SnapDeals - Theory Report

Protocol Labs

November 2021

Abstract

This documents is a technical report about the security of FIP019 (aka "SnapDeals") which allows for adding storage deals into a native CC (commited capacity) sector without need of resealing.

Contents

1 Introduction **1**
1.1 Background 2

2 Protocols **4**
2.1 Starting point: the interactive protocol 4
2.2 SnapDeals: Non-interactive sector update 5

3 Security Analysis **6**
3.1 Step 1 6
3.2 Step 2 7
3.2.1 Empirical analysis 7
3.2.2 Analysis based on Kolmogorov complexity 8

A Challenge Bucketing **11**

1 Introduction

Filecoin is a decentralized storage network that turns cloud storage into an algorithmic market. The market runs on a blockchain with a native protocol

token (also called "Filecoin"), which Storage Providerss earn by providing storage to clients and mining blocks.

A *sector* is the default unit of storage in Filecoin, Storage Providerss add storage capacity and gain power in units of sectors. Each sector has a size (currently, we allow only for currently 32GBs or 64GBs as sector size) and a lifetime determined by the Storage Provider at the moment of the creation. Each sector contains a *replica*. Replicas are incompressible encoding of data that can be agreed through deals in the Storage Market (ie, clients data) or an incompressible encoding of a string of zeros. We refer to the former as Sectors with Deals, while the latter are called Committed Capacity (CC) sectors. A CC sector can upgrade to a deals sector and this is currently done by re-sealing.

This document presents SnapDeals (FIP019): a protocol that allows Storage Provider to transform a CC sector in a deals sector in a cheaper and more efficient way than resealing. Snapdeals also allows for a single message protocol for onboarding data into CC sectors, getting rid of multiple rounds of communication.

1.1 Background

A Proof of Space (PoS, see for example [DFKP15]) is a protocol that allows a prover to convince a verifier that he has a minimum specified amount of space (ie, used storage). More precisely in a PoS protocol we have two main sub-protocols:

- *Initialization*: on public input a size N, an advice (eg, vector of random data) of length N is created. The advice is stored by the prover, while the verifier knows a commitment to it. This is a one-time setup phase.
- *Execution*: the verifier and the prover run a protocol and the verifier outputs reject/accept. Accept means that the verifier is convinced that the prover stores the advice. This phase can be repeated many times.

A PoS is sound if a verifier interacting with a malicious prover who stores a fraction of the advice that has size $N' < N$, and runs in at most T steps during the execution phase, outputs accept with small probability (ie, soundness error). We refer to this property as *space-hardness* and the value

$$\epsilon = (N - N')/N$$

is called the *spacegap*.

In Filecoin, a replica R is created by Storage Provider running the initialization phase of the Stacked-DRGs Proof of Space [Fis19] and then using the the advice to encode users data. More precisely, the replica is created via the following steps:

- *PreCommit*: on input some data D (encoded as vector of size N), the Storage Provider computes the PoS advice S (ie, the labels of the last layers in the Staked-DRGs graph[1]) and the replica $R = S + D$. The Storage Providerss also compute commitments to the data ($\mathsf{Comm}_D =$ the root of the Merkle tree constructed over D), to the labels of the graph and to R ($\mathsf{Comm}_R =$ the root of the Merkle tree constructed over R).

  Both R and D are vectors of $N = 2^{30}$ components and each component is a 32 bytes element that we see as field element Fr32 in the arithmetic field of BLS12-381; $+$ is the component-wise sum in the field.

  We do not give the details here, but it is known that the replica R has the same space hardness property as the original advice S (ie, the same pebbling analysis as the in the original Stacked-DRGs PoS can be used).

- *ProveCommit*: Storage Provider waits 150 epochs[2] and then takes the random seed from the beacon and generates c challenges for the index set as $c_i = \mathsf{H}_1(\mathsf{tag}, \mathsf{seed}, i) \in \{0, 1, \ldots, N-1\})$ for $i = 1, 2, \ldots, c$. For each challenge index c_i, the Storage Provider constructs the SNARK proof that the elements S_{c_i} and R_{c_i} (ie, the c_i-th components of S and R, respectively) were correctly generated and match the commitments posted on-chain. Finally, the Storage Provider posts the proof on-chain.

[1] A commitment to the data Comm_D is added inside the labeling hash function (eg, $label = \mathsf{H}(\mathsf{Comm}_D, parents)$ in order to tie the PoS instantiation to the specific data D and avoid that a malicious Storage Provider adaptively chooses D such that $S + D$ is efficiently compressible.

[2] In Filecoin, time is divided into discrete units we call epochs. Storage Providerss have semi-synchronised clocks that indicates the current epoch.

2 Protocols

2.1 Starting point: the interactive protocol

The starting point for SnapDeals is a simple 3-message protocol (2 round of interaction) that works as follows:

1. Storage Provider has CC sector with a replica R storing zeros (or more in general compressible data). To update this to a sector with deals, he declares a list of clients data D and posts on chain Comm_D; then the Storage Provider waits 150 epochs and then takes the random seed from the random beacon;

2. Storage Provider expands the random seed getting randomness field element ρ_i, updates the replica R to R^* computing

$$R_i^* = R_i + D_i \cdot \rho_i$$

   for all $i \in \{0, 1, \cdots, N-1\}$ (sum and product in the arithmetic field) and posts Comm_{R^*} (the root of the Merkle tree constructed over the vector R^*) on chain;

3. Storage Provider waits 150 epochs and then takes the random seed from the beacon and generates c challenges for the index set $\{0, 1, \ldots, N-1\}$ as $c_i = \mathsf{H}_1(\mathsf{tag}, \mathsf{seed}, i)$ for $i = 1, 2, \ldots, c$. For each challenge index c_i, the Storage Provider constructs the SNARK proof that the element $R^*_{c_i}$ was correctly generated and matches the commitment posted on-chain. Finally, the Storage Providerss posts the proof on-chain.

Security: We can set the parameter c in such a way that with large probability, a Storage Provider that has some errors in R^* does not pass the verification on the posted proof. More precisely, if

$$c = \frac{-s}{\log_2(1-x)} \tag{1}$$

the we can say that an R^* with a fraction x of wrong components does not pass the last step of the protocol with probability larger than $1 - 2^{-s}$.
Once we know that R^* is correctly generated, then R^* has the same space hardness property as the original replica R because the values added to it are random and can not be used to compress a component.

Efficiency: Encoding function is simple to prove and lightweight to invert (once we have the inverse of the ρ_is and the old replica R, decoding is trivial). Proving overhead for the ProveCommit step is significantly reduced (10x improvement only in snark generation, and no need of recomputing the Stacked-DRGs graph labels) with respect to re-sealing.

On the other hand, the footprint of this protocol is not optimal: 3 message on-chain (one for declaring new data to be included in the CC sector, one for declaring replica update, one for proving replica update). Moreover, the whole process needs at least around 300 epochs (plus proving time) to complete, which translates in around 3 hours. In order to avoid the last two downsides, we propose SnapDeals, described in the next section.

2.2 SnapDeals: Non-interactive sector update

This is the non interactive version of the previous 3-rounds protocol, using a similar encoding. More in details:

1. Storage Provider has CC sector with a replica R. To updated this, it takes a list of data D, computes Comm_D and uses Comm_D in order to derive randomness $\{\rho_j\}_{j\in J}$. More precisely, divide a sector in N/K subsets of size K components and let $\rho_j = \mathsf{H}_2(\mathsf{tag}, \mathsf{Comm}_D, \mathsf{Comm}_R, j)$ for $j \in \{0, 1, \ldots, K-1\}$, then update the replica R to R^* as

   $$R^*_{jK+i} = R_{jk+i} + D_{jK+i} \cdot \rho_j$$

   for all $i \in \{0, 1, \ldots, N/K - 1\}$.

   The Storage Provider also computes Comm_{R^*} and uses it to locally generate c challenges to prove R^* was correctly generated.

   The declaration of sector upgrade with deals D, the commitments $\mathsf{Comm}_D, \mathsf{Comm}_{R^*}$ and the SNARK proof of correctness and matching for the challenged indices is posted on-chain.

Note that we have a value ρ_j shared among K components (resulting in $2^{30}/K$ different hash computations) in order to decrease the number of hash computations. This implies being able to reduce the cost of 2^30 modular inversion while inverting the encoding function. Note that 2^30 modular inversions are needed if we use a different hash for each node.

3 Security Analysis

Similar to the interactive protocol (Section 2.1), in SnapDeals the number of challenges c needed guaranteed the correctness of R^* can be chosen using the formula (1). For example, in the current implementation we have $c = 1376$, which guarantees that more than 99% of R^* is correct with overwhelming probability ($s = 20$).

On the other hand, the fact that the randomness values ρ_j are now computed by the Storage Provider itself opens to a new grinding attack (not possible in the interactive protocol) where the malicious Storage Provider could try different values for the data D in order to find values that allows him to compress the original replica R. More precisely, the goal of the adversary is to increase the original value of the spacegap ϵ and get an R^* where more than ϵN components can be not stored and re-computed on the spot with less than T steps (see Section 1.1).

In this section, we show that an adversary with a limited budget of attempts (ie, we assume that at most 2^λ attempts are feasible) can increase the spacegap of only a small value δ. The proof consists in two steps:

- **Step 1**: we show that an adversary with budget 2^λ can only control at most λ bits of the bit representation of the string R. See Section 3.1.
- **Step 2**: we show that controlling at most λ bits allows for limited spacegap amplification. We do this using two different techniques: one based on empirical assumption (Section 3.2.1) and the other using the Kolmogorov complexity (Section 3.2.2).

3.1 Step 1

Remember that $R = (R_1, \ldots, R_N)$, $D = (D_1, \ldots, D_N)$, and $R^*_i = R_i + D_i \cdot \rho_j$ with $\rho_j = \mathsf{H}_2(\mathsf{Comm}_D, \mathsf{Comm}_R, j)$ and is re-used for a subset of K sequential components. The vector R is fixed, and we consider an adversary that can try at most 2^λ different values for the vector D.

Claim: The adversary we consider can only control (by fliping) at most λ bits of the bit representation of the string R.

Proof: First observe that the adversary can choose the vector $(D_1 \cdot \rho_1, \ldots, D_N \cdot \rho_{N/K})$ in a set of vectors of cardinality at most 2^λ (each D gives at most one vector). Moreover, trivially each vector defines at most one value for the string R^*. This implies that the power of adversary we consider is limited to choose R^* in a set of at most 2^λ values. Secondly, observe that saying that the adversary can control x bits of a string implies that he can choose the string in a set of cardinality 2^x (for each bit, he can choose to flip or not, this x times). Finally, we can concluded using a "by contradiction" argument: if the adversary could control $\lambda + 1$, then following our second point before he would be able to choose R^* in a set of $2^{\lambda+1}$ values, but this contradicts our first point.

3.2 Step 2

Observe that the goal of the adversary to get a larger spacegap is getting an R^* where a fraction of components are set to compressible values (eg, zeros in the last 2 bytes) instead of being random components as in the original replica. In the following we use the verb "compress" to indicate this. Given the claim of Step 1, we can now rephrase the problem as follows:

> We have a random string R, with N components, 32 bytes each. We give the adversary a budget of λ bit flips for the bit representation of R. How much can he compress R?

3.2.1 Empirical analysis

In this section we present an empirical analysis about how much the adversary can compress R using at most λ bit flipping.

The best strategy for the adversary is to inspect R and flip the bits that are at the end of in the longest substrings of zeros. For example, consider a substring of the form $s = 0000000100000000$ in R. Flipping only 1 bit flip, 2 bytes are set to zero. When this happens in a component (32 bytes), it allows to a 6.25% compression of the component.

However, strings as s are not very likely to happen in R. Instead of computing the probability for the substring s (for each possible length), we just assume that the adversary can find λ substrings each of length[3] 16.

[3]We consider strings of length 16 because compressing shorter strings for 256 bits is not worth (since compressing + placeholding is essentially the same as not compressing).

This event has very low probability, but we consider it as the status quo for R for our analysis (giving this advantage to the adversary improves the security we claim).

Moreover, we assume that the adversary finds each one of the string of the form of $s = 000000010000000$ (or equivalent) in λ different components, and we assume that when an adversary finds 16 bits in a component that he can compress into 1 bit, he finds it in all the components that share the same ρ_i. This results in a total compression of

$$\lambda \cdot 0.0625 \cdot K$$

components out of $N = 2^{30}$.

In our implementation, we consider $K = 2^{21}$ and, for the argument above, we claim that the spacegap is at most 1% more than the original one.

3.2.2 Analysis based on Kolmogorov complexity

The Kolmogorov complexity measures the complexity of objects in terms of the minimum amount of bits required to represent them.

Let T be a deterministic Turing machine and x a bit string. We say that (T, α) is a (possibly inefficient) description of x if $T(\alpha) = x$. The Kolmogorov complexity $C_T(x)$ of a bit string x with respect to a deterministic Turing machine T is defined as:

$$C_T(x) = \min_{\alpha \in \{0,1\}^*} \{|\alpha| : T(\alpha) = x\}$$

It has been proved that the Kolmogorov complexity with respect to different Turing machines is invariant only up to a constant that depends on the reference Turing machine. For this reason, from now on we express the Kolmogorov complexity using the universal Turing machine U as a reference machine $C(x) = C_U(x)$. A bit string x is *c-incompressible* if $C(x) \geq |x| - c$.

Theorem: Given a n-bit string S that is V-incompressible, for any adversary that modifies S into S' by flipping λ bits of his choice, we have that the incompressibility of S' is at least $V - \lambda \cdot [\log(n) + 2\log(\log(n) + 1)]$.

Proof: Note that $C(S') = C(S) + \lambda \cdot (\log(n) + 2\log(\log(n) + 1))$. Indeed, $\lambda \cdot (\log(n) + 2\log(\log(n) + 1))$ are the bits needed in order to be

able to store the λ bit positions where the adversary flipped the bits using self delimiting codes [ACF+21]. Assume by contradiction that $C(S') < n-V+\lambda\cdot(\log(n)+2\log(\log(n)+1))$, then $C(S) < n-V$, which is not possible.

Given that in Filecoin the security model assumes a rational adversary, with an abuse of notation we say that the replica R is an ϵn-incompressible string[4] with $n = 256N$. Recall that we consider the adversary that can flip at most λ bits of his choice. To be conservative, given that we have $2^{30}/K = 1024 = 2^{10}$ different hashes, we apply the compressibility lower bound on vector of size $N' = 230/K = 2^{20}$ nodes, which corresponds to 2^{28} bits (given that nodes are 256 bits each).

According to the lower bound, we get that an adversary that can flip $\lambda = 80$ bits of his choice on R can not compress R to a string which has compressibility smaller than $N'\cdot 2^8 - 80(\log(N'\cdot 2^8) + 2\cdot\log(\log(N'\cdot 2^8)) + 1)$.

In order to be even more conservative, we are using N instead of N' in the compressibility reduction factor. We are then considering that R can not be compressed to a string which is less compressible than $2^{30}/K\cdot 2^8 - 80(\log(2^{30}\cdot 2^8) + 2\log(\log(2^{30}\cdot 2^8)) + 1) = 2^{30}/K\cdot 2^8 - 80(38+12)$. As a consequence, we have that an adversary can compress a total of $80(38+12)\cdot K$ bits out of 238 bits This means that, for $K = 2^{20}$ and $2^{10} = 1024$ different hashes, we have that the additional spacegap is $< 1.53\%$ (see calculation here). If we consider 20% original spacegap, the additional spacegap is $< 1.91\%$ (see calculation here).

On Uncontrolled bit Flips and Compressibility

We have shown that an adversary with a budget of 80 bit flips can affect compressibility of an original replica by less than 2%. One could argue that for the way the construction works, those flips are not coming alone, meaning that those flips also modify other bits with respect to the original replica, and this could harm incompressibility even more. This is not the case, since those additional modifications are not under the control of the adversary herself.

[4] We know that the vector R is (ϵ, T)-space hard. More precisely we know that computing a random component of R starting from an input of at most $(1-\epsilon)N$ components costs more than T steps with probability larger than $\epsilon/2$. Now we assume that no adversary chooses to pay the the expected cost of $\epsilon/2 * T$ because of the rationality assumption. And therefore, we can interpret the space-hardness property as "computing any component of R starting from an input of length at most $(1-\epsilon)N$ does not happen".

This means that there is no way she can exploit such uncontrolled changes. Moreover, we observe that if there was a way for which the incompressible string could be compressed by introducing uncontrolled changes, this would also affect the original replicas, given the way replica $R = S + D$ is formed starting from the incompressible advice S.

A Challenge Bucketing

Due to implementation efficiency, we are here analysing the possibility of bucketing challenges instead of having a global set of uniform challenges.

Analysis

Given a string S of length N, doing X challenges with λ bits of security ensures that percentage Y was correctly encoded. More formally, we have that $(Y)^X = 2^{-\lambda}$.

This means that storing less than Y and answering challenges happens with probability smaller than $2^{-\lambda}$. Now, instead of asking for X challenges on the whole string S, we want to partition the string S into substrings $S_1, ..., S_k$, with $|S_i| = N/k$ asking X/k challenges in each substring.
We refer to x_i as the portion of the string stored in each S_i. This means that the probability of respectively storing $s_1, ..., s_k$ portion of $S_1, ..., S_k$ and pass the verification is bounded by $(s_1)^{X/k} \cdot ... \cdot (s_k)^{X/k}$.
Which is the best strategy of choosing $s_1, ..., s_k$?

In order to answer the question we need to maximise the function $F(s_1, ..., s_k) = (s_1)^{X/k} \cdot ... \cdot (s_k)^{X/k}$.
First, we need to evaluate the gradient of the function F.

$$\nabla F = [\partial F/\partial s_1, ..., \partial F/\partial s_k] =$$
$$= [X/k \cdot (s_1^{X/k-1}) \cdot (s_2 \cdot ... \cdot s_k)^{X/k}, ..., X/k \cdot (s_k^{X/k-1}) \cdot (s_1 \cdot ... \cdot s_{k-1})^{X/k}]$$

We have a restriction on F: we consider $s_1, ..., s_k$ such that $(s_1 + ... + s_k) \cdot (N/k) = Y \cdot N$, that is $(s_1 + ... + s_k)/k = Y$ (note that if this is not the case an attacker would store more than a Y fraction of the string, which is not an adversarial strategy). We define $G(s_1, ..., s_k) = (s_1 + ... + s_k)/k - Y$.
We consider now the auxiliary function

$$L = F(s_1, ..., s_k) - \gamma \cdot G(s_1, ..., s_k)$$

It follows

$$\nabla L = (\partial L/\partial s_1, ..., \partial L/\partial s_k, \partial L/\partial \gamma) = (\nabla F - \gamma/k, G).$$

In order to find maximums/minimums we need the gradient of L to be 0.

$$\nabla L = (\nabla F - \gamma/k, G) = 0$$
$$\Rightarrow [X/k \cdot (s_1^{X/k-1}) \cdot (s_2 \cdot ... \cdot s_k)^{X/k} - \gamma/k, ..., X/k \cdot (s_k^{X/k-1}) \cdot (s_1 \cdot ... \cdot s_{k-1})^{X/k} - \gamma/k,$$
$$(s_1 + ... + s_k)/k - Y] = 0$$

This translates in a system of equations of the form

- $\partial F/\partial s_1 - \gamma/k = 0$
  ...
- $\partial F/\partial s_k - \gamma/k = 0$

- $G(s_1, ..., s_k) = 0.$

which means

- $X/k \cdot (s_1^{X/k-1}) \cdot (s_2 \cdot ... \cdot s_k)^{X/k} = \gamma/k$
- $X/k \cdot (s_2^{X/k-1}) \cdot (s_1 \cdot s_3 \cdot ... \cdot s_k)^{X/k} = \gamma/k$
  ...
- $X/k \cdot (sk^{X/k-1}) \cdot (s_1 \cdot ... \cdot s_{k-1})^{X/k} = \gamma/k$
- $(s_1 + ... + s_k)/k = Y$

It is easy to see that the first k equations give as a result

1. $s_i = 0$ for some i (one or more)
2. $s_1 = ... = s_k$.

The first relation is trivial. For the second relation we give an example for the first two equations. The rest is only an iteration of the following step:

We know that it must be

- $X/k \cdot (s_1^{X/k-1}) \cdot (s_2 \cdot ... \cdot s_k)^{X/k} = \gamma/k$
- $X/k \cdot (s_2^{X/k-1}) \cdot (s_1 \cdot s_3 \cdot ... \cdot s_k)^{X/k} = \gamma/k$

This means that we can write the above as

- $X/k \cdot (s_1^{X/k-1}) \cdot (s_2 \cdot ... \cdot s_k)^{X/k} = X/k \cdot (s_2^{X/k-1}) \cdot (s_1 \cdot s_3 \cdot ... \cdot s_k)^{X/k}$
- $X/k \cdot (s_2^{X/k-1}) \cdot (s_1 \cdot s_3 \cdot ... \cdot s_k)^{X/k} = \gamma/k.$

It follows

- $(s_1^{X/k-1}) \cdot (s_2 \cdot ... \cdot s_k)^{X/k} = (s_2^{X/k-1}) \cdot (s_1 \cdot s_3 \cdot ... \cdot s_k)^{X/k}$
- $X/k \cdot (s_2^{X/k-1}) \cdot (s_1 \cdot s_3 \cdot ... \cdot s_k)^{X/k} = \gamma/k.$

If we divide both members of the first equation by

$$(s_1^{X/k-1}) \cdot (s_2^{X/k-1}) \cdot (s_3 \cdot ... \cdot s_k)^{X/k}$$

we obtain

- $s_2 = s_1$
- $X/k \cdot (s_2^{X/k-1}) \cdot (s_1 \cdot s_3 \cdot ... \cdot s_k)^{X/k} = \gamma/k.$

If one repeats the process with the second and the third equations, obtains $s_3 = s_2$ and so on and so forth.

So now the equations are:

- $s_2 = s_1$
- $s_3 = s_2$ ...
- $s_k = s_{k-1}$
- $(s_1 + ... + s_k)/k = Y.$

Which translates into

- $s_1 = s_2 = ... = s_k =: s$
- $k \cdot s/k = Y \rightarrow s = Y.$

This suggests that the best strategy in order to maximise the probability of winning is to store the same percentage in each substring, namely a percentage Y in each of them.

References

[ACF+21] Giuseppe Ateniese, Long Chen, Danilo Francati, Dimitrios Papadopoulos, and Qiang Tang. Verifiable capacity-bound functions: A new primitive from kolmogorov complexity. Cryptology ePrint Archive, Report 2021/162, 2021. https://ia.cr/2021/162.

[DFKP15] Stefan Dziembowski, Sebastian Faust, Vladimir Kolmogorov, and Krzysztof Pietrzak. Proofs of space. In *Advances in Cryptology - CRYPTO 2015 - 35th Annual Cryptology Conference, Santa Barbara, CA, USA, August 16-20, 2015, Proceedings, Part II*, pages 585–605, 2015.

[Fis19] Ben Fisch. Tight proofs of space and replication. In *Advances in Cryptology - EUROCRYPT 2019 - 38th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Darmstadt, Germany, May 19-23, 2019, Proceedings, Part II*, pages 324–348, 2019.