. A transaction is a list of inputs—unspent transactions in the blockchain—and a list of outputs—addresses to which to transfer the coins, whose unit is a “satoshi”, equal to $10^{-8}$ Bitcoins or BTCs. To ensure that only the owner can spend his coins, each input and output is accompanied by a *script*. For outputs, this “locking” script contains the conditions under which the output can be redeemed (*scriptPubKey*), while for inputs, an “unlocking” script contains a cryptographic signature (*scriptSig*) as proof that these conditions have been met. These *scripts* are sequences of instructions (*opcodes*) that get executed by special nodes called miners. To prevent Denial-of-Service (DoS) attacks exploiting computationally-intensive instructions, most nodes only accept

[1] This article extends a paper presented at ESORICS 2016.

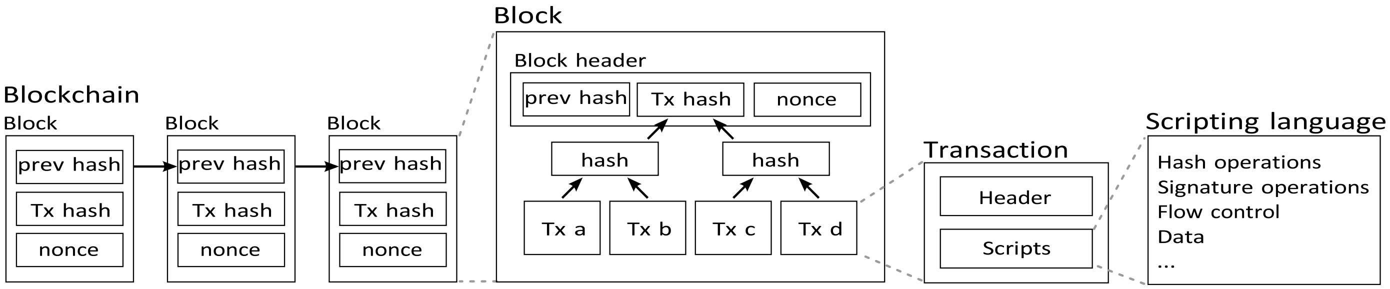

Fig. 1. The blockchain data structure. This forms the basis of the public, append-only ledger where all transactions are recorded.

the five *standard scripts*:

1) *Public-Key* The unlocking script must sign the transaction under this key.
2) *Pay-to-Public-Key-Hash (P2PKH)* The unlocking script must provide a public key that hashes to the given value, and must then sign the transaction.
3) *Multi-Signature* An M-of-N ( $N \leq 15$ ) multi-signature scheme provides N public keys, and requires M signatures in the unlocking script.
4) *Pay-to-Script Hash (P2SH)* This script is the hash of a non-P2SH standard transaction. The unlocking script provides the full script hashing to this value and any necessary signatures. This script is typically used to shorten the length of multi-signature transactions.
5) *Data Output (OP_RETURN)* The output cannot be redeemed, but can be used to store up to 40 arbitrary bytes, such as human-readable messages.

For a transaction to be valid, it must contain all the required fields, all signatures must be correct, and the scripts must be standard. This is a task that miners undertake for a small fee. In addition, *non-standard* scripts using different sequences of opcodes can be included in blocks for higher fees. We discuss these in the context of a recent SHA1 collision in Section 6.2.

### 2.2 Mining and Consensus

To ensure that no coin is used more than once, every transaction is made public through a global, append-only ledger called the *blockchain*, consisting of *blocks* combining transactions in a Merkle Tree. New blocks become a part of the blockchain through a process called *mining*: miners need to find a value (nonce) such that the hash of a block's header is less than a given target $h(hdr||nonce) < T$. The idea behind this *proof-of-work* (PoW) scheme is that the probability of creating the next block is proportional to the miner's computational power, and because miners receive transaction fees, they are incentivized to validate transactions and blocks. A summary is shown in Fig. 2.

Due to the probabilistic nature of mining, the presence of adversaries, and networking delays, miners may disagree on the current state of the blockchain. This is known as a *fork*. To deal with this issue, there are hard-coded blocks included in the clients, known as *checkpoints*, starting from the first block, called the *genesis block*. In addition, honest (non-adversarial) miners work on the longest blockchain they become aware of, when other nodes announce new blocks and transactions.

```
input  : Bitcoin block
output: valid or invalid
/* Verify block header                                  */
Verify Hash(block header) < target
Verify Merkle hash
Verify Hash(prev block) = prev_hash
/* Verify each transaction input in block               */
foreach transaction input in the block do
    Check that referenced output transaction exists and hasn't
     already been spent
    Verify signatures
end
```

Fig. 2. Procedure to verify a block's cryptographic primitives.

These temporary forks enable *double spending*: an adversary can have different transactions in different branches of the fork using the same inputs but different outputs. However, because the probability of "deep" forks where branches differ in the top $N$ blocks drops exponentially in $N$, receivers usually wait for multiple confirmation blocks.

### 2.3 Network

The last key component is the Peer-to-Peer (P2P) network for distributed operation. Transactions and blocks are broadcast by nodes to their peers, and then relayed further to flood the network if they meet the relay policies (to prevent DoS attacks). Not every node is a miner or necessarily has access to the full chain: "lightweight" clients that use Simple Payment Verification (SPV) only download headers and the relevant transactions (with the corresponding Merkle Trees).

Over time, the need for extensions or bugfixing motivates protocol changes. Since not all nodes upgrade at the same time, this may introduce forks. If the validation rules in the upgrade become stricter, then the protocol remains backwards-compatible, resulting in a *softfork*. A *hardfork*, on the other hand, is not backwards-compatible, and thus requires the entire network to upgrade, as old software would reject new transactions and blocks as invalid.

## 3 Broken Hashing Primitives

In this section, we look at the cryptographic hash functions in Bitcoin, and analyze the effect of a break in one of the properties of first and second pre-image and collision resistance.

TABLE 1
Summary of the effects on Bitcoin for different types of hash breakage.

| **Breakage** | **Address Hash** (*$H_A$*) | **Main Hash** (*$H_M$*) |
|---|---|---|
| Collision | Repudiate payment | Steal and destroy coins |
| Second pre-image | Repudiate payment | Double spend and steal coins |
| Pre-image | Uncover address | Complete failure of the blockchain |

## 3.1 Hashing in Bitcoin

In the original Bitcoin paper [2], the concrete primitives used are not specified: there were no "addresses" but just public keys, and the hash used for mining and the Merkle tree was just referred to as a hash function. The current Bitcoin implementation uses two hash functions.

**Main Hash** This hash function has an output of $256$ bits and requires applying SHA256 twice: $H_M(x) = SHA256(SHA256(x))$. It is the hash used for *mining* (Proof-of-Work): miners need to find a nonce such that the double SHA256 hash of a block header is less than a "target" hash. It is also used to hash transactions within a block into a *Merkle Tree*, a structure which summarizes the transactions present within a block. Finally, it is the hash used for transactions signed with a user's private key.

**Address Hash** The second hash function is used as part of the Pay-to-Public-Key-Hash (P2PKH) and the Pay-to-Script-Hash (P2SH) scripts. Its output is 160 bits, and it is concretely instantiated as $H_A(x) = RIPEMD160(SHA256(x))$.

## 3.2 Modeling Hash Breakage

In this section, we analyze how hashes break in terms of their building blocks.

### *3.2.1 Identifying Hashing Building Blocks*

A good cryptographic hash function $h(x)$ should offer three properties:

1) *Pre-image resistance* Given $y$ it is hard to find $x$ with $h(x) = y$.
2) *Second pre-image resistance* Given $x_1$, it is hard to find $x_2 \neq x_1$ with $h(x_1) = h(x_2)$.
3) *Collision resistance* It is hard to find distinct $x_1 \neq x_2$ such that $h(x_1) = h(x_2)$.

where "hard" refers to computational infeasibility. This is because hash functions have a fixed-length output, so collisions always exist.

We consider attacks against $H_A$ and $H_M$ abstractly, so that our arguments can be extended for any future version that uses the same structure. Currently, $H_A$ and $H_M$ are built using RIPEMD160 and SHA256. Table 1 contains a summary of our results, while we relate the attacks we discover back to the concrete primitives in Section 6.

## 3.3 Main Hash

In this section, we analyze the main hash $H_M$, which is used for mining, in Merkle Trees, and with signatures. We discuss all three use-cases separately.

### 3.3.1 Mining

We first investigate pre-image attacks against the block headers under two different attack scenarios, before turning to collision and second pre-image attacks.

**Attack 1: Pre-Image against Fixed Merkle Root** We show that the probability that an adversary with access to a preimage oracle can break mining is negligible. Miners search for block headers whose $n$-bit hash is below a target, which we assume starts with $d$ zeros. This assumption only introduces up to 1 bit of extra work, as there is always a unique $d$ with $T \leq 2^d < 2T$, for any target $T$.

If the adversary controls $b \leq n$ bits of the input, there are $2^b$ possible inputs to the hash function. These need to map to one of the $2^{n-d}$ values in the range $[0, 0^d 1^{n-d})$, and will be uniformly distributed across $2^n$ values. This gives the expected number of $b$-bit pre-images as $E[\#\ \text{pre-images}] = 2^b \cdot (2^{n-d})/(2^n) = 2^{b-d}$. The adversary can only query the pre-image oracle for specific target hashes. Because there are $2^{b-d}$ $b$-bit pre-images, distributed across the $2^{n-d}$ values, the probability that a given hash in $[0, 0^d 1^{n-d})$ has a $b$-bit pre-image is: $P[\text{correct pre-image}] = (2^{b-d})/(2^{n-d}) = 2^{b-n}$. This probability does not depend on $d$, as one might expect. This is because by increasing $d$ to reduce the number of valid hashes, the adversary also reduces the expected number of $b$-bit pre-images. Assuming the adversary is allowed $2^a$ queries to the oracle, the probability of breaking mining becomes $P[\text{success}] = 2^a \cdot 2^{b-n} = 2^{a+b-n}$.

To calculate $b$, we explore all fields in the block header. The version number (nVersion), as well as the hashes of the previous block header (hashPrevBlock), and of the current Merkle root hash (hashMerkleRoot) are fixed. However, the adversary has partial control over the remaining fields in the header. For the timestamp field (nTime), the value can be within 7200 seconds of the current median/average, giving the adversary approximately 13 bits of freedom. Moreover, the adversary has complete control over the 32 bits of the nonce (nNonce). The nBits field $0xAABBCCDD$ describes the target difficulty as $0xBBCCDD \cdot 256^{0xAA-3}$, with the protocol only checking that the produced number is at most the target value given by the consensus. At the time of writing, the target value is $0x18038b85$, granting the adversary approximately 27 bits of freedom.

Together the fields give $b = 72$. With $n = 256$, and allowing $2^{80}$ calls to the oracle, the probability of success is only $2^{80+72-256} = 2^{-104}$, which is negligible.

**Attack 2: Pre-Image against Variable Merkle Root** By varying the Merkle root, an adversary can break mining, though by the discussion of Attack 1, this cannot be achieved by simply reordering or excluding transactions. Instead the adversary must work backwards, by querying the oracle for a target Merkle hash and repeatedly querying the oracle to reconstruct the entire

Merkle tree. This would normally fail, as the transactions generated would not be valid due to incorrect signatures,[2] but Bitcoin does not enforce a minimum number of transactions in a block. Hence, miners can mine blocks with just the coinbase transaction which generates new coins, and which has a variable-length input of up to 100 bytes that is controlled by miners. A malicious miner with access to the pre-image oracle can then:

1) Pick an arbitrary target $T$ and get a pre-image for $H_M(a||x||b) = T$ where the desired $x$ is the `hashMerkleRoot` field, and $a, b$ are the remaining fields in a block header. Because the root is 256 bits, there is a pre-image with high-probability, but if not, repeat with some other random target $T'$.
2) Pick a length $l$ for the script, and fix all other fields for the coinbase transaction. Solve $H_M(a'||y||b') = x$ where $a', b'$ are the remaining fields for the coinbase transaction. Because the number of free bytes is up to 100, there is an $l$-bit pre-image $y$ with high probability. The miner then generates a coinbase transaction using $a', y, b'$ and combines it into a block using $a, b$. This block will have a hash of $T$ as desired.

**Collisions, Second Pre-Images** Collisions and second preimages are only useful for mining if the pre-images start with $d$ zeros. Assuming the pre-images contain valid transactions and signatures, a miner can fork the chain, but this only occurs with negligible probability.

### 3.3.2 Merkle Trees

**Altering existing blocks** A similar argument as for mining (Attack 1) shows that an adversary cannot find a valid second pre-image of an entire block except with negligible probability. Pre-images do not give the adversary new information, as they already accompany the hash value. Collisions are also not useful, as both values are attacker-controlled and cannot alter existing blocks.

**Attacking new blocks** For new blocks and transactions, an adversary with sufficient network control can use a collision or second pre-image to split the network, reject both blocks or reverse transactions, thus enabling double-spending. Preimages are again not useful, as they always accompany the hashed value.

### 3.3.3 Main Hash Usage in Signatures

In Bitcoin, signatures are over messages hashed with $H_M$. Therefore, a second pre-image attack or a collision on $H_M$ can be used to destroy and possibly steal coins: an adversary can ask for a signature on an innocuous transaction (e.g., pay 1 satoshi from address $X$ to address $Y$), but transmit a malicious one instead (e.g., pay 100 BTC from address $X$ to address $Z$) since there are enough bytes that the adversary controls to guarantee success with high probability. Note that for this attack to succeed, the adversary must still specify the same unspent transaction $X$ belonging to the victim for the signature to be valid, but can alter the amount (bounded by the total number of Bitcoins present in address $X$), and the destination.

TABLE 2
Effects of a broken signature scheme.

| Breakage | Effect |
|---|---|
| Selective forgery | Steal coins from public key |
| Integrity break | Claim payment not received |
| Repudiation | - |

### 3.4 Address Hash

The address hash is used in two contexts. First, in Bitcoin addresses, using Pay-to-Public-Key-Hash (P2PKH) scripts: an address is essentially $y = H_A(p) = RIPEMD160\ (SHA256\ (p))$ where $p$ is the public key (together with a checksum). Payments to addresses only use the hashed value $y$, but transactions to addresses require the full public key $p$ and the signature on the transaction. The second use is in Pay-to-Script-Hash (P2SH) scripts. A P2SH is $y = H_A(s)$ where $s$ is a standard script, typically a multi-signature transaction. Payments to a P2SH script do not reveal the pre-image, but transactions spending the coins require it and the signatures of the corresponding parties. We discuss them jointly, since the only difference between a P2PKH and a P2SH in this context is the number of required signatures.

**Pre-image** For previously spent outputs, or for reused addresses, $H_A$ is already accompanied by its pre-image. A pre-image thus can only reveal the public key(s) for unspent outputs. This has minimal privacy consequences since public keys are not tied to real identities, but it could enable an offline attack on the key. Assuming that the key was not chosen with bad randomness and there is no weakness in the signature scheme, the probability of success is still negligible.

**Second pre-image** A second pre-image gives the adversary access to a different public key or script with the same hash. However, because the adversary does not control the corresponding private key, he cannot use this to change existing transactions or create new ones. This is because preimages (whether a key or a script) are only revealed and verified when spent in transactions.

**Collision** Collisions are similar, though in this case both public keys are under the adversary's control, and again the adversary does not have access to the private keys. In both scenarios, there is a question of non-repudiation external to the protocol itself: by presenting a second pre-image of a key used to sign a transaction, a user/adversary can claim that his coins were stolen.

## 4 Broken Signature Primitives

In this section, we describe the use of digital signatures in Bitcoin, and analyze how a break in their unforgeability, integrity, or non-repudiation impacts Bitcoin. We summarize our results in Table 2.

[2] Although transaction malleability can allow the same signature to be valid for two different transactions [3].

### 4.1 Digital Signatures in Bitcoin

Bitcoin's digital signature scheme is the Elliptic Curve Digital Signature Algorithm (ECDSA) with the secp256k1 parameters, and is used to sign the main hash $H_M$ of transactions.

### 4.2 Modeling Signature Breakage Variants

The security of digital signature schemes is usually discussed in terms of three properties, which we define as follows:

1) *Unforgeability* No-one can sign a message $m$ that validates against a public key $p$ without access to the secret key $s$.
2) *Integrity* A valid signature $\{m\}_s$ does not validate against any $m' \neq m$.
3) *Non-repudiation* A valid signature $\{m\}_s$ does not validate against any public key $p' \neq p$.

where there is an implicit "except with negligible probability", due to hashing.

These properties are linked and a breakage in one usually implies a breakage in the others. In addition, they are often discussed in a much more abstract way: non-repudiation refers to the property that the signature proves to all parties the origin of the signature, but, in this case, we introduce it in a way that is more akin to Duplicate Signature Key Selection (DSKS) attacks [4].

### 4.3 Broken Signature Scheme Effects

We now analyze a break in each of these properties separately, starting with the last two, as neither of them can lead to an attack on their own.

**Integrity** In order for a break in the integrity of the signature scheme to be useful in Bitcoin, a signature of $H_M(m)$ must also be valid for $H_M(m')$. This involves $H_M$ in a non-trivial way, so we discuss this further in Section 5, but note that transaction malleability can cause the issuer of a transaction to think that his payment was not confirmed [3].

**Non-repudiation** For non-repudiation, we note that for transactions, even if a signature verifies under a different key, the address hashes of the two public keys must match. A break thus involves $H_A$, so we discuss this case further in Section 5.

**Unforgeability** When it comes to unforgeability, we can distinguish between various types of breaks [5]: *Total break* to recover the private key, *universal forgery* to forge signatures for all messages, *selective forgery* to forge signature on a message of the adversary's choice, and *existential forgery* to produce a valid signature that is not already known to the adversary.

Because the message to be signed must be the hash of a valid transaction, an existential forgery is not sufficient since the probability that it corresponds to a valid message is negligible. Selective forgery on the other hand can be used to drain a victim's wallets. From this perspective, selective forgery and a total break have the same effect. However, as we discuss later, the type of breakage influences how to upgrade to a new system. It is worth noting that an adversary does not necessarily have access to a user's public key, since addresses that have not been reused are protected by the address hash $H_A$.

TABLE 3
Multi-breakage effects: combining broken hashes and signatures.

| | Signature Property | | |
|---|---|---|---|
| **Hash Property** | **Selective forgery** | **Integrity break** | **Repudiation** |
| **Address Hash ($H_A$)** | | | |
| Collision | Repudiate transaction | - | Change existing payment† |
| Second pre-image | Steal all coins | - | Change existing payment |
| Pre-image | Steal all coins | - | - |
| **Main Hash ($H_M$)** | | | |
| Collision | Steal coins | Steal coins† | - |
| Second pre-image | Steal coins | Double spend† | - |
| Pre-image | - | - | - |

† Achieving this requires a modification of definitions. See text for details.

## 5 Multi-Breakage

In this section, we analyze how combinations of breakages in different primitives can impact Bitcoin. Because $H_A$ and $H_M$ are not used together, we only consider a break in the signature algorithm in combination with a break in one of the two hashes. The results are summarized in Table 3.

### 5.1 Address Hash and Signature Scheme

**Signature Forgery** Combining a selective forgery with a *first or second pre-image* break of the address hash can be used to steal all coins that are unspent. Generating two public keys $p, p'$ with $H_A(p) = H_A(p')$ (*collision*) whose signatures the adversary can forge does not have a direct impact, since the adversary controls both addresses. However, it appears as if two different users are attempting to use the same coin, thus raising a question of repudiation, which we discuss in Section 6.

**Signature Integrity** As the messages signed for transactions do not involve $H_A$, this combination does not increase the adversary's power.

**Signature Repudiation** A *pre-image* attack on $H_A$ is not useful as the public key is already known. For a *second pre-image*, assume that given a message $m$ (the hash of a transaction) and a public key $p$, an oracle returns $p'$ such that $H_A(p) = H_A(p')$ and the signature of $m$ under $p$ also validates against $p'$. Since the same signature validates for both keys, an adversary can replace $p$ by $p'$ in the unlocking script. Though this does not give the adversary immediate monetary gain, a transaction in the blockchain has been partially replaced.

For *collisions*, assume that given a message $m$, an oracle returns two public keys $p, p'$ such that $H_A(p) = H_A(p')$ and the signature of $m$ under $p$ validates under $p'$. If the adversary does not have access to the private keys, he cannot sign the transaction. Otherwise, the effect is identical to the second pre-image case, where the adversary can replace part of a transaction in the blockchain.

### 5.2 Main Hash and Signature Scheme

**Signature Forgery** As explained in Section 3.3, none of the potential attacks using the hash $H_M$ required a break in the signature scheme. The partial exceptions were mining under a pre-image break, and transactions with second pre-image or collision breaks. We discuss each possibility below.

For *mining*, a pre-image attack is useful when working backwards from a fixed target to get a pre-image for the Merkle root and turn

TABLE 4
Effects of concrete primitive breakage on the current version of Bitcoin.

| Breakage | Effect |
|---|---|
| **SHA256** | |
| Collisions | Steal and destroy coins |
| Second pre-image | Double spend and steal coins |
| Pre-image | Complete failure |
| **RIPEMD160** | |
| Any of the above | Repudiate payments |
| **ECDSA** | |
| Selective forgery | Steal coins |
| Integrity break | Claim payment not received |
| Repudiation | - |

it into a tree of transactions. The problem identified in Section 3.3 was that there is only negligible probability that the transactions refer to valid, unspent outputs, so a forgery does not solve this issue. Finally, for *transactions*, collisions and second pre-images on their own can be used to destroy coins, or steal coins. If the adversary can also forge signatures, he is guaranteed to be able to steal coins no matter what address they went to, as long as it is not protected by the address hash.

**Signature Integrity** A collision or a second pre-image attack trivially breaks the integrity of the scheme as messages are always hashed, and reduces to the case discussed in Section 3.3, so we modify the definitions slightly to consider a joint break in the two algorithms.

A *collision integrity* oracle given a public key $p$ produces $m, m'$ such that the signature of $H_M(m)$ is also valid for $H_M(m')$. The adversary can ask for a signature on an innocent transaction, but transmit the malicious one with the still valid signature. Unlike in the regular collision case, the two hashes $H_M(m)$ and $H_M(m')$ are different. Hence, the adversary cannot just replace the transaction in the block, but he can opt never to transmit the innocent one instead.

A *second pre-image integrity* oracle given a public key $p$ and a message $m$ produces $m'$ such that the signature of $H_M(m)$ is also valid for $H_M(m')$. This case also resembles the break on just $H_M$, but, again, because the hashes are not equal, the adversary cannot simply replace an existing transaction, unless it has not yet been confirmed in a block. This can split the network and destroy coins.

**Signature Repudiation** The non-repudiation property of the signature scheme necessarily involves a break of $H_A$, as was explained in Section 4.3. This combination therefore does not increase the adversary's power.

## 6 Current Bitcoin Implementation

In this section, we revisit the current Bitcoin implementation, in the context of its choice of primitives, non-standard scripts, and contingency plans, using observations from the previous sections.

### 6.1 Current Cryptographic Primitives

In the current implementation of Bitcoin, $H_A(x) = RIPEMD160(SHA256(x))$ , and $H_M(x) = SHA256(SHA256(x))$. Because there are no critical breaks for $H_A$, a break in RIPEMD160 is not cause for concern. Moreover, because $H_M$ only uses SHA256, an attack against SHA256 is equivalent to an attack against $H_M$. We can thus summarize the effect of concrete primitive breakage in Table 4.

### 6.2 Non-Standard Scripts

As explained in Section 2.1, the Bitcoin scripting language extends beyond the 5 standard types of transactions. In part due to the higher fees associated with them, non-standard transactions represent less than 0.02% of all Bitcoins in circulation,[3] but are more versatile and can include opcodes for calculating SHA1, SHA256, and RIPEMD160 hashes. By using these primitives, one can create "challenges" involving the primitives, so that, for instance, a user needs to create a collision to redeem a certain coin.

One set of such challenges was created by one of the early Bitcoin developers asking for collisions on individual and combined primitives.[4] The challenge for a SHA1 collision was claimed using the collision found by Stevens et al. [6], for a bounty of 2.48BTC, the equivalent of approximately $2,800 at the time. Although the USD/BTC exchange rate fell temporarily as a result of these news, it quickly recovered, since SHA1 is not an integral part of the Bitcoin protocol. However, this collision highlights the need to anticipate the breakage of primitives, since non-standard transactions allow collision and pre-image attacks to be used even when the core protocol is not yet broken. However, as we explain in Sections 6.3 and 6.4 the existing contingency plans are not sufficient.

### 6.3 Existing Contingency Plans

A break of the primitives has interested the community from the early days of Bitcoin. Informal recommendations by Satoshi in forums evolved into a "wiki" page which describes contingency plans for "catastrophic failure[s]" [1]. Such a failure for primitives is defined in terms of an adversary that can defeat the algorithm with "a few days of work" [1], and the focus is on notifying users, and protecting the `OP_CHECKSIG` operation to prevent people from stealing coins.

Concretely, for a "severe, 0-day failure of SHA-256" [1], the plans propose switching to a new hashing algorithm $H'$, and hard-coding known public keys with unspent outputs as well as the Merkle root of the blockchain under $H'$. For a broken signature scheme, if the attacker cannot recover the private key, and there is a drop-in replacement using the same key-pair, the plan is to simply switch over to the new algorithm. Otherwise, the new version of Bitcoin "should automatically send old transactions somewhere else using the new algorithm" [1].

[3] https://p2sh.info/dashboard/db/non-standard-outputs-statistics, accessed 2017-04-11.

[4] See https://bitcointalk.org/index.php?topic=293382.0.

### 6.4 Potential Migration Pitfalls

The contingency plans suggest that "code for all of this should be prepared" [1], but no such mechanism currently exists. Moreover, no plans are in place for a break in RIPEMD160. Since sudden breaks are unlikely, neither is cause for immediate concern, but should be included in future plans.

**Broken SHA256** By our analysis, it is clear that new transactions should not use a broken hash. However, existing historical transactions and blocks cannot be altered, except in a majority mining attack. Thus, hard-coding public keys, and rehashing the entire blockchain are more prudent than necessary. It should be noted that a sudden break necessitates a hardfork for Bitcoin.

**Broken ECDSA** For a broken ECDSA, a transition is indeed easy if there is a drop-in replacement and the private key is safe. Otherwise, a gradual transition scheme is necessary as users will need to manually switch over to a new key pair.

### 6.5 Recommendations

In this section, we make recommendations to more properly anticipate primitive breakage. Recognizing that there are financial considerations in addition to the technical ones, we do not propose a full upgrade mechanism, but merely make suggestions to the Bitcoin developers and community.

First of all, our analysis reinforces the idea that users should not reuse addresses, not just for privacy reasons, but also because they protect against some types of primitive breakage. For instance, if the signature scheme is broken, addresses are still protected by the hash.

The plans for a sudden breakage should address when to freeze the blockchain, and whether to roll back transactions in the case of a sudden break. Moreover, the centralized approach of hard-coding well-known keys is perhaps not entirely in line with Bitcoin's decentralized philosophy and can lead to lost coins. If keys are to be hard-coded, there is a trade-off between complexity and risking making coins unspendable: developers must decide whether the migration would occur at once, or whether new key pairs should be distributed periodically. An alternative and perhaps better approach would be to use Zero-Knowledge Proofs to tie the old address still protected by their hash to the new public key.

Given that sudden breaks are unlikely, there is a need for a separate plan for weakened primitives. Based on our analysis, we recommend the following:

- Introduce a minimum number of transactions per block to increase the difficulty of performing the preimage attack against the mining header target (Proof-of-Work or PoW) using the coinbase transaction.
- To migrate from old addresses, whether due to a weakened hash or signature scheme, introduce new address types using stronger hashing and signature schemes. This can be achieved with a softfork by making transactions appear to old clients as "pay-to-anybody", akin to how P2SH was introduced.
- Instead of using nested hashes for $H_A$, $H_M$, combine primitives in a way that increases defense-in-depth (see related work in Section 8).
- Given that $H_M$ has multiple use-cases, consider whether each of its functions should have a different instantiation, whether through distinct primitives, by pre-pending different values, or by using an HMAC with different keys.
- Consider a hardfork in response to a weakened $H_M$, with re-designed headers and transactions, and without any use of the old primitives.

A softfork is insufficient for properly upgrading a weakened hash function $H_M = H_1$ to the stronger $H_2$, because $H_M$ forms the core of the PoW scheme. Specifically, since any changes must be backwards compatible, the old validation rules must still apply, so for every new block, $H_1(hdr) < T$, where the target $T$ is still calculated by the same algorithm. New blocks would also need to satisfy some additional constraint $H_2(hdr') < T'$, where the target $T'$ is calculated independently and $hdr'$ is the block header, possibly excluding some fields. As a result, new clients would have to solve two PoW computational puzzles. Though every instance of $H_1$ (transaction, Merkle root, etc.) could be accompanied by an instance of $H_2$, blocks and transactions are fundamentally identified by their $H_1$ hash, which an attacker could exploit. There are also questions of incentives, and whether new iterations of Bitcoin would still use a PoW scheme, but this is left as future work.

## 7 OTHER CRYPTOCURRENCIES

Bitcoin is the largest cryptocurrency by market capitalization and adoption, but both derivatives ("altcoins") and completely different designs have spread. Although discussing them all is out of scope, we identify a few common threads amongst those which use Proof-of-Work (PoW) schemes.

The first class of altcoins consists of those which are forks of Bitcoin with additional types of transactions supported. For these currencies, our analysis applies verbatim, but needs to be extended to these new types of operations. As an example, Namecoin introduces transactions for registering and updating `.bit` domains. The `name_new d/<name>` command creates a transaction whose outputs include $H_A(r \,||\, \mathrm{name})$, where $r$ is a nonce. This acts as a pre-order for the domain `name.bit`, which is then updated by its owner after the transaction has been in 12 confirmed blocks. This pre-order commitment phase is necessary to ensure no-one can steal the name of the address, which is fully registered in a subsequent `name_firstupdate` command which reveals the name, the nonce and the DNS configuration values, signing the transaction with the user's key.

Any updates for a domain require signatures, so any novel exploit would involve the `name_new` and `name_firstupdate` commands. Unlike Bitcoin, however, a pre-image attack on $H_A$ allows recovering *a* random nonce and domain name. Namecoin rules do not seem to preclude $name\_new$ transactions with the same hash, but if the preimage recovers the original $(r, name)$ pair, an attacker can pre-register the domain, by publishing his own `name_new` transaction before the original user's transaction is confirmed. In practice, however, this attack would require pre-image oracles for both hash functions used in $H_A$ (SHA256, RIPEMD160), implying that there is a pre-image oracle for $H_M$ as well, making the entire currency insecure.

The second set of cryptocurrencies consists of those which separate the roles of $H_M$ and replace it by two different hash functions, say $H_I$ for integrity checking and $H_P$ for the PoW. If $H_I$ breaks, one can still steal coins, by creating collisions in transactions, which are hashed before they are signed, as also explained in Section 3.3.3. However, since $H_I$ is not used for PoW, an attack on $H_I$ cannot directly be used to exploit mining. More importantly, as explained in Section 3.3, even a full pre-image oracle on $H_P$ is not sufficient for a breakdown of the currency. This is because the probability of finding a valid pre-image for a fixed target is negligible, as the adversary does not control sufficiently many header bits. Instead, as also identified in Section 3.3, an adversary needs to be able to alter the Merkle root at will, which is protected by the different $H_I$, requiring both to be exploited for a successful attack. Example currencies employing this scheme include Litecoin and Dogecoin, where $H_I$ = SHA256(SHA256), and $H_P$ is the `scrypt` key derivation function. Ethereum is similar, with $H_I$ using Keccak-256 and $H_P$ using Ethash, though it should be noted that due to the different structure of the blockchain, our results are not directly applicable, even though our methodology is.

Primecoin has a different PoW mechanism. It is a fork of Bitcoin using the same structures (and only the single $H_M$), but its PoW requires finding long chains of prime numbers $p_i$ which satisfy $p_{i+1} = 2p_i \pm 1$ , [5] with $p_0 = k \cdot H_M(\text{header}) \pm 1$ for some $k$. An adversary with access to a pre-image oracle can attack this scheme by re-using existing chains: if $(p_i)$ is a valid chain for header $h$ and the difficulty has not changed, then $(p_i)$ is also a valid chain for $h/q$ for any factor $q$ of $h$, and for $h \cdot r$, where $r$ is a small factor of $k$ (to ensure that $h \cdot r$ remains a valid hash). Note that once more, this exploit depends on using the pre-image oracle twice: once for the overall header, and once for the coinbase transaction, showing that it is necessary to increase the minimum number of transactions per block.

Finally, there are currencies which use a hybrid PoW and Proof-of-Stake or Proof-of-Burn scheme. For instance, Peercoin is a fork of Bitcoin that uses the same PoW and structure as Bitcoin, but also allows "minting" coins based on their age for a Proof-of-Stake approach. As a result, the attacks identified in this paper still apply, but the attack surface increases to include the new transactions, and needs to be analyzed separately, as we did with Namecoin above.

## 8 Related Work

Identifying how hash collisions break the security of protocols such as TLS, IPSec, and SSH was recently investigated in [7]. More recently, researchers identified vulnerabilities with the cryptographic function used in the IOTA cryptocurrency, which allowed them to create syntactically valid transactions with the same hash, but which pay out different amounts [8]. In terms of the primitives used in Bitcoin, attacks against RIPEMD160 pre-images [9] and collisions [10] as well as SHA256 collisions [11] and pre-images [12] only work for a reduced number of rounds, and incrementally improve upon brute-force solutions. Certain ECDSA parameters can lead to Duplicate Signature Key Selection, where an adversary can create a different key $P'$ that validates against a correct signature under a key $P$ [4]. Such research indicates that Bitcoin primitives are indeed under attack, highlighting the need for anticipating their breakage.

More generally, for combining hashes effectively, [13] shows that simultaneous collisions for multiple hash functions are not much harder to find than individual ones. [14] shows that even when the underlying compression functions behave randomly but collisions are easy to generate, finding collisions in the concatenated hash $h_1(x)||h_2(x)$ and the XOR hash $h_1(x)\oplus h_2(x)$ requires $2^{n/2}$ queries. However, when the hash functions use the Merkle-Damgård (MD) construction, there is a generic pre-image attack against the XOR hash with complexity $\tilde{O}(2^{\frac{5n}{6}})$ [15]. MD hash functions also behave poorly against pre-image attacks, allowing one to find second pre-images of length $2^{60}$ for RIPEMD160 in $2^{106} \ll 2^{160}$ time [16]. If an adversary can further find many collisions on an MD construction, he can also find pre-images that start with a given prefix (Chosen Target Forced Prefix) [17].

## 9 Conclusions

We presented the first systematic analysis of the effect of broken primitives on Bitcoin. Our analysis reveals that some breakages cause serious problems, whereas others are inconsequential. The main vectors of attack involve collisions on the double SHA256 hash or attacking the signature scheme, which directly enable coin stealing. In contrast, a break of the hash used in addresses has minimal impact, since they do not meaningfully protect the privacy of a user. Our analysis has also uncovered more subtle attacks. For example, the existence of another public key with the same hash as an address in the blockchain enables parties to claim that they did not make a payment. Such attacks show that an attack on a cryptographic primitive can have social rather than technical implications. We leave the economic impact of such attacks as future work. Because our analysis abstracts away from the concrete primitives, our general results extend to future versions that use a similar structure, as well as altcoins and other blockchain-based schemes.

We uncovered a lack of defense-in-depth in Bitcoin. In most cases, the failure of a single property in one cryptographic primitive is as bad as multiple failures in several primitives at once. For future versions of Bitcoin, we recommend including various redundancies such as properly combined hash functions, and requiring a minimum number of transactions per block. Bitcoin's migration plans are currently under-specified, and offer at best an incomplete solution if primitives get broken. We offer some initial guidelines for making the cryptocurrency more robust, both for a sudden break, but also in response to weakened primitives. However, future discussions should directly involve the Bitcoin developers and community to propose plans that would be in line with their expectations.

## References

[1] Bitcoin Wiki, "Contingency plans," https://en.bitcoin.it/wiki/Contingency plans, May 15, 2015, accessed: 2016-02-11.

[5] Technically, Cunningham chains of first or second kind, or bi-twin chains.

[2] S. Nakamoto, “Bitcoin: A peer-to-peer electronic cash system, http://bitcoin.org/bitcoin.pdf,” 2008.

[3] C. Decker and R. Wattenhofer, “Bitcoin transaction malleability and MtGox,” in *European Symposium on Research in Computer Security (ESORICS)*, 2014.

[4] S. Blake-Wilson and A. Menezes, “Unknown key-share attacks on the station-to-station (STS) protocol,” in *International Workshop on Practice and Theory in Public Key Cryptography (PKC)*, 1999.

[5] S. Goldwasser, S. Micali, and R. L. Rivest, “A digital signature scheme secure against adaptive chosen-message attacks,” *SIAM Journal on Computing (SICOMP)*, 1988.

[6] M. Stevens, E. Bursztein, P. Karpman, A. Albertini, and Y. Markov, “The first collision for full SHA-1,” https://shattered.io/static/ shattered.pdf, February 23, 2017, accessed: 2017-04-11.

[7] K. Bhargavan and G. Leurent, “Transcript collision attacks: Breaking authentication in TLS, IKE, and SSH,” in *Annual Network and Distributed System Security Symposium (NDSS)*, 2016.

[8] E. Heilman, N. Narula, T. Dryja, and M. Virza, “IOTA vulnerability report: Cryptanalysis of the curl hash function enabling practical signature forgery attacks on the IOTA cryptocurrency,” https://github.com/mit-dci/tangled-curl/blob/master/ vuln-iota.md, September 7, 2017, accessed: 2017-09-08.

[9] C. Ohtahara, Y. Sasaki, and T. Shimoyama, “Preimage attacks on step-reduced RIPEMD-128 and RIPEMD-160,” in *International Conference on Information Security and Cryptology (Inscrypt)*, 2010.

[10] F. Mendel, T. Peyrin, M. Schlaffer, L. Wang, and S. Wu, “Improved¨ cryptanalysis of reduced RIPEMD-160,” in *International Conference on the Theory and Application of Cryptology and Information Security (ASIACRYPT)*, 2013.

[11] F. Mendel, T. Nad, and M. Schlaffer, “Improving local collisions:¨ New attacks on reduced SHA-256,” in *Annual International Conference on the Theory and Applications of Cryptographic Techniques (EUROCRYPT)*, 2013.

[12] D. Khovratovich, C. Rechberger, and A. Savelieva, “Bicliques for preimages: Attacks on Skein-512 and the SHA-2 family,” in *International Workshop on Fast Software Encryption (FSE)*, 2012.

[13] A. Joux, “Multicollisions in iterated hash functions. application to cascaded constructions,” in *Annual International Cryptology Conference (CRYPTO)*, 2004.

[14] J. J. Hoch and A. Shamir, “On the strength of the concatenated hash combiner when all the hash functions are weak,” in *International Colloquium on Automata, Languages and Programming (ICALP)*, 2008.

[15] G. Leurent and L. Wang, “The sum can be weaker than each part,” in *Annual International Conference on the Theory and Applications of Cryptographic Techniques (EUROCRYPT)*, 2015.

[16] J. Kelsey and B. Schneier, “Second preimages on $n$-bit hash functions for much less than $2^n$ work,” in *Annual International Conference on the Theory and Applications of Cryptographic Techniques (EUROCRYPT)*, 2005.

[17] J. Kelsey and T. Kohno, “Herding hash functions and the nostradamus attack,” in *Annual International Conference on the Theory and Applications of Cryptographic Techniques (EUROCRYPT)*, 2006.

**Keywords**: Bitcoin, alt-coins, crypto-currencies, broken primitives, SHA256, ECDSA, hashing pre-images, hashing collisions

## Author Bios:

**Ilias Giechaskiel** is a DPhil (PhD) student in the Department of Computer Science at the University of Oxford, focusing on hardware security. He holds an MPhil in Advanced Computer Science from the University of Cambridge and a BA in Mathematics from Princeton University. He has also interned at Bloomberg, Microsoft, Dropbox, Microsoft Research, and Jump Trading, often taking on security-oriented projects.

**Cas Cremers** is a full professor in the Department of Computer Science at the University of Oxford. His research focuses on information security, in particular the formal analysis of security protocols. This work ranges from developing mathematical foundations for protocol analysis to the development of analysis tools, notably the Scyther tool and the Tamarin prover. He has also been involved in protocol standardisation, including the improvement of the ISO/IEC 9798 & 11770 and IETF TLS 1.3 standards, and applied cryptography, leading to the development of new security requirements and protocols.

**Kasper Rasmussen** is an Associate Professor in the Computer Science Department at the University of Oxford. He joined the department in 2013 and in 2015 was awarded a University Research Fellowship from the Royal Society in London. Prior to being at Oxford, Kasper Rasmussen spent two years as a post-doc at University of California, Irvine.

Kasper Rasmussen did his Ph.D. with prof. Srdjan Capkun at the Department of Computer Science at ETH Zurich (Switzerland), where he worked on security issues relating to secure time synchronization and secure localization with a particular focus on distance bounding. His thesis won the "ETH Medal" for an outstanding dissertation from the Swiss Federal Institute of Technology and he was additionally awarded the Swiss National Science Foundation (SNSF), Fellowship for prospective researchers.