A Predictive Model for the Zcash Verification Delay

Fabian Stiehle
Technische Universität Berlin
Berlin, Germany
stiehle@campus.tu-berlin.de

ABSTRACT

Zcash enables privacy preserving transactions by employing Zero-Knowledge (ZK) proofs. As in Bitcoin, reaching consensus involves propagation of new blocks across the peer-to-peer network. This propagation delay limits scalability and performance. Additionally, recent work suggests that propagation delay also has a direct and measurable impact on security guarantees. However, a vital part of the overall propagation is the verification, as each node in the network will first verify and store the new block locally before forwarding it. Thus, understanding and modeling block verification can greatly help investigate and improve the overall propagation delay. And yet, only very simplistic models to simulate the verification delay exist. In this work, we present a new model for predicting the verification delay of a Zcash block. We compare it with the currently used model in literature and show that, under same conditions, it performs magnitudes better, with a mean absolute error as low as 3ms, compared to 11ms, and a R^2 score as high as 0.91, compared to 0.25 of the currently used model.

1 INTRODUCTION

Bitcoin provides a digital currency through a distributed ledger that is in practice: immutable, non-repudiable, highly available and fully transparent. While some use cases benefit from these transparency guarantees, they also raise serious privacy concerns. Zcash addresses this issue by employing Zero-Knowledge (ZK) proofs to allow confidential transactions. In both systems, reaching consensus across different blockchain nodes involves propagation of new blocks across the peer-to-peer network. The time delay caused by this is usually called the propagation delay [4]. While it is well known that the propagation delay limits scalability and performance factors such as inter-block time and overall throughput [15], recent work suggests that propagation delay also has a direct and measurable impact on security guarantees. A longer propagation delay may lead to a higher occurrence of stale blocks and blockchain forks [4], which in turn weakens the overall security guarantees of the blockchain system [6][13].

However, a vital part of the overall propagation delay is the *verification delay*. During propagation, a peer will receive a new block, verify it against the consensus rules, and on successful verification forward the block to its own peers (store-verify-forward). Thus, the propagation delay not only includes the delay necessary to transmit the block over the network to the next peer (transmission delay), but also incurs time necessary to verify the block locally (verification delay).

Thus, understanding and modeling block verification can help investigate and improve the overall propagation delay, and therefore, current limitations of scalability related properties, but also understand and improve security guarantees of the blockchain system overall. And yet, only very simplistic models, based purely on block size, exist. This work aims to give a better understanding of the Zcash verification process and present a predictive model that allows for better analysis of the entire Zcash system. To achieve this we: (1) Give an overview of the Zcash verification process based on code analysis and the protocol specification, (2) Present a model for predicting the verification delay of a Zcash block that performs magnitudes better with a mean absolute error as low as 3ms, compared to 11ms, and a R^2 score as high as 0.91, compared to 0.25 of the currently used model in literature; and, (3) Perform a benchmark of the Zcash client and its employed cryptographic primitives to inform and evaluate our model.

The remainder of the paper is structured as follows: First, we give a general introduction to Zcash. We then give an informal primer to ZK proofs and ZK succinct non-interactive argument of knowledge (ZK SNARK). We then introduce the necessary Zcash specific terminology and then give a concrete description of the verification process focusing on cryptographic operations. Next, we discuss a naive model as currently used in literature and present our improved model. Finally, we evaluate and compare both models and discuss their qualities and limitations.

2 ZCASH

The foundation of our analysis forms a detailed analysis of the Zcash verification process. We will first give a general overview of the Zcash system, we will then informally introduce ZK and ZK SNARKs. We then go into the Zcash specific terminology and outline the verification process and the employed cryptographic primitives. Our description is based on the protocol specification, the Zcash documentation, and code analysis of the Zcash client version 3.0 [10].[1]

Zcash is a Bitcoin code fork that enables privacy-preserving transactions through *shielded transactions*. A shielded transaction keeps values, with currency symbol *ZEC*, and sender and receiver confidential. To still verify the transaction's validity Zcash employs ZK proofs (see Section 2.1). Zcash still supports *transparent transactions*, which are essentially native Bitcoin transactions. Transparent transactions give no confidentiality guarantees and rely on the standard UTXO model (c.f. [11][14]). Outputs of transparent transactions are held by *transparent addresses*. Outputs of shielded transactions are held by *shielded addresses*. It is possible to transfer ZEC between both address types.

Zcash's major network upgrades are usually denoted by name. The latest major upgrade was denoted *Heartwood* and is active

[1] "Zcash 3.0" https://github.com/zcash/zcash, "Zcash Documentation" https://zcash.readthedocs.io/en/latest/ [Accessed: 09-Sep-2020]

since July 2020. Most noteworthy is *Sprout*, which was the release version, and *Sapling*. Sapling introduced changes to how shielded transactions are encoded and verified (c.f. Section 2.3).

2.1 ZK and ZK SNARKs

Given two parties: the *verifier* and the *prover*, ZK proofs enable the prover to convince the verifier of the validity of a statement without revealing anything besides the truth of its validity [8]. Goldreich et. al. showed that a ZK proof can be constructed from any NP problem if combined with a commitment scheme. We can apply the notion of a witness directly to ZK proofs. A prover can prove membership of witness $w \in L$, where L denotes an NP language, by sending w to the verifier, which can verify $w \in L$ in polynomial time [7]. ZK proofs can be summarised to have three core properties [9]:

- Completeness: Given a witness, the prover can convince the verifier of the validity of a statement.
- Soundness: A malicious prover cannot deceive the verifier in accepting a false statement.
- Zero-Knowledge: The proof does not reveal the witness or anything else besides the validity of the statement.

Blum et al. defined non-interactive ZK proofs, where the (sometimes extensive) interaction between prover and verifier can be replaced by a common reference string (called public parameters in Zcash) [1]. Furthermore, Brassard et. al. introduced ZK arguments, which relax the soundness property of ZK proofs to only capture "computational feasible" ways of deceiving the verifier (computational soundness), which makes ZK arguments more feasible in practice [2]. Putting these definitions together, we get Zero-Knowledge succinct non-interactive argument of knowledge (ZK SNARK), where succinct limits the witness size to allow for more efficient verification by the verifier.

In practice, the statement to be proven is often encoded as a program, which is then compiled to a quadratic arithmetic program [5][12].[2] Zcash Sprout was based on a libsnark fork following the protocol specification of the Pinocchio protocol [12], with the Sapling upgrade, Zcash switched to the rust library bellman[3] based on Groth [9]. Groth introduced ZK SNARKs with faster verification times and smaller proof sizes.

2.2 Terminology

A block may contain a sequence of transparent and shielded transactions. Both transaction types can contain transparent inputs and outputs. An input references, and thereby spends, an output in a previous transaction. Transactions contain a transparent value pool. Inputs add ZEC to the balance. An output withdraws and defines a transparent address to which the balance is transferred to. The balance cannot be negative. The positive balance is the transaction fee. The transaction fee, as well as the mining reward for coin generation, are collected by the miner through outputs in the *coinbase* transaction, it includes a coinbase input, that does not reference a previous output. A shielded transaction is either a Sprout or a Sapling shielded transaction. A Sprout shielded transaction contains *JoinSplit descriptions*. A Sapling shielded transaction contains *Spend descriptions* with shielded spends and *Output descriptions* with shielded outputs. Descriptions are separately balanced and on positive balance output ZEC to the transparent value pool. If a shielded transaction contains transparent outputs, they are called *Deshielding*. If a shielded transaction contains transparent inputs they are called *Shielding*.

Each Zcash user generates a *spending key*, from which further keys are derived, such as a payment address. For simplicity, we will mostly generalize and simply refer to spending key for each key of the set of all possible keys that can be derived from it.

A *note* can be a Sprout note or a Sapling note. A note represents a value that is spendable by a recipient. The recipient is defined by the public spending key. Notes are encrypted for the recipient only to provide confidentiality. For a given note, a commitment and *nullifier* are computed. A note is spendable if the commitment is publicly revealed on the blockchain but the nullifier is not. The nullifier must be revealed if the note is spent. The nullifier can only be calculated correctly if the private spending key to the note is known. Note, commitment, and nullifier cannot be linked without knowledge of the plaintext note.

A note is spent by revealing the nullifier and proving in ZK that it was calculated correctly from the correct private spending key. Therefore, the privacy guarantees of Zcash stem from the fact that a ZK proof verifies that a proper note commitment and nullifier was revealed, but not which. The nullifier proves spending was authorized as the correct private spending key was used. When the nullifier is revealed, it is checked for uniqueness among all revealed nullifiers to prevent double-spending. Figure 1 gives an overview of the used terminology in the context of the verification process.

2.3 Verification

This Section gives a general overview of how a block is verified in Zcash. We will omit trivialities such as checks for well-formedness and certain details for simplicity, in general we will focus on computational intense operations.

The verification of the block includes (1) verifying the block header, which includes verifying the Proof-of-Work hash, (2) verifying all transactions in the block, and, (3) resolving blockchain forks and storing the block to disk or memory.

First, a block's header is verified. This amounts to verifying the Proof-of-Work hash. Zcash uses Equihash, the hash is computed over the 1487 Byte large block header and must satisfy the difficulty filter specified in the consensus rules [10].

Second, the transactions are verified. Transaction verification differs based on transaction type. For simplicity, we will only consider Pay-To-Public-Key-Hash scripts for transparent transactions.

2.3.1 Transparent transactions. Transparent transactions follow the same verification process as in Bitcoin. An input script combined with an output script must evaluate to true. For Pay-To-Public-Key-Hash this amounts to proving the knowledge of the private key part sk corresponding to the transparent address (public key) hash pk_{hash} specified in the output script it spends. To bind this authorization to the transaction, and prevent transaction malleability, the

[2] Also, See: "libsnark: a C++ library for zkSNARK proofs" https://github.com/scipr-lab/libsnark [Accessed: 09-Sep-2020]
[3] "bellman" https://github.com/zkcrypto/bellman [Accessed: 09-Sep-2020]

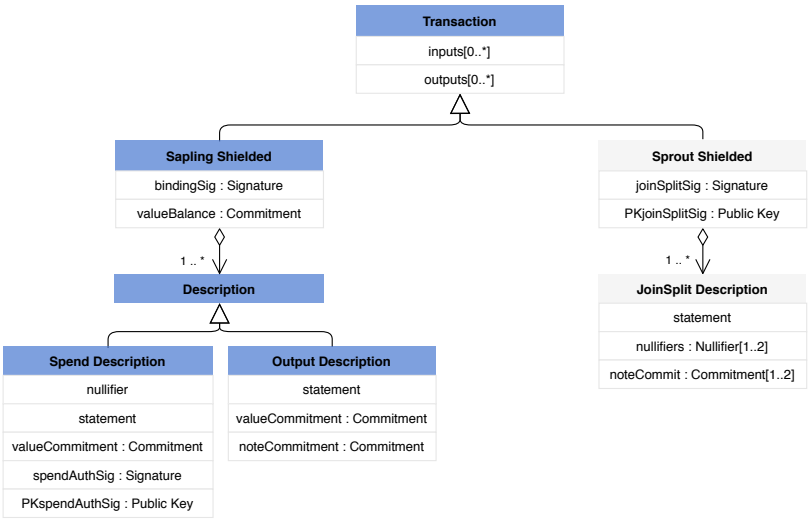


Figure 1: A simplified data model for shielded transactions. The legacy Sprout transaction is marked in grey (light). [Notation: UML]

input includes an ECDSA signature *sig* generated by signing the hash of the transaction with *sk*. The input also includes the public key part *pk* of the signature *sig*. To verify, the script proceeds as follows:

(1) *OP_EQUALVERIFY*: Verify that *pk* in the input script hashes to pk_{hash} in the output script.
(2) *OP_CHECKSIG*: Hash the transaction and verify that, using *pk*, the provided signature *sig* produces the correct transaction hash.

2.3.2 Shielded transactions. Shielded transactions cannot make use of the script system. Verification differs for JoinSplit descriptions or Spend and Output descriptions. Each transaction may contain a sequence of JoinSplit descriptions or Spend and Output descriptions. We will generalize verification for shielded transactions by explaining three distinct properties that have to be ensured:

(1) Authorization: A note can only be spent on knowledge of the private spending key.
(2) Non-Malleability: The Zcash ZK proving systems are malleable [10]. Additional measures must be employed to ensure non-malleability.
(3) Balance: A transaction's inputs/spends and outputs must balance.

We will outline how each description type achieves these properties and then how verification is performed.

JoinSplit descriptions. JoinSplit descriptions spend $[1, 2]$ notes and a value from the transparent value pool and create $[1, 2]$ new notes and add to the transparent value pool. Each description contains a JoinSplit statement which contains the ZK SNARK proof.

- Authorization: For each of the notes, a nullifier is revealed. The JoinSplit statement proves that each nullifier was computed from the correct note and spending key.
- Non-Malleability: The transaction includes a signature over the transaction hash $joinSplitSig$ signed by the private key part of a key pair that is generated for each transaction. The public key part $pk_{joinSplitSig}$ is included in the transaction to enable verification. The private key part is part of the input of the ZK proof in JoinSplit statement.
- Balance: For JoinSplits, balance is proven through the ZK proof contained in the JoinSplit statement.

To verify JoinSplit descriptions contained in a transaction the following steps are performed (in this order):

(1) A hash over the transaction is created.
(2) Verify that, using $pk_{joinSplitSig}$, $joinSplitSig$ produces the correct transaction hash.
(3) For each JoinSplit description: the JoinSplit statements (ZK proofs) are verified.

Spend and Output descriptions. Spend and Outputs descriptions spend one note or create one note respectively. Both descriptions include a Pedersen commitment to the value of the input or output note respectively. Pedersen commitments are homomorphic. Each description contains a Spend or Output statement respectively, which contains the ZK SNARK proof.

- Authorization: Each Spend description includes a RedJubjub signature over the transaction hash $spendAuthSig$ demonstrating knowledge of the private spending key of the note it spends. The signature $spendAuthSig$ and the corresponding public key part $pk_{spendAuthSig}$ is included in the Spend description. Once a note is spent a nullifier is revealed. The Spend statement proves that the nullifier was computed from the correct note, value commitment, and spending key. Each Output description produces a new note and commitment. The Output statement proves that the commitment was computed from the correct note and value commitment.

- Non-Malleability: *spendAuthSig* binds all Spend descriptions to the transaction. The balance signature, described momentarily, binds all Output statements to the transaction hash.
- Balance: As Spend and Output statements include separate proofs, the balance is proven outside the ZK proof. The homomorphic property of Pedersen commitments is used, which allows for addition and subtraction, to ensure balance of all value commitments contained in the transaction. The target balance *valueBalance* is explicitly encoded as commitment in the transaction. A RedJubub signature over the transaction hash *bindingSig* is included in the transaction. The signature can be verified by deriving the verification key from the Spend and Output value commitments and the balance *valueBalance* commitment.

To verify Spend and Output descriptions contained in a transaction the following steps are performed (in this order):

(1) A hash over the transaction is created.
(2) For each Spend description: Verify that, using $pk_{spendAuthSig}$, the *spendAuthSig* signature produces the correct transaction hash. Verify the Spend statement.
(3) For each Output description: Verify the output statement.
(4) Derive the verification key from the Spend and Output value commitments and the balance *valueBalance* commitment. Verify that, using the verification key, the *bindingSig* signature produces the correct transaction hash.

Note. : Since the Overwinter upgrade Zcash reuses transaction hashes so that they don't have to be recreated for multiple verification steps across one transaction.[4] During the verification, state is flushed to disk if necessary based on the available memory size.

3 BENCHMARK SETUP

To inform and evaluate our model we performed two benchmark runs with different setups. One on a HDD disk setup and more limited RAM, to show the influence of Disk I/O and RAM on the verification delay. One with a SSD disk setup and more available RAM, to simulate realistic client hardware used (c.f. Table 1). On both systems we performed a black-box like benchmark capturing the complete block verification process. This amounts to the time the client processes the `ProcessNewBlock` function during synchronization with the network. To allow for fine grained analysis, we separately performed benchmarks of singular computational steps, such as verifying singular inputs or spends, performed during verification (As identified in Section 2.3).

Table 1: Benchmark setup.

Setup	HDD system	SSD system
Host Spec	Dual-Core Intel Core i5 2,6GHz	Six-Core AMD Ryzen 5 2600X 3,60GHz
RAM, SWAP	5GB, 2GB	12GB, 2GB

[4]See *ZIP: 143, Transaction Signature Validation for Overwinter* https://zips.z.cash/zip-0143 and *ZIP: 243, Transaction Signature Validation for Sapling* https://zips.z.cash/zip-0243 [Both Accessed: 02-Sep-2020].

To improve repeatability and automation of our benchmark we compiled and run the Zcash 3.0 client inside a Docker container. The code to our benchmark and evaluation is publicly available.[5] All our runs were executed on systems with specification as summarized in Table 1. We perform our benchmark from block height 525,000 on-wards, which is the latest Zcash checkpoint. Our evaluation and graphs are based on different samples from 220,000 blocks from block height 695,581 up to height 915,578.

4 MODEL

Already, from our description given in Section 2.3, we can identify relevant variables for the block verification delay. While verifying the block header is a constant operation for all blocks, the verification steps differ greatly based on the included transaction types, and, more precisely, on the amount of inputs, JoinSplit descriptions, Spend descriptions and Output descriptions.

Now consider a model d_b predicting the verification delay $\hat{t}$ in the form of

$$\hat{t} = d_b(s_b) = k + \beta s_b$$

similar as in [3]. Where s_b is the block size and β a constant coefficient. That is, d_b assumes a linear relationship between block size s_b and verification delay. In Figure 2 we illustrate the behaviour of the model versus the real captured verification delay during our benchmark on our HDD and SSD system respectively. For comparison, we parametrisize β, k by linear regression and find $\beta = 2.232\mu s, k = 28445.511\mu s$ on our HDD and $\beta = 0.910\mu s, k = 9647.37\mu s$ on our SSD system. Additionally, we use $\beta = 0.3796\mu s$ and $k = 0\mu s$ as in [3], where β was taken from [6].

We can clearly see that d_b is not able to explain the high variance. The visualisation suggests that the block size is only a weak indicator for the verification delay. To further illustrate this we show the influence of each Kibibyte of a block on the verification delay in Figure 3a. It is important to notice the distribution of block sizes of our sample, while after around 40KiB the added delay seems to become constant, most of our samples exhibit a different behaviour with high variation that cannot be explained by the block size. An interesting question is also, if the verification of singular inputs, JoinSplit descriptions, Spend descriptions and Output descriptions is affected by the block size. Figure 3b, Figure 3c, Figure 3d, Figure 3e, shows the influence of a Kibibyte on the median verification delay of one singular input, JoinSplit description, Spend description and Output description, by dividing the median delay of the operation for a given block size by the block size. While we have an initial overhead the impact of each Kibibyte on the verification delay decreases and is nearing $0\mu s$. Notice the distribution and behaviour as compared to Figure 3a. We see a clear pattern of decreasing verification delay per Kibibyte and not much variance. An exception is Figure 3b, where we see an increased variance for smaller blocks. We assume that this is due to different scripts utilized in transparent inputs.

From our analysis in Section 2.3, one could conclude that the verification steps performed are in an at least linear relationship with the block size, as most steps include hashing the transaction. If each input, for example, would perform a hash operation, verifying inputs would incur $O(n^2)$, as the hash operation is dependent on transaction size, which is in turn, again, partly dependent on

[5]https://github.com/fstiehle/zcash-benchmark

Figure 2: d_b and the verification delay of 200,000 observed blocks (blue) during our benchmark on the HDD and SSD system. We parametrisized d_b by linear regression (green) using a sample of 20,000 blocks and find $\beta = 2.232\mu s, k = 28445.511\mu s$ on our HDD and $\beta = 0.910\mu s, k = 9647.37\mu s$ on our SSD system. Additionally, we show $\beta = 0.3796\mu s, k = 0\mu s$ (pink). The plot is zoomed in to better visualize the behaviour.

the amount of inputs. However, as we've also noted in Section 2.3, hashes are reused across verification steps. This is in line with our observations in Figure 3b, Figure 3c, Figure 3d, Figure 3e, where we see an initial overhead, followed by a decreasing impact of size on verification delay and eventually constant behaviour near $0\mu s$. We can therefore conclude, that the block size does not impact singular operations, beyond an initial overhead.

We now present an improved model d_v based on inputs, spends, shielded outputs and JoinSplits.

$$\hat{t} = d_v(n_v, n_s, n_o, n_j) = k + \beta_v n_v + \beta_s n_s + \beta_o n_o + \beta_j n_j$$

Where n_v is the number of inputs, n_s is the number of spends, n_o the number of shielded outputs and n_j the number of JoinSplits. With intercept k we can simulate the overhead of verifying the block header, hashing transactions and possible I/O operations. $\beta_v, \beta_s, \beta_o, \beta_j$ are constant coefficients used as weights.

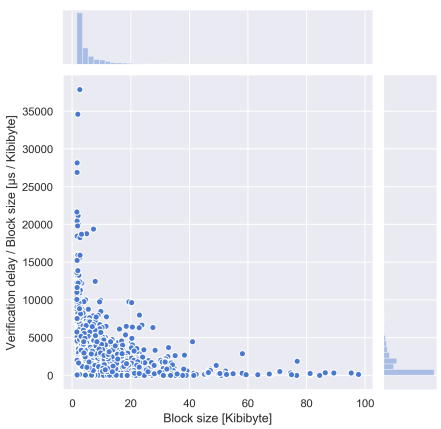
In case of Zcash Sapling, d_v can be, in most cases, simplified by $n_j = 0$. Coefficients for d_b and d_v can be found by performing a simple benchmark on a local client. We can then parametrizise the model by using the mean or median time (depending on the quality of the sample distribution) of the operations. For optimal performance coefficients can be discovered by linear regression.

5 EVALUATION

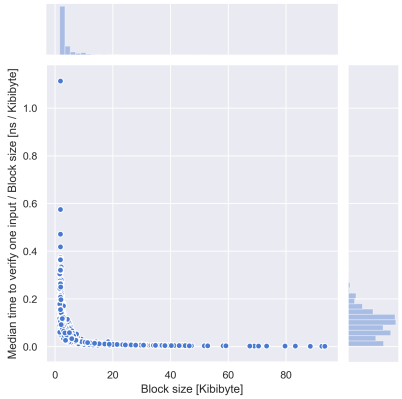
We evaluate both models by calculating the mean absolute error (MAE) and the R^2 score. We evaluate both models by predicting 10,000 and 200,000 blocks. We use a sample of 15,000 and 220,000 blocks and use 5,000 and 20,000 blocks for parametrization. To capture and compare the optimal performance of both models, we use linear regression to discover coefficients.

We also show our evaluation for 15,000 blocks with only blocks that contain transparent transactions exclusively for a SSD system. Here we also use 5,000 blocks for parametrization and predict 10,000 blocks. Table 2 summarises our results. In Figure 5 we compare the estimated verification delay with the actual captured verification delay and the R^2 score of both models for our SSD benchmark.

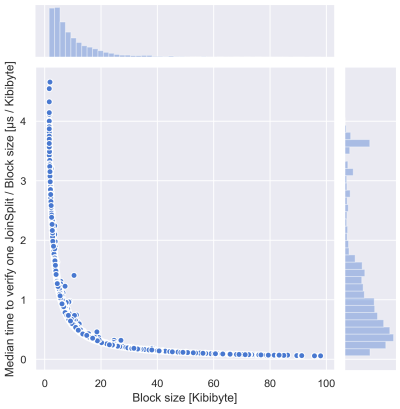
d_v is still a simple enough model that can be easily parametrisized by using simple benchmarks of the cryptographic primitives employed in Zcash. We suggest parametrization as given in Table 2. Despite its simplicity, and the low sample size used for parametrization, our model produces significantly better results than the current model used in literature. In Figure 4 we show our model in comparison with d_b and the real verification delay observed in relation with the block size for 10,000 predicted blocks for both our HDD and SSD benchmarks. This highlights, once again, the non-linearity of the block size and verification delay. We draw a line between each data point per block size to better visualize the general trend. It shows that, while for our HDD system our model continuously underestimates the impact of size on the verification delay, it still performs better than d_b. While it is interesting to show the influence of slow disk I/O and limited RAM on the verification delay, we consider our SSD benchmark as the more realistic system. Here, we can see how d_v aligns very closely with the general trend of the Benchmark, even for some of the extreme outliers with block size larger than 100 Kibibyte it is able to get near perfect hits. In Figure 6 we given a concrete example by showing a 200 block excerpt from our 10,000 blocks prediction for d_b and d_v and the real measured verification time. Here, one can similarly see that our improved model is in general much closer to the real measured time and is even able to closely predict outliers.



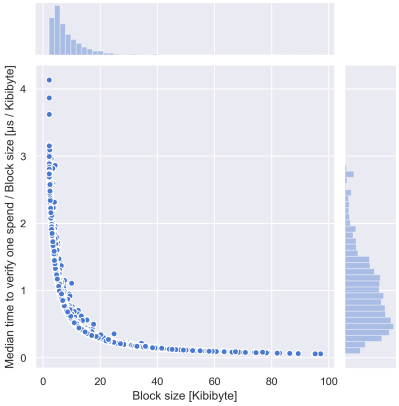
(a) Verification delay / Block size



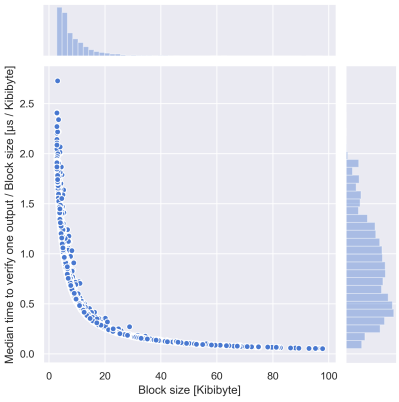
(b) Median time to verify one input / Block size



(c) Median time to verify one JoinSplit / Block size



(d) Median time to verify one spend / Block size



(e) Median time to verify one output / Block size

Figure 3: Influence of block size on different metrics as observed with 5000 blocks on our SSD system.

Table 2: Model evaluation. We give error and R^2 compared to benchmarks performed on our HDD and SSD system. We evaluate both models by predicting 10,000 and 200,000 blocks. We use a total sample of 15,000 and 250,000 blocks and use 5,000 and 20,000 blocks for parametrization.

System	Model	Parameter [μs]	MAE [μs]	MAE% [%]	R^2 score
Train: 5,000, Predict: 10,000 blocks					
HDD system	d_b	$\beta = 4.345, k = 8784.760$	31195.803	91.080	0.091
	d_v	$\beta_v = 246.312, \beta_s = 39760.496, \beta_o = 9862.146, \beta_j = 10999.119, k = 13209.042$	22051.201	64.381	0.165
SSD system	d_b	$\beta = 1.717, k = 3584.715$	10594.982	78.336	0.252
	d_v	$\beta_v = 61.411, \beta_s = 16912.591, \beta_o = 5726.675, \beta_j = 5359.094, k = 4468.949$	3109.079	22.987	0.907
Train: 20,000, Predict: 200,000 blocks					
HDD system	d_b	$\beta = 2.232, k = 28445.511$	42462.803	97.924	0.119
	d_v	$\beta_v = 139.676, \beta_s = 25227.674, \beta_o = 12607.155, \beta_j = 10784.519, k = 21760.549$	30031.503	69.256	0.403
SSD system	d_b	$\beta = 0.910, k = 9647.374$	15126.399	92.067	0.137
	d_v	$\beta_v = 40.339, \beta_s = 12067.658, \beta_o = 5782.956, \beta_j = 5349.659, k = 5928.899$	7112.943	43.293	0.718
Transparent only: Train: 5,000, Predict: 10,000 blocks					
SSD system	d_b	$\beta = 3.448, k = 3223.436$	4181.409	81.233	0.008
	d_v	$\beta_v = 113.271, k = 3563.299$	4181.722	81.239	-0.006

In fact, for our SSD benchmark predicting 10,000 blocks, our model is magnitudes more accurate than d_b. Our model was able to predict the verification delay with a mean absolute error of 3ms, compared to 11ms of d_b. A more direct comparison of d_v and d_b can be seen in Figure 5, where we show the R^2 score of 0.9 for d_v and 0.25 for d_b. The low score of d_b suggests that d_b makes only slightly better predictions than the median. Figure 5 implies that both models have difficulties explaining the variations for blocks with a small verification delay. Here, d_v still performs much better

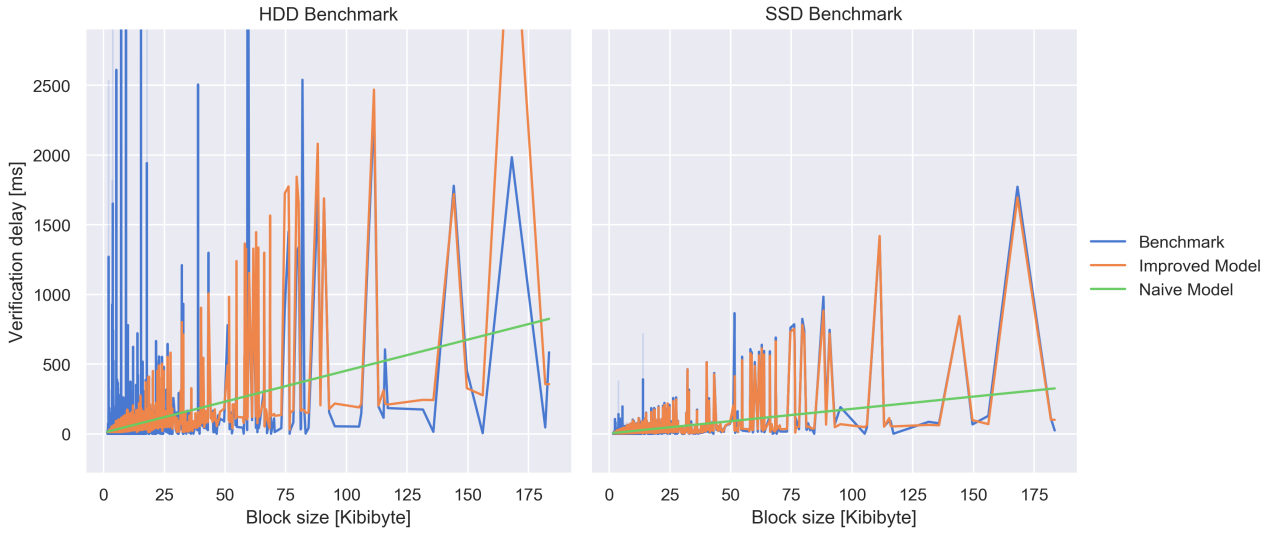
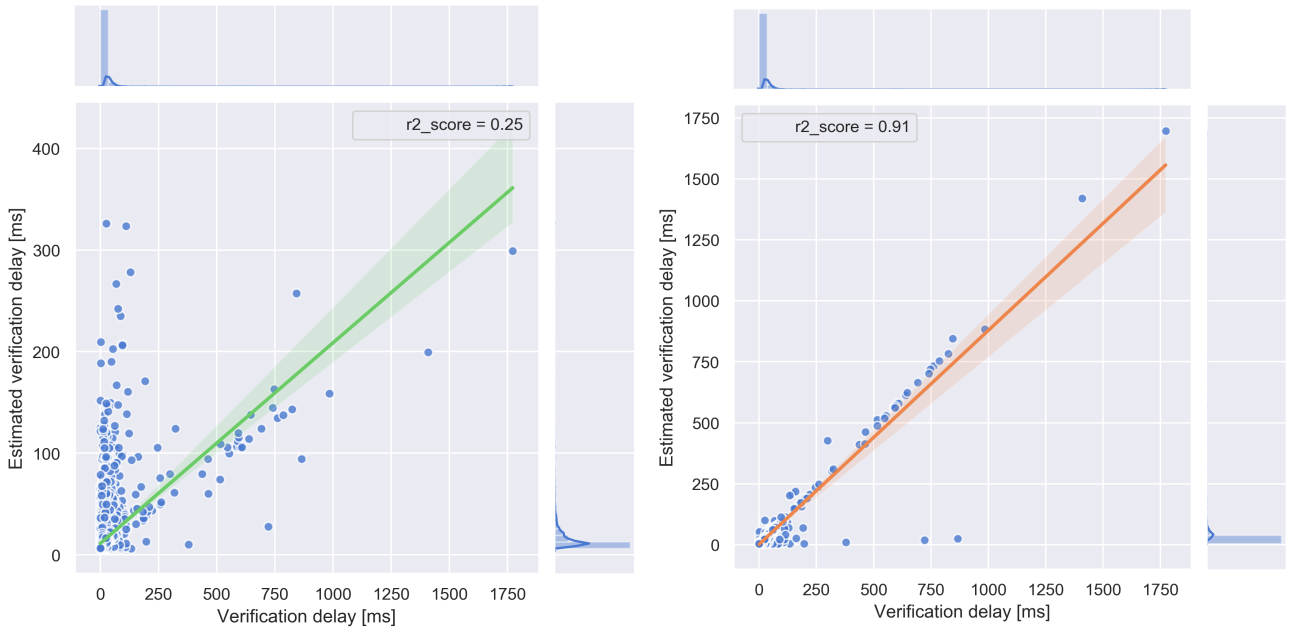


Figure 4: The naive model d_b (green) compared with our improved model d_v (orange) and the verification delay of 10,000 observed blocks obtained during our benchmark (blue). We used a sample of 5,000 blocks to parametrisize both models. Note that the plot is zoomed in to better visualize the behaviour.



(a) **(b)**

Figure 5: R^2 score obtained by comparing estimates made by d_b (a) and d_v (b) with the verification delay obtained by our benchmark (blue) performed on our SSD system at a 95% confidence level. The model were parametrisized as given in Table 2.

but we can also observe some unexplained variations. A small verification delay often correlates with many transparent transactions and only a few (if any) shielded. Both models under-perform with $R^2 = 0$ for blocks only containing transparent transactions. Both models are not able to explain the variance in the verification delay of singular inputs. We assume that our decision to only consider

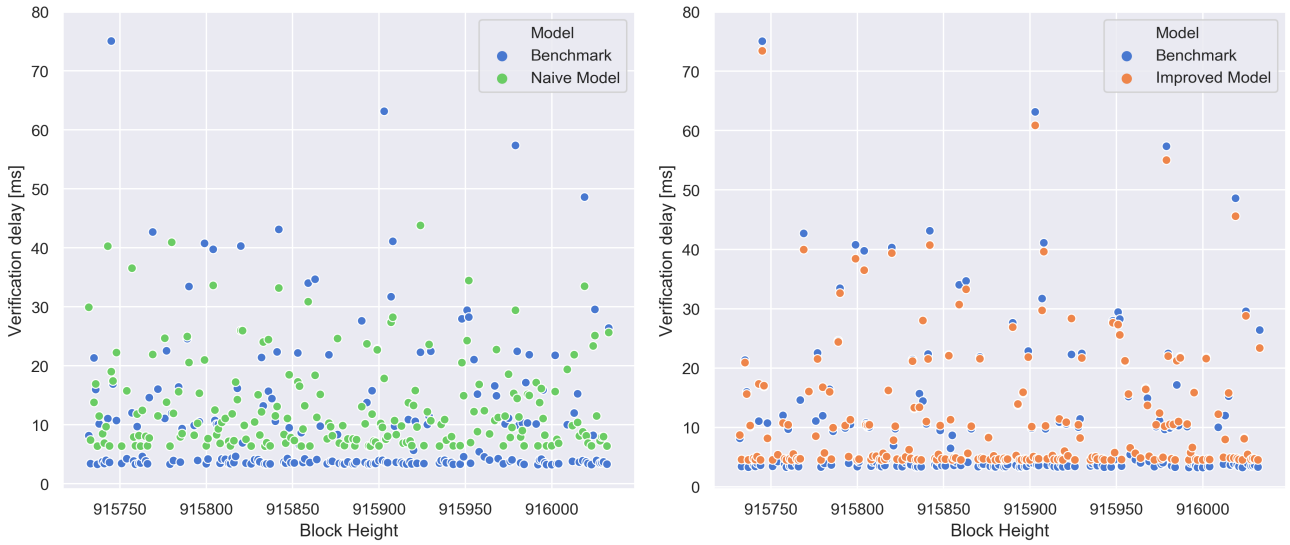


Figure 6: 200 block excerpt from our 10,000 blocks prediction as predicted by d_b (a) and d_v (b) compared with the real verification delay (blue) on our SSD system.

Pay-To-Public-Key-Hash scripts and abstract away from the scripting system led to this poor result. We leave the improvement of our model for transparent transactions as future work.

6 RELATED WORK

The Zerocash paper presents a simulation for a potential Zerocash blockchain system. The simulation models the verification time for ZK transactions as a constant of 10ms [13]. Zcash, however, has evolved significantly from the Zerocash specification. Decker et al. investigate the propagation delay in an early version of Bitcoin. They argue for a strong correlation between block size and overall propagation delay. They show a constant behaviour between size and propagation delay after an initial overhead. The study, however, was conducted in 2013 on an early Bitcoin version with a block size limit of 500kB. Nevertheless, they argue that the verification delay is a major factor for the overall propagation delay. They show that an increased propagation delay increases the likelihood of blockchain forks and, thus, suggest an improved verification delay as a possible countermeasure [4]. In contrast, Gervais et al. simulation of Bitcoin suggests a linear relationship between block size and propagation delay up to a point, followed by an exponential relationship [6]. Both works capture the verification delay only implicitly through the overall propagation delay. Daniel et al. present an inference model for connections between nodes in the Zcash network, they use a d_b like model (with $k = 0\mu s$), which we have used in this work as a comparative model, to estimate the verification delay [3].

7 CONCLUSION

We showed how Zcash verifies transactions on a conceptual level. We presented an improved model that performs significantly better than the current model used in literature with a mean absolute error as low as 3ms, compared to 11ms, and a R^2 score as high as 0.91, compared to 0.25 of the currently used model. We've shown that the block size is not a good indicator for the verification delay. Our improved model is still simple to parametrizise and performs especially well on modern hardware. However, our model was not able to explain the high variance in the verification delay of transparent transactions. This limits the model's generality and applicability for other UTXO based blockchains such as Bitcoin. Future work invested in improved prediction of the delay caused by transparent transactions promises to improve the model's generality and overall performance.

REFERENCES

[1] Manuel Blum, Paul Feldman, and Silvio Micali. 1988. Non-interactive zero-knowledge and its applications. In *Proceedings of the twentieth annual ACM symposium on Theory of computing - STOC '88*. ACM Press, Chicago, Illinois, United States, 103–112. https://doi.org/10.1145/62212.62222

[2] Gilles Brassard, David Chaum, and Claude Crépeau. 1988. Minimum disclosure proofs of knowledge. *J. Comput. System Sci.* 37, 2 (Oct. 1988), 156–189. https://doi.org/10.1016/0022-0000(88)90005-0

[3] Erik Daniel, Elias Rohrer, and Florian Tschorsch. 2019. Map-Z: Exposing the Zcash Network in Times of Transition. *arXiv:1907.09755 [cs]* (July 2019). http://arxiv.org/abs/1907.09755 arXiv: 1907.09755.

[4] Christian Decker and Roger Wattenhofer. 2013. Information propagation in the Bitcoin network. In *IEEE P2P 2013 Proceedings*. IEEE, Trento, Italy, 1–10. https://doi.org/10.1109/P2P.2013.6688704

[5] Rosario Gennaro, Craig Gentry, Bryan Parno, and Mariana Raykova. 2013. Quadratic Span Programs and Succinct NIZKs without PCPs. In *Advances in Cryptology – EUROCRYPT 2013*, David Hutchison, Takeo Kanade, Josef Kittler, Jon M. Kleinberg, Friedemann Mattern, John C. Mitchell, Moni Naor, Oscar Nierstrasz, C. Pandu Rangan, Bernhard Steffen, Madhu Sudan, Demetri Terzopoulos, Doug Tygar, Moshe Y. Vardi, Gerhard Weikum, Thomas Johansson, and Phong Q. Nguyen (Eds.). Vol. 7881. Springer Berlin Heidelberg, Berlin, Heidelberg, 626–645. https://doi.org/10.1007/978-3-642-38348-9_37 Series Title: Lecture Notes in Computer Science.

[6] Arthur Gervais, Ghassan O. Karame, Karl Wüst, Vasileios Glykantzis, Hubert Ritzdorf, and Srdjan Capkun. 2016. On the Security and Performance of Proof of Work Blockchains. In *Proceedings of the 2016 ACM SIGSAC Conference on*

Computer and Communications Security. ACM, Vienna Austria, 3–16. https://doi.org/10.1145/2976749.2978341

[7] Oded Goldreich, Silvio Micali, and Avi Wigderson. 1986. Proofs that yield nothing but their validity and a methodology of cryptographic protocol design. In *27th Annual Symposium on Foundations of Computer Science (sfcs 1986)*. IEEE, Toronto, ON, Canada, 174–187. https://doi.org/10.1109/SFCS.1986.47

[8] S Goldwasser, S Micali, and C Rackoff. 1985. The knowledge complexity of interactive proof-systems. In *Proceedings of the seventeenth annual ACM symposium on Theory of computing - STOC '85*. ACM Press, Providence, Rhode Island, United States, 291–304. https://doi.org/10.1145/22145.22178

[9] Jens Groth. 2016. On the Size of Pairing-Based Non-interactive Arguments. In *Advances in Cryptology – EUROCRYPT 2016*, Marc Fischlin and Jean-Sébastien Coron (Eds.). Vol. 9666. Springer Berlin Heidelberg, Berlin, Heidelberg, 305–326. https://doi.org/10.1007/978-3-662-49896-5_11 Series Title: Lecture Notes in Computer Science.

[10] Daira Hopwood, Sean Bowe, Taylor Hornby, and Nathan Wilcox. 2020. Zcash Protocol Specification Version 2020.1.14. https://github.com/zcash/zips/blob/master/protocol/protocol.pdf

[11] Satoshi Nakamoto. 2008. Bitcoin: A Peer-to-Peer Electronic Cash System. https://bitcoin.org/bitcoin.pdf

[12] B. Parno, J. Howell, C. Gentry, and M. Raykova. 2013. Pinocchio: Nearly Practical Verifiable Computation. In *2013 IEEE Symposium on Security and Privacy*. IEEE, Berkeley, CA, 238–252. https://doi.org/10.1109/SP.2013.47

[13] Eli Ben Sasson, Alessandro Chiesa, Christina Garman, Matthew Green, Ian Miers, Eran Tromer, and Madars Virza. 2014. Zerocash: Decentralized Anonymous Payments from Bitcoin. In *2014 IEEE Symposium on Security and Privacy*. IEEE, San Jose, CA, 459–474. https://doi.org/10.1109/SP.2014.36

[14] Florian Tschorsch and Bjorn Scheuermann. 2016. Bitcoin and Beyond: A Technical Survey on Decentralized Digital Currencies. *IEEE Communications Surveys & Tutorials* 18, 3 (2016), 2084–2123. https://doi.org/10.1109/COMST.2016.2535718

[15] Xiwei Xu, Ingo M Weber, and Mark Staples. 2019. *Architecture for blockchain applications*. OCLC: 1091598117.