On Bitcoin Security in the Presence of Broken Cryptographic Primitives

Ilias Giechaskiel, Cas Cremers, Kasper B. Rasmussen

21st European Symposium on Research in Computer Security (ESORICS),
Heraklion, Crete, Greece, Sep. 2016

On Bitcoin Security in the Presence of Broken Cryptographic Primitives

Ilias Giechaskiel, Cas Cremers, and Kasper B. Rasmussen

University of Oxford, United Kingdom
ilias.giechaskiel@cs.ox.ac.uk, cas.cremers@cs.ox.ac.uk, kasper.rasmussen@cs.ox.ac.uk

Abstract. Digital currencies like 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 frequently, 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 identify the core cryptographic building blocks and analyze the ways in which they 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 observations on, and suggestions for, the Bitcoin migration plans 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 [46]. However, such plans are handwavy and incomplete at best: no adequate transition mechanism has been built into Bitcoin, and no plans for partial breakage (or weakening of a primitive) have been considered. Primitives rarely break abruptly, but instead they break gradually. With hash functions, for example, it is common that first a single collision is found. This is then later generalized to multiple collisions, and only later do arbitrary collisions become feasible to compute. In parallel, the complexity of attacks (such as collisions) decreases to less-than-brute-force, and computational power increases. Finally, quantum computing will make some attacks easier, e.g., by Grover's pre-image attack [20], or Shor's algorithm for discrete log computation [40].

Hence, 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.

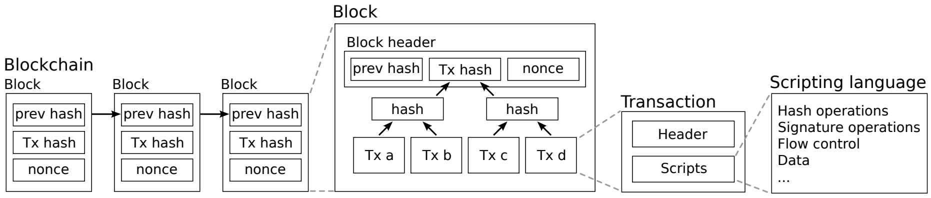

Fig. 1. The blockchain data structure. This forms the basis of the public, append-only ledger where all transactions are recorded.

Contributions We provide the first systematic analysis of the impact of broken primitives on Bitcoin. By analyzing the failure of primitive properties, both in isolation and in combination, we describe precisely the range of consequences different breaks have, and pinpoint their exact cause. For example, the flexibility of the coinbase transaction is the reason why mining becomes trivial if an adversary can easily compute pre-images of SHA256 hashes. In our analysis, we introduce an oracle model for hash functions that unifies and extends several existing types of breakage, allowing us to analyze more realistic attacks. 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 concrete suggestions regarding future iterations of the cryptocurrency in response to entirely broken and partially weakened primitives.

Overview We provide background in Section 2 and propose our adversary model in Section 3. We next analyze the effects of broken primitives: hashing in Section 4, signature schemes in Section 5, and combinations of primitive breaks in Section 6. We revisit the current Bitcoin implementation and its contingency plans in Section 7. We discuss related work in Section 8 and conclude in Section 9.

2 Background

In this section, we give a description of Bitcoin, the popular peer-to-peer (P2P) cryptocurrency introduced in 2008 by Satoshi Nakamoto [34]. Figure 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 that determines the owners of coins (Section 2.1), and it is up to miners to ensure 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). We do not consider components outside the main protocol, such as wallets.

2.1 Transactions and Scripts

Bitcoin is an electronic cash system [34], 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 that get executed by special nodes called miners. To prevent Denial-of-Service (DoS) attacks exploiting computationally intensive instructions, most nodes only accept 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 which 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. Though some non-standard scripts can be accepted by some miners for a higher fee, we do not cover these in our analysis.

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 [33]. 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 do the work, which includes validating transactions and blocks. A summary is shown in Figure 2, with the full procedure at [45].

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. This way, nodes eventually reach consensus [10, 17].

```
input  : Bitcoin block
output : valid or invalid
/* Verify block header                                                  */
Verify $\mathsf{Hash}(\textit{block header}) < \mathsf{target}$
Verify Merkle hash
Verify $\mathsf{Hash}(\textit{prev block}) = \mathsf{prev\_hash}$
/* 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. If a miner or a group of collaborating miners (called a *pool*) is in control of a high enough proportion of the total computational power (51% [29], or even less [16]), then they can possibly destabilize the system.

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 System and Adversary Model

In this section we describe our Bitcoin model and discuss the adversary's goals and powers in the presence of broken cryptographic primitives. We distinguish between 4 entities: senders, receivers, miners, and networking nodes. Senders and receivers, collectively referred to as users, wish to exchange Bitcoins via transactions. They care about the amount of money under their control, but not about the details of the underlying system.

Transactions are transmitted via the underlying P2P network. Miners have their own (possibly different) copy of the blockchain, and have different hashing capacities. For our model, we consider pools as single miners with a large hashing capability. We distinguish between two adversary roles: user and miner. As a user, the adversary aims to make money, either by successfully double spending or by spending from another user's wallet. As a miner, the adversary controls a proportion $\alpha < 0.5$ of the mining power. We assume the adversary controls a proportion β of the nodes in the P2P network, so that he can attempt to split the network temporarily in the presence of a suitable vulnerability, but cannot be confident that such attempt will succeed.

We consider the economic aspects of Bitcoin out of scope, and we also do not consider developers as a threat. Finally, we do not investigate adversarial attacks of an individual miner against his own pool, thus allowing us to consider pools as single entities of more mining power.

4 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. We generalize these into a single property called chosen-format bounded pre-image resistance.

4.1 Hashing in Bitcoin

In the original Bitcoin paper [34], 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, going back to at least version 0.1.0 [35] uses two hash functions.

Main Hash This hash function has an output of 256 bits and requires applying SHA256 twice: $H_M(x) = \mathrm{SHA256}\left(\mathrm{SHA256}\left(x\right)\right)$. 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 (see [39] for details).

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 $H_A(x) = \mathrm{RIPEMD160}\left(\mathrm{SHA256}\left(x\right)\right)$.

4.2 Modeling Hash Breakage

In this section we analyze how hashes break in terms of their building blocks, and introduce our oracle model for their breakage.

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 $h(x_1) = h(x_2)$.

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 ($2n$ calls)
Bounded pre-image	All of the above	Complete failure of the blockchain (n calls)

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

where "hard" refers to computational infeasibility, since 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. To relate the attacks we discover back to the concrete primitives in Section 7, we show in Appendix A that for collisions and second pre-images, only one of the two nested hashes needs to be broken, while for pre-images both need to be broken.

Modeling Hash Breakage Variants The three properties discussed above do not accurately capture all types of breakages, which typically exploit the internal structure of the hash function. Thus, an adversary might have more control over the structure of the pre-image or the target value. For example, mining expects the hash to be smaller than a given target, a property which cannot be expressed using traditional pre-image oracles, as we show in Section 4.3.

For this reason, we introduce a more general oracle model to enable our analysis. We call the oracle a *chosen-format bounded pre-image oracle P*, which on input (a, b, y_l, y_h, i) returns an x_i such that $y_l \leq h(a||x_i||b) \leq y_h$ or $\perp$ if none exists. Thus, the oracle returns a value $X_i = a||x_i||b$ such that its beginning and end are caller-supplied, and its hash is within a given target range. Moreover, the oracle is deterministic such that the same x_i is returned each time and $x_i \neq x_j$ for $i \neq j$ and if given an optional parameter s, the returned x_i has size s bits. That is to say, the oracle can be called multiple times to get different pre-images, and the user is also able to specify the length of the pre-image in bits.

In Appendix B, we motivate these parameters and show that our oracle captures breakages in the three properties. We summarize our results in Table 1.

4.3 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.

4.3.1 Mining

Pre-Image against Fixed Merkle Root We analyze the probability that an adversary with access to a pre-image oracle can break mining. Miners search for block headers whose n-bit hash is below a target, which we assume starts with d zeros. Since the target can be higher by at most a factor of 2, the assumption introduces up to 1 bit of extra work.

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 $0x180928f0$, granting the adversary 28 bits of freedom.

Together the fields give $b = 73$. With $n = 256$, and allowing 2^{80} calls to the oracle, the probability of success is only $2^{80+73-256} = 2^{-103}$, which is negligible.

Pre-Image against Variable Merkle Root An adversary may attempt to reorder or exclude transactions to try different roots, but by the same argument the success probability is negligible. However, the adversary can 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, for instance due to incorrect signatures, 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 [39]. 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.

Bounded Pre-Image An adversary with access to our chosen-format, bounded pre-image oracle P can simply call P on $(hdr, \perp, 0, y_t, 0, s)$, where y_t is the target hash, hdr is the beginning of the block header, and $s = 32$ is the size of the required nonce such that $0 \leq H_M(hdr||nonce) \leq y_t$. This completely breaks mining and requires half as many calls to the oracle compared to the above attack using the simple pre-image oracle.

Collisions, Second Pre-Images Collisions and second pre-images 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.

4.3.2 Merkle Trees

Altering existing blocks A similar argument as for mining 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. This can occur even with invalid pre-images: a similar situation occurred when some miners generated invalid blocks which were not detected by clients [1]. Pre-images are again not useful, as they always accompany the hashed value.

4.3.3 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 to address X), but transmit a malicious one instead (e.g., pay 100 BTC to address Y) since there are enough bytes that the adversary controls to guarantee success with high probability.

Though external to the protocol, signatures of H_M are also used by Bitcoin developers to transmit alerts. A pre-image attack again does not give useful information to the adversary, as the pre-image always accompanies the signature. Collisions are also not useful, as the adversary cannot sign them. However, a second pre-image allows the adversary to reuse an old signature on a new alert.

4.4 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 $y = H_A(p) = \text{RIPEMD160}(\text{SHA256}(p))$ where p is the public key (together with a checksum [4]). 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

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

Table 2. Effects of a broken signature scheme.

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 pre-images (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.

5 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.

5.1 Digital Signatures in Bitcoin

Bitcoin's digital signature scheme is the Elliptic Curve Digital Signature Algorithm (ECDSA) with the `secp256k1` [43] parameters, and is used to sign the main hash H_M of transactions. These signatures can be over different parts of the message based on the *hashtype* [39], leading to transaction malleability attacks [13], as the same transaction can be encoded multiple ways without invalidating the signature. The signature scheme is also used for alerts by developers to announce critical information. The signature is over the main hash H_M of the entire alert structure. The effects on alerts are not summarized in the table as they are external to the protocol.

5.2 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 [9].

5.3 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 6, but note that transaction malleability can cause the issuer of a transaction to think that his payment was not confirmed [13].

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 6. For the alert mechanism, however, if given a message m and a public key p, one can find p' (with its secret key s') such that $\{m\}_{s'}$ validates against p, then an adversary can send fake alert messages. This can have an external impact on Bitcoin, for instance by asking users to manually shut down clients.

Unforgeability When it comes to unforgeability, we can distinguish between various types of breaks [19]: *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.

	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	-	-
Bounded pre-image	All of the above	-	Change existing payment
Main Hash (H_M)			
Collision	Steal coins	Steal coins†	-
Second pre-image	Steal coins	Double spend†	-
Pre-image	-	-	-
Bounded pre-image	Steal coins	All of the above	-

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

Table 3. The effects of a multi-breakage: combining broken hashes and signatures.

6 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. Since the extra power of our oracle is not needed, we discuss breakage in terms of the three traditional properties. The results are summarized in Table 3.

6.1 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 7.

Signature Integrity As the messages signed for alerts or 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.

6.2 Main Hash and Signature Scheme

Signature Forgery As explained in Section 4.3, 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, alerts with collisions, 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 it into a tree of transactions. The problem identified in Section 4.3 was that there is only negligible probability that the transactions refer to valid, unspent outputs, so a forgery does not solve this issue. For *alerts*, collisions require forgery. Though the effect of signing and transmitting two different alert messages with the same hash is unclear, it could potentially be used to cause external effects to Bitcoin by making the different messages ask the users to take different actions. 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 4.3, 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 5.3. This combination therefore does not increase the adversary's power.

7 Current Bitcoin Implementation

In this section, we revisit the current Bitcoin implementation, its choice of primitives and contingency plans, using observations from the previous sections.

7.1 Current Cryptographic Primitives

In the current version of Bitcoin, $H_A(x) = \text{RIPEMD160}(\text{SHA256}(x))$, and $H_M(x) = \text{SHA256}(\text{SHA256}(x))$. Because there are no critical breaks for H_A, a

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

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

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.

7.2 Existing Contingency Plans

A break of the primitives has interested the community from the early days of Bitcoin. Informal recommendations by Satoshi in forums [36, 37] evolved into a "wiki" page which describes contingency plans for "catastrophic failure[s]" [46]. Such a failure for primitives is defined in terms of an adversary that can defeat the algorithm with "a few days of work" [46], and the focus is on notifying users (since alerts may be compromised), and protecting the `OP_CHECKSIG` operation to prevent people from stealing coins.

Concretely, for a "severe, 0-day failure of SHA-256" [46], 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" [46].

7.3 Potential Migration Pitfalls

The contingency plans suggest that "code for all of this should be prepared" [46], 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 migration 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.

7.4 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 periodic alert-like messages would be used to distribute new key pairs 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 pre-image 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 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.
- Since alerts are external to the Bitcoin mechanism itself, send alerts using a new signature and hash scheme to new clients, and duplicate the message using old primitives for old clients.
- 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, fundamentally blocks and transactions are 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.

8 Related Work

Since no other systematic analysis exists regarding primitive breakage for Bitcoin, we consider papers which have focused on Bitcoin security in general, and also explore related work focusing on the security of the primitives themselves.

Bitcoin Multiple papers have identified or formalized properties such as stability and anonymity in Bitcoin and other cryptocurrencies [10, 17, 44]. Anonymity and privacy issues have also been explored extensively [3, 8, 41, 42].

Research on adversarial miners has shown that there are infinitely many Nash equilibria for mining strategies [29], and some strategies allow miners controlling $\alpha < 50\%$ of the power to gain disproportionate rewards [12,15,16]. Other research has demonstrated that double spending attacks are practical against Bitcoin fast payment scenarios [24, 25], with some further focus on causing a network split [18] or isolating victims from other peers in the P2P nework [21].

[5] focuses on the economics of Bitcoin, including the effect of a history revision, which is discussed in the contingency plans [46]. [13] investigated transaction malleability attacks which were prevalent in 2014.

Cryptographic Primitives For combining hashes, [23] shows simultaneous collisions for multiple hash functions are not much harder to find than individual ones. [22] 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 $\tilde{O}\left(2^{5n/6}\right)$ [30].

Neither MD hashes [11] nor $h\left(h\left(x\right)\right)$ [14] behave as random oracles. 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 [27]. 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) [26]. This notion can be extended to Chosen Target Forced Midfix attacks and it was proven that at least $2^{2n/3}/L^{1/3}$ queries to the compression function are needed where L is the maximum length of the pre-image [2] .

Attacks against RIPEMD160 pre-images [38] and collisions [32] as well as SHA256 collisions [31] and pre-images [28] only work for a reduced number of rounds, and typically only 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 [9]. Implementations of ECDSA can also be vulnerable to side-channel attacks [47], an attack which has also been practically demonstrated against Bitcoin [6]. Finally, [7] showed how hash collisions break the security of protocols like TLS, IPSec, and SSH.

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.

We uncovered a worrying 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 Alert: Some miners generating invalid blocks. `https://bitcoin.org/en/alert/2015-07-04-spv-mining` (July 4 2015), accessed: 2016-02-11
2. Andreeva, E., Mennink, B.: Provable chosen-target-forced-midfix preimage resistance. In: International Conference on Selected Areas in Cryptography (SAC) (2011)
3. Androulaki, E., Karame, G.O., Roeschlin, M., Scherer, T., Čapkun, S.: Evaluating user privacy in Bitcoin. In: Financial Cryptography and Data Security (FC) (2013)
4. Antonopoulos, A.M.: Mastering Bitcoin: Unlocking Digital Crypto-Currencies. O'Reilly Media, Inc., 1st edn. (2014)
5. Barber, S., Boyen, X., Shi, E., Uzun, E.: Bitter to better — how to make Bitcoin a better currency. In: Financial Cryptography and Data Security (FC) (2012)

6. Benger, N., Pol, J., Smart, N.P., Yarom, Y.: “ooh aah... just a little bit” : A small amount of side channel can go a long way. In: Cryptographic Hardware and Embedded Systems (CHES) (2014)
7. Bhargavan, K., Leurent, G.: Transcript collision attacks: Breaking authentication in TLS, IKE, and SSH. In: Annual Network and Distributed System Security Symposium (NDSS) (2016)
8. Biryukov, A., Khovratovich, D., Pustogarov, I.: Deanonymisation of clients in Bitcoin P2P network. In: ACM Conference on Computer and Communications Security (CCS) (2014)
9. Blake-Wilson, S., Menezes, A.: Unknown key-share attacks on the station-to-station (STS) protocol. In: International Workshop on Practice and Theory in Public Key Cryptography (PKC) (1999)
10. Bonneau, J., Miller, A., Clark, J., Narayanan, A., Kroll, J., Felten, E.: SoK: Research perspectives and challenges for Bitcoin and cryptocurrencies. In: IEEE Symposium on Security and Privacy (SP) (2015)
11. Coron, J.S., Dodis, Y., Malinaud, C., Puniya, P.: Merkle-Damgård revisited: How to construct a hash function. In: Annual International Cryptology Conference (CRYPTO) (2005)
12. Courtois, N.T., Bahack, L.: On subversive miner strategies and block withholding attack in Bitcoin digital currency. ArXiv e-prints 1402.1718 (2014), `http://arxiv.org/abs/1402.1718`
13. Decker, C., Wattenhofer, R.: Bitcoin transaction malleability and MtGox. In: European Symposium on Research in Computer Security (ESORICS) (2014)
14. Dodis, Y., Ristenpart, T., Steinberger, J., Tessaro, S.: To hash or not to hash again?(in) differentiability results for H^2 and HMAC. In: Annual International Cryptology Conference (CRYPTO) (2012)
15. Eyal, I.: The miner’s dilemma. In: IEEE Symposium on Security and Privacy (SP) (2015)
16. Eyal, I., Sirer, E.G.: Majority is not enough: Bitcoin mining is vulnerable. In: Financial Cryptography and Data Security (FC) (2014)
17. Garay, J., Kiayias, A., Leonardos, N.: The Bitcoin backbone protocol: Analysis and applications. In: Annual International Conference on the Theory and Applications of Cryptographic Techniques (EUROCRYPT) (2015)
18. Gervais, A., Ritzdorf, H., Karame, G.O., Capkun, S.: Tampering with the delivery of blocks and transactions in Bitcoin. In: ACM Conference on Computer and Communications Security (CCS) (2015)
19. Goldwasser, S., Micali, S., Rivest, R.L.: A digital signature scheme secure against adaptive chosen-message attacks. SIAM Journal on Computing (SICOMP) (1988)
20. Grover, L.K.: A fast quantum mechanical algorithm for database search. In: Annual ACM Symposium on Theory of Computing (STOC) (1996)
21. Heilman, E., Kendler, A., Zohar, A., Goldberg, S.: Eclipse attacks on Bitcoin’s peer-to-peer network. In: USENIX Security Symposium (USENIX Security) (2015)
22. Hoch, J.J., Shamir, A.: 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)
23. Joux, A.: Multicollisions in iterated hash functions. application to cascaded constructions. In: Annual International Cryptology Conference (CRYPTO) (2004)
24. Karame, G.O., Androulaki, E., Roeschlin, M., Gervais, A., Čapkun, S.: Misbehavior in bitcoin: A study of double-spending and accountability. In: ACM Transactions on Information and System Security (TISSEC) (2015)

25. Karame, G.O., Androulaki, E., Čapkun, S.: Double-spending fast payments in Bitcoin. In: ACM Conference on Computer and Communications Security (CCS) (2012)
26. Kelsey, J., Kohno, T.: Herding hash functions and the nostradamus attack. In: Annual International Conference on the Theory and Applications of Cryptographic Techniques (EUROCRYPT) (2006)
27. Kelsey, J., Schneier, B.: 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)
28. Khovratovich, D., Rechberger, C., Savelieva, A.: Bicliques for preimages: Attacks on Skein-512 and the SHA-2 family. In: International Workshop on Fast Software Encryption (FSE) (2012)
29. Kroll, J.A., Davey, I.C., Felten, E.W.: The economics of Bitcoin mining, or Bitcoin in the presence of adversaries. In: Workshop on the Economics of Information Security (WEIS) (2013)
30. Leurent, G., Wang, L.: The sum can be weaker than each part. In: Annual International Conference on the Theory and Applications of Cryptographic Techniques (EUROCRYPT) (2015)
31. Mendel, F., Nad, T., Schläffer, M.: Improving local collisions: New attacks on reduced SHA-256. In: Annual International Conference on the Theory and Applications of Cryptographic Techniques (EUROCRYPT) (2013)
32. Mendel, F., Peyrin, T., Schläffer, M., Wang, L., Wu, S.: Improved cryptanalysis of reduced RIPEMD-160. In: International Conference on the Theory and Application of Cryptology and Information Security (ASIACRYPT) (2013)
33. Merkle, R.C.: A digital signature based on a conventional encryption function. In: Annual International Cryptology Conference (CRYPTO) (1987)
34. Nakamoto, S.: Bitcoin: A peer-to-peer electronic cash system, `http://bitcoin.org/bitcoin.pdf` (2008)
35. Nakamoto, S.: Bitcoin source code v0.1.0: Util.h. `https://github.com/trottier/original-bitcoin/blob/4184ab26345d19e87045ce7d9291e60e7d36e096/src/util.h` (2009), accessed: 2016-02-11
36. Nakamoto, S.: Dealing with SHA-256 collisions (msg #6). `https://bitcointalk.org/index.php?topic=191.msg1585#msg1585` (June 14 2010), accessed: 2016-02-11
37. Nakamoto, S.: Hash() function not secure (msg #28). `https://bitcointalk.org/index.php?topic=360.msg3520#msg3520` (July 16 2010), accessed: 2016-02-11
38. Ohtahara, C., Sasaki, Y., Shimoyama, T.: Preimage attacks on step-reduced RIPEMD-128 and RIPEMD-160. In: International Conference on Information Security and Cryptology (Inscrypt) (2010)
39. Okupski, K.: Bitcoin developer reference working paper. `http://enetium.com/resources/Bitcoin.pdf` (2015), accessed: 2016-02-11
40. Proos, J., Zalka, C.: Shor's discrete logarithm quantum algorithm for elliptic curves. Quantum Information & Computation (2003)
41. Reid, F., Harrigan, M.: An analysis of anonymity in the Bitcoin system. In: Security and Privacy in Social Networks (2013)
42. Ron, D., Shamir, A.: Quantitative analysis of the full Bitcoin transaction graph. In: Financial Cryptography and Data Security (FC) (2013)
43. Standards for Efficient Cryptography: Sec 2: Recommended elliptic curve domain parameters version 2.0. `http://www.secg.org/sec2-v2.pdf` (2010)
44. Tschorsch, F., 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`

45. Bitcoin Wiki: Protocol rules. https://en.bitcoin.it/wiki/Protocol_rules (March 11, 2014), accessed: 2016-02-11
46. Bitcoin Wiki: Contingency plans. https://en.bitcoin.it/wiki/Contingency_plans (May 15, 2015), accessed: 2016-02-11
47. Yarom, Y., 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

A Breaking Nested Functions

In this section, we investigate the three main hashing properties, for a function $h = h_1 \circ h_2$ which is a composition of two hash functions. We show that for collisions and second pre-images, only one of the two nested hashes needs to be broken, while for pre-images both need to be broken.

Pre-image resistance h is broken only when both h_1, and h_2 are broken. In one direction, assume that we have a pre-image algorithm for h, that returns x on input y. Then, to find a pre-image for y under h_2, run the algorithm on $h_1(y)$ for output x. If $h_2(x) = y$, then x is a pre-image for y under h_2. Else $h_2(x) \neq y$ and $(h_2(x), y)$ forms a collision (or second pre-image) for h_1. Conversely, if there is an algorithm for both h_1 and h_2 pre-images, then to get a pre-image of y under h, one finds a pre-image x_1 of y under h_1, and then a pre-image x_2 of x_1 under h_2. x_2 is then a pre-image of y under h.

Second pre-image resistance h is only as strong as the inner function h_2. In one direction, assume that given x_1 one can find $x_2 \neq x_1$ such that $h_2(x_1) = h_2(x_2)$. Then clearly $h(x_1) = h(x_2)$.[1] In the other direction, assuming that given x_1, one can find $x_2 \neq x_1$ such that $h(x_1) = h(x_2)$, then either $h_2(x_1) = h_2(x_2)$ for a second pre-image attack on h_2 or $h_2(x_1) \neq h_2(x_2)$ for a collision (and second pre-image of $h_2(x_1)$) on h_1.

Collision resistance h is again only as strong as h_2. A collision (x_1, x_2) for h_2 is clearly a collision for h, and a collision (x_1, x_2) for h is either a collision for h_2 or $(h_2(x_1), h_2(x_2))$ is a collision for h_1.

B Generalizing Hash Oracles

In this section, we first motivate the parameters in our oracle model and then show that our oracle generalizes traditional primitive breakage. We remind the reader that our oracle P on input (a, b, y_l, y_h, i) returns an x_i such that $y_l \leq h(a||x_i||b) \leq y_h$ or $\perp$ if none exists, with $x_i \neq x_j$ when $i \neq j$.

First of all, specifying a, b, and the length of the input forces pre-images and collisions to follow the format of transactions and block headers. Using bounds on the target range is necessary to describe some attacks against the proof-of-work (PoW) scheme. In addition, the oracle needs an index parameter to ensure that the adversary is polynomially bounded: when there is no length restriction on the pre-image, there are potentially infinitely many pre-images, and exponentially many for a fixed-length input. Finally, $x_i \neq x_j$ for $i \neq j$ so that the adversary can

[1] The same can be said if h_1 is vulnerable to second pre-image attacks and h_2 is vulnerable to first pre-image attacks.

access as many distinct pre-images as desired. These returned values are distinct, without gaps, i.e., if the oracle returns $\perp$ on i it should also return $\perp$ on $i+1$, so that the adversary can stop querying the oracle after receiving a $\perp$. We now show how an adversary with access to P can break the three hash properties.

Pre-image Getting a pre-image of y amounts to calling P on $(\perp, \perp, y, y, 0)$, so the adversary can break pre-image resistance with a single call to the oracle.

Second pre-image Getting a second pre-image given x is almost identical, but potentially requires two oracle calls: call P on $(\perp, \perp, h(x), h(x), 0)$, and if that returns x, call P on $(\perp, \perp, h(x), h(x), 1)$.

Collision Getting a collision is not as straightforward. Let $h : \{0,1\}^* \to \{0,1\}^n$ be the hash function in question. First of all, it is not always the case that every $y \in \{0,1\}^n$ has a pre-image (let alone two), even though probabilistically this holds true for a well-designed hash function. For instance, consider h', where $h'(x) = 1$ when $h(x) = 0$, and $h'(x) = h(x)$ otherwise. Then, h' is strong if h is strong, but does not hit 0. However, by exploiting the pigeonhole principle and binary search, one can make $\lg(n)$ calls to the oracle to generate a collision.

The idea is to call P on $(\perp, \perp, y_l, y_h, y_h - y_l + 2)$. If the oracle returns anything but $\perp$, there are more pre-images than possible hashes within the range $[y_l, y_h]$. Then, one can perform a binary search with initial $y_l = 0^n$, $y_h = 1^n$ to determine a value y that has at least 2 pre-images.

Chosen-prefix collision To get a chosen-prefix collision, i.e. given p find two values $x \neq x'$ such that $h(p||x) = h(p||x')$, one performs the same procedure as for getting a normal collision, but with $a = p$.