

Analysis of a Blockchain Protocol Based on LDPC Codes

Massimo Battaglioni^{1,*}, Paolo Santini¹, Giulia Rafaiani¹, Franco Chiaraluce¹ and Marco Baldi¹

¹Department of Information Engineering, Università Politecnica delle Marche, Ancona, 60131, Italy

Abstract

In a blockchain Data Availability Attack (DAA), a malicious node publishes a block header but withholds part of the block, which contains invalid transactions. Honest full nodes, which can download and store the full ledger, are aware that some data are not available but they have no formal way to prove it to light nodes, i.e., nodes that have limited resources and are not able to access the whole blockchain data. A common solution to counter these attacks exploits linear error correcting codes to encode the block content. A recent protocol, called SPAR, employs coded Merkle trees and low-density parity-check codes to counter DAAs. In this paper, we show that the protocol is less secure than claimed, owing to a redefinition of the adversarial success probability. As a consequence we show that, for some realistic choices of the parameters, the total amount of data downloaded by light nodes is larger than that obtainable with competing solutions.

Keywords

Blockchain, data availability attacks, LDPC codes, SPAR protocol

1. Introduction

A blockchain can be seen as an ordered list of blocks, each containing a set of transactions occurred among the participants of a peer-to-peer network. The recent discovery of Data Availability Attacks (DAAs) represents a new threat against blockchain security. Since the DAA introduction in [2], there has been a growing research interest in finding efficient countermeasures to this type of attacks, possibly leading to new blockchain models with improved scalability and security (e.g., [13, 8, 1, 5]).

In fact, scalability, which is related to the ability of supporting large transaction rates, represents one of the main issues in most existing blockchains [14]. The straightforward solution of increasing the block size raises a series of further concerns. In fact, the larger the block size the smaller the number of nodes able to download the full blockchain and, indeed, to participate in the network as *full nodes*, verifying the validity of new blocks and of every contained transaction. More peers would rather participate in the network as *light nodes*, which, due to

DLT 2022: 4th Distributed Ledger Technology Workshop, June 20, 2022, Rome, Italy

*Corresponding author.

✉ m.battaglioni@univpm.it (M. Battaglioni); p.santini@univpm.it (P. Santini); g.rafaiani@univpm.it (G. Rafaiani); f.chiaraluce@univpm.it (F. Chiaraluce); m.baldi@univpm.it (M. Baldi)

ORCID 0000-0002-8539-4007 (M. Battaglioni); 0000-0003-0631-3668 (P. Santini); 0000-0003-0029-5104 (G. Rafaiani); 0000-0001-6994-1448 (F. Chiaraluce); 0000-0002-8754-5526 (M. Baldi)



© 2022 Copyright for this paper by its authors. Use permitted under Creative Commons License Attribution 4.0 International (CC BY 4.0).

CEUR Workshop Proceedings (CEUR-WS.org)

their limited resources, store only a squeezed version of the blockchain [10] and, consequently, cannot autonomously verify the validity of transactions.

Light nodes aim at downloading as less data as possible. For instance, they may store only the block headers, which unambiguously identify the content of the blocks. However, in a setting with relatively few full nodes, collusion among them is more probable; this makes light nodes more susceptible to DAAs. In fact, the aim of a DAA is to make at least one light node accept a block which has not been fully disclosed to the network. This can happen if and only if honest full nodes are prevented from preparing *fraud proofs*, *i.e.*, demonstrations that the block is invalid [13, 3].

One of the most promising countermeasures to DAAs consists in encoding the blocks through some error correcting code. Encoding introduces redundancy and distributes the information of each transaction across all the codeword symbols, so that recovering a small portion of an encoded block may be enough to retrieve the entirety of its contents through decoding. This strategy, combined with a sampling process in which light nodes ask for fragments of an encoded block and then gossip them to the connected full nodes, ensures that malicious block producers are forced to reveal enough pieces of the invalid block [3]. An alternative to transactions encoding is to change the protocol in such a way that a group of light nodes can collaboratively (among themselves) and autonomously (from full nodes) verify blocks [5]. Another option is to decouple the consensus rules from the transaction validity rules [1, 4].

In a recent paper [13], Yu et al. proposed SPAR, a blockchain protocol which uses Low-Density Parity-Check (LDPC) codes to counter DAAs; LDPC codes for this specific application have then been studied in [8, 9]. SPAR comes as an improvement of the protocol in [3], the latter using two-dimensional Reed-Solomon codes, whose parameters have been optimized in [12]. The authors of SPAR study the protection against DAAs in case the adversary aims to prevent honest full nodes from successfully decoding the block, which is a strict requirement to settle a proper fraud proof. In [13], this situation is investigated assuming the adversary operates by withholding pieces of the encoded block; under a coding theory perspective, this gets modeled as a transmission over an erasure channel. They conclude that, unless the adversary is able to find stopping sets (which is an NP-hard problem [6]), SPAR guarantees that the success probability of a DAA is sufficiently small even when light nodes download a small amount of data besides the block header. As a consequence, SPAR claims improvements in all the relevant metrics [13, Table 1].

Our contribution In this paper we study the security of the SPAR protocol. Namely, we recompute the adversarial success probability with the consideration that deceiving *at least* a single light node is a success for the attacker, which is the same scenario considered in [3]. This yields a sampling cost that is much larger than the expected one, thus penalizing the light nodes participating in the network. Moreover, we show that the total amount of data that light nodes have to download (header size plus sampling cost) is actually larger than that of competing solutions such as [3].

Paper organization The paper is organized as follows. In Section 2 we describe the notation and some background. In Section 3 we introduce a general framework to study DAAs. In Section

4 we provide some numerical results. Finally, in Section 5 we draw some conclusions.

2. Notation and background

In this section we establish the notation used throughout the paper, and recall some background notions.

2.1. Mathematical notation

Given two integers a and b , we use $[a, b]$ to indicate the set of integers x such that $a \leq x \leq b$. For a set A , we use $|A|$ to denote its cardinality. We denote with \mathbb{F}_q the finite field with q elements. Given a vector \mathbf{v} , we use $\text{supp}(\mathbf{v})$ to denote its support, *i.e.*, the set containing the positions of its non-zero entries and $w_{\text{H}}(\mathbf{v})$ to denote its Hamming weight, that is, the size of its support. Given an integer l and a set A , A^l is the set of vectors of length l taking entries in A . Given a matrix \mathbf{M} , $m_{i,j}$ denotes its entry at row i and column j , $\mathbf{M}_{i,:}$ denotes the i -th row, and $\mathbf{M}_{:,j}$ denotes the j -th column. Given a set A , $\mathbf{M}_{:,A}$ (respectively, $\mathbf{M}_{A,:}$) represents the matrix formed by the columns (respectively, rows) of \mathbf{M} indexed by A .

We denote by Concat the string concatenation function and by $H_b(\cdot)$ the binary entropy function. Moreover, we denote by Hash a cryptographic hash function, with codomain D . Given some vector \mathbf{a} , we use $\mathcal{T}(\mathbf{a})$ to denote a generic hash tree structure constructed from \mathbf{a} and using Hash as underlying function. The root of the tree is denoted as $\mathcal{T}.\text{Root}(\mathbf{a})$; it generically takes values in D^l and is a one-way function. With analogous notation, by $\mathcal{T}.\text{Proof}(\mathbf{a}, i)$ we refer to the proof that the i -th entry of \mathbf{a} is a leaf in the base layer of the tree. Notice that, when Hash is properly chosen, then for any pair of strings $\mathbf{a} \neq \mathbf{a}'$ we have $\mathcal{T}.\text{Root}(\mathbf{a}) \neq \mathcal{T}.\text{Root}(\mathbf{a}')$ and, for any index i , $\mathcal{T}.\text{Proof}(\mathbf{a}, i) \neq \mathcal{T}.\text{Proof}(\mathbf{a}', i)$ with overwhelming probability (say, not lower than $1 - 2^{-256}$ for modern hash functions); therefore, for the sake of simplicity, in the following we assume the absence of root and proof collisions.

2.2. LDPC codes

LDPC codes are a family of linear codes characterized by parity-check matrices having a relatively small number of non-zero entries compared to the number of zeros. Namely, if an LDPC $\mathbf{H} \in \mathbb{F}_q^{r \times n}$ has full rank $r < n$ and row and column weight in the order of $\log(n)$ and $\log(r)$, respectively, then it defines an LDPC code with length n and dimension $k = n - r$, with code rate $R = \frac{k}{n}$. If all the rows (columns, respectively) of \mathbf{H} have the same Hamming weight, we denote it as w (v , respectively). The associated code is $\mathcal{C} = \{\mathbf{c} \in \mathbb{F}_q^n \mid \mathbf{c}\mathbf{H}^\top = \mathbf{0}\}$, where \top denotes transposition. The rows of the parity-check matrix define the code *parity-check equations*, that is,

$$\sum_{j=1}^n c_j h_{i,j} = 0, \quad \forall i \in [1, r], \quad \forall \mathbf{c} \in \mathcal{C}. \quad (1)$$

Equivalently, any code can be represented in terms of a generator matrix $\mathbf{G} \in \mathbb{F}_q^{k \times n}$, which forms a basis for \mathcal{C} .

In an Erasure Channel (EC), some of the codeword symbols are replaced with the erasure symbol ϵ . To this end, we express the action of an EC as $\mathbf{c} + \mathbf{e}'$, where \mathbf{c} is the input sequence and $\mathbf{e}' \in \{0, \epsilon\}^n$, with ϵ such that, conventionally, $\epsilon + a = \epsilon, \forall a \in \mathbb{F}_q$. A decoding algorithm for the EC aims to obtain a codeword by substituting each erasure with an element from \mathbb{F}_q . In the case of LDPC codes, the most common decoder used over the EC is the peeling decoder [7]. This algorithm works by expressing (1) as a linear system, where the unknowns are exactly the erased symbols. Due to the sparsity of \mathbf{H} , with large probability the linear system will include several univariate equations, *i.e.*, containing only one erasure. Each of these equations can be solved to compute the corresponding unknown, which is then substituted into all the other equations. This procedure is iterated until all the unknowns are found or, at some point, the linear system does not contain any univariate equation, *i.e.*, all the unsolved equations contain at least two unknowns. In the former case we have a decoding success, while in the latter case we have a failure, due to a *stopping set* [11], *i.e.*, a set of symbols participating to parity-check equations that contain at least two unknowns each. If all the symbols forming a stopping set are erased, peeling decoding fails. The *stopping ratio* β of an LDPC code is defined as the minimum stopping set size divided by n .

2.3. Components of the SPAR protocol

SPAR is based on a novel hash tree called Coded Merkle Tree (CMT), combined with an ad-hoc *hash-aware* peeling decoder.

Coded Merkle Tree A CMT is a hash tree which is constructed from ℓ linear codes $\{\mathcal{C}^{(1)}, \dots, \mathcal{C}^{(\ell)}\}$ over \mathbb{F}_q ; the i -th code has length n_i and dimension k_i . Each code $\mathcal{C}^{(i)}$ is defined by the systematic generator matrix $\mathbf{G}^{(i)} = [\mathbf{I}_{k_i} \mid \mathbf{A}_i]$, with $\mathbf{A}_i \in \mathbb{F}_q^{k_i \times (n_i - k_i)}$ and \mathbf{I}_{k_i} being the identity matrix of size k_i . The CMT uses an integer b which must be a divisor of all codeword length values n_1, \dots, n_ℓ . Furthermore, one needs to have partitions for the sets $[1, n_i]$, for $i \in [1, \ell - 1]$. Namely, we have $\mathcal{S}_i = \{S_1^{(i)}, \dots, S_{k_{i+1}}^{(i)}\}$ which is a partition of $[1, n_i]$, such that the $S_j^{(i)}$ are all disjoint and each one contains b elements, since $k_{i+1} = n_i/b$. Starting from $\mathbf{c} \in \mathcal{C}^{(1)}$, we build the associated CMT $\mathcal{T}'(\mathbf{c})$ as follows:

1. set $i = 1$;
2. for $j \in \{1, \dots, k_{i+1}\}$, set

$$u_j = \text{Concat} \left(\text{Hash} \left(\mathbf{c}_{z_1}^{(i)} \right), \dots, \text{Hash} \left(\mathbf{c}_{z_b}^{(i)} \right) \right),$$

with $\{z_1, \dots, z_b\} = S_j^{(i)}$;

3. encode $\mathbf{u} = [u_1, \dots, u_{k_{i+1}}]$ as¹ $\mathbf{c} = \mathbf{u}\mathbf{G}^{(i+1)}$;
4. if $i < \ell - 1$, increase i and restart from step 2), otherwise set $\mathcal{T}'.\text{Root}(\mathbf{c}) = \mathbf{u}$.

¹Notice that, when LDPC codes are considered, encoding is conveniently performed using the parity-check matrix rather than the generator matrix. This implementation detail does not affect the conclusions of our analysis but, considering encoding with the parity-check matrix, we would unnecessarily burden the notation. Therefore, we stick to encoding with the generator matrix.

Hash-aware peeling decoder A hash-aware peeling decoder, described in [13, Section 4.3], is an algorithm that decodes a set of ℓ words which are expected to constitute a CMT. Namely, let $\{\mathbf{x}^{(1)}, \dots, \mathbf{x}^{(\ell)}\}$, where $\mathbf{x}^{(i)} \in \{\mathbb{F}_q \cup \{\epsilon\}\}^{n_i}$, be the words to be decoded. The hash-aware peeling decoder works in a top-down fashion and, at every iteration, uses the peeling decoder strategy (*i.e.*, recover erasures that participate in univariate parity-check equations) for any layer of the CMT. Additionally, the hash-aware peeling decoder verifies the consistency between symbols of connected layers of the tree via hash functions, whilst the symbols are recovered. Decoding fails whenever a stopping set or a failed parity-check equation is met, just like the conventional peeling decoder. Furthermore, the hash-aware peeling decoder fails in case check consistency fails for some layer. Finally, an undetected error is met (but not recognized by the decoder) if the decoded sequence is a codeword, but not the original one.

3. A general framework to study DAAs

In this section we present a general framework to study DAAs, and then apply it to the SPAR protocol. For brevity, we only give the fundamentals of the model; for further details concerning DAAs, we refer the interested reader to [13, 3].

3.1. A general model for DAAs

We consider a game in which an adversary \mathcal{A} exchanges messages with m players $\mathcal{P}_1, \dots, \mathcal{P}_m$, who cannot communicate among themselves. Each player has access to an oracle \mathcal{O} , who can only perform polynomial time operations. Every list of transactions is seen as a vector $\mathbf{u} \in \mathbb{F}_q^k$. We assume that the following information is publicly available:

- a validity function $f : \mathbb{F}_q^k \mapsto \{\text{False}, \text{True}\}$, which depends on the blockchain rules and on its current status;
- a k -dimensional code $\mathcal{C} \subseteq \mathbb{F}_q^n$ with generator matrix \mathbf{G} ;
- two hash trees $\mathcal{T}, \mathcal{T}'$ (the former is constructed upon the uncoded data \mathbf{u} , while the latter is constructed upon the codeword $\mathbf{c} = \mathbf{u}\mathbf{G}$).

The game proceeds as follows:

1. \mathcal{A} chooses $\mathbf{u} \in \mathbb{F}_q^k$ such that $f(\mathbf{u}) = \text{False}$ and a vector $\tilde{\mathbf{c}} \in \mathbb{F}_q^n$;
2. \mathcal{A} challenges the players with (h_u, h_c) , where $h_u = \mathcal{T}.\text{Root}(\mathbf{u})$, $h_c = \mathcal{T}'.\text{Root}(\tilde{\mathbf{c}})$;
3. each player \mathcal{P}_i selects $J_i \subseteq [1, n]$ with size s ;
4. \mathcal{A} receives $U = \bigcup_{i=1}^m J_i$;
5. to reply to a query containing the index i , \mathcal{A} must send $\{\tilde{c}_i, \mathcal{T}.\text{Proof}(\tilde{\mathbf{c}}, i)\}$; \mathcal{A} is free to choose which queries to reply and which ones to ignore;
6. if a player does not receive a valid reply for any of his queries, then he discards (h_u, h_c) ;
7. the players gossip all the valid answers to \mathcal{O} , that aims to produce a proof for one of the following facts:
 - a) $\exists \tilde{\mathbf{c}} \notin \mathcal{C}$, such that $\mathcal{T}'.\text{Root}(\tilde{\mathbf{c}}) = h_c$;
 - b) $\exists \tilde{\mathbf{c}} \in \mathcal{C}$ such that $\mathcal{T}'.\text{Root}(\tilde{\mathbf{c}}) = h_c$, $\tilde{\mathbf{c}} = \tilde{\mathbf{u}}\mathbf{G}$ and $\mathcal{T}.\text{Root}(\tilde{\mathbf{u}}) \neq h_u$;
 - c) $\exists \tilde{\mathbf{c}} \in \mathcal{C}$ such that $\mathcal{T}'.\text{Root}(\tilde{\mathbf{c}}) = h_c$, $\tilde{\mathbf{c}} = \tilde{\mathbf{u}}\mathbf{G}$, $\mathcal{T}.\text{Root}(\tilde{\mathbf{u}}) = h_u$ and $f(\tilde{\mathbf{u}}) = \text{False}$.

Let us also define two properties.

Property 1. Soundness: *if a player accepts (h_u, h_c) , then \mathcal{O} will be able to recover $\tilde{\mathbf{c}}$ (and $\tilde{\mathbf{u}}$) within a finite maximum delay.*

Property 2. Agreement: *if a player accepts (h_u, h_c) , then all the other players will accept (h_u, h_c) within a finite maximum delay.*

Clearly, if \mathcal{A} wins the game, which happens with probability γ , soundness and agreement are caused to fail. We denote by γ the Adversarial Success Probability (ASP), *i.e.*, the probability that \mathcal{A} wins a random execution of the game.

It can be easily seen that, in our model, the players $\mathcal{P}_1, \dots, \mathcal{P}_m$ correspond to the light nodes connected to a malicious node modeled by \mathcal{A} . The oracle \mathcal{O} instead represents the fact that we assume any light node must be connected to at least one honest full node wishing to broadcast fraud proofs. We remark that the hypotheses and properties that underlie our model are the same under which DAAs have been studied in the literature [13, 3, 8, 12]. Finally, our model does not fix any hash tree, nor code family; thus, it can be used to study several blockchain networks. We now proceed by describing how SPAR adapts to such a model, but it can be easily seen that also the protocol proposed in [3] fits into the model.

3.2. DAAs in the SPAR protocol

In SPAR, the CMT is instantiated using the code design procedure considered in [7], which produces an ensemble of LDPC codes whose parity-check matrices have at most column weight v and at most row weight w . As mentioned in Section 2.3, besides the CMT, SPAR requires the use of another hash tree, denoted by \mathcal{T} and considered as a standard Merkle tree.

Let $\mathbf{u} \in \mathbb{F}_q^k$ denote the list of transactions of a new block. Then, a correctly constructed header contains $h_u = \mathcal{T}.\text{Root}(\mathbf{u})$ and $h_c = \mathcal{T}.\text{Root}(\mathbf{c})$, with $\mathbf{c} = \mathbf{u}\mathbf{G}^{(1)}$. However, in case of a DAA, the word $\tilde{\mathbf{c}} = \mathbf{c} + \mathbf{e}$ upon which h_c is constructed may be any vector picked from \mathbb{F}_q^n . The authors of SPAR study the protection of the protocol against DAAs; namely, they initially consider the following two cases:

- a) if $\tilde{\mathbf{c}}^{(i)} \notin \mathcal{C}^{(i)}$, then the proof consists in sending the value of all the symbols that participate in a failed parity-check equation, except for one of them, together with their CMT proofs; we refer to such a proof as *parity-check equation incorrect-coding proof*;
- b) if $\tilde{\mathbf{c}} = \mathbf{c}$ but $f(\mathbf{u}) = \text{False}$, the adversary succeeds only if the samples received by the oracle are not enough to allow the recovery of \mathbf{u} from $\tilde{\mathbf{c}}$ through decoding.

The scenario where the oracle finds a hash inconsistency is also considered, in which case \mathcal{O} can broadcast a fraud proof to the light nodes, called here *hash inconsistency incorrect-coding proof*.

The following bound for the ASP is derived [13, Theorem 1]:

$$\gamma \leq \max \left\{ (1 - \alpha_{\min})^s, 2^{\max_i \{H_b(\alpha_i)n_i + ms \log_2(1 - \alpha_i)\}} \right\} \quad (2)$$

where α_i is the *undecodable ratio* of $\mathcal{C}^{(i)}$, that is, the minimum fraction of coded symbols the adversary needs to make unavailable in order to prevent the oracle from full decoding,

$\alpha_{\min} = \min_i(\alpha_i)$, and s is the number of queries performed by each light node. Therefore, if the oracle is not able to decode due to the presence of a stopping set, the adversarial success probability computed in [13] is the probability that *exactly* one player receives an answer to all its queries.

We argue here, instead, that a sufficient condition to break the soundness and agreement as defined in [3, 13], and recalled in Section 3.1 is actually that *at least* one player accepts an invalid block.

Proposition 1. *In SPAR, an adversary cannot cause the soundness and agreement to fail with probability*

$$\gamma \leq \min\{1, \max\{1 - [1 - (1 - \alpha_{\min})^s]^m, t_2\}\}, \quad (3)$$

where $t_2 = 2^{\max_i\{b(\alpha_i)n_i + ms \log(1 - \alpha_i)\}}$.

Proof. According to Property 1, the soundness fails if at least one player accepts the block header, but the oracle will not be able to dispatch a fraud proof. The probability that exactly one player accepts the challenge is lower than or equal to $(1 - \alpha_{\min})^s$ and, therefore, the probability that exactly one player discards the challenge is larger than or equal to $1 - (1 - \alpha_{\min})^s$. Considering that there are m players, the probability that all of them discard the block is larger than or equal to $[1 - (1 - \alpha_{\min})^s]^m$. So, finally, the probability that at least a player accepts the block is lower than or equal to

$$1 - [1 - (1 - \alpha_{\min})^s]^m.$$

The rest of the proof is as in [13, Theorem 1]. □

4. Numerical examples

Let us consider the code parameters proposed in [13] as a benchmark. It is shown in [13, Table 2] that the most favourable value of the stopping ratio of the constructed ensemble (β^*) is obtained when $w = 8$ and the code rate is $R = 1/4$, from which $v = 6$ easily follows. As in [13] we consider two cases: a *strong adversary* (SA) able to find stopping sets and erase the corresponding symbols, and a *weak adversary* (WA) unable to find them and hence forced to erase random symbols. For the SA, the undecodable ratio is $\alpha^* = \beta^* = 12.4\%$; instead, in case of WA we have $\alpha^* = 47\%$ [13]. According to [13, Table 2], when $n = 4096$, the probability that the code stopping ratio α is smaller than the ensemble stopping ratio is relatively small ($3.2 \cdot 10^{-4}$).

In Table 1 we report the upper bound (2) and the newly assessed upper bound (3) on the ASP, for some values of s , considering $n = 4096$ and $m = 1024$; notice that the new value is never smaller than the previously computed upper bound. Clearly, this may have severe security consequences.

As obvious and expected, the upper bound on ASP is a decreasing function of the number of samples s . So, once a target adversarial success probability is chosen, it is possible to compute a lower bound for the value of s each player needs to ask in order to stay below it, simply by inverting (2) and (3). Considering the same parameters as above ($n = 4096$ and $m = 1024$) we obtain the results in Table 2.

Table 1Upper bound values from (2) and (3) for $m = 1024$, $n = 4096$.

s	Upper bound on γ [13]		New upper bound on γ	
	WA (2)	SA (2)	WA (3)	SA (3)
8	$6.23 \cdot 10^{-3}$	≈ 1	≈ 1	≈ 1
35	$2.24 \cdot 10^{-10}$	$9.72 \cdot 10^{-3}$	$2.29 \cdot 10^{-7}$	≈ 1
200	≈ 0	$3.17 \cdot 10^{-12}$	≈ 0	$3.24 \cdot 10^{-9}$
2000	≈ 0	≈ 0	≈ 0	≈ 0

Table 2Values of s (obtained by inverting (2) and (3)) for $m = 1024$, $n = 4096$ and different values of γ .

γ	Lower bound on s [13]		New lower bound on s	
	WA	SA	WA	SA
10^{-2}	8	35	19	88
10^{-5}	19	87	30	140
10^{-10}	37	174	48	227

We notice that the actual number of samples asked by each node is much larger than expected, resulting in a larger sampling cost S , which increases linearly with s as follows [13]

$$S = s \left(\frac{B}{k} + [y(b-1) + yb(1-R)] \log_{bR} \frac{k}{Rt} \right),$$

where B is the block size, y is the hash size and b , introduced in Section 2.3, is the number of batched hashes in each layer. Finally, t is the number of hashes in the root and determines the header size $H = t\ell_{\mathcal{H}}$, where $\ell_{\mathcal{H}} = 256$ is the binary length of the digests.

In Table 3, we assess the sampling cost S , normalized with respect to the block size B , assuming $m = 1024$, $R = 1/4$, $k = 1024$ symbols, $B = 1$ MB, $b = 8$, $t = 256$ hashes and some different values of the ASP γ . A comparison with the optimized ASBK protocol [12], named after the original authors' initials, is also reported, for which we have considered the same block size, and codes defined over a field of size 2^{256} . As expected, the optimized ASBK protocol results in smaller sampling costs than the SPAR protocol (this also held true for the original ASBK protocol [13, Fig. 4]).

Table 3Sampling cost S normalized to the block size B for $m = 1024$, $n = 4096$ and different values of γ .

γ	Lower bound on S/B [13]		New lower bound on S/B		Lower bound on S/B [12]
	WA	SA	WA	SA	
10^{-2}	0.0233	0.1019	0.0553	0.2563	0.0278
10^{-5}	0.0533	0.2534	0.0874	0.4077	0.0358
10^{-10}	0.1078	0.5068	0.1398	0.6611	0.0435

However, it should be noticed that SPAR has the advantage of relying on a fixed header size whereas in ASBK the header size increases as the square root of the block size. Therefore,

considering the same setting, in Tables 4, 5 and 6 we have compared the total amount of downloaded data D (sampling cost plus header size) using SPAR, to that obtained using the optimized ASBK protocol. The tables also report the header size H for the optimized ASBK protocol, when $B = 1$ MB, $B = 10$ MB and $B = 100$ MB, respectively. The header size for SPAR does not depend on the block size and its value is $t\ell_{\mathcal{H}} = 8.192$ kB. Notice that this amount of data must be downloaded by any light node during the regular course of the protocol, independently of the malicious behaviour of some full nodes, possibly resulting in the additional download of fraud proofs.

Table 4

Total amount of downloaded data normalized to the block size $B = 1$ MB for $m = 1024$, $n = 4096$ and different values of γ .

γ	New lower bound on D/B		Lower bound on D/B [12]	H [kB] [12]
	WA	SA	-	-
10^{-2}	0.0635	0.2645	0.0454	20.411
10^{-5}	0.0956	0.4159	0.0544	
10^{-10}	0.148	0.6693	0.0639	

Table 5

Total amount of downloaded data normalized to the block size $B = 10$ MB for $m = 1024$, $n = 4096$ and different values of γ .

γ	New lower bound on D/B		Lower bound on D/B [12]	H [kB] [12]
	WA	SA	-	-
10^{-2}	0.009	0.0386	0.0099	52.244
10^{-5}	0.0137	0.0609	0.0123	
10^{-10}	0.0214	0.0983	0.0154	

Table 6

Total amount of downloaded data normalized to the block size $B = 100$ MB for $m = 1024$, $n = 4096$ and different values of γ .

γ	New lower bound on D/B		Lower bound on D/B [12]	H [kB] [12]
	WA	SA	-	-
10^{-2}	0.0012	0.0051	0.0022	158.03
10^{-5}	0.0018	0.008	0.0025	
10^{-10}	0.0028	0.013	0.0031	

We observe that, for relatively small and moderate values of the block size, despite the larger header size, the use of the ASBK protocol is preferable even if a weak adversary is taken into account. Instead, when the block size is large, SPAR is very convenient in the presence of a weak adversary, but still more costly than ASBK if the adversary is strong.

5. Conclusion

By carefully analyzing the SPAR protocol we have shown that the actual sampling cost required by the scheme, in order to achieve target security guarantees, is much larger than that initially expected. Moreover, it is shown that, in many practical scenarios, the amount of data that light nodes have to download is larger than that of other well-known schemes.

References

- [1] Mustafa Al-Bassam. Lazyledger: A distributed data availability ledger with client-side smart contracts. <https://arxiv.org/pdf/1905.09274.pdf>, 2019.
- [2] Mustafa Al-Bassam, Alberto Sonnino, and Vitalik Buterin. Fraud and data availability proofs: Maximising light client security and scaling blockchains with dishonest majorities. <https://arxiv.org/pdf/1809.09044.pdf>, 2019.
- [3] Mustafa Al-Bassam, Alberto Sonnino, Vitalik Buterin, and Ismail Khoffi. Fraud and data availability proofs: Detecting invalid blocks in light clients. In N. Borisov and C. Diaz, editors, *Financial Cryptography and Data Security, FC 2021*, volume 12675 of *Lecture Notes in Computer Science*, pages 279–298. Springer, Berlin, Heidelberg, 2021.
- [4] Matteo Bernardini, Diego Pennino, and Maurizio Pizzonia. Blockchains meet distributed hash tables: Decoupling validation from state storage. In *CEUR Workshop Proceedings, 2nd Distributed Ledger Technology Workshop (DLT)*, volume 2334, pages 43–55, Pisa, Italia, February 2019.
- [5] Steven Cao, Swanand Kadhe, and Kannan Ramchandran. CoVer: Collaborative light-node-only verification and data availability for blockchains. In *Proceedings of the 2020 IEEE International Conference on Blockchain (Blockchain 2020)*, pages 45–52, Rhodes, Greece, November 2020.
- [6] Karunakaran Murali Krishnan and Priti Shankar. Computing the stopping distance of a Tanner graph is NP-hard. *IEEE Transactions on Information Theory*, 53(6):2278–2280, 2007.
- [7] Michael G. Luby, Michael Mitzenmacher, M. Amin Shokrollahi, and Daniel A. Spielman. Efficient erasure correcting codes. *IEEE Transactions on Information Theory*, 47(2):569–584, 2001.
- [8] Debarbab Mitra, Lev Taus, and Lara Dolecek. Concentrated stopping set design for coded Merkle tree: Improving security against data availability attacks in blockchain systems. In *Proceedings of the International Symposium on Information Theory (ISIT 2020)*, pages 136–140, Los Angeles, CA, USA, June 2020.
- [9] Debarbab Mitra, Lev Taus, and Lara Dolecek. Concentrated stopping set design for coded Merkle tree: Improving security against data availability attacks in blockchain systems. <https://arxiv.org/pdf/2010.07363.pdf>, 2021.
- [10] Satoshi Nakamoto. Bitcoin: A peer-to-peer electronic cash system. : <https://bitcoin.org/bitcoin.pdf>, 2008.
- [11] Tom Richardson and Rüdiger Urbanke. *Modern Coding Theory*. Cambridge University Press, 2008.
- [12] Paolo Santini, Giulia Rafaiani, Massimo Battaglioni, Franco Chiaraluce, and Marco Baldi.

- Optimization of a Reed-Solomon code-based protocol against blockchain data availability attacks. In *Proceedings of IEEE International Conference on Communications (ICC) 2022*, Seoul, South Korea and virtual, May 2022.
- [13] Mingchao Yu, Saeid Sahraei, Songze Li, Salman Avestimehr, Sreeram Kannan, and Pramod Viswanath. Coded Merkle tree: Solving data availability attacks in blockchains. In J. Bonneau and N. Heninger, editors, *Financial Cryptography and Data Security, FC 2020*, volume 12059 of *Lecture Notes in Computer Science*, pages 114–134. Springer, Cham, 2020.
- [14] Qiheng Zhou, Huawei Huang, Zibin Zheng, and Jing Bian. Solutions to scalability of blockchain: A survey. *IEEE Access*, 8:16440–16455, 2020.