Master Practicum Report

Evaluating the effect on counter-measures preventing the inclusion of off-topic data into Bitcoin's blockchain

Anton Ehrmanntraut
Department of Computer Science
Chair of Computer Science II (Secure Software Systems)

Prof. Dr. Alexandra Dmitrienko
First Reviewer

Abstract

Blockchain-based cryptocurrencies use the blockchain data structure as persistent, distributed, append-only, and timestamped ledger. These properties make blockchains attractive to use as storage for data that is not related to the payment processing itself. In the case of Bitcoin's blockchain, recent studies show that this inclusion of off-topic data has already been executed in practice. In fact, arguably objectionable content is already stored on Bitcoin's blockchain; this potentially illegal content has the possibility to put participants of the network at legal risk, imposing a threat to the entire network.

To overcome this risk, possible countermeasures were discussed. This paper analyzes a countermeasure by Seeg, which exploits the one-way property of the address generation. This paper analyzes this approach and identifies this type of countermeasure as maximally strong with respect to principles that can be considered compatible with the philosophy of Bitcoin, motivating an evaluation of the strength of this countermeasure. By performing a brute-force exponential-time inversion on the address generation, an implementation of a circumvention is obtained.

The economic cost for executing this brute-force circumvention can be considered as a reasonable upper bound on the effectiveness of the countermeasure. Hence, using publicly available benchmark figures, we estimate the cost of the circumvention, to assess the effectiveness of any approach attempting to prevent off-topic inclusions while adhering to the (social) principles of Bitcoin.

Contents

1 Introduction

Bitcoin, the decentralized, distributed, pseudonymous P2P electronic cash system introduced in 2009, makes use of blockchains, distributed data structures, as permanent tamper-proof and append-only log of transactions, enforcing irreversibility of these transactions. By design, the blockchain is distributed inside the P2P network to all (full) nodes of the Bitcoin, and specifically every network node maintains a local copy of the entire blockchain. While many Bitcoin users can rely on "simplified payment verification" and/or blockchain indexing servers to avoid downloading a full copy of the blockchain, full nodes – maintaining a complete up-to-date copy of the blockchain – are required in the network to to independently verify the transaction ledger [1].

The beneficial properties of the blockchain storage – distributed, tamper-proof, time-stamped, among others – have incentivized actors to use Bitcoin's blockchain as permanent storage of data unrelated to Bitcoin's payment processing. Antonopoulos for example observes that transactions record a digital fingerprint to establish a proof-of-existence for specific files on a specific date.

In fact, the use of Bitcoin's blockchain as storage for data unrelated to payment processing is controversial, he claims. While some see this use as abusive and worry about bloat of blockchain and payment processing, others see such (unintended) use of Bitcoin's technology as legitimate and valuable [2, p. 155]. Most such inclusions of off-topic data observed in practice are negligible in size and content, but nevertheless more sophisticated methods have allowed actors to include entire files such as images, or PDF files [3].

Besides the issues already mentioned by Antonopoulos, the Bitcoin community observed potential legal issues in connection with this data inclusions [4]. Illegal content in the blockchain makes participants of the network legally liable, who need to maintain a copy of the blockchain for operation.

Hence, the insertion of objectionable content has the potential to jeopardize Bitcoin (and generalizing, all blockchain-based cryptocurrencies), rendering participation illegal in certain jurisdictions. Seeg [5, pp. 1–2] also observes that these deliberate inclusions have the possibility to manipulate the bitcoin price, i.e. expecting falling prices caused by endangered legality, and hence give an attacker serious financial incentive for executing such an attack, inserting malicious data, and profiting off, e.g., futures contracts.

Several approaches were discussed to prevent this type of off-topic data inclusion. These range from monitoring incoming data [6, Sec. IV.A], consensus-based deletions of harmful portions of the blockchain [7], [8], or maintaining balances instead of transactions [9].

Approach and Contribution

In general, we can expect that increasingly more effective countermeasures require increasingly radical methods, which in turn, make adoption of that countermeasure increasingly controversial and unlikely. Hence from a theoretical perspective, when given a consensus on which methods are considered so controversial that they are rejected, we can determine a maximal effectiveness of any countermeasure that is considered acceptable in the sense of the given consensus.

Due to the multitude of different interest groups and actors that constitute Bitcoin's community, a precise consensus of the approved methods available when designing an effective countermeasure seems unlikely. Nevertheless, we can identify a minimal set of general principles that can be considered defining for Bitcoin, that is "even if all Bitcoin

users decide to adopt any of these changes, the resulting cryptocurrency can no longer be considered 'Bitcoin' because it has diverged too much from the original design" [10].

In light of these principles, we make plausible that a proof-of-ownership countermeasure, a generalization of proposed countermeasures by Seeg [5] and Matzutt et al. [6], resp., are maximally strong. These build upon the cryptographic security of Bitcoin's key generation scheme, and requires actors to prove that they hold the private key of their respective addresses. A brute-force exponential-time inversion on the involved cryptographic primitives leads to a trivial circumvention of that proof-of-ownership countermeasure. Under reasonable assumptions on the one-way property of these primitives, the computational resources necessary for that specific brute-force circumvention serve as a sensible upper bound for the strength of the countermeasure.

Having established this circumvention, we focus on the main part of the paper. As first result, a stochastic model of expected inclusion cost, in combination with a sensible estimate of the currently available computing efficiency on the cryptographic primitives, allows us to assess the effectiveness of the proposed countermeasure, and in turn by maximality, any countermeasure adhering the aforementioned principles. Furthermore, as secondary result, we assess the impact on variations in bitcoin price and available computing efficiency on the countermeasure's effectiveness.

Outline

Following this introduction, Section 2 gives an overview of the transaction processing in the Bitcoin network, and then demonstrates how the inclusion of off-topic data into the blockchain is currently implemented.

Section 3 clarifies the above-mentioned principles, which makes it immediately clear that the countermeasure proposed by Seeg cannot be further strengthened without violating the principles. A slight generalization of his proposed countermeasure is presented, followed by an implementation of a brute-force circumvention.

On the basis of this implementation, Section 4 establishes a stochastic model estimating the expected cost involved in executing a circumventing and inserting arbitrary data into the blockchain. From this, the above mentioned main results are calculated.

Finally, Section 5 concludes with a summary of the findings, and discusses potential improvements in the countermeasure.

2 Related Work

3 Data Inclusion in the Blockchain

This section describes the state-of-the-art method how the transaction structure of the Bitcoin blockchain enables the inclusion of arbitrary data in the blockchain. For this, we first outline the transaction processing of the Bitcoin network on the basis on its scripting language. Then, describes the techniques how this structure can be exploited for data inclusion, and sensibly restrict our considerations on one specific technique type.

3.1 Transaction Structure and Processing

In the Bitcoin network, the transfer of electronic coins is realized using *transactions*. From a structural perspective, every transaction specifies a set of *transaction input* and a set of *transaction outputs*, with every input referencing a previous transaction's output. Following these references, each transaction can be considered as a vertex of a directed acyclic graph, where its references the directed edges. Sources of that graph correspond to (mined) coinbase transactions.

To give this transaction authenticity, an eligible user attaches to each reference some form of digital signature. By verifying each signature associated with the preceding transactions in the graph, one can follow and verify the chain of ownership of the coins. Hence, strictly speaking, 'ownership of coins' in the Bitcoin network is the ability to create a transaction referencing outputs not already referenced, using ownership of private keys to generate valid signatures. Using the blockchain as a timestamping server prevents double-spending of transaction outputs.

Bitcoin implements the authenticity system of these references using a challenge–response authentication between the referenced transaction outputs and the transaction inputs.

- Every output contains a *locking script*, which can be understood as the cryptographic puzzle or challenge, which was defined when the transaction was constructed.

- A reference to an output from an input is authentic if the corresponding *unlocking script* can solve the challenge imposed by the referenced output.

Bitcoin provides several methods for implementing locking and unlocking scripts, and as example, the widespread pay-to-public-key script types are presented: In this scenario, a payee Bob expects the payer Alice to publish a valid transaction with his public key embedded in the locking script of an output x. Then, to transfer these coins to Charlie, Bob constructs a transaction t with an input referencing x, adding to the unlocking script a signature of message t, signed with his private key, corresponding to the public key embedded in x. Then Bob can add the output with the locking script expected by Charlie, and submit the transaction; cf. [2, Chap. 6–7].

The network verifies this reference by checking the signature of the unlocking script against the public key embedded in the locking script.

Bitcoin employs a *Elliptic Curve Digital Signature Algorithm* (using parameters from SEC2-secpk256k1) as implementation of a public-key system. Henceforth, we refer to $G \in \mathbb{F}_p^2$ as the base point of the elliptic curve group (over the finite field of prime order p). For $x \in \mathbb{N}$, let $x \cdot G$ denote the elliptic curve point multiplication by a scalar.

Type	locking script / address	unlocking script
P2PK	pubkey $x \cdot G$	sig of tx from privkey x
P2PKH	hash of public key $H(x \cdot G)$	$x \cdot G$, sig of tx from x
P2MS	pubkeys $x_1 \cdot G, x_2 \cdot G, \ldots, x_n \cdot G$	sigs of tx from $x_{i_1}, \ldots, x_{i_m}$
P2SH	hash $H(s)$ of redeem script s	s plus additional data y
Null Data	$\leq 80\,\mathrm{B}$ arbitrary data	(unspendable)

Table 3.1: Overview of the discussed standard address types: P2PK (pay-to-public-key), P2PKH (pay-to-public-key-hash), P2MS (pay-to-multisig), P2SH (pay-to-script-hash), Null Data. For pay-to-multisig (P2MS), $1 \leq m \leq n \leq 3$ is required to be considered standard. See following text for a definition of redeem script.

Following Table 3.1 summarizes the five standard locking script types (i.e. address types) with their corresponding unlocking script. Any other forms of locking scripts are currently being rejected by network participants. Hash function H is implemented in Bitcoin as composition of first SHA256, followed by RIPEMD160.

The P2SH (pay-to-script-hash) type requires further explanation: this address type is a generalization of the challenge–response authentication employed by the network, and allow users to create their own challenges and responses using a limited scripting language. A locking script consists of the hash of the challenge script s (called *redeem script*), while the unlocking script must contain the corresponding redeem script (transferred off-band to the payee), and additional parameters y that the author of the redeem script s requires as input to s to be processed, checking authenticity of the payee, i.e. s accepts on input y.

3.2 Script-Based Data Inclusion

As was introduced above, the Bitcoin network employs a blockchain as a distributed data structure to prevent double-spending of transaction outputs, i.e. multiple references to the same output. The blockchain acts as timestamping mechanism, by providing "a system for participants to agree on a single history on the order [of transactions] in which they were received" [1, p. 307], hence storing every transaction with its relevant information in that data structure on time of submission.

This data structure disallows any modification on the agreed set of transactions *by design*, to maintain the chain of ownership induced by the transactions. Hence the locking and unlocking scripts of every transaction are stored in the blockchain, essentially indefinitely, and distributed to each participant to verify the chain of ownership.

Inclusion on Outputs / Locking Scripts

Null Data transactions already give actors the possibility to create (unspendable) transactions that contain up to $80\,\mathrm{B}$ of arbitrary data payload. This imposed size limit is deliberately low to limit blockchain bloat, and to make this inclusion costly and not suitable for large quantities of data; cf. [2, pp. 155–156].

Therefore, more sophisticated techniques for data inclusion build upon the fact, that the network does not, and cannot test if the variable parts of the locking scripts, e.g. pubkey(s) $x \cdot G$ in P2PK (P2MS) transactions, hashes $H(x \cdot G)$, $H(s)$ in P2PKH, P2SH transactions, are in fact, generated using the usual intended way. For example, consider payload y of $65\,\mathrm{B}$. The network accepts and stores a P2PK transaction with y as 'fake' public key in its locking script, even though $y \neq x \cdot G$ for all x (in fact, even if y does not conform to the syntax of a EC point). Again, this means that the coins 'sent' to the fake address y

cannot be spent, i.e. *burnt* coins, just as it is the case of Null Data transactions. Unlike Null Data outputs, a multitude of P2PK outputs can be present in a transaction, making the technique using P2PK outputs more cost-efficient. Refer to the paper of Sward et al. [11] for an in-depth discussion of efficiencies using this technique.

Inclusion on Inputs / Unlocking Scripts

While the previously presented inclusion techniques insert their payload in the locking scripts, Sward et al. [11] analyzed inclusion approaches using P2SH unlocking scripts. Again considering payload y, they construct a specialized redeem script s and unlocking script s', such that y is a substring in s'. To now perform the inclusion into the blockchain, first one publishes a transaction containing P2SH output with $H(s)$, then another transaction referencing this output, containing s' as unlocking script. This results in two valid transactions, and even require no burnt coins. In fact, Sward et al. showed that this technique outperforms every known inclusion technique using locking scripts, given the fact that unlocking scripts have a much higher size limitation in comparison to locking scripts, which have a fixed format.

    Nevertheless, we will exclude an analysis of this inclusion technique in this report. These complex transaction types might be the easiest to ban or limit, if data inclusion using P2SH were to become a problem. Since most P2SH transaction effectively implement a multisig scheme, a consensus for imposing a fixed structure for redeem scripts might be established, especially if data inclusion using unlocking scripts causes major threats.

4 Countermeasure and Circumvention

The following section first gives a detailed definition of the proof-of-ownership countermeasure. After specifying the functionality and demonstrating its intended effect, we develop an efficient brute-force method to circumvent this countermeasure. The last part of the section gives details on how to implement such circumvention in practice.

A straightforward idea for preventing data inclusion would make the network 'forget' stored information after processing, or allow the network to remove undesirable data, or recognize bad transactions using a heuristic, before they are distributed and stored. However, all these preventions conflict with certain properties of the network, that are universally considered characterizing for the Bitcoin system. Hence, the adoption of such countermeasure would transform the Bitcoin system to such an extent, that the resulting electronic cash system would in general not be considered 'Bitcoin'.

While there is no central authority stating these properties, the corresponding *Bitcoin Wiki* text can serve as a representative for the consensus of the community:

> "All changes and upgrades to the protocol should strive to maintain and reinforce these Principles of Bitcoin
>
> [1.] 21 million coins.
>
> [2.] No censorship: Nobody should be able to prevent valid txs from being confirmed.
>
> [3.] Open-Source: Bitcoin source code should always be open for anyone to read, modify, copy, share.
>
> [4.] Permissionless: No arbitrary gatekeepers should ever prevent anybody from being part of the network (user, node, miner, etc).
>
> [5.] Pseudonymous: No ID should be required to own, use Bitcoin.
>
> [6.] Fungible: All coins are equal and should be equally spendable.
>
> [7.] Irreversible Transactions: Confirmed blocks should be set in stone. Blockchain History should be immutable" [12].

The mentioned countermeasures above would contradict principles 4, 7, at least.

Hence we continue our analysis only on countermeasures that are compatible with these properties, since all nonconforming countermeasures might face serious opposition to adoption from the community. From these principles appear to follow, that these countermeasures must respect following propositions, when determining if a submitted transaction contains no bad data:

(a) Every honest user can convince the system implementing the countermeasure in question that its submitted transaction is not bad, without relying on third parties, i.e. using only the private key (by principle 2, 5, 6).

(b) Further assessment needs to be made in a transparent way, especially without an authoritative figure (by principle 4),

(c) it needs to be made *before* publication of the transaction (by principle 7).

As a consequence of (a), honest addresses are those derived from a private key, as otherwise, proof asserting honesty requires a third party. This implies that all addresses possible to generate are clean in general. While (b) and (c) could allow for further consensus-approved content filters deciding which addresses are clean, the decentralized nature of the Bitcoin network, together with (b) and (c), make efficient filters particularly hard to implement.

Hence, without loss of generality, we need to assume a submitted destination address as clean if the owner of that address can prove the existence of the corresponding private key, that is, ownership of the address.

Matzutt et al. already pointed out that, under this setup, pseudonymous identifiers (cf. principle 5) always open the possibility to insert data by encoding them into these identifies: no matter how 'existence of the private key' is implemented, users willing to insert content into the blockchain always have the possibility to generate arbitrarily many addresses until suitable ones encoding the content are found [6, Sec. III.C].

Following section introduces a Proof-of-ownership countermeasure that appears to be the strongest one respecting these propositions, motivating the evaluation of this countermeasure and its circumvention, which directly incorporates the approach outlined by Matzutt et al.

4.1 Proof-of-Ownership Countermeasure

In his Master's thesis *Hardening Bitcoin against off-topic data inclusion*, Seeg proposes a countermeasure to this problem using a proof-of-ownership mechanism, which he calls "preimage solution" [5]. He considers an address as clean (i.e. without arbitrary data) if and only if the owner of that address can prove control, that is, ownership of the address. Such proof of ownership is derived from the 'private' preimage part of the address, similar to a classic challenge–response authentication. The wanted effect follows from the assumption that given 'public' part of a non-clean address, one cannot efficiently derive 'private' preimage required for the proof of ownership.

For P2PK address types, this translates into a proof using signatures, which certify the control of the corresponding private key. That is, in order for Alice to send coins to Bob, he sends her out-of-band his public key $x \cdot G$, and – this is the addition of the countermeasure – a signature σ verifying message $m = H(x \cdot G)$, generated using private key x. This signature proves that Bob is in control if his private key. Then, Alice can construct a P2PK transaction with recipient key/address $x \cdot G$, and when submitting the transaction to the network, she attaches σ which proves that Bob indeed controls that address. (This entails that σ needs to be transmitted out-of-band to Alice before the transaction.) Miners can verify σ using the public key $x \cdot G$ embedded in the transaction, approving the output address of the transaction.

This countermeasure protects against data inclusion: if an adversary wishes to embed message into public key y, he cannot efficiently derive x such that $x \cdot G = y$, required in order to sign the proof σ. Thus, given valid σ, the transaction in question can be safely included into the blockchain.

In the case of P2PKH and P2SH addresses, proofs are constructed in a similar manner. For P2PKH addresses $H(x \cdot G)$, signed message m remains identical. Only the public key $x \cdot G$ is additionally transferred such that σ can be verified by the network. For P2SH addresses $H(s)$, the redeem script s serves as preimage/proof σ. Again, this prevents data inclusion due to the one-way property of hash function H. Note that in the case of P2PKH and P2SH transactions, Seeg proposed a slightly weaker approach, exploiting the choice of H as composition of two hash methods, and using the first digest as σ.

In fact, this type of countermeasure appears to be one of the strongest compatible with the Principles of Bitcoin. We already observed that existence of a private key needs to be taken as sufficient proof that the public address is clean, and this countermeasure

appropriately takes respective signatures σ as such proof of existence. Addresses that withstand the proposed test hence are precisely the addresses that are derived from private keys.

Additionally, other countermeasures that respect these mentioned principles build on top of the one-way property of the same cryptographic functions in a similar matter, e.g. "self-verifying account identifiers" proposed by Matzutt et al. [6, Sec. IV.D]. Therefore, this countermeasure and the respective circumvention act as representative for all one-way-based approaches.

4.2 Circumvention Using Partial Preimages

In essence, the proposed countermeasure builds upon preimage resistance of the cryptographic functions translating a private key into a public address. That is, for given address derivation function f, required preimage x for the proof-of-ownership cannot be efficiently derived from image $f(x)$, whereas only $f(x)$ is stored in the blockchain. Attempts to embed chosen y as address into the blockchain hence become computationally hard, since finding x such that $f(x) = y$ is required. For given address type, the address derivation function f is precisely the procedure translating a private key into the locking script, as outlined in Table 3.1. From the implementation follows, that the countermeasures builds upon the (presumed) one-way property of hash function H (in the case of P2SH addresses) and elliptic curve multiplication (in the case of P2PK, P2PKH, P2MS addresses).

No matter the chosen function f, the trivial approach to find a preimage to value $y = f(x)$ is via *brute force*: we test pairwise different preimage candidates $x_1, x_2, \ldots$ until we find x_i such that $f(x_i) = y$. The preimage x_i can then be used to generate suitable proofs for the image (e.g. signatures σ). Nevertheless, successfully executing such a preimage attack is highly unlikely due to the large search space. In fact, if that attack were possible, an adversary could gain control of the funds assigned to any public key (by inverting the public-key function f).

While a full preimage attack is out of reach, *partial* preimage attacks are achievable nonetheless: Given a prefix bit-length n, we define x as *partial preimage* of y when $f(x)$ equals y *in the first n leftmost (most significant) bits*, i.e. $f(x) = y\|z$ where z is an arbitrary suffix.

For uniformly chosen private key candidate x_i, we can assume that the respective n-bit prefix of the public key $f(x_i)$ is again uniformly chosen from the range $[0:2^n]$. By this, the probability of x_i being a partial preimage of y increases to $1/2^n$, which, after sufficiently many repetitions and sufficiently small n, leads to suitable partial preimage x; the corresponding public key $f(x)$ then contains the wanted payload y as prefix. In terms of the proposed countermeasure, one would continue by constructing transaction containing output address $f(x)$ and proof σ from x. Submitting both would then successfully circumvent the countermeasure and include y into the blockchain, as prefix of $f(x)$.

While the notion of prefixes is clear in the case of hash digests involved in P2PKH and P2SH addresses, a prefix of a public key $x \cdot G = (a, b) \in \mathbb{F}_p^2$ involved in P2PK addresses is not necessarily obvious. We define the n-bit prefix of $x \cdot G$ as the n most significant bits of first coordinate a of that curve point. (From the perspective of the SEC-defined curve point representation used in Bitcoin, this corresponds to bits 9 to 264. In fact, first byte of the SEC representation is fixed.)

Note that the relative strength of this countermeasure cannot be amplified by hardening the function f. While harder f make computation of partial preimages increasingly difficult, so are honest users inhibited by harder address generation and transaction verification.
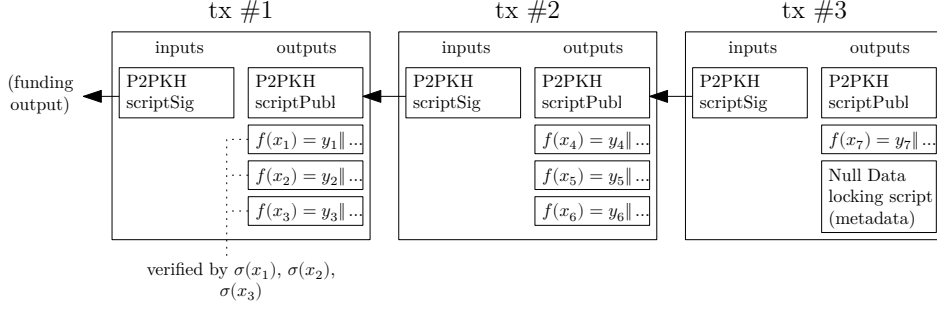
Figure 4.1: Proposed scheme to construct transaction from the prepared addresses, which are partial preimages of the payload's fragments. Submitting tx #1, #2, #3 together with the respective proofs $\sigma(x_1), \ldots, \sigma(x_7)$ would then circumvent the countermeasure. Arrows from input to output signify which output the respective input is referencing; that is, the *logical* direction is from right to left, the *chronological* direction from left to right. Usually, more than three payload-holding outputs can be bundled into a single transaction.

4.3 Design and Implementation

Having provided a framework showing the theoretical possibility to circumvent such countermeasure, following section describes the necessary steps to perform such inclusion in practice. That is, including payload data into the blockchain, where the data is embedded into ordered output addresses of transactions. Additionally, following procedure generates private keys to each of the used output addresses, such that proofs of ownership can be deduced as required in the countermeasure, and hence the generated transactions are approved. It is to be noted, that, in the context of Seeg's proposal, investing in a rainbow table could significantly increase performance in comparison to a full brute force approach. Nevertheless we omit rainbow tables, since this issue could easily be solved when refining the countermeasure, e.g. including transaction hash in the proof message m to be signed. Cf. the "freshness property" in the approach proposed by Matzutt et al. [6, pp. 368–369].

The procedure is given with a chosen prefix length n. Performance and cost directly depend on this parameter, and a fit choice for n is discussed in the next section. Without loss of generality, we assume the payload length to be a multiple of n.

Detailed Description of the Procedure

Preparation. First, we partition the payload into fragments $y_1, y_2, \ldots, y_p$ of length n, and use a binary tree as efficient data structure to access the fragments, sorting them by their lexicographic order.

Partial preimage loop. Then, we have to find a suitable partial preimage x_i for every fragment y_i with $1 \le i \le p$ in the tree. In each iteration (of the outer loop), we generate many private key candidates x by brute force (inner loop), until the prefix of $f(x)$ matches any fragment remaining in the tree. We then can assign matching fragment y_i the private key $x_i := x$, and remove y_i from the tree. This leads to a mapping $y_i \mapsto x_i$ for all $1 \le i \le p$.

The implementation is trivial for the P2SH method: let s be an injective function transferring a nonce m into a valid redeem script. We start off with a random nonce m, and test $H(s(m))$, increment m, test again, and so forth. Having found m with prefix of $H(s(m))$ matching one fragment in the tree, we can assign that fragment the private key $s(m) =: x_i$.

In the case of the P2PK method, this translates to testing products $m \cdot G$, $(m+1) \cdot G$, $(m+2) \cdot G, \ldots$ from random starting nonce m. Using the axioms of the group formed by the elliptic curve, we can avoid costly elliptic curve multiplications: starting with random nonce m, we cache public key $P := m \cdot G$. When incrementing m, we add generator point

to P, that is $P := P + G$, and ensure invariant $P = m \cdot G$. Again, having found m with prefix of P matching one fragment in the tree, we assign that fragment private key $m =: x_i$. The P2PKH method can be implemented similarly, by testing hashes $H(P)$.

Transaction construction. From assigned private keys, we can construct transactions containing the respective public keys $f(x_1), f(x_2), \ldots$ in order, embedding the prefixes $y_1, y_2, \ldots$. That is, we construct a transaction with a single funding input, and add corresponding outputs containing the public keys as destination addresses, whereas every output must send a value of at least the non-dust amount of 546 sat. If the size of the transaction exceeds a defined limit, we add an output referred by the next transaction as funding input, as depicted in Figure 4.1. This also ensures that the network limit of 100 kB per transaction is followed. A terminal Null Data output allows embedding of metadata, facilitating reconstruction of the payload. Since we have chained the transactions using funding inputs, the transaction identifier of the last transaction can serve as locator for the data in the blockchain. It is recommended to use P2PK(H) scripts as funding inputs, since the signatures in the respective unlocking scripts parts effectively prevent modifications to the transaction, for example an unwanted reordering of the transaction outputs.

Signature generation and submission. Again, having the private keys $x_1, x_2, \ldots$, it is trivial to generate the proofs confirming ownership, e.g. the signatures as defined by the proposed proof-of-ownership countermeasure. Submitting both the proofs and the previously generated transaction then successfully embeds the entire payload into the blockchain. Note that this approach allows for the payload to be split up between multiple blocks in the blockchain.

Computational Properties

We carefully ensured that the inner iterations of the partial preimage loop, the major part constituting the computational hardness, is particularly efficient, especially by using a binary tree to quickly search for matching fragments, and caching the elliptic curve point P. Hence, it seems unlikely that modifications on the iterations could increase performance.

Likewise, it is plausible that the performance of this algorithm cannot be considerably increased. Any other modifications would need to employ sophisticated techniques to limit the search space of potential nonces, which would directly attack the underlying cryptographic primitives (SHA256 and elliptic curve cryptography).

We further observe that the preimage loop can easily be parallelized on multiple cores and computers, especially allowing efficient implementations on GPUs.

Overview of the Implementation

5 Increased Cost of Data Inclusion

After having demonstrated the possibility to circumvent the proposed countermeasure by brute force, this section attempts to evaluate the effectiveness of that countermeasure by taking into consideration the computational cost of the circumvention. We assume a (not necessarily adversarial) user, attempting to use the blockchain as data storage. We will see that the user's chosen prefix length for the partial collisions directly determines the inclusion cost, hence we first model the inclusion cost mathematically. Then, by giving further estimations of essential parameters, we are able to predict by which factor the proposed countermeasure using proof-of-ownership signatures increases inclusion cost with respect to current, unprotected situation using fake addresses, as presented in Section 3.2

5.1 Modeling Inclusion Cost

Essentially, the choice of a suitable prefix length is a trade-off: On the one hand, the number of required keypairs, and hence needed transaction fees, are inversely proportional to the chosen prefix length; on the other hand, the probability for a partial collision decreases exponentially with respect to the prefix length, and in turn, computation cost increases exponentially.

To precisely model the cost of an inclusion of an arbitrary payload into the blockchain, restricted with the proposed countermeasure, let us denote

- prefix length with n,
- payload length with N,
- cost of a single hash operation with c,
- transaction fee for a single address with f.

Moreover, we denote $m = N/n$ as the number of required addresses, (or equivalently, the number of payload fragments) and X as the random variable of required hash operations. From this, total cost C of an inclusion can be given with

$$C = cX + fm = cX + fN/n.$$

To precisely determine random variable X, we remind ourselves of the general iterative procedure of the brute-force algorithm: In each iteration, the algorithm determines a keypair such that its prefix matches one of the payload fragments not yet assigned a keypair, and then assigns that fragment the keypair. That is, in iteration i, the algorithm chooses private keys randomly, until generated public key partially collides with one of the $m-i$ remaining unassigned payload fragments. In the worst case, all fragments are pairwise different, hence one iteration can be viewed as repeated independent Bernoulli trials until success (i.e. partial collision found), with success probability $(m - i)/2^n$. Referring to required hashes in iteration i with X_i, we thus observe that the random variable X_i follows a geometric distribution with parameter $(m-i)/2^n$. For the total number of required hash operations X, we reach (with H_p the m-th harmonic number)

$$X = \sum_{i=1}^{m} X_i, \quad E[X] = \sum_{i=1}^{m} E[X_i] = \sum_{i=1}^{m} \frac{2^n}{i} = 2^n H_m,$$

and conclude for the expected total cost C

$$E[C] = 2^n H_m c + fm. \tag{5.1}$$

Type	script size	payload length	addresses per tx	total cost in BTC	f in sat	f in USD
P2PK	35 B	32 B	2267	0.022 375 52	987.01	7.70×10^{-2}
1-of-3 P2MS	105 B	32 B	2625	0.014 775 40	562.07	4.50×10^{-2}
P2PKH	25 B	20 B	2934	0.026 018 14	886.78	7.09×10^{-2}
P2SH	23 B	20 B	3117	0.027 016 12	866.73	6.93×10^{-2}
1-of-3 P2FMS	201 B	65 B	1425	0.012 591 40	883.60	7.07×10^{-2}

Table 5.1: Overview of the different transaction types. Second column denotes the scripts size for a transaction output, holding an address (resp. three addresses in the case of multisig), third column the maximum payload length per address. Fourth column gives how many addresses can be gathered in a single transaction (not exceeding the size limit of 100 kB), fifth column the total cost for submitting this transaction, that is with smallest non-dust amount (546 sat) per output plus transaction fees of 10 sat per byte. Value f is obtained by dividing total cost by number of addresses, further assuming current exchange rate of 8000 USD per BTC.

5.2 Estimation of Parameters

Before we can determine the cost increase caused by the proposed countermeasure, we need to give sensible estimations for the cost parameters c and f involved in the previously stated cost model (5.1). While fees f for publishing an address are directly determined by the miners' transaction fee rate, hashing cost depends on the efficiency of the hardware used for the brute-force process. For this, we estimate the parameter indirectly using hashrates reported by users of the tools *VanitySearch* by Pons [13] and *Hashcat* by Steube et al. [14].

Transaction Fees and Burnt Coins

The fees required for publishing the address into the blockchain, i.e. parameter f, can easily be estimated using average bytes required for a transaction output. For this, we construct the largest possible transaction using a single P2PK coinbase input to fund the outputs with non-dust burns, and adding outputs until reaching network's size limit of 100 kB. Multiplying with miner's transaction fees, and adding smallest non-dust burn amount to each output yields the value for parameter f. Table 5.1 gives this parameter in absolute values for all three methods, using transaction fees and Bitcoin exchange rate at point of writing. That is, network's transaction fee of 10 sat per byte, and 8000 USD per bitcoin. Note that the P2MS strategy, arranging pubkeys in 1-of-3 multisig outputs, is currently the least expensive one in terms of fees required. Even though multisig outputs cause more overhead in the constructed transaction, required non-dust amount is counted per output, hence, grouping three pubkeys into one transaction output saves on burnt bitcoins.

Since the P2PK method performs worst in this regard, we discard this method and focus on the related method P2MS, besides the fact that these two methods only differ in their transaction construction, yet are identical in their brute-force procedure.

Hash Cost Estimation

For the P2PKH method, we can rely on user reports on their performance using the tool *VanitySearch*, which allows users to create *vanity addresses*. These Bitcoin addresses contain a human-readable message as prefix in their Base-58 representation; cf. [2, pp. 82–83]. Similarly to the presented tool in previous section, *VanitySearch* brute-forces private

Hardware	wattage	est. freq.	c in USD	ref.
P2PKH				
GTX 1660 TI	120 W	960 MHz	4.51×10^{-15}	[15, #396]
GTX 1650	75 W	510 MHz	5.31×10^{-15}	[15, #374]
RTX 2080 Super	250 W	2000 MHz	4.51×10^{-15}	[15, #396]
RTX 2080 TI (MSI)	300 W	2500 MHz	4.33×10^{-15}	[15, #343]
Tesla V100-SXM3-32GB	300 W	2500 MHz	4.33×10^{-15}	[15, #619]
P2SH				
GTX 1080 TI	250 W	2400 MHz	3.76×10^{-15}	[16]
RTX 2080 TI	300 W	3700 MHz	2.93×10^{-15}	[17]

Table 5.2: Selected user's reports of their brute-force frequencies on specific hardware. For the P2PKH method, frequency was directly taken from reported *VanitySearch* speeds. For the P2SH method, SHA256 hash frequency reported from *Hashcat* was divided by factor 2, as explained in the respective section. We estimate cost parameter c for the P2PKH by first researching estimated power consumption of the GPU under full load, and assuming energy cost of 0.13 USD per kW h.

keys, until the hashed public key has the desired prefix; this public key hash is precisely the one used to create P2PKH hashes.

Hence, due to the computational similarity, it seems reasonable to estimate possible computational cost for the P2PKH method from reported frequencies of *VanitySearch* on selected hardware. We assume that the computational effort of testing if a public key hash partially collides with one of the remaining unassigned payloads is negligible. Table 5.2 gives estimations for cost c bases on selected user reports form the *Bitcointalk* forums. We assume an energy cost of 0.13 USD per kW h.

In a similar matter, one can use the performances in computing SHA256 digests as estimation for the cost of the P2SH method. As was outlined in previous section, the brute-force procedure for the P2SH method constructs from nonce candidate x a 57-byte long redeem script $s(x)$, and computes the P2SH address as the hash digest $H(x)$, that is by definition

$$x \mapsto \text{RIPEMD160}(\text{SHA256}(s(x))). \tag{5.2}$$

The primitive hash functions RIPEMD160 and SHA256 are highly similar in architecture and performance; their respective Merkle–Damgård transforms also operate on the same block size of 512 bits. Cf. for example the *ECRYPT Benchmarking of Cryptographic Systems* [18]. Therefore, the effort required in above procedure (Eq. 5.2) can be considered twice as large as one SHA256 digest computation (in the sense of input length ≤ 512 bit).

Likewise to previous estimation using *VanitySearch*, we can use hash frequencies published by users of the password cracking tool *Hashcat*. Dividing the reported SHA256 hash frequency by factor 2 thus yields an approximate frequency accomplishable for the P2SH brute-force procedure. (We can assume that the GPU can be programmed to perform RIPEMD160 calculations in similar speed.) Again, refer to Table 5.2 for estimations on cost c bases on selected user reports. These numbers should explicitly not be understood to represent the current state of hardware efficiency, but rather as figures attainable with available off-the-shelf hardware. While non-GPU hardware has the potential to be more energy-efficient, either the hardware cannot be easily adapted (e.g. in the case of ASIC hardware), or the frequencies are too low to perform the brute-force algorithm in sensible time.

Both the P2PKH and the P2SH method perform the same hash operations, only the former has to additionally perform a elliptic-curve addition. Comparing the two methods,
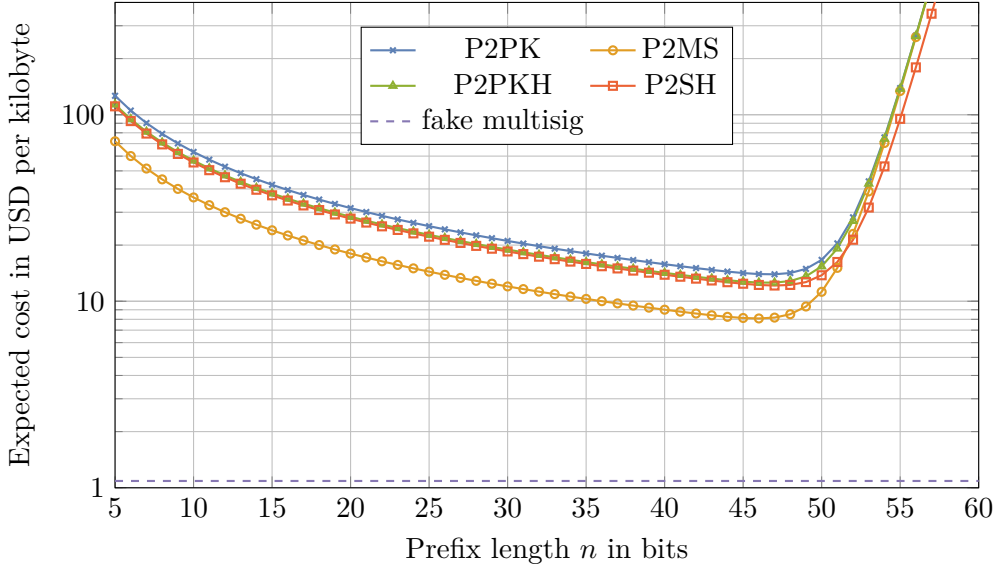
Figure 5.1: Log-linear plot of expected total cost $E[C]$ (see Equation 5.1) with $N = 10\,\text{kB}$ for inclusion methods P2PKH ($c = 4.50 \times 10^{-15}$ USD, $f = 7.90 \times 10^{-2}$ USD), P2MS ($c = 4.50 \times 10^{-15}$ USD, $f = 4.50 \times 10^{-2}$ USD), P2SH ($c = 3.0 \times 10^{-15}$ USD, $f = 6.93 \times 10^{-2}$ USD). For comparison, dotted line indicates constant efficiency achievable using fake addresses in multisig outputs (P2FMS).

we can already observe that the digest calculation in the P2PKH method significantly contributes to total computation time, next to the elliptic-curve arithmetic.

Specialized software for cracking ECDSA keypairs does not appear to exist, hence we cannot estimate parameter c in the same manner. We therefore omit a precise cost estimation of the P2PK/P2MS method, and use the cost parameter c of the P2PKH method as upper bound.

5.3 Comparison with State-of-the-Art Methods

This section compares the impact of the proposed inclusion method with the current situation, having no countermeasures. To begin with, we choose following rough but sensible estimates for computation cost parameter c for methods P2MS, P2PKH, P2SH, based on our observations from previous section. We estimate $c_{\text{ms}} = c_{\text{pkh}} = c_{\text{p2pkh}} = 4.50 \times 10^{-13}$ USD, $c_{\text{sh}} = 3.00 \times 10^{-15}$ USD. Also stated in previous section, fee parameter f directly follows from Bitcoin exchange rate and mining fees, and is displayed in Table 5.1.

From the results of Sward et al. [11, cf. Table 3], we can infer that the best present method of including arbitrary data using fake addresses exploits fake public keys (in their extended 65 byte representation) arranged in 1-of-3 multisig outputs, not being inhibited by any countermeasure requiring proof signatures (P2FMS). Relative cost of inclusion using this state-of-the-art method can easily be estimated using same formula (Eq. 5.1), assigning $c = 0$, maximum prefix size $n = 65\,\text{B}$, and computing f identically to previous procedure (see also table Table 5.1). This gives us a constant cost efficiency of 1.09 $\text{USD}\,\text{kB}^{-1}$ as benchmark.

Figure 5.1 plots the expected cost per byte payload, with respect to prefix length n, using $N = 10\,\text{kB}$ and previously stated parameters, for each method. As comparison, the constant relative cost of the state-of-the-art method is highlighted by the dashed horizontal line.

	opt. prefix length	exp. cost in USD per kB	rel. increase w.r.t. P2FMS
P2PK	47	13.9	12.8
P2MS	46	8.08	7.43
P2PKH	47	12.5	11.5
P2SH	47	12.1	11.2

Table 5.3: Minimum point n and minimum value E[C] from plot in Figure 5.1. Rightmost column gives the relative increase with respect to the constant efficiency achievable using fake addresses in multisig outputs.

We confirm that determining the most cost-effective prefix length corresponds to finding the local minimum of the cost function. Furthermore, the plot confirms the trade-off nature of the problem: left of the minimum point, cost increases proportional due to more addresses required; cost right of the minimum point is dominated by the exponentially increasing brute force computation cost.

We give minimum cost per kilobyte payload in Table 5.3, and obtain following result: defending the Bitcoin network against unwanted data inclusion using proof-of-ownership signatures could allow for an 7-fold increase of inclusion cost, at current situation.

Nevertheless, it needs to be highlighted that optimal cost efficiency can only be obtained using significant computation time. For example, using the fastest GPU for the P2SH inclusion method from Table 5.2, performing the brute-force algorithm with 10 kB payload already requires more than 84 hours of computation. In fact, real-time inclusion using non-specialized hardware seems out of reach for prefix lengths as low as 28 bits, requiring more than a minute of computation time on a typical personal computer for payload length of $N = 100$ B (assuming hash frequency of 10 MHz).

5.4   Variations in Parameters

We established a model estimating cost of inclusion from parameters N, f and c, corresponding to payload size resp. the economic situation of the Bitcoin network (transaction fees, bitcoin exchange rate) resp. computational cost of the hashing procedure. It is therefore logical to investigate the impact on changes on this parameter, especially since a continuation of the exponentially growing trend of computation efficiency is expected. To keep the analysis reasonable, we restrict the parameter space to $10^{-20} \leq c \leq 10^{-14}$ for computation cost, and for the bitcoin price p to range 10^3 USD/BTC $\leq p \leq 10^6$ USD/BTC. Due to the transcendental nature of the cost function E[C] with respect to prefix length n, it seems out of reach to formulate a closed-form solution to finding optimal prefix length and respective total cost, given parameters N, f, c. Hence, numerical methods are applied to determine minimal cost, given numerical values of the parameters.

Before discussing the impact of p and c, we can observe that relative savings from larger payload size N are effectively negligible, that is, total cost is near-linear with respect to payload size; this trend is independent of all other parameters. Therefore, we continue our analysis by looking at the cost of an inclusion *relative* to payload size, while additionally fixing $N = 10$ kB.

In order to derive parameter f from bitcoin exchange rate p, we fix the P2MS method and network transaction fees of 10 sat per byte. This is motivated by the fact that 10 sat per byte can be considered a lower bound on the network's transaction cost, and assuming this value, P2MS method is the cheapest one. Accordingly, the final results of the following calculation can be interpreted as a reasonable lower bound.
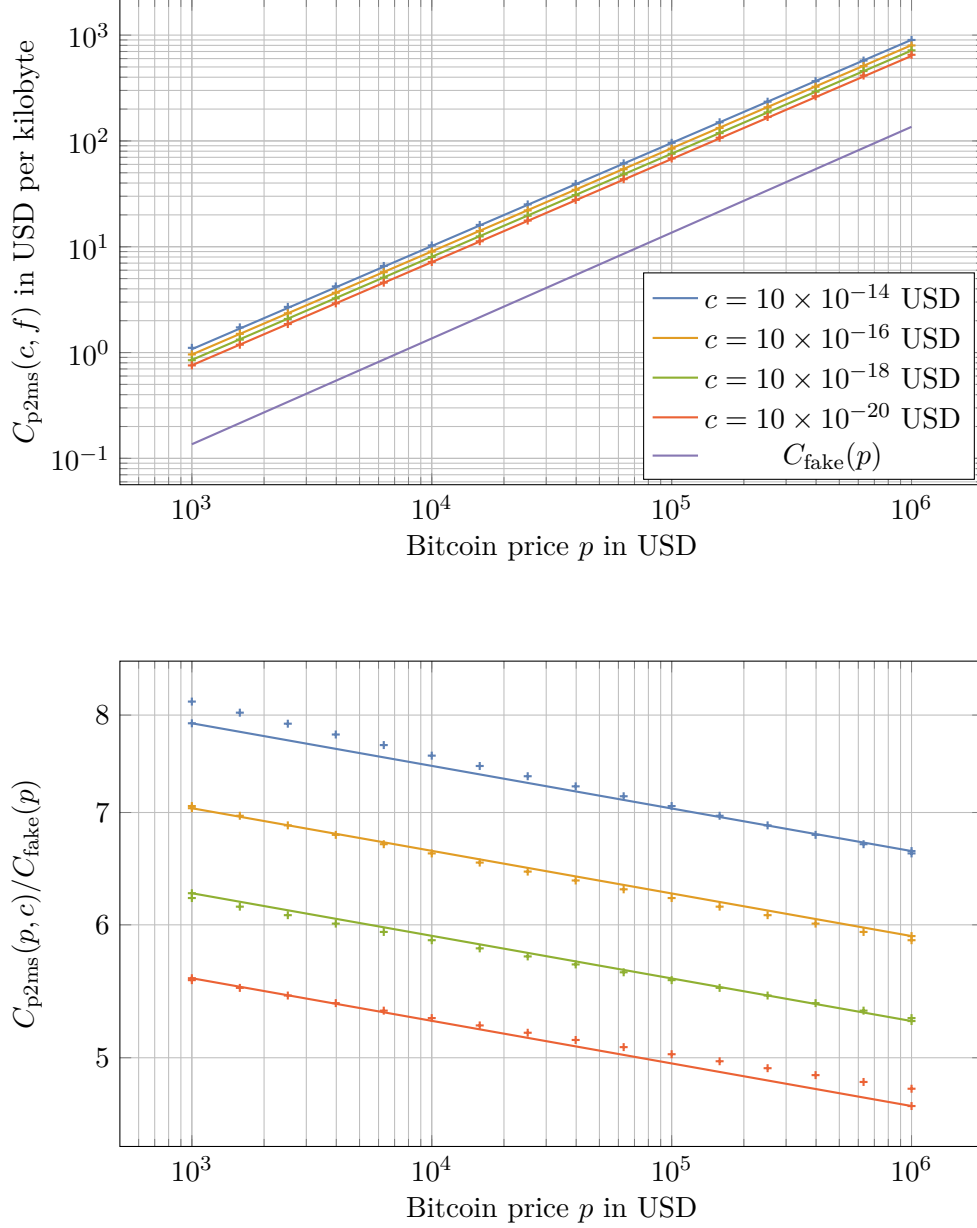
Figure 5.2: Top figure plots the optimal relative inclusion cost for the P2MS method with respect to fixed parameter c and variable bitcoin price p in a log-log scale. For comparison, red line shows the linear relative inclusion cost using fake keys. Marked points denote numerically determined values, while the respective lines show the fit specified by the established regression model. Bottom figure is set up identically, but plots the respective effect factor of the countermeasure, i.e. the relative increase of inclusion cost with respect to P2FMS.

As was already performed in Section 5.2, we can define parameter f in units of sat, and multiply with p to obtain parameter f in terms of USD. Combining with given parameter c, let $C_{\text{p2ms}}(p, c)$ be the optimal relative inclusion cost in USD per byte, determined by numerically finding optimal prefix length n. By same methods, we can generalize the inclusion cost using fake keys, as

$$C_{\text{fake}}(p) = \frac{N}{65\,\text{B}} \cdot f_{\text{p2fms}} \cdot p.$$

Cf. Equation 5.1 and Table 5.1.

In order to state the effect of bitcoin price and computation cost quantitatively, we proceed with numerically computing $C_{\text{p2ms}}(p, c)$ for c and p in the previously fixed respective ranges; both price p and c in $10^{0.2}$-fold increments, yielding 416 data points.

We continue by presenting a bivariate log-log regression model to explaining the dependent data points from the independent variables, that is,

$$\log_{10} \frac{C_{\text{p2ms}}(p, c)}{\text{USD}\,\text{B}^{-1}} = \alpha + \beta \log_{10} \frac{p}{\text{USD}} + \gamma \log_{10} \frac{c}{\text{USD}} + \epsilon_i, \qquad (5.3)$$

with unknown parameters α, β, γ, and error terms ϵ_i. Usual least-squares-error method provides $\alpha \approx -5.5378$, $\beta \approx 0.974\,61$, $\gamma \approx 0.025\,315$.

It is explicitly to be remarked that this model does not precisely explain the data; this becomes clear when considering the visible patterns in the residual graph of this regression. Nevertheless, the model appears to be sufficiently precise to estimate the effect of the variables p, c on relative inclusion cost: in fact, mean square error $< 10^{-5}$, and maximum residual is $\max |\epsilon_i| < 0.0129$. Consequently, the model predicts inclusion cost with a relative error of at most $3.01\,\%$. See also Figure 5.2, which plots observed data points against the fit in a log-log plot.

Having established that model, we can derive from parameters β and γ following observations: First, as we expected, relative inclusion cost $C_{\text{p2ms}}(p, c)$ is almost linear with respect to bitcoin price c. Since sub-linear, we expect that $C_{\text{p2ms}}(p, c)$ continues to converge towards $C_{\text{fake}}(p)$, which grows linear with respect to p. Nevertheless, this trend appears to be minor; a 100-fold bitcoin price increase lowers relative inclusion cost by no more than $11\,\%$.

Second, computation cost c has – independent of bitcoin price p – only slim effects on inclusion cost: lowering computation cost by factor 10, for example, would reduce relative inclusion cost by $5.66\,\%$. (Moore's law would estimate such cost decrease of factor 10 in a time span of 7 years.)

We conclude: even though effectiveness of the proposed countermeasure decreases both with increasing bitcoin price and decreasing computational cost, effectiveness remains stable even for extreme values. For our established bounds of parameters, this means $4.7 \leq C_{\text{p2ms}}(p, c)/C_{\text{fake}}(p) \leq 8.1$.

6 Conclusion

6.1 Summary

Section 2 gave an overview of the current method to include off-topic data using fake addresses, exploiting the presence of data fields in the transaction that are essential for payment processing in honest transactions. This exploit depends on the difficulty to verify the authenticity of (generalized) destination addresses, i.e. the network cannot distinguish between honest addresses and payload data masquerading as an address.

In consideration with the (community-defined) principles of the Bitcoin network, we generalize a countermeasure proposed by Seeg. In principle, this proof-of-ownership considers a transaction clean if and only if the sender of that transaction has control of the private components of the destination addresses. That is, if and only if the sender can provide signatures proving his control. We argued in Section 3 that such countermeasure can be considered maximally strong with respect to the principles of the network, motivating the analysis of the effectiveness of that countermeasure.

From the considered countermeasure immediately follows a parallelizable brute-force approach to circumvent that countermeasure, outlined in Section 3.2. Section 3.3 gives an abstract overview of the specific details of an implementation. Specifically, the implementation allows a trade-off between computation time required for generation, and transaction fee required for publication into the network. Furthermore, the construction generates addresses in such a way that the straightforward reconstruction of the payload data is possible.

Section 4 gives a stochastic model that estimates the cost required by the brute-force circumvention, using the outlined implementation, assuming the presence of the considered countermeasure. Then, required parameters (price and computing cost) are estimated using publicly available data on hardware performance, implying that, at the current date, the proposed countermeasure would increase the cost of arbitrary data inclusion by factor 7. Variations in these parameters show that this factor is in the range $[4, 8]$ for realistic values, considering the best-performing P2MS method. Furthermore, our regression model implies that inclusion cost circumventing the countermeasure is dominated by Bitcoin price in a near-linear relationship. Impact on computing cost c however is heavily sublinear, with a growth behavior proportional to c^γ, $\gamma \ll 1$ (cf. Eq. 5.3), implying that improvements in hardware lead to minor effects on the effectiveness of the countermeasure.

Besides the increase in inclusion cost, the required computation time (even if cheap, relatively speaking) might already discourage actors wishing to include arbitrary data into the blockchain. In other words, the countermeasure ensures that a large data inclusion within a reasonable time is only possible if a lot of parallel computing power is available.

6.2 Outlook

If inclusion of data into the blockchain were to become a major and existing problem, adoption of Seeg's countermeasure could contribute to protecting against this attack. While the additional checks imposed by the countermeasure appear to be in accordance with the general principles of the network, it comes with drawbacks concerning general usability and increased verification work for miners. Furthermore, a parallelization of the brute-force process might make data inclusion a feasible option for some actors.

Following our assessment that the proposed countermeasure is maximally strong with respect to the principles, we observe that the effectiveness of the countermeasure directly depends on the computational hardness of the outline brute-force algorithm. While replacements in the underlying cryptographic primitives of Bitcoin are expected to be controversial, these could have a significant impact on the circumvention's hardness, and hence can be employed to further improve the effectiveness of the countermeasure.

For example, a sensible strengthening could be achieved by replacing the implementation of Bitcoin's script hash function H to a memory-hard hash function (e.g. *scrypt* [19]). With applicable parameters, this adoption would make the parallelization of the brute-force process (memory-)costly, but without strongly impeding miners computing the hash digests during transaction verification. Thus, combined with a ban of P2PK/P2MS transactions (i.e. restricting outputs to H-digests), the time required to compute the respective keypairs for a data inclusion could be moved out of reasonable bounds.

Nevertheless, with the presence of the discussed countermeasure, it is unreasonable to expect an adversary attempting to deliberately insert large problematic data. For one, even without the discussed replacement of the hash function H, such large-scale inclusion attack would require significant hashing power, which would be rather invested in gaining a relevant portion of the network's hashrate. For another, the inclusion of large data would significantly increase transaction fees, effectively translating this attack into a denial-of-service attack, whereas the latter can be performed by easier means.

List of Figures

List of Tables

References

[1] S. Nakamoto, "Bitcoin: A peer-to-peer electronic cash system," in *Mastering Bitcoin: programming the open blockchain*, Second edition. Sebastopol, CA: O'Reilly, 2008, pp. 305–316, reprint of the original, ISBN: 978-1-4919-5438-6.

[2] A. M. Antonopoulos, *Mastering Bitcoin: programming the open blockchain*, Second edition. Sebastopol, CA: O'Reilly, 2017, ISBN: 978-1-4919-5438-6.

[3] R. Matzutt, J. Hiller, M. Henze, J. H. Ziegeldorf, O. Hohlfeld, and K. Wehrle, "A quantitative analysis of the impact of arbitrary blockchain content on bitcoin," in *Financial Cryptography and Data Security, 22nd International Conference, FC 2018*, Nieuwpoort, Curaçao: Springer Berlin Heidelberg, Feb. 28, 2018, pp. 420–438. DOI: `10.1007/978-3-662-58387-6_23`.

[4] Bitcoin Wiki, "Weaknesses," 2011. `https://en.bitcoin.it/w/index.php?title=Weaknesses&oldid=14350` (accessed Jul. 3, 2020).

[5] A. Seeg, "Hardening bitcoin against off-topic data inclusion attacks," Master thesis, Universität Würzburg, 2018.

[6] R. Matzutt, M. Henze, J. H. Ziegeldorf, J. Hiller, and K. Wehrle, "Thwarting unwanted blockchain content insertion," in *2018 IEEE International Conference on Cloud Engineering (IC2E)*, Orlando, FL, Apr. 2018, pp. 364–370. DOI: `10.1109/IC2E.2018.00070`.

[7] G. Ateniese, B. Magri, D. Venturi, and E. Andrade, "Redactable blockchain – or – rewriting history in bitcoin and friends," in *2017 IEEE European Symposium on Security and Privacy (EuroS&P)*, Paris, France, Apr. 2017, pp. 111–126. DOI: `10.1109/EuroSP.2017.37`.

[8] I. Puddu, A. Dmitrienko, and S. Capkun, "μchain: How to Forget without Hard Forks," 2017. Cryptology ePrint Archive: `2017/106`. [Online]. Available: `https://eprint.iacr.org/2017/106`.

[9] J. D. Bruce, "The mini-blockchain scheme," Cryptonite.info, 2017. `https://cryptonite.info/files/mbc-scheme-rev3.pdf` (accessed Jul. 3, 2020).

[10] Bitcoin Wiki, "Prohibited changes," 2016. `https://en.bitcoin.it/w/index.php?title=Prohibited_changes&oldid=60827` (accessed Jul. 3, 2020).

[11] A. Sward, I. Vecna, and F. Stonedahl, "Data insertion in bitcoin's blockchain," *Ledger*, vol. 3, Apr. 3, 2018, ISSN: 2379-5980. DOI: `10.5195/ledger.2018.101`.

[12] Bitcoin Wiki, "Principles of bitcoin," 2017. `https://en.bitcoin.it/w/index.php?title=Principles_of_Bitcoin&oldid=63920` (accessed Jul. 3, 2020).

[13] J.-L. Pons, "VanitySearch," GitHub.com, 2019. `https://github.com/JeanLucPons/VanitySearch` (accessed Jul. 6, 2020).

[14] J. Steube, P. Schmidt, G. Gristina, J.-C. Delaunay, and J. Gosney, "Hashcat," 2020. `https://hashcat.net/hashcat/` (accessed Jul. 3, 2020).

[15] DaveF (pseudo.), "Re: VanitySearch (Yet another address prefix finder)," response #299, Bitcointalk.org, Apr. 29, 2019. `https://bitcointalk.org/index.php?topic=5112311.msg50823897#msg50823897` (accessed Jul. 4, 2020).

[16] J. M. Gosney, "8x Nvidia GTX 1080 Ti Hashcat Benchmarks," GitHub.com, Apr. 29, 2020. `https://gist.github.com/epixoip/ace60d09981be09544fdd35005051505` (accessed Jul. 4, 2020).

[17] M. Celik, "2x 2080 Ti Hashcat Benchmarks," GitHub.com, Oct. 24, 2018. `https://gist.github.com/codeandsec/1c1f2c7bd81abba6fa9736b061944675` (accessed Jul. 4, 2020).

[18] D. J. Bernstein and T. Lange, Eds., "Measurements of hash functions, indexed by machine," ECRYPT Benchmarking of Cryptographic Systems, Jul. 3, 2020. `https://bench.cr.yp.to/results-hash.html` (accessed Jul. 6, 2020).

[19] C. Percival, "Scrypt: A new key derivation function," presented at the BSDCan, May 9, 2009. [Online]. Available: `https://www.bsdcan.org/2009/schedule/attachments/87_scrypt.pdf` (accessed Jul. 7, 2020).