Time/Space Tradeoffs for Generic Attacks on Delay Functions

Kasper Green Larsen[1] and Mark Simkin[2]

[1] Aarhus University, larsen@cs.au.dk
[2] Flashbots, mark@univariate.org

Abstract. Delay functions have the goal of being inherently slow to compute. They have been shown to be useful for generating public randomness in distributed systems in the presence of a dishonest majority of network participants; a task that is impossible to solve without such functions, due to Cleve's seminal impossibility result (STOC 1986). Currently, little is known on how to construct secure delay functions or how to analyze the security of newly proposed candidate constructions.
In this work, we explore the time/space tradeoffs of generic attacks for a large class of potential delay function designs. We consider delay functions F^T, which are computed as

$$F^T := \underbrace{F(\dots F(F}_{T}(x))\dots),$$

where $F : [N] \to [N]$ is some round function, in the presence of an adversary, who is given an advice string of some bounded size, has oracle access to F, and would like to compute F^T on a random input x using less than T sequential oracle calls.
We show that for both random and arbitrary functions F there exist non-trivial adversaries, who successfully evaluate F^T using only $T/4$ sequential calls to oracle F, when given a large enough advice string. We also show that there exist round functions F for which the adversary cannot compute F^T using less than $T/2$ sequential queries, unless they receive a large advice string or they can perform a large number of oracle queries to F in parallel.

1 Introduction

Delay functions, first introduced by Goldschlag and Stubblebine [GS98], are functions that are inherently slow to compute, no matter how much parallel computing power is available. Slightly more formally, these are functions that cannot be evaluated by low depth circuits, unless the circuit size is super-polynomial related to the input length. Treating the depth of a circuit as a proxy for the physical time needed to evaluate the corresponding function and assuming that parties in a distributed network can measure time, has fascinating consequences in both theoretical and applied cryptography.

Boneh et al. [BBBF18], for example, show that delay functions allow for overcoming Cleve's impossibility result [Cle86], which states that n parties cannot flip an unbiased coin in a distributed network, when half or more of the processors are faulty.[3] Being able to flip coins securely in the presence of a dishonest majority is of significant importance for modern day blockchains and following the results of Boneh et al., several constructions of randomness beacons based on delay functions have been proposed [CMB23]. Motivated by the application outlined above and various other ones in the context of blockchains, delay functions have received a significant amount of interest [BBBF18, Pie19, Wes19, EFKP20, MSW20, RS20, ARS24] in recent years.

Unfortunately, right now we do not know how to construct delay functions that are as secure and as efficient as one might hope for and our ability to assess the security of new candidate delay

[3] The general idea of using time-based cryptography to overcome Cleve's impossibility originally appeared in the work of Boneh and Naor [BN00].

functions is quite limited. Broadly speaking, there are two main approaches for assessing their security and both of them are constrained in their power.

The *reductionist approach* is to base the security of a given delay function upon the presumed hardness of some computational problem. This approach presupposes that there are "good" problems to reduce to, but as of now this does not seem to be the case. Delay functions built upon the assumption that repeated squaring in groups of unknown order cannot be parallelized, an assumption introduced by Cai et al. [CLSY93], all face practical obstacles. They either rely on a trusted setup or require performing highly efficient (low depth) arithmetic operations over class groups, which are hard to implement in practice. Alternatively, delay functions can be built upon the sole assumption that certain non-parallelizable languages and one-way functions exist [ARS24], but the practicality of the resulting constructions remains yet to be seen.

The *cryptanalytic approach* is try and break a given candidate delay function and to declare it secure, when no meaningful attacks have been found for a sufficiently long time. In recent years, several new delay function designs have been proposed by different researchers [BBBF18, DMPS19, LM23, KMT22], but many of these proposed constructions have been broken shortly after their initial publication [PT24, BFH+24, DDJ24]. Although these attacks are bad news for the delay functions they target, they may also provide valuable insights for developing new, more secure designs, serving as guiding principles for future delay functions.

The existing attacks [BBBF18, PT24, BFH+24, DDJ24] specifically target the algebraic structure underlying the delay functions they consider, thus they provide somewhat limited insights into what more general design principles for delay functions could be.

1.1 Our Contribution

In this work we put on our cryptanalytic hat and explore time/space tradeoffs for generic attacks on a large class of delay functions. Specifically, we focus on delay functions F^T that are comprised of a round function $F : [N] \to [N]$ with $[N] := \{0, \ldots, N-1\}$, which is repeatedly applied to itself. In other words, we consider delay functions with arbitrary round functions of the form

$$F^T := \underbrace{F(\ldots F(F}_{T}(x)) \ldots).$$

This class of delay functions encompasses virtually all existing candidate delay functions, with the exception of a recent work by Abram, Roy, and Simkin [ARS24]. We consider an adversary $\mathcal{A} = (\mathcal{A}_0, \mathcal{A}_1)$, who has oracle access to the function F and who aims to evaluate $F^T(x)$ on a random challenge x using less than T sequential calls to the oracle F. The adversary $\mathcal{A}_0$ is initially allowed to perform an unbounded preprocessing step, which is independent of the challenge x, and return an advice string of size S. This advice string along with a random challenge x is then given to $\mathcal{A}_1$, who aims to output $F^T(x)$ using at most $\widetilde{T}$ with $\widetilde{T} < T$ sequential oracle calls to F, where during each call the adversary can evaluate F in at most P different points in parallel.

In this work, we explore the relationship between all of the involved parameters, namely between the size S of the advice string, the number of required and sufficient oracle calls $T, \widetilde{T}, P$, and the size of the domain N. We show several somewhat complementary results.

First, using rather simple attack ideas, we show that non-trivial adversaries exist for random and arbitrary functions. For example, we show that for some range of parameters, there exists an efficient adversary for arbitrary F that successfully computes F^T using $\widetilde{T} = T/4$ sequential oracle

calls with constant success probability. Slightly more formally, this result is captured in the theorem statement below.

Theorem 1 (Informal). *For any round function $F : [N] \to [N]$, any $T \in \mathbb{N}$, there exists an adversary $\mathcal{A} = (\mathcal{A}_0, \mathcal{A}_1)$ that outputs an advice string of size $S = \mathcal{O}\left(\frac{N \lg N}{T}\right)$, makes at most $\widetilde{T} = T/4$ sequential and at most P parallel queries, such that and*

$$\Pr\left[\mathcal{A}_1(x, \sigma) = F^T(x) : \sigma \leftarrow \mathcal{A}_0(F)\right] > 1/2,$$

where the probability is taken over the choice of $x \in [N]$ and the random coins of the adversary.

The above theorem tells us that for some parameter ranges we have successfully found an adversarial strategy that works, but it leaves open the question, whether there are much better attacks for arbitrary functions F.

Using an encoding argument, a proof technique that has been used with great success in data structure lower bounds [PD06, PV10, Lar12a, Lar12b, BL13, VZ13, CKL18, LS20, LY20, LLYZ23], we show that there is a limit to how successful an adversary can be. Concretely, we show that there exist round functions for which *any* adversary that wants to do less than $T/2$ sequential oracle calls needs to have a large advice string and/or needs to perform many parallel oracle calls. Proving this statement is rather involved and constitutes the main technical contribution of this work. Slightly more formally, we get the following theorem.

Theorem 2 (Informal). *There exist round functions $F : [N] \to [N]$, such that for any adversary $\mathcal{A} = (\mathcal{A}_0, \mathcal{A}_1)$ with*

$$\Pr\left[\mathcal{A}_1(x, \sigma) = F^T(x) : \sigma \leftarrow \mathcal{A}_0(F)\right] > 1/2,$$

either the number of sequential queries $\widetilde{T} \geq T/2$ or (roughly speaking) it must hold that

$$(S + \widetilde{T})\widetilde{T}P = \Omega(N).$$

Are these Attacks practical? We note that the goal of this work is to explore the asymptotic landscape of time/space tradeoffs for generic attacks on delay functions. We have not optimized the hidden constants in our results and leave tightening hidden constants and exploring the practicality of the proposed attacks for future work.

1.2 Related Works

Several works [RSS20, KLX20, RS20] investigate the computational hardness of the repeated squaring assumption of Cai et al. [CLSY93]. Rotem, Segev, and Shahaf [RSS20] rule out a large class of potential constructions of delay functions from known-order cyclic groups. Katz, Loss, and Xu [KLX20] as well as Rotem and Segev [RS20] show that under certain simplifying assumptions reducing the sequential time needed for performing repeated squaring in groups of unknown order is as hard as factoring. Boneh et al. [BBBF18] investigate delay functions based on certain types of permutation polynomials and explore several kinds of algebraic attacks on those. Peikert and Tang [PT24] show that a new hardness assumption over lattices, introduced by Lai and Malavolta [LM23] with the goal of constructing delay functions, is false. All these works have a focus that is different from ours, as we care about generic attacks, whereas these works focus on highly specific algebraic structures.

Several different attacks and their time/space tradeoffs have also been explored in the context of a specific, by now defunct, candidate delay function called Minroot [LMP+23]. Ideas underlying some of these attacks are similar to those underlying our attacks on arbitrary round functions, but whereas their attacks and analysis focuses on Minroot specifically (in an idealized model), we show that this type of attack can be made to work for arbitrary round functions (without an idealized model).

Somewhat more loosely related, but similar in spirit is the topic of function inversion [Hel80]. Here, an adversary is given oracle access to a function F, can perform an unbounded preprocessing step to output an advice string of length S and is then asked to invert F on a random output y, i.e. the adversary is given y and asked to find an x, such that $F(x) = y$, while making as few queries to the oracle F as possible. In his seminal work, Hellman [Hel80] proved that non-trivial time/space tradeoffs exist for random functions and suggested that they also exist for arbitrary functions. A subsequent work by Fiat and Naor [FN91] has then rigorously proven the time/space tradeoffs for arbitrary functions. More recently, this topic has seen a renewed interest [CK19, CHM20, GGPS23], which has led to new bounds.

We stress that despite their superficial similarities, function inversion and our model in this work are fundamentally different; especially when it comes to proving limits on the power of adversaries in the respective settings. Lower bounds in function inversion need to argue that, given some advice string of bounded size, the adversary has to make at least some number of oracle queries *in total* to compute the output. In our setting, the output can be computed using only T sequential queries, but we need to prove that an adversary cannot compute it using significantly less that T *sequential* queries, unless it effectively queries most of the domain using parallel queries.

2 Model

Definition 1 (Iterated Functions). *Let $T \in \mathbb{N}$ and let $F : [N] \to [N]$ be a function. We define*

$$F^T := \underbrace{F(\dots F(F}_{T}(x)) \dots)$$

to be the T-wise iterated function with round function F.

Definition 2 (Parallel and Sequential Oracle Queries). *Let $\mathcal{A}$ be an algorithm with query access to oracle $\mathcal{O}$. Separate queries by $\mathcal{A}$ to $\mathcal{O}$ are said to be sequential, if they happen one after another. They are said to be parallel, if they happen simultaneously. Any algorithm $\mathcal{A}$ is allowed to do both types of queries.*

Definition 3 ($(S, P, \widetilde{T})$-admissible adversaries). *Let $S, P, \widetilde{T} \in \mathbb{N}$. An adversary $\mathcal{A} = (\mathcal{A}_0, \mathcal{A}_1)$ is said to be $(S, P, \widetilde{T})$-admissible, if $\mathcal{A}_0$ on some given input always returns a bit string of length at most S and if $\mathcal{A}_1$ with access to oracle $\mathcal{O}$ makes at most P parallel queries at once in a single round and at most $\widetilde{T}$ sequential rounds of queries.*

Definition 4 (Sequentiality). *Let $S, P, T \in \mathbb{N}$, let $\epsilon \in \mathbb{R}$ with $0 \leq \epsilon \leq 1$, and let $F : [N] \to [N]$ be a round function. We say iterated function F^T is (S, P, ϵ)-secure against, if for all $(S, P, \epsilon T)$-admissible adversaries $\mathcal{A} := (\mathcal{A}_0, \mathcal{A}_1)$, it holds that*

$$\Pr\left[\mathsf{Exp}^{\mathsf{SEQ}}_{\mathcal{A},F,T} = 1\right] \leq \frac{1}{2},$$

where experiment $\mathsf{Exp}^{\mathsf{SEQ}}_{\mathcal{A}}$ *is depicted in Figure 1 and where the probability is taken over the random choice of the challenge.*

$\mathsf{Exp}^{\mathsf{SEQ}}_{\mathcal{A},F,T}$

$\sigma \leftarrow \mathcal{A}_0(F)$
$x \leftarrow [N]$
$y \leftarrow \mathcal{A}_1^F(\sigma, x)$
if $y = F^T(x)$
  return 1
else
  return 0

Fig. 1. The delay function sequentiality experiment

Definition 5 (Evaluation Trace). *Let* $T \in \mathbb{N}$*, let* $F : [N] \to [N]$*, and let* $x \in [N]$*. The evaluation trace of iterated function* F^T *on input* x *is defined as*

$$\textsc{Trace}(F^T, x) := \{x, F(x), F(F(x)), \ldots, F^T(x)\}.$$

3 Upper Bounds

In this section, we present several simple adversarial strategies. In Section 3.1, we show that when F is random and when the number of required iterations T is large enough, then exists an adversary that can compute F^T in $T/4$ steps using no preprocessing at all. In Section 3.2, we then go on to consider arbitrary round functions F. First we show in Theorem 3 that a sufficient amount of space allows for an adversary for arbitrary F that uses no parallelism, but only does $T/4$ sequential queries. We then show in Theorem 5 how to exploit parallelism to reduce the size of the required advice string at the cost of also now requiring the adversary to perform $3T/4$ sequential steps.

3.1 Random Round Functions

Definition 6. *Let* $F : [N] \to [N]$ *be a uniformly random function. Then for any* $T \geq 8 \cdot \sqrt{N}$*, the iterated function* F^T *is not* $(0, 1, 1/4)$*-secure, i.e. there exists a* $(0, 1, T/4)$*-admissible adversary* $\mathcal{A}$ *with*

$$\Pr\left[\mathsf{Exp}^{\mathsf{SEQ}}_{\mathcal{A},F,T} = 1\right] > \frac{1}{2}.$$

Proof. To prove the theorem, we will construct an adversary which correctly computes the output of F^T using at most $T/4$ many sequential queries to oracle F. The idea behind the attack is very simple. Consider the (directed) functional graph of F, i.e. the graph (V, E) with vertices $V := \{1, \ldots, N\}$ and edges $E := \{(u, v) \mid u \in \mathbb{N} \wedge F(u) = v\}$. Evaluating F on a point x corresponds to walking along an edge in the corresponding functional graph. Our attacker will simply hope that the given challenge x is either directly on a cycle or on a path into a cycle, such that walking $T/4$ steps on the graph starting from x will complete a full walk over the cycle. If this is the case, then the adversary

does not need to perform any more queries to the oracle F, since computing F^T just corresponds to running laps on the cycle that is known after $T/4$ sequential queries.

For an input $x \in [N]$, let E_x be the indicator variable that is one, if there exists $i \in [T/4]$ with $F^i(x) \in \textsc{Trace}(F^{i-1}, x)$, where $\textsc{Trace}(F^0, x) := x$. For x and $i \in [T/4]$, let $E_{x,i}$ be the indicator variable that is one, if $F^i(x) \in \textsc{Trace}(F^{i-1}, x)$. We observe that

$$\Pr[E_x] = \Pr\left[E_{x,1} \vee \cdots \vee E_{x,T/4}\right] = 1 - \Pr\left[\neg E_{x,1} \wedge \cdots \wedge \neg E_{x,T/4}\right].$$

Since F is a uniformly random function, it holds that

$$\begin{aligned}
&\Pr\left[\neg E_{x,1} \wedge \cdots \wedge \neg E_{x,T/4}\right] \\
&= \Pr[\neg E_{x,1}] \cdot \Pr[\neg E_{x,2} \mid \neg E_{x,1}] \cdots \Pr\left[\neg E_{x,T/4} \mid \neg E_{x,1} \wedge \cdots \wedge \neg E_{x,T/4-1}\right] \\
&= \prod_{i=1}^{T/4} \left(1 - \frac{i}{N}\right) \leq \prod_{i=1}^{T/4} e^{-i/N} = e^{-1/N \cdot \sum_{i=1}^{T/4} i} \leq e^{-\frac{T^2}{32N}}.
\end{aligned}$$

Plugging in the bound for T from the theorem statement, we get that

$$\Pr[E_x] \geq 1 - e^{-2} > 1/2.$$

Finally, we observe that, if E_x happens, then

$$F^T(x) \in \textsc{Trace}(F^T, x) = \textsc{Trace}(F^{T/4}, x),$$

and the adversary can simply retrieve the output $F^T(x)$ from the already computed partial trace, without querying the oracle F any more. □

3.2 Arbitrary Round Functions

Theorem 3. *For any $T \in \mathbb{N}$ and any $F : [N] \to [N]$, there exists a $(\frac{8N \lg N}{T}, 1, T/4)$-admissible adversary $\mathcal{A}$ with*

$$\Pr\left[\mathsf{Exp}^{\mathsf{SEQ}}_{\mathcal{A},F,T} = 1\right] > \frac{1}{2}.$$

Proof. To prove the theorem statement, we construct a $(\frac{8N \lg N}{T}, 1, T/4)$-admissible adversary $\mathcal{A} = (\mathcal{A}_0, \mathcal{A}_1)$. During preprocessing, adversary $\mathcal{A}_0$ picks $x_i \in [N]$ for $i \in [4N/T]$ independently and uniformly at random and stores $(x_i, F^{3T/4}(x_i))$. The advice string σ is the set of these pairs. The bit length of the advice string produced by $\mathcal{A}_0$ is $\frac{8N \lg N}{T}$.

Upon receiving a challenge $x \in [N]$, adversary $\mathcal{A}_1$ checks whether advice σ contains the pair $(x, F^{3T/4}(x))$. If σ contains the pair, then $\mathcal{A}_1$ uses $T/4$ sequential calls to oracle F to compute $F^T(x)$ from the stored value $F^{3T/4}(x)$. If σ does not contain the pair, then $\mathcal{A}_1$ computes $F(x)$ and checks whether σ contains $(F(x), F^{3T/4}(F(x)))$ and proceeds as above. $\mathcal{A}_1$ repeats this process until they match a pair in σ or until they compute $F^{T/4}(x)$ from x using $T/4$ sequential calls. If no pair was matched at this point, then $\mathcal{A}_1$ checks whether $\left|\textsc{Trace}(F^{T/4}, x)\right| < T/4+1$. If this is the case, then the evaluation trace contains repeated values, meaning that the iterated evaluation is in a cycle in the functional graph of F and thus $\mathcal{A}_1$ can compute $F^T(x)$ from the existing trace, without querying the oracle F any further. If, however, $\left|\textsc{Trace}(F^{T/4}, x)\right| = T/4 + 1$, then $\mathcal{A}_1$ outputs $\perp$ and aborts. It is clear that $\mathcal{A}_1$ either correctly computes $F^T(x)$ with $T/4$ sequential calls to F or outputs $\perp$.

Let us analyze the success probability of the adversary above. Let hit be the event that for some value $z \in \textsc{Trace}(F^{T/4}, x)$, it holds that $(z, F^{T/4}(z)) \in \sigma$ and without loss of generality, we now assume that $\left|\textsc{Trace}(F^{T/4}, x)\right| = T/4+1$, as the the adversary wins trivially otherwise. To win the sequentiality security experiment, the adversary needs

$$\Pr[\mathsf{hit}] > 1/2,$$

where the probability is taken over the random choices of $\mathcal{A}_0$ and the choice of the challenge x.

To analyze this probability, we observe that the choice of σ is independent of x and the corresponding evaluation trace. We can therefore consider the equivalent experiment of first picking x, computing the corresponding evaluation trace up to $T/4$, and then selecting pairs for σ uniformly at random. Then, we get that

$$\Pr[\neg\mathsf{hit}] < \left(1 - \frac{(T/4)}{N}\right)^{4N/T} \leq \left(e^{-T/4}\right)^{4/T} = e^{-1} < 1/2.$$

Thus it follows that the adversary $\mathcal{A}$ will be successful with probability strictly greater than half. □

Corollary 4. *Let $T \in \mathbb{N}$. For any iterated function $F^T : [N] \to [N]$ that is $(S, 1, 1/4)$-secure, it must hold that $S \cdot T \in \mathcal{O}(N \lg N)$.*

Let us now explore how the ability to do parallel queries can be exploited.

Theorem 5. *Let $T, P \in \mathbb{N}$ and let $S = 6\,(N/T - P) \lg N$. For any round function $F : [N] \to [N]$, there exists a $(S, P+1, 2T/3)$-admissible adversary $\mathcal{A}$ with*

$$\Pr\left[\mathsf{Exp}^{\mathsf{SEQ}}_{\mathcal{A},F,T} = 1\right] > \frac{1}{2}.$$

Proof. The to space constraints, the proof is deferred to Section A.

Corollary 6. *Let $T, P \in \mathbb{N}$. For any iterated function $F^T : [N] \to [N]$ that is $(S, P+1, 2/3)$-secure, it must hold that $(S + P) \cdot T \in \mathcal{O}(N \lg N)$.*

4 Lower Bound

After having established that adversaries with large enough advice and sufficient sequential and parallel time can win the iterated function security experiment with probability greater than $1/2$ no matter the round function F, let us now show the converse. That is, there exist functions F for which the adversary cannot evaluate F^T using $\widetilde{T} \leq T/2$ sequential oracle calls, unless they have many bits of advice and/or perform many calls in parallel.

For the purpose of this lower bound, it will be convenient to view the adversary $\mathcal{A}$ as a data structure D of $S \ll N$ bits, such that for a uniform random $x \in [N]$, it allows for quickly computing $F^T(x)$. We require D to return the correct result on a random input with probability at least $1/2$. We now show the following theorem.

Theorem 7. *For $N \geq c$, where $c \in \mathbb{N}$ is a large enough constant, there exists a functions $F : [N] \to [N]$, such that any data structure D that answers queries $F^T(x)$ on uniformly random inputs x correctly with probability at least $1/2$, must either have $\widetilde{T} \geq T/2$ or satisfy*

$$(S + \widetilde{T} \lg N) \max\{\widetilde{T} P \lg^5(\widetilde{T}), T\} = \Omega(N).$$

To prove the theorem statement, let us define a hard distribution over round functions F.

Hard Input Distribution. Let the round function $\mathbf{F}$ be obtained by drawing a uniform random permutation π with a single cycle over the elements $\{1, \ldots, N-1\}$ and letting $\mathbf{F}(0) = \pi(1)$, $\mathbf{F}(\mathbf{F}(0)) = \pi(2)$, $\mathbf{F}^k(0) = \pi(k)$ for $1 \leq k \leq N-1$ and $\mathbf{F}^N(0) = 0$. The function $\mathbf{F}$ is thus obtained by drawing a uniform random directed cycle on the elements/nodes $[N]$ and letting $\mathbf{F}(x)$ be the successor of the node x on the cycle.

Henceforth, we thus think of the problem as if we are given a uniform random directed cycle on the elements $[N]$ as input, and we get to store S bits. A query is a uniform random element $x \in [N]$. We then have $\widetilde{T}$ (sequential) rounds of evaluating the successor of P nodes. The goal is to determine the T-th successor of x along the cycle. We use $\mathbf{F}^i(x)$ to denote the i-th successor of x along the cycle.

Proof Framework. We now introduce the main ideas and general framework used to prove our lower bound. On an intuitive level, if $\widetilde{T} < T$, then the data structure has an insufficient number of rounds of oracle calls to $\mathbf{F}$ to merely follow the "successor pointers" on the cycle, starting from x, then $\mathbf{F}(x)$, $\mathbf{F}(\mathbf{F}(x))$ and so forth. Hence it must either "guess" intermediate nodes between x and $\mathbf{F}^T(x)$ by randomly querying elements in $[N]$, or it must use the S bits of advice to store "short cuts". The former strategy requires a large number of total queries, i.e. $\widetilde{T} \cdot P$ must be large. The latter strategy requires many advice bits, i.e. S must be large. To formalize this, we want to argue that without many queries and advice, the data structure must be very *lucky* if it is to correctly compute $\mathbf{F}^T(x)$. Turning this into a solid argument is however far from trivial. Our approach for doing so is based on an *encoding argument*, a technique which has been used to great success in data structure lower bounds, see e.g. [PD06, PV10, Lar12a, Lar12b, BL13, VZ13, CKL18, LS20, LY20, LLYZ23].

The general idea of an encoding argument is to show that an efficient data structure D implies an efficient binary encoding of a randomly chosen input cycle/function $\mathbf{F}$. We view such an encoding as consisting of two parts, an *encoder* and a *decoder*. The encoder receives $\mathbf{F}$ as input and must send a prefix-free message to the decoder. The decoder must be able to uniquely recover $\mathbf{F}$ from the message. If $H(\cdot)$ denotes binary Shannon entropy, then it follows from Shannon's source coding theorem that the expected length of the message must be at least $H(\mathbf{F}) = \lg((N-1)!)$ bits. The key step is now to show that a too efficient data structure D, can be used to give an encoding into fewer bits, yielding an information theoretic contradiction.

To simplify our proof, we make two further modifications to this framework. First, our lower bound holds also for randomized data structures D that errs with probability up to $1/2$ on any given input-query pair (F, x). By Yao's principle (fixing the random coins), this also implies the existence of a deterministic data structure with the same $\widetilde{T}, P, S$ that errs with probability at most $1/2$ over the random choice of $\mathbf{F}$ and a uniform random query $\mathbf{X} \in [N]$. We thus assume D is a deterministic data structure with this property.

Next, it will be convenient for the encoder and decoder to agree on a set of random queries. We thus let the encoder and decoder share access to B uniform random and independently chosen elements $\mathbf{X}_1, \ldots, \mathbf{X}_B \in [N]$ for a $B \leq N/(8\widetilde{T})$ to be chosen later. By independence, we have $H(\mathbf{F} \mid \mathbf{X}_1, \cdots, \mathbf{X}_B) = H(\mathbf{F}) = \lg((N-1)!)$. It follows that any prefix free encoding of $\mathbf{F}$ must use at least $\lg((N-1)!)$ bits in expectation, even if the encoder and decoder share access to $\mathbf{X}_1, \ldots, \mathbf{X}_B$.

4.1 Overall Encoding Scheme

With the encoding framework introduced above, we now present the overall ideas in our encoding scheme. Our strategy is to show that being able to answer the random queries $\mathbf{X}_1, \ldots, \mathbf{X}_B$ on $\mathbf{F}$

reveals $\tilde{\Omega}(B)$ bits of information about $\mathbf{F}$. If a data structure D has only S bits of advice, this will eventually lead to a contradiction if B is sufficiently larger than S. However for this claim to be true, we need that many of the queries $\mathbf{X}_1, \ldots, \mathbf{X}_B$ are answered correctly, and that the queries are *sufficiently different*. Intuitively, if $\mathbf{X}_i$ and $\mathbf{X}_j$ are two queries close to each other on the cycle defining $\mathbf{F}$, then answering both of them may reuse evaluations of $\mathbf{F}$ and thus reveal little or no new information. With these considerations in mind, we thus define an *interesting input*.

Definition 7. *We say that* $\mathbf{F}, \mathbf{X}_1, \ldots, \mathbf{X}_B$ *is an* interesting input *for a data structure* D *with* $\widetilde{T}$ *sequential queries, if there exists* $m \geq B/8$ *distinct indices* $i_1, \ldots, i_m \in \{1, \ldots, B\}$ *such that* $\mathbf{X}_{i_j}$ *is answered correctly by* D *on input* $\mathbf{F}$ *for every* i_j *and* $\mathbf{X}_{i_j} \neq \mathbf{F}^k(\mathbf{X}_h)$ *for any* $1 \leq k \leq 2\widetilde{T}$ *and* $h \neq i_j$*. We say such an index* i_j *is* useful*.*

Notice the crucial property of our definition requiring that $\mathbf{X}_{i_j} \neq \mathbf{F}^k(\mathbf{X}_h)$ for any $1 \leq k \leq 2\widetilde{T}$. This ensures that even if D evaluates $\mathbf{F}(\mathbf{X}_i)$ in round one for every i, then $\mathbf{F}(\mathbf{F}(\mathbf{X}_i))$ in round two and so forth for $\widetilde{T}$ rounds, there will be no overlap in the discovered nodes as the $\mathbf{X}_i$'s are spaced by at least $2\widetilde{T}$ on the cycle. We will later show that many inputs are interesting.

In our encoding, we will consider the function evaluations $\mathbf{F}(X)$ made by the data structure D implemented on $\mathbf{F}$ when the queries are $\mathbf{X}_1, \ldots, \mathbf{X}_B$. Let us denote by $\mathbf{Y}^i_1, \ldots, \mathbf{Y}^i_{B \cdot P} \in [N]$ the list of function evaluations made by D in the i-th sequential round when answering the queries $\mathbf{X}_1, \ldots, \mathbf{X}_B$. That is, in the i-th round, D evaluates $\mathbf{F}(\mathbf{Y}^i_1), \ldots, \mathbf{F}(\mathbf{Y}^i_{BP})$. We order the evaluations such that $(i, j) < (i', j')$ if either $i < i'$ or if $i = i'$ and $j < j'$.

With the interpretation of $\mathbf{F}$ as a cycle, each function evaluation $\mathbf{F}(\mathbf{Y}^i_j)$ made by D *discovers* one directed edge $(\mathbf{Y}^i_j, \mathbf{F}(\mathbf{Y}^i_j))$ on the cycle. We define the *view* $\mathbf{V}^i_j$ of D after evaluations $\mathbf{Y}^1_1, \ldots, \mathbf{Y}^i_{j-1}$ as all discovered edges $(\mathbf{Y}^{i'}_{j'}, \mathbf{F}(\mathbf{Y}^{i'}_{j'}))$ after the function evaluations $\mathbf{F}(\mathbf{Y}^{i'}_{j'})$ with $(1, 1) \leq (i', j') < (i, j)$.

The edges in a view $\mathbf{V}^i_j$ naturally reveal *chains* of nodes on the cycle corresponding to $\mathbf{F}$. The first element on a chain is called the *head*, and is the only element on the chain for which its predecessor in $\mathbf{F}$ has not yet been discovered (the predecessor of an X is the Y such that $\mathbf{F}(Y) = X$). The *tail* is likewise the only element on the chain for which its successor has not been discovered. For a node $X \in [N]$, we let $\mathbf{c}^i_j(X)$ denote the list of nodes on the chain containing X in $\mathbf{V}^i_j$. The list is ordered in the natural ordering with the head first and the tail last. Observe that in the view $\mathbf{V}^1_1$ corresponding to before any function evaluations are made, all nodes X reside in a singleton chain $\mathbf{c}^1_1(X) = X$. We need to also count the number of nodes on the chain from X and until the tail of the chain. Let $\mathbf{t}^i_j(X)$ denote this number of nodes. Note that $\mathbf{t}^i_j(X) = 1$ if X is the tail of $\mathbf{c}^i_j(X)$ and $\mathbf{t}^i_j(X) = |\mathbf{c}^i_j(X)|$ if X is the head.

With these definitions, we now take two alternative approaches to encoding $\mathbf{F}$ depending on the chains in the *final view* $\mathbf{V}^\infty := \mathbf{V}^{\widetilde{T}}_{BP+1}$. Note that the final view corresponds to when the data structure D has finished answering all queries $\mathbf{X}_1, \ldots, \mathbf{X}_B$. For an interesting input $\mathbf{F}, \mathbf{X}_1, \ldots, \mathbf{X}_B$, let $\mathcal{I} = \{\mathbf{I}_1, \ldots, \mathbf{I}_{B/8}\}$ be the first (i.e. of smallest values) $B/8$ useful indices. If at least half the indices $j \in \mathcal{I}$ have $|\mathbf{t}^\infty(\mathbf{X}_j)| := |\mathbf{t}^{\widetilde{T}}_{BP+1}(\mathbf{X}_j)| = |\mathbf{t}^{\widetilde{T}+1}_1(\mathbf{X}_j)| > 2\widetilde{T}$, then we say the final view has *long chains*. Otherwise, we say it has *short chains*. We present a different encoding in these two cases. The overall encoding strategy described above is shown as Algorithm 1, where $\mathrm{bin}(z)$ for an integer z promised to be in $[2^k]$ is a length-k binary encoding of the integer z and $a \circ b$ is the concatenation of binary strings a and b.

Algorithm 1: ENCODE$(F, X_1, \ldots, X_B, D)$

Input: Function $F : [N] \to [N]$ corresponding to a cycle, queries $X_1, \ldots, X_B \in [N]$, data structure D.
Result: Prefix free encoding of F (binary string).

Build D on F.
if $(F, X_1, \ldots, X_B)$ *is not an interesting input for* D **then**
  Interpret F as an integer f in $[(N-1)!]$.
  return $0 \circ \mathrm{bin}(f)$
Let a be the S bits of advice of D on input F.
Let $\mathcal{I} = \{I_1, \ldots, I_{B/8}\}$ be the first $B/8$ useful indices in $\{1, \ldots, B\}$.
Execute all queries $X_1, \ldots, X_B$ on D and compute views V_j^i for all $1 \le i \le \widetilde{T}$ and $1 \le j \le BP + 1$.
Let $L \subseteq \mathcal{I}$ be the set of indices $j \in \mathcal{I}$ such that $|t^\infty(X_j)| > 2\widetilde{T}$.
if $|L| < B/16$ **then**
  return $10 \circ a \circ$ ENCODESHORTCHAINS$(F, X_1, \ldots, X_B, \mathcal{I} \setminus L, D)$
else
  return $11 \circ a \circ$ ENCODELONGCHAINS$(F, X_1, \ldots, X_B, L, D)$

The encoding procedure is accompanied by a decoding/reconstruction procedure, shown as Algorithm 2. In the decoding procedure, we use the notation $b[i]$ for a binary string $b \in \{0,1\}^*$ to denote the i-th bit of b, with $b[0]$ being the first bit. We use the notation $b[i : j]$ to denote $b[i] \circ \cdots \circ b[j]$ and we use $b[i : \infty]$ to denote the suffix of b starting at the i-th bit.

Algorithm 2: DECODE$(b, X_1, \ldots, X_B, D)$

Input: binary string $b \in \{0,1\}^*$ with $b =$ENCODE$(F, X_1, \ldots, X_B, D)$ for some $F : [N] \to [N]$ corresponding to a cycle, queries $X_1, \ldots, X_B \in [N]$, data structure D.
Result: The function $F : [N] \to [N]$.

if $b[0] = 0$ **then**
  Interpret $b[1 : \infty]$ as an integer f in $[(N-1)!]$.
  return F *corresponding to* f
Let $a = b[2 : 1 + S]$.
if $b[1] = 0$ **then**
  return DECODESHORTCHAINS$(b[2 + S, \infty], a, X_1, \ldots, X_B, D)$
else
  return DECODELONGCHAINS$(b[2 + S, \infty], a, X_1, \ldots, X_B, D)$

In the next sections, we give the two encoding and decoding procedures for the cases of short and long chains. The number of bits used by the two encoding procedures is shown in the following the lemmas.

Lemma 8. *If* $N \ge 2^7$, $B\widetilde{T}P \le N/4$ *and* $B(T+1) \le N$, *then the encoding length of* ENCODESHORTCHAINS *is no more than*

$$\lg((N-1)!) + 6 + 2\lg N - B/16$$

bits.

Lemma 9. *If* $B\widetilde{T}P\lg^5(4\widetilde{T}) \le N/2^{20}$, *then the encoding length of* ENCODELONGCHAINS *is no more than*

$$\lg((N-1)!) + 1 + \widetilde{T}(1 + \lg N) - B/32$$

bits.

We will prove the two results in the following sections and instead proceed to analyse the expected length of the encoding and derive the lower bound.

First, observe that Encode adds $2 + S$ bits to encoding produced by either EncodeShortChains or EncodeLongChains. The savings of $B/16$ or $B/32$ in Lemma 8 and Lemma 9 must be enough to make up for this as well as the additional bits in the two lemmas. We thus set $B = 32(S + \widetilde{T}(1 + \lg N) + 22)$. We conclude that either $(S + \widetilde{T} \lg N)\widetilde{T}P \lg^5(\widetilde{T}) = \Omega(N)$, or $(S + \widetilde{T} \lg N)T = \Omega(N)$, or both encodings use no more than $\lg((N-1)!) - 22$ bits.

Assume for the sake of contradiction that the encodings use no more than $\lg((N-1)!) - S - 22$ bits. If E denotes the event that the random input $\mathbf{F}, \mathbf{X}_1, \ldots, \mathbf{X}_B$ is interesting, the expected length of the encoding is the bounded by

$$\begin{aligned} &1 + (1 - \Pr[E])(1 + \lg((N-1)!)) + \Pr[E](S + \lg((N-1)!) - S - 22) \\ \leq &2 + \lg((N-1)!) - \Pr[E]22. \end{aligned}$$

We now invoke the following observation to show that the input is interesting with good probability:

Observation 8. *For any data structure D, the random input $\mathbf{F}, \mathbf{X}_1, \ldots, \mathbf{X}_B$ is an interesting input with probability at least $1/7$.*

Inserting this bound for $\Pr[E]$ gives an expected encoding length of at most $2 + \lg((N-1)!) - 22/7 < \lg((N-1)!)$, i.e. a contradiction. We thus conclude that either $(S + \widetilde{T} \lg N)\widetilde{T}P \lg^5(\widetilde{T}) = \Omega(N)$, or $(S + \widetilde{T} \lg N)T = \Omega(N)$. This proves Theorem 7.

We conclude this section by proving the above observation regarding the probability of obtaining an interesting input.

Proof (of Observation 8). Consider a fixed index $i \in \{1, \ldots, B\}$. Since $\mathbf{X}_i$ is uniform random in $[N]$, it follows from the guarantees of D that $\mathbf{X}_i$ is answered correctly with probability at least $1/2$ over the choice of $\mathbf{F}$. At the same time, conditioned on $\mathbf{X}_1, \ldots, \mathbf{X}_{i-1}, \mathbf{X}_{i+1}, \ldots, \mathbf{X}_B$, we have that $\mathbf{X}_i$ is still uniform random in N. Hence $\mathbf{X}_i \neq \mathbf{F}^k(\mathbf{X}_h)$ for all $1 \leq k \leq 2p$ and $h \neq i$ with probability at least $(N - 2B\widetilde{T})/N = 1 - 2B\widetilde{T}/N \geq 1 - 1/4$. A union bound implies that $\mathbf{X}_i$ is both answered correctly and has $\mathbf{X}_i \neq \mathbf{F}^k(\mathbf{X}_h)$ for all $1 \leq k \leq t$ and $h \neq i$ with probability at least $1/4$ (i.e. i is useful). The expected number of useful i is thus at least $B/4$. This also implies that the expected number of i that is not useful is no more than $3B/4$. By Markov's inequality, the probability that there are more than $7B/8$ i that are not useful is thus at most $(3/4)/(7/8) = 6/7$.

4.2 Encoding with Short Chains

In this section, we describe the encoding and decoding procedures for the case when the final view has short chains. Recall that this implies the existence of a subset $\mathcal{I}' \subseteq \mathcal{I}$ with $|\mathcal{I}'| = B/16$ such that for all $j \in \mathcal{I}'$, we have $|t^\infty(X_j)| \leq 2\widetilde{T}$. The key observation is that for every $j \in \mathcal{I}'$, the chain $t^\infty(X_j)$ containing X_j in the final view is different from the chain $t^\infty(F^T(X_j))$ containing the answer $F^T(X_j)$ to the query X_j. This is because T is assumed larger than $2\widetilde{T}$ and less than $2\widetilde{T}$ nodes succeed X_j on its chain. That they are on different chains intuitively means that the answer to the query X_j reveals new information not discovered while evaluating the function F. This is precisely what we exploit to compress F. In our encoding, we assume an ordering on the chains in a view V_j^i. The concrete ordering is not important, as long as it is consistent with the ordering used in decoding. One choice could be to order them based on the indices of the head of the chains, i.e. a smaller index head means earlier in the ordering.

The encoding procedure is shown as Algorithm 3 and we will describe the ideas in the encoding by refering to the pseudocode.

Algorithm 3: EncodeShortChains$(F, X_1, \ldots, X_B, \mathcal{I}, D)$

Input: Function $F : [N] \to [N]$ corresponding to a cycle, queries $X_1, \ldots, X_B \in [N]$, indices $\mathcal{I} \subseteq \{1, \ldots, B\}$ with $|\mathcal{I}| \geq B/16$, data structure D.
Result: Prefix free encoding of F (binary string).

Build D on F.
Execute all queries $X_1, \ldots, X_B$ on D and compute views V_j^i for all $1 \leq i \leq \widetilde{T}$ and $1 \leq j \leq BP$.
Let $\mathcal{I}' \subseteq \mathcal{I}$ be the first $B/16$ indices in $\mathcal{I}$.
Let $b \leftarrow \mathrm{bin}(\mathcal{I}')$ be a binary encoding of $\mathcal{I}'$ as a subset of the indices $\{1, \ldots, B\}$ using $\lceil \lg \binom{B}{B/16} \rceil$ bits.
Let $m \in [N]$ be the number of chains in V^∞. Update $b \leftarrow b \circ \mathrm{bin}(m)$.
Let L be an initially empty list of indices.
for $i = 1, \ldots, \widetilde{T}$ **do**
  for $j = 1, \ldots, BP$ **do**
    if Y_j^i *is the tail of the chain* $c_j^i(Y_j^i)$ **then**
      Append the rank r_j^i of $c_j^i(F(Y_j^i))$ among all chains in $V_j^i \setminus \{c_j^i(Y_j^i)\}$ to L.
Interpret the list L as a number $\ell \in [(N-1)(N-2)\cdots m]$ and update $b \leftarrow b \circ \mathrm{bin}(\ell)$.
Let $V^\star \leftarrow V^\infty$ equal the final view and $c^\star(X)$ denote the chain in $V^\star$ containing a node $X \in [N]$.
Let K be an initially empty list of indices.
for $i = 1, \ldots, B/16$ **do**
  Let j be the i-th index in $\mathcal{I}'$.
  while $c^\star(X_j) \neq c^\star(F^T(X_j))$ **do**
    Let $x \leftarrow t^\star(X_j)$ denote the number of nodes from X_j and until the tail of $c^\star(X_j)$.
    Let $y \leftarrow |c^\star(F^T(X_j))| - t^\star(F^T(X_j))$ denote the number of nodes from the head and up just before $F^T(X_j)$ in $c^\star(F^T(X_j))$.
    if $x + y = T$ **then**
      Add the edge from the tail of $c^\star(X_j)$ to the head of $c^\star(F^T(X_j))$ and merge the chains in $V^\star$.
    else
      Append the rank in $V^\star \setminus \{c^\star(X_j)\}$ of the chain whose head is the successor of the tail in $c^\star(X_j)$ to K. Then merge the two chains in $V^\star$ by inserting the corresponding edge.
Update $b \leftarrow b \circ \mathrm{bin}(|K|)$ with $|K| \in [N]$.
Interpret the list K as a number $k \in [(m-1)(m-2)\cdots(m-|K|)]$ and update $b \leftarrow b \circ \mathrm{bin}(k)$.
$b \leftarrow b \circ$ EncodeRemainingView$(F, V^\star)$.
return b

Algorithm 4: EncodeRemainingView$(F, V^\star)$

Input: $F : [N] \to [N]$ corresponding to a cycle, view $V^\star$ consistent with F.
Result: Prefix free encoding of F from $V^\star$.

Let L be an initially empty list of indices.
Let $n \leftarrow |V^\star| - 1$.
for $i = 1, \ldots, n$ **do**
  Let c_1 be the chain in $V^\star$ with the smallest rank.
  Append the rank in $V^\star \setminus \{c_1\}$ of the chain whose head is the successor of the tail in c_1 to L. Then merge the two chains in $V^\star$ by inserting the corresponding edge.
Interpret the list L as a number $\ell \in [(|V^\star| - 1)(|V^\star| - 2)\cdots 1]$.
return $\mathrm{bin}(\ell)$

The first part of EncodeShortChains (Algorithm 3) constructs the data structure D and executes the queries to compute the different views V_j^i (lines 1-2). We then select the first $B/16$ indices $\mathcal{I}' \subseteq \mathcal{I}$. Recall that when Encode invokes EncodeShortChains, the list $\mathcal{I}$ contains indices j of queries that are answered correctly and where $|t^\infty(X_j)| \leq 2\widetilde{T} < T$. In line 4 we then encode these indices $\mathcal{I}'$ such that the decoder will be able to recover these indices.

In lines 5-11 we now encode the final view V^∞ such that the decoder may reconstruct this view. The way we encode it, is to simply write down the edges discovered while D performs the function evaluations $F(Y_j^i)$ during its execution. This will allow the decoder to simulate the execution of D as well. Up to this point, there is no saving of bits in the encoding.

Lines 12-23 if where we finally exploit that D managed to answer the queries while not connecting the chains $c^\infty(X_j)$ and $c^\infty(F^T(X_j))$. Concretely, the for-loop iterations over all the queries in $\mathcal{I}'$ in turn. For each one, it repeatedly merges chains (lines 16-22) until X_j and $F^T(X_j)$ are on the same chain. The crucial point is that when the number of elements succeeding X_j on its chain, plus the number of elements up to and including $F^T(X_j)$ on its chain equals T, then we know that there must be an edge from the tail of the chain containing X_j to the head of the other chain. This saves us the encoding of an edge, i.e. the information is contained in the answer to the query. The second crucial point is that we save the encoding of one edge for every single index in $\mathcal{I}'$. This is because the indices in $\mathcal{I}'$ are *useful* and thus spaced by at least T. Furthermore, all edges inserted in the while-loop for a fixed X_j are inserted between X_j and $F^T(X_j)$.

Finally, we complete the encoding in line 25 by invoking EncodeRemainingView. This process simply encodes how to merge the remaining chains using $\lceil \lg((|V^\star| - 1)!) \rceil$ bits for a view $V^\star$ with $|V^\star|$ many chains.

With the above in mind, the decoding procedure is shown as Algorithm 5.

We briefly discuss it while refering to the pseudocode. In lines 1-3, we read off $\mathcal{I}'$, $|V^\infty|$ and the list L produced by EncodeShortChains. Lines 4-9 now simulates the execution of the data structure D on queries $X_1, \ldots, X_B$ on the function F. Note that this is done without direct access to F. Instead, since the decoding procedure is provided with the advice bits a, it can first determine all function evaluations that D would make in the first sequential round. That is, we know $Y_1^1, \ldots, Y_{BP}^1$. Now in order to simulate D, we need to know the answer to all the function evaluations $F(Y_i^1)$. But this information is precisely what is stored in L. Once we know the results of all function evaluation in the first sequential round, we can continue simulating D for the second round and so forth until completion. This obtains the final view V^∞. Furthermore, and critically, we also finish simulating D on all the queries. Since all queries in $\mathcal{I}'$ are answered correctly by D, the decoding procedure now knows $F^T(X_j)$ for every $j \in \mathcal{I}'$ (line 10).

In lines 11-21 we now re-execute all the merges of chains that EncodeShortChains performed (in lines 14-22 of that procedure). The key observation here is that we know the query answers and thus can still carry out the test $x + y = T$?. We thus reconstruct $V^\star$ as it was during encoding, just before invoking EncodeRemainingView. We feed this $V^\star$ to DecodeRemainingView, which merely reads off the merges necessary to reduce the number of chains in $V^\star$ to one. This also uniquely determines the function F.

We have thus argued correctness of our encoding and decoding procedure for handling views with short chains. Let us next analyse the size of the encoding.

Algorithm 5: DecodeShortChains$(b, a, X_1, \ldots, X_B, D)$

Input: binary string $b \in \{0,1\}^*$ with $b =$EncodeShortChains$(F, a, X_1, \ldots, X_B, \mathcal{I} \setminus L, D)$ for some $F : [N] \to [N]$ corresponding to a cycle, advice a of data structure D on input F, queries $X_1, \ldots, X_B \in [N]$, data structure D.

Result: The function $F : [N] \to [N]$.

Read and remove first $\lceil \lg \binom{B}{B/16} \rceil$ bits of b to recover $\mathcal{I}'$.
Read and remove first $\lceil \lg N \rceil$ bits of b to recover $m = |V^\infty|$.
Read and remove first $\lceil \lg((N-1)!/(m-1)!) \rceil$ bits of b to recover L.
for $i = 1, \ldots, \widetilde{T}$ **do**
    From a, queries $X_1, \ldots, X_B$, $Y_j^{i'}$ and $F(Y_j^{i'})$ with $i' < i$, simulate D to determine all Y_j^i, i.e. all function evaluation made by D in sequential round i. This also determines the view V_{BP+1}^{i-1} (end of round $i-1$).
    for $j = 1, \ldots, BP$ **do**
        if *Y_j^i is the tail of the chain $c_j^i(Y_j^i)$* **then**
            Read and remove the first element r_j^i of L. We learn that $F(Y_j^i)$ is the head of the chain of rank r_j^i in $V_j^i \setminus \{c_j^i(Y_j^i)\}$. We can thus compute $F(Y_j^i)$ and the next view V_{j+1}^i.
Let $V^\star \leftarrow V^\infty$ equal the final view.
From simulating D on queries X_j, compute $F^T(X_j)$ for every $j \in \mathcal{I}'$.
Read and remove first $\lceil \lg N \rceil$ bits to recover $|K|$.
Read and remove first $\lceil \lg((m-1)!/(m-|K|-1)!) \rceil$ bits to recover K.
for $i = 1, \ldots, B/16$ **do**
    Let j be the i-th index in $\mathcal{I}'$.
    while $c^\star(X_j) \neq c^\star(F^T(X_j))$ **do**
        Let $x \leftarrow t^\star(X_j)$.
        Let $y \leftarrow |c^\star(F^T(X_j))| - t^\star(F^T(X_j))$.
        if $x + y = T$ **then**
            Add the edge from the tail of $c^\star(X_j)$ to the head of $c^\star(F^T(X_j))$ and merge the chains in $V^\star$.
        else
            Remove the first element from K. This determines the chain whose head is the successor of the tail in $c^\star(X_j)$. Then merge the two chains in $V^\star$ by inserting the corresponding edge.
return DecodeRemainingView$(b, V^\star)$

Algorithm 6: DecodeRemainingView$(b, V^\star)$

Input: binary string $b \in \{0,1\}^*$, view $V^\star$ consistent with F.

Result: The function $F : [N] \to [N]$.

Read the $\lceil \lg((|V^\star|-1)! \rceil$ bits in b to recover L.
Let $n \leftarrow |V^\star| - 1$.
for $i = 1, \ldots, n$ **do**
    Let c_1 be the chain in $V^\star$ with the smallest rank.
    Remove the first element r of L and interpret it as the rank in $V^\star \setminus \{c_1\}$ of the chain whose head is the successor of the tail in c_1 to L. Then merge the two chains in $V^\star$ by inserting the corresponding edge.
return *The unique function F consistent with the single chain in $V^\star$*

Encoding Size with Short Chains. Examining the EncodeShortChains procedure, we see that it writes

$$\begin{aligned}
&\left\lceil \lg \binom{B}{B/16} \right\rceil + 2\lceil \lg N \rceil + \left\lceil \lg \left(\frac{(N-1)!}{(m-1)!} \right) \right\rceil + \left\lceil \lg \left(\frac{(m-1)!}{(m-|K|-1)!} \right) \right\rceil \\
&+\lceil \lg((|V^\star|-1)!) \rceil \\
\leq & 6 + \lg \binom{B}{B/16} + 2\lg N + \lg \left(\frac{(N-1)!}{(m-1)!} \right) + \lg \left(\frac{(m-1)!}{(m-|K|-1)!} \right) \\
&+\lg((|V^\star|-1)!) \\
\leq & 6 + (B/16)\lg(16e) + 2\lg N + \lg \left(\frac{(N-1)!}{(m-|K|-1)!} \right) + \lg((|V^\star|-1)!)
\end{aligned}$$

bits, where $V^\star$ here denotes the $V^\star$ that is passed to the procedure EncodeRemainingView. Now observe that $|V^\star| = m - |K| - B/16$. This is because the loop in lines 13-21 of EncodeShortChains performs precisely $|K|$ merges that add an element to K and one merge for each index $i \in \mathcal{I}'$ which does not add to K. As mentioned earlier, this is precisely where our saving comes from. The above is thus upper bounded by

$$\begin{aligned}
&6 + (B/16)\lg(16e) + 2\lg N + \lg \left(\frac{(N-1)!}{(m-|K|-1)!} \right) \\
&+\lg((m-|K|-B/16-1)!) \\
&\leq \lg((N-1)!) + 6 + (B/16)\lg(16e) + 2\lg N - (B/16)\lg(m-|K|-B/16).
\end{aligned}$$

Since m is the number of chains in the final view V^∞ and D performs at most $B\widetilde{T}P$ function evaluations throughout the encoding procedure, we have $m \geq N - B\widetilde{T}P$. At the same time, we have that the loop in lines 14-22 of EncodeShortChains only adds elements to K when merging chains within the first T steps of X_j on the cycle for $j \in \mathcal{I}'$. Hence $|K| \leq (B/16)T$. We thus get a number of bits of at most

$$\lg((N-1)!) + 6 + 2\lg N - (B/16)\lg((N - B\widetilde{T}P - (B/16)(T+1))/(16e)).$$

If $N \geq 2^7$, $B\widetilde{T}P \leq N/4$ and $B(T+1) \leq N$, then $\lg((N - B\widetilde{T}P - (B/16)(T+1))/(16e)) \geq \lg((N/2)/(16e)) \geq 1$ and the above is at most $\lg((N-1)!) + 6 + 2\lg N - (B/16)$. This completes the proof of Lemma 8.

4.3 Encoding with Long Chains

In this section, we describe the encoding and decoding procedures for the case when the final view has long chains. Recall that this implies the existence of a subset $\mathcal{I}' \subseteq \mathcal{I}$ with $|\mathcal{I}'| = B/16$ such that for all $j \in \mathcal{I}'$, we have $|t^\infty(X_j)| > 2\widetilde{T}$. Our main observation is that chains may only grow this long if some of the function evaluations $F(Y_j^i)$ made by D return nodes on the cycle that reside on chains of length longer than 1. Since there are no more than $B\widetilde{T}P$ many chains of length longer than 1, this allows us to save bits in encoding such $F(Y_j^i)$. Unlike in the short chains case, the savings in compression size thus come from encoding the view V^∞ resulting from executing the queries $X_1, \ldots, X_n$ on D.

Algorithm 7: EncodeLongChains$(F, X_1, \ldots, X_B, \mathcal{I}, D)$

Input: Function $F : [N] \to [N]$ corresponding to a cycle, queries $X_1, \ldots, X_B \in [N]$, indices $\mathcal{I} \subseteq \{1, \ldots, B\}$ with $|\mathcal{I}| \geq B/16$, data structure D.
Result: Prefix free encoding of F (binary string).

Build D on F.
Execute all queries $X_1, \ldots, X_B$ on D and compute views V_j^i for all $1 \leq i \leq \widetilde{T}$ and $1 \leq j \leq BP$.
Let $\mathcal{I}' \subseteq \mathcal{I}$ be the first $B/16$ indices in $\mathcal{I}$.
Let $b \leftarrow \text{bin}(\mathcal{I}')$ be a binary encoding of $\mathcal{I}'$ as a subset of the indices $\{1, \ldots, B\}$ using $\lceil \lg \binom{B}{B/16} \rceil$ bits.
Let $\ell_j^i \leftarrow \min\{2\widetilde{T} + 1, t_1^{i+1}(X_j)\} - \min\{2\widetilde{T} + 1, t_1^i(X_j)\}$ for each $i \in \{1, \ldots, \widetilde{T}\}$ and $j \in \{1, \ldots, B\}$.
for $j \in \mathcal{I}'$ **do**
    Compute an index $h_j \in \{0, \ldots, \lg(\widetilde{T}) - 3\}$ such that there are at least $\Delta_{h_j} := 2^{h_j}/(h_j + 1)^2$ indices q with $\ell_j^q - 1 \geq \widetilde{T}/2^{h_j+3}$ (such h_j always exists).
    Let $S_{h_j} \subseteq \{1, \ldots, \widetilde{T}\}$ be the first Δ_{h_j} indices q with $\ell_j^q - 1 \geq \widetilde{T}/2^{h_j+3}$.
    Update $b \leftarrow b \circ \text{bin}(h_j) \circ \text{bin}(S_{h_j})$, where $\text{bin}(S_{h_j})$ is encoded as a Δ_{h_j}-sized subset of $\{1, \ldots, \widetilde{T}\}$.
for $i = 1, \ldots, \widetilde{T}$ **do**
    Let L_i be an initially empty list of indices.
    Let $V' \leftarrow V_1^i$ and use $c'(X)$ and $t'(X)$ as the equivalent definitions of c_j^i and t_j^i for the view V'.
    for $j \in \mathcal{I}'$ **do**
        if $i \in S_{h_j}$ **then**
            while $t'(X_j) \leq 2\widetilde{T}$ *and the tail of* $c'(X_j)$ *equals* Y_k^i *for some* k **do**
                Let $z \leftarrow |c'(F(Y_k^i))|$.
                $b \leftarrow b \circ \text{binAdapt}(\lceil \lg z \rceil)$.
                if $z > 1$ **then**
                    Let $r \in [\lfloor B\widetilde{T}P/2^{\lceil \lg z \rceil - 1} \rfloor]$ be the rank of the chain $c'(F(Y_k^i))$ among all chains c in V' with $|c| \in (2^{\lceil \lg z \rceil - 1}, 2^{\lceil \lg z \rceil}]$.
                    $b \leftarrow b \circ \text{bin}(r)$.
                else
                    if *the singleton chain* $c'(F(Y_k^i)) = F(Y_k^i)$ *has* $Y_\ell^i = F(Y_k^i)$ *for some* $\ell \in [BP]$ **then**
                        $b \leftarrow b \circ 0 \circ \text{bin}(\ell)$.
                    else
                        $b \leftarrow b \circ 1 \circ \text{bin}(F(Y_k^i))$ with $F(Y_k^i) \in [N]$.
                Add the edge from the tail of $c'(X_j)$ to $F(Y_k^i)$ and merge the chains in V'.
    for $j = 1, \ldots, BP$ **do**
        if Y_j^i *is the tail of the chain* $c'(Y_j^i)$ **then**
            Append the rank r_j^i of $c'(F(Y_j^i))$ among all chains in $V' \setminus \{c'(Y_j^i)\}$ to L_i.
    Let $m_i \in [N]$ denote $|L_i|$ and update $b \leftarrow b \circ \text{bin}(m_i)$.
    Interpret the list L_i as a number $\ell_i \in [(|V'| + m_i - 1)(|V'| + m_i - 2) \cdots |V'|]$ and update $b \leftarrow b \circ \text{bin}(\ell_i)$.
$b \leftarrow b \circ$ EncodeRemainingView(F, V^∞).
return b

We will next go over the steps of the encoding procedure while referring to the pseudocode shown in Algorithm 7.

Lines 1-4 are somewhat uneventful. Here we simply start by executing the queries $X_1, \ldots, X_B$ on D to compute the views V_j^i. We then pick the first $B/16$ indices $\mathcal{I}'$ among the indices $\mathcal{I}$ given as argument. Recall that ENCODE invokes ENCODELONGCHAINS with $\mathcal{I}$ being a set of useful queries X_j with $|t^\infty(X_j)| > 2\widetilde{T}$. These are queries in which the data structure managed to grow the length of its chain by much more than one per sequential round (there are $\widetilde{T}$ sequential rounds). The encoding will exploit this and show that such long chains may be used to compress F.

In line 5, we compute the increase in the number of nodes succeeding X_j in its chain during sequential round i. The two min-expression ensures that we will only count an increase in number of nodes while the chain extends no more than $2\widetilde{T}$ beyond X_j. The reason for this cap, is that since the queries $j \in \mathcal{I}'$ are useful, they are spaced by at least this much on the cycle. This will guarantee that all edges/function evaluations that we exploit to compress F are disjoint for different X_j's.

In lines 6-9, we iterate over the useful $j \in \mathcal{I}'$. For each of these, we look for a "typical" increase in the number of nodes on X_j's chain. Concretely, we determine the indices h_j in line 7, guaranteeing that there are at least $\Delta_{h_j} := 2^{h_j}/(h_j+1)^2$ rounds where the chain length grows by at least $\widetilde{T}/2^{h_j+3} + 1$. At the end of this section, we will show that such an index h_j always exists:

Lemma 10. *If $t_1^{\widetilde{T}+1}(X_j) > 2\widetilde{T}$, then there exists an index $h_j \in \{0, \ldots, \lg(\widetilde{T}) - 3\}$ such that there are at least $\Delta_{h_j} := 2^{h_j}/(h_j+1)^2$ indices i with $\min\{2\widetilde{T}+1, t_1^{i+1}(X_j)\} - \min\{2\widetilde{T}+1, t_1^i(X_j)\} \geq \widetilde{T}/2^{h_j+3} + 1$.*

We then encode a number of sequential rounds in which the chain of X_j grows by at least $\widetilde{T}/2^{h_j+3}+1$. The reason for encoding such rounds, is that it is expensive (bit-wise) to encode a round in which the chain grows. Thus the chain has to grow sufficiently much to afford encoding the round. The exact details will be clearer when we analyse the encoding length.

In lines 10-31, we now wish to encode the views V_1^i at the beginning of each sequential round. The overall goal with these steps, is to allow the decoder to recover V^∞. The encoding of sequential round i begins with going over the useful indices in $\mathcal{I}'$. For each such j, if sequential round i was one of the rounds where the increase in X_j's chain length is sufficiently large, we encode the chain merges that X_j is involved in. In particular, lines 15-26 encode all function evaluations $F(Y_k^i)$ that grow the chain of X_j. For each such function evaluation, we check the length of the chain $c'(F(Y_k^i))$ that we merge with. This length is precisely the increase in the length of X_j's chain. If we merge with a non-singleton ($z > 1$) chain, we will save bits by encoding the chain $c'(F(Y_k^i))$ as an index into the set of chains of length in $(2^{\lceil \lg z \rceil - 1}, 2^{\lceil \lg z \rceil}]$. Since any chain with a length in this interval has at least $2^{\lceil \lg z \rceil - 1}$ many edges and D makes no more than $B\widetilde{T}P$ function evaluations in total, the number of such chains is no more than $\lfloor B\widetilde{T}P/2^{\lceil \lg z \rceil - 1} \rfloor$. The crucial point is that the number of bits we spend on encoding the chain drops with the increase z in chain length. If instead we merge with a singleton chain, we check whether that chain is itself involved in a merge in sequential round i (line 22). If so, then we know that the head/tail of the singleton chain is one of the Y_ℓ^i. Since there are only BP function evaluations per sequential round, this allows us to save in the encoding length. Finally, if the singleton is not involved in a merge, we pay an expensive encoding of the chain (line 25). Fortunately, this will be made up for by the savings in other steps of the encoding.

Notice that in line 17, we encode the value $\lceil \lg z \rceil$ above using a special binAdapt(x) encoding. This encoding is prefix-free and uses a number of bits proportional to the value $x \geq 0$. It starts with $\lceil \lg(x+1) \rceil$ 0's, followed by a 1. This gives a unary encoding of $\lceil \lg(x+1) \rceil$. It then has $\lceil \lg(x+1) \rceil$

bits giving a binary encoding of x. Decoding such a number is straight forward: read bits until the first 1 is encountered. This recovers $\lceil \lg(x+1) \rceil$. Then read the next $\lceil \lg(x+1) \rceil$ bits to recover recover x. The number of bits used to encode an integer $x \geq 0$ is thus no more than

$$2\lceil \lg(x+1) \rceil \leq 2 + 2\lg(x+1).$$

In lines 27-29 we complete the encoding of all merges taking place in sequential round i. This is done by going over all Y_j^i that still correspond to a tail of a chain and merely encoding the successor $F(Y_j^i)$ as an index into the remaining chains. This step will make no savings in bits, but also will not lose any. The crucial point is that upon completing the encoding of one sequential round i, we have encoded all merges that occur in the round. This will allow the decoder to determine the view at the beginning of round $i+1$.

Finally, once all $\widetilde{T}$ sequential rounds have been completed, we encode the merges necessary for completing the final view V^∞ to a single chain (similarly to the short chains case).

The corresponding decoding procedure, DecodeLongChains, shown as Algorithm 8 simply reconstructs the views V_1^i in the order produced by the encoding. For completeness, we here outline the steps. First, in lines 1-4, we recover $\mathcal{I}'$, h_j and S_{h_j} for each $j \in \mathcal{I}'$. With this information, the decoding procedure can now simulate the i sequential rounds of D. In round i, we start by processing all $j \in \mathcal{I}'$ for which $i \in S_{h_j}$. Here we read off (lines 11-23) all the function evaluations/chain merges that extends the chain of X_j in round i (unless it grows beyond $2\widetilde{T}$ nodes succeeding X_j). In lines 24-28 we then decode all other function evaluations in round i. Note that decoding crucially uses in line 8 that the decoder has access to the advice a, the queries $X_1, \ldots, X_B$ and the view V_1^i. Since the view V_1^i contains all edges corresponding to function evaluations in previous sequential rounds, the decoding procedure can thus simulate D up to round i and determine all function evaluations $F(Y_j^i)$ made by D in round i. After processing sequential round i, we have recovered all function evaluations made by D in round i and thus V_1^{i+1} equals V' (as updated in line 29). Finally, we use the procedure from decoding short chains, DecodeRemainingView (Algorithm 6), to recover F.

Encoding Size with Long Chains. Analysing the encoding length with long chains is a bit more tricky than with short chains.

Examining EncodeLongChains in Algorithm 7, we first have $\lceil \lg \binom{B}{B/16} \rceil$ bits to encode $\mathcal{I}'$. For each $j \in \mathcal{I}'$, we use $\lceil \lg(\lg(\widetilde{T}) - 2) \rceil$ bits to encode h_j and $\lceil \lg \binom{\widetilde{T}}{\Delta_{h_j}} \rceil$ bits for S_{h_j}. We will argue that the remaining part of the encoding makes savings that can pay for these costs.

First, observe that upon completion of lines 10-31 in Algorithm 7, we use $\lceil \lg((|V^\infty|-1)!) \rceil$ bits to encode the merges of the chains in V^∞. Encoding the lists L_i costs $\lceil \lg((|V_i'|+m_i-1)!/(|V_i'|-1)!) \rceil$, where V_i' is the value of V' upon reaching line 31 in iteration i of the for-loop. We also pay $\lceil \lg N \rceil$ to encode $|L_i|$ for each i. The key point is that if we ignore the rounding $\lceil \cdot \rceil$, then all factors in $\lg((|V^\infty|-1)!) + \sum_i \lg((|V_i'|+m_i-1)!/(|V_i'|-1)!)$ correspond to terms in $\lg((N-1)!)$. Furthermore, there is precisely one term in $(N-1)!$ missing for every edge added in line 26. Moreover, each of these terms is at least $|V^\infty| \geq N - B\widetilde{T}P$ large. If a total of Z edges are added in line 26, we then have

$$\lg((|V^\infty|-1)!) + \sum_i \lg((|V_i'|+m_i-1)!/(|V_i'|-1)!) \leq \lg((N-1)!) - Z\lg(N - B\widetilde{T}P).$$

Let us now partition these Z edges depending on the index $j \in \mathcal{I}'$ and $i \in S_{h_j}$ causing us to add the edges in line 26. That is, we let Z_j^i denote the number of edges added in line 26 for a fixed

Algorithm 8: DecodeLongChains$(b, a, X_1, \ldots, X_B, D)$

Input: binary string $b \in \{0,1\}^*$ with $b =$EncodeLongChains$(F, a, X_1, \ldots, X_B, \mathcal{I} \setminus L, D)$ for some $F : [N] \to [N]$ corresponding to a cycle, advice a of data structure D on input F, queries $X_1, \ldots, X_B \in [N]$, data structure D.

Result: The function $F : [N] \to [N]$.

Read and remove first $\lceil \lg \binom{B}{B/16} \rceil$ bits of b to recover $\mathcal{I}'$.
for $j \in \mathcal{I}'$ **do**
  Read and remove first $\lceil \lg(\lg(\widetilde{T}) + 2) \rceil$ bits to recover h_j.
  Read and remove first $\lceil \lg \binom{\widetilde{T}}{\Delta_{h_j}} \rceil$ bits to recover S_{h_j}.
Let V_1^1 be the view with all nodes as singleton chains.
for $i = 1, \ldots, \widetilde{T}$ **do**
  Let $V' \leftarrow V_1^i$ and use $c'(X)$ and $t'(X)$ as the equivalent definitions of c_j^i and t_j^i for the view V'.
  From V' and the advice a, simulate D until sequential round i to determine the function evaluations $F(Y_j^i)$ it makes in round i.
  for $j \in \mathcal{I}'$ **do**
    if $i \in S_{h_j}$ **then**
      while $t'(X_j) \leq 2\widetilde{T}$ *and the tail of* $c'(X_j)$ *equals* Y_k^i *for some* k **do**
        Read and remove $\lceil \lg z \rceil \leftarrow |c'(F(Y_k^i))|$ from b.
        if $\lceil \lg z \rceil > 0$ **then**
          Read and remove the rank r of the chain $c'(F(Y_k^i))$ among all chains c in V' with $|c| \in (2^{\lceil \lg z \rceil - 1}, 2^{\lceil \lg z \rceil}]$ from b.
          Add the edge from Y_k^i to the head of the chain $c'(F(Y_k^i))$ to V' and merge the two chains.
        else
          Read and remove the first bit q from b.
          if $q = 0$ **then**
            Read and remove $\lceil \lg(BP) \rceil$ bits from b to recover ℓ such that $F(Y_k^i) = Y_\ell^i$.
            Add the edge from Y_k^i to Y_ℓ^i and merge the two chains.
          else
            Read and remove $\lceil \lg(N) \rceil$ bits from b to recover $F(Y_k^i)$.
            Add the edge from Y_k^i to $F(Y_k^i)$ and merge the two chains.
  Read and remove first $\lceil \lg N \rceil$ bits to recover $m_i = |L_i|$.
  Read and remove $\lceil \lg((|V'| + m_i - 1)!/(|V'| - 1)!) \rceil$ bits from b to recover L_i.
  for $j = 1, \ldots, BP$ **do**
    if Y_j^i *is the tail of the chain* $c'(Y_j^i)$ **then**
      Read and remove the first element r_j^i of L_i. We learn that $F(Y_j^i)$ is the head of the chain of rank r_j^i in $V' \setminus \{c'(Y_j^i)\}$. We can thus compute $F(Y_j^i)$, insert the edge from Y_j^i to $F(Y_j^i)$ and merge the two chains in V'.
  $V_1^{i+1} \leftarrow V'$.
$V^\infty \leftarrow V_1^{\widetilde{T}+1}$.
return DecodeRemainingView(b, V^∞)

$j \in \mathcal{I}'$ and $i \in S_{h_j}$. We want to argue that the number of bits added to the encoding in lines 15-26 due to this fixed j and i is smaller than $Z_j^i \lg(N - B\widetilde{T}P)$.

For this argument, note that since $i \in S_{h_j}$, we have $\ell_j^i := \min\{2\widetilde{T}+1, t_1^{i+1}(X_j)\} - \min\{2\widetilde{T}+1, t_1^i(X_j)\} \geq \widetilde{T}/2^{h_j+3}+1$. The total length of the chains corresponding to the Z_j^i edges is thus at least ℓ_j^i. Consider one of these Z_j^i edges e, and let z_e denote the increase in $t'(X_j)$ due to the edge. The edge first adds binAdapt($\lceil \lg z_e \rceil$) to the encoding, which we argued above cost no more than $2 + 2\lg(\lceil \lg z_e \rceil + 1) \leq 2 + 2\lg\lg(4z_e)$ bits.

If $z_e > 1$, we then pay $\lceil \lg(B\widetilde{T}P/2^{\lceil \lg z_e \rceil - 1}) \rceil \leq 2 + \lg(B\widetilde{T}P) - \lg z_e$. The *saving* due to the edge e is thus at least $\lg(N - B\widetilde{T}P) - \lg(2^4 B\widetilde{T}P \lg^2(4z_e)/z_e)$ bits.

If $z_e = 1$ and we enter line 23, we pay $1 + \lceil \lg BP \rceil \leq 2 + \lg BP$ for the edge e in line 23, plus the $2 + 2\lg\lg 4 = 4$ bits from binAdapt($\lceil \lg z_e \rceil$). The edge e thus *saves* at least $\lg(N - B\widetilde{T}P) - \lg(2^6 BP)$ bits.

Comparing the two cases above, we see that the *saving per increase in* $t'(X_j)$ is $\lg(N - B\widetilde{T}P) - \lg(2^6 BP)$ in the latter case and $(\lg(N - B\widetilde{T}P) - \lg(2^4 B\widetilde{T}P \lg^2(4z_e)/z_e))/z_e$ in the former case. Since the expensive case in line 25 can only occur once (since no further merges occur for the chain), the above two cases must account for at least $\ell_j^i - 1$ of the increase in $t'(X_j)$. The total saving is smallest when this is due to a single edge with $z_e = \ell_j^i - 1 \geq \widetilde{T}/2^{h_j+3}$, giving a total saving of at least $\lg(N - B\widetilde{T}P) - \lg(2^4 B\widetilde{T}P \lg^2(4\widetilde{T})/(\widetilde{T}/2^{h_j+3})) = \lg(N - B\widetilde{T}P) - \lg(2^{h_j+7} BP \lg^2(4\widetilde{T}))$. The expensive case in line 25 may happen once, giving a *negative* saving of $\lg(N - B\widetilde{T}P) - \lg N - 1$. The Z_j^i edges for the fixed $j \in \mathcal{I}'$ and $i \in S_{h_j}$ thus give a saving of at least

$$\lg\left(\frac{(N - B\widetilde{T}P)^2}{2^{h_j+7} BP \lg^2(4\widetilde{T}) N}\right)$$

bits. This happens for each of the Δ_{h_j} choices of $i \in S_{h_j}$. Also accounting for the $\lceil \lg \binom{\widetilde{T}}{\Delta_{h_j}} \rceil \leq 1 + \Delta_{h_j} \lg(e\widetilde{T}/\Delta_{h_j})$ bits spent encoding S_{h_j} now gives a saving of

$$\Delta_{h_j} \lg\left(\frac{(N - B\widetilde{T}P)^2 \Delta_{h_j}}{e 2^{h_j+7} \widetilde{T} BP \lg^2(4\widetilde{T}) N}\right).$$

Using that $\Delta_{h_j} = 2^{h_j}/(h_j+1)^2$ and $h_j \leq \lg(\widetilde{T}) - 3$, this is at least

$$\Delta_{h_j} \lg\left(\frac{(N - B\widetilde{T}P)^2}{e 2^7 (h_j+1)^2 \widetilde{T} BP \lg^2(4\widetilde{T}) N}\right) \geq \Delta_{h_j} \lg\left(\frac{(N - B\widetilde{T}P)^2}{e 2^7 \widetilde{T} BP \lg^4(4\widetilde{T}) N}\right).$$

If we now assume that $B\widetilde{T}P \lg^5(4\widetilde{T}) \leq N/2^{20}$, then the above saving is at least

$$\Delta_{h_j} \lg\left(\frac{(N/2)^2}{e 2^7 N^2/(2^{20} \lg(4\widetilde{T}))}\right) \geq \Delta_{h_j} \lg(2^7 \lg(4\widetilde{T})) \geq \lg(2^7 \lg(4\widetilde{T})).$$

Finally accounting for the $\lceil \lg(\lg(\widetilde{T}) - 2) \rceil \leq \lg(2\lg(\widetilde{T}))$ bits spend on encoding h_j, we get a saving of at least $\lg(2^7 \lg(4\widetilde{T})) - \lg(2\lg(\widetilde{T})) \geq 6$ bits for each $i \in \mathcal{I}'$.

Recalling that we spend $\lceil \lg N \rceil$ on encoding $|L_i|$ for each i, the total encoding length is thus bounded by

$$\lg((N-1)!) + \left\lceil \lg \binom{B}{B/16} \right\rceil - 6B/16 + 1 + \widetilde{T}(1+\lg N).$$

This is at most

$$\begin{aligned} &\lg((N-1)!) + 1 + \widetilde{T}(1+\lg N) - (B/16)\lg(2^6/(e16)) \\ \leq &\lg((N-1)!) + 1 + \widetilde{T}(1+\lg N) - (B/32). \end{aligned}$$

This completes the proof of Lemma 9.

The indices h_j exist (Proof of Lemma 10). Here we give the proof of Lemma 10 showing that it is possible to find the indices h_j.

Proof (of Lemma 10). Observe that for $t_1^{\widetilde{T}+1}(X_j) > 2\widetilde{T}$, we have

$$\sum_{i=1}^{\widetilde{T}} \min\{2\widetilde{T}+1, t_1^{i+1}(X_j)\} - \min\{2\widetilde{T}+1, t_1^{i}(X_j)\} \geq 2\widetilde{T}.$$

Using $\delta_i := \min\{2\widetilde{T}+1, t_1^{i+1}(X_j)\} - \min\{2\widetilde{T}+1, t_1^{i}(X_j)\}$ for short, this also implies

$$\sum_{i=1}^{\widetilde{T}} (\delta_i - 1) \geq \widetilde{T}. \tag{1}$$

Now assume for the sake of contradiction that there is no index $h_j \in \{0, \ldots, \lg(\widetilde{T}) - 3\}$ such that there are at least $\Delta_{h_j} := 2^{h_j}/(h_j+1)^2$ indices i with $\delta_i \geq \widetilde{T}/2^{h_j+3} + 1$. Then we have

$$\sum_{i=1}^{\widetilde{T}} (\delta_i - 1) = \sum_{h=1}^{\lg(\widetilde{T})-3} \sum_{i:\widetilde{T}/2^{h+3}+1 \leq \delta_i < \widetilde{T}/2^{h+2}+1} (\delta_i - 1) + \sum_{i:\delta_i \geq \widetilde{T}/2^3+1} (\delta_i - 1).$$

Observe that $\Delta_0 = 1$ and thus our contradictory assumption implies that there are no indices i with $\delta_i \geq \widetilde{T}/2^3 + 1$. We thus upper bound the sum by

$$\begin{aligned} \sum_{i=1}^{\widetilde{T}} (\delta_i - 1) &\leq \sum_{h=1}^{\lg(\widetilde{T})-3} \sum_{i:\widetilde{T}/2^{h+3}+1 \leq \delta_i < \widetilde{T}/2^{h+2}+1} \left(\widetilde{T}/2^{h+2} - 1\right) \\ &\leq \sum_{h=1}^{\lg(\widetilde{T})-3} \left(2^h/(h+1)^2 - 1\right)\left(\widetilde{T}/2^{h+2} - 1\right) \\ &\leq \frac{\widetilde{T}}{4} \sum_{h=1}^{\lg(\widetilde{T})-3} \frac{1}{(h+1)^2} < \frac{\pi^2 \widetilde{T}}{24} < \widetilde{T}. \end{aligned}$$

This contradicts (1).

References

ARS24. Damiano Abram, Lawrence Roy, and Mark Simkin. Time-based cryptography from weaker assumptions: Randomness beacons, delay functions and more. Cryptology ePrint Archive, Report 2024/769, 2024. URL: https://eprint.iacr.org/2024/769. 1, 1.1

BBBF18. Dan Boneh, Joseph Bonneau, Benedikt Bünz, and Ben Fisch. Verifiable delay functions. In Hovav Shacham and Alexandra Boldyreva, editors, *Advances in Cryptology – CRYPTO 2018, Part I*, volume 10991 of *Lecture Notes in Computer Science*, pages 757–788, Santa Barbara, CA, USA, August 19–23, 2018. Springer, Cham, Switzerland. doi:10.1007/978-3-319-96884-1_25. 1, 1.2

BFH+24. Alex Biryukov, Ben Fisch, Gottfried Herold, Dmitry Khovratovich, Gaëtan Leurent, María Naya-Plasencia, and Benjamin Wesolowski. Cryptanalysis of algebraic verifiable delay functions. In Leonid Reyzin and Douglas Stebila, editors, *Advances in Cryptology - CRYPTO 2024 - 44th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 18-22, 2024, Proceedings, Part III*, volume 14922 of *Lecture Notes in Computer Science*, pages 457–490. Springer, 2024. 1

BL13. Karl Bringmann and Kasper Green Larsen. Succinct sampling from discrete distributions. In Dan Boneh, Tim Roughgarden, and Joan Feigenbaum, editors, *45th Annual ACM Symposium on Theory of Computing*, pages 775–782, Palo Alto, CA, USA, June 1–4, 2013. ACM Press. doi:10.1145/2488608.2488707. 1.1, 4

BN00. Dan Boneh and Moni Naor. Timed commitments. In Mihir Bellare, editor, *Advances in Cryptology – CRYPTO 2000*, volume 1880 of *Lecture Notes in Computer Science*, pages 236–254, Santa Barbara, CA, USA, August 20–24, 2000. Springer Berlin Heidelberg, Germany. doi:10.1007/3-540-44598-6_15. 3

CHM20. Dror Chawin, Iftach Haitner, and Noam Mazor. Lower bounds on the time/memory tradeoff of function inversion. In Rafael Pass and Krzysztof Pietrzak, editors, *TCC 2020: 18th Theory of Cryptography Conference, Part III*, volume 12552 of *Lecture Notes in Computer Science*, pages 305–334, Durham, NC, USA, November 16–19, 2020. Springer, Cham, Switzerland. doi:10.1007/978-3-030-64381-2_11. 1.2

CK19. Henry Corrigan-Gibbs and Dmitry Kogan. The function-inversion problem: Barriers and opportunities. In Dennis Hofheinz and Alon Rosen, editors, *TCC 2019: 17th Theory of Cryptography Conference, Part I*, volume 11891 of *Lecture Notes in Computer Science*, pages 393–421, Nuremberg, Germany, December 1–5, 2019. Springer, Cham, Switzerland. doi:10.1007/978-3-030-36030-6_16. 1.2

CKL18. Diptarka Chakraborty, Lior Kamma, and Kasper Green Larsen. Tight cell probe bounds for succinct Boolean matrix-vector multiplication. In Ilias Diakonikolas, David Kempe, and Monika Henzinger, editors, *50th Annual ACM Symposium on Theory of Computing*, pages 1297–1306, Los Angeles, CA, USA, June 25–29, 2018. ACM Press. doi:10.1145/3188745.3188830. 1.1, 4

Cle86. Richard Cleve. Limits on the security of coin flips when half the processors are faulty (extended abstract). In *18th Annual ACM Symposium on Theory of Computing*, pages 364–369, Berkeley, CA, USA, May 28–30, 1986. ACM Press. doi:10.1145/12130.12168. 1

CLSY93. Jin-yi Cai, Richard J. Lipton, Robert Sedgewick, and Andrew Chi-Chih Yao. Towards uncheatable benchmarks. In *Proceedings of the Eigth Annual Structure in Complexity Theory Conference, San Diego, CA, USA, May 18-21, 1993*, pages 2–11. IEEE Computer Society, 1993. URL: https://doi.org/10.1109/SCT.1993.336546. 1, 1.2

CMB23. Kevin Choi, Aathira Manoj, and Joseph Bonneau. Sok: Distributed randomness beacons. In *44th IEEE Symposium on Security and Privacy, SP 2023, San Francisco, CA, USA, May 21-25, 2023*, pages 75–92. IEEE, 2023. 1

DDJ24. Nico Döttling, Jesko Dujmovic, and Antoine Joux. Space-lock puzzles and verifiable space-hard functions from root-finding in sparse polynomials. In Elette Boyle and Mohammad Mahmoody, editors, *TCC 2024: 22nd Theory of Cryptography Conference, Part III*, volume 15366 of *Lecture Notes in Computer Science*, pages 431–459, Milan, Italy, December 2–6, 2024. Springer, Cham, Switzerland. doi:10.1007/978-3-031-78020-2_15. 1

DMPS19. Luca De Feo, Simon Masson, Christophe Petit, and Antonio Sanso. Verifiable delay functions from supersingular isogenies and pairings. In Steven D. Galbraith and Shiho Moriai, editors, *Advances in Cryptology – ASIACRYPT 2019, Part I*, volume 11921 of *Lecture Notes in Computer Science*, pages 248–277, Kobe, Japan, December 8–12, 2019. Springer, Cham, Switzerland. doi:10.1007/978-3-030-34578-5_10. 1

EFKP20. Naomi Ephraim, Cody Freitag, Ilan Komargodski, and Rafael Pass. Continuous verifiable delay functions. In Anne Canteaut and Yuval Ishai, editors, *Advances in Cryptology – EUROCRYPT 2020, Part III*, volume 12107 of *Lecture Notes in Computer Science*, pages 125–154, Zagreb, Croatia, May 10–14, 2020. Springer, Cham, Switzerland. doi:10.1007/978-3-030-45727-3_5. 1

FN91. Amos Fiat and Moni Naor. Rigorous time/space tradeoffs for inverting functions. In *23rd Annual ACM Symposium on Theory of Computing*, pages 534–541, New Orleans, LA, USA, May 6–8, 1991. ACM Press. doi:10.1145/103418.103473. 1.2

GGPS23. Alexander Golovnev, Siyao Guo, Spencer Peters, and Noah Stephens-Davidowitz. Revisiting time-space tradeoffs for function inversion. In Helena Handschuh and Anna Lysyanskaya, editors, *Advances in Cryptology – CRYPTO 2023, Part II*, volume 14082 of *Lecture Notes in Computer Science*, pages 453–481, Santa Barbara, CA, USA, August 20–24, 2023. Springer, Cham, Switzerland. doi:10.1007/978-3-031-38545-2_15. 1.2

GS98. David M. Goldschlag and Stuart G. Stubblebine. Publicly verifiable lotteries: Applications of delaying functions. In Rafael Hirschfeld, editor, *FC'98: 2nd International Conference on Financial Cryptography*, volume 1465 of *Lecture Notes in Computer Science*, pages 214–226, Anguilla, British West Indies, February 23–25, 1998. Springer Berlin Heidelberg, Germany. doi:10.1007/bfb0055485. 1

Hel80. Martin E. Hellman. A cryptanalytic time-memory trade-off. *IEEE Trans. Inf. Theory*, 26(4):401–406, 1980. 1.2

KLX20. Jonathan Katz, Julian Loss, and Jiayu Xu. On the security of time-lock puzzles and timed commitments. In Rafael Pass and Krzysztof Pietrzak, editors, *TCC 2020: 18th Theory of Cryptography Conference, Part III*, volume 12552 of *Lecture Notes in Computer Science*, pages 390–413, Durham, NC, USA, November 16–19, 2020. Springer, Cham, Switzerland. doi:10.1007/978-3-030-64381-2_14. 1.2

KMT22. Dmitry Khovratovich, Mary Maller, and Pratyush Ranjan Tiwari. MinRoot: Candidate sequential function for ethereum VDF. Cryptology ePrint Archive, Report 2022/1626, 2022. URL: https://eprint.iacr.org/2022/1626. 1

Lar12a. Kasper Green Larsen. The cell probe complexity of dynamic range counting. In Howard J. Karloff and Toniann Pitassi, editors, *44th Annual ACM Symposium on Theory of Computing*, pages 85–94, New York, NY, USA, May 19–22, 2012. ACM Press. doi:10.1145/2213977.2213987. 1.1, 4

Lar12b. Kasper Green Larsen. Higher cell probe lower bounds for evaluating polynomials. In *53rd Annual Symposium on Foundations of Computer Science*, pages 293–301, New Brunswick, NJ, USA, October 20–23, 2012. IEEE Computer Society Press. doi:10.1109/FOCS.2012.21. 1.1, 4

LLYZ23. Tianxiao Li, Jingxun Liang, Huacheng Yu, and Renfei Zhou. Tight cell-probe lower bounds for dynamic succinct dictionaries. In *64th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2023, Santa Cruz, CA, USA, November 6-9, 2023*, pages 1842–1862. IEEE, 2023. 1.1, 4

LM23. Russell W. F. Lai and Giulio Malavolta. Lattice-based timed cryptography. In Helena Handschuh and Anna Lysyanskaya, editors, *Advances in Cryptology – CRYPTO 2023, Part V*, volume 14085 of *Lecture Notes in Computer Science*, pages 782–804, Santa Barbara, CA, USA, August 20–24, 2023. Springer, Cham, Switzerland. doi:10.1007/978-3-031-38554-4_25. 1, 1.2

LMP+23. Gaëtan Leurent, Bart Mennink, Krzysztof Pietrzak, Vincent Rijmen, Alex Biryukov, Benedikt Bunz, Anne Canteaut, Itai Dinur, Yevgeniy Dodis, Orr Dunkelman, Ben Fisch, Ilan Komargodski, Nadia Heninger, Maria Naya Plasencia, Leo Perrin, Christian Rechberger, Gil Segev, Martin Stam, Stefano Tessaro, Benjamin Wesolowski, Mike Schaffstein, Dankrad Feist, Herold Gottfried, Antonio Sanso, Mark Simkin, and Dmitry Khovratovich. Analysis of minroot. 2023. URL: https://inria.hal.science/hal-04320126/file/minrootanalysis2023.pdf. 1.2

LS20. Kasper Green Larsen and Mark Simkin. Secret sharing lower bound: Either reconstruction is hard or shares are long. In Clemente Galdi and Vladimir Kolesnikov, editors, *SCN 20: 12th International Conference on Security in Communication Networks*, volume 12238 of *Lecture Notes in Computer Science*, pages 566–578, Amalfi, Italy, September 14–16, 2020. Springer, Cham, Switzerland. doi:10.1007/978-3-030-57990-6_28. 1.1, 4

LY20. Mingmou Liu and Huacheng Yu. Lower bound for succinct range minimum query. In Konstantin Makarychev, Yury Makarychev, Madhur Tulsiani, Gautam Kamath, and Julia Chuzhoy, editors, *52nd Annual ACM Symposium on Theory of Computing*, pages 1402–1415, Chicago, IL, USA, June 22–26, 2020. ACM Press. doi:10.1145/3357713.3384260. 1.1, 4

MSW20. Mohammad Mahmoody, Caleb Smith, and David J. Wu. Can verifiable delay functions be based on random oracles? In Artur Czumaj, Anuj Dawar, and Emanuela Merelli, editors, *ICALP 2020: 47th International Colloquium on Automata, Languages and Programming*, volume 168 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 83:1–83:17, Saarbrücken, Germany, July 8–11, 2020. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik. doi:10.4230/LIPIcs.ICALP.2020.83. 1

PD06. Mihai Puatracscu and Erik D. Demaine. Logarithmic lower bounds in the cell-probe model. *SIAM J. Comput.*, 35(4):932–963, 2006. URL: https://doi.org/10.1137/S0097539705447256. 1.1, 4

Pie19. Krzysztof Pietrzak. Simple verifiable delay functions. In Avrim Blum, editor, *ITCS 2019: 10th Innovations in Theoretical Computer Science Conference*, volume 124, pages 60:1–60:15, San Diego, CA, USA, January 10–12, 2019. Leibniz International Proceedings in Informatics (LIPIcs). doi:10.4230/LIPIcs.ITCS.2019.60. 1

PT24. Chris Peikert and Yi Tang. Cryptanalysis of lattice-based sequentiality assumptions and proofs of sequential work. In Leonid Reyzin and Douglas Stebila, editors, *Advances in Cryptology – CRYPTO 2024, Part V*, volume 14924 of *Lecture Notes in Computer Science*, pages 129–157, Santa Barbara, CA, USA, August 18–22, 2024. Springer, Cham, Switzerland. doi:10.1007/978-3-031-68388-6_6. 1, 1.2

PV10. Mihai Patrascu and Emanuele Viola. Cell-probe lower bounds for succinct partial sums. In Moses Charika, editor, *21st Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 117–122, Austin, TX, USA, January 17–19, 2010. ACM-SIAM. doi:10.1137/1.9781611973075.11. 1.1, 4

RS20. Lior Rotem and Gil Segev. Generically speeding-up repeated squaring is equivalent to factoring: Sharp thresholds for all generic-ring delay functions. In Daniele Micciancio and Thomas Ristenpart, editors, *Advances in Cryptology – CRYPTO 2020, Part III*, volume 12172 of *Lecture Notes in Computer Science*, pages 481–509, Santa Barbara, CA, USA, August 17–21, 2020. Springer, Cham, Switzerland. doi:10.1007/978-3-030-56877-1_17. 1, 1.2

RSS20. Lior Rotem, Gil Segev, and Ido Shahaf. Generic-group delay functions require hidden-order groups. In Anne Canteaut and Yuval Ishai, editors, *Advances in Cryptology – EUROCRYPT 2020, Part III*, volume 12107 of *Lecture Notes in Computer Science*, pages 155–180, Zagreb, Croatia, May 10–14, 2020. Springer, Cham, Switzerland. doi:10.1007/978-3-030-45727-3_6. 1.2

VZ13. Elad Verbin and Qin Zhang. The limits of buffering: A tight lower bound for dynamic membership in the external memory model. *SIAM J. Comput.*, 42(1):212–229, 2013. URL: https://doi.org/10.1137/110842211. 1.1, 4

Wes19. Benjamin Wesolowski. Efficient verifiable delay functions. In Yuval Ishai and Vincent Rijmen, editors, *Advances in Cryptology – EUROCRYPT 2019, Part III*, volume 11478 of *Lecture Notes in Computer Science*, pages 379–407, Darmstadt, Germany, May 19–23, 2019. Springer, Cham, Switzerland. doi:10.1007/978-3-030-17659-4_13. 1

A Missing Proof of Theorem 5

To prove the theorem statement, we construct a $(S, P+1, 2T/3)$-admissible adversary $\mathcal{A} = (\mathcal{A}_0, \mathcal{A}_1)$. During preprocessing, adversary $\mathcal{A}_0$ picks $x_i \in [N]$ for $i \in [3N/T - P]$ independently and uniformly at random and stores $(x_i, F^{T/3}(x_i))$. The advice string σ is the set of these pairs. The bit length of the advice string produced by $\mathcal{A}_0$ is $2 \lg N \, (3N/T - P)$.

Upon receiving challenge x, adversary $\mathcal{A}_1$ picks $x_1, \ldots, x_P \in [N]$ independently and uniformly at random. Then, $\mathcal{A}_1$ computes $F^{T/3}(x), F^{T/3}(x_1), \ldots, F^{T/3}(x_P)$ using $T/3$ sequential and $P+1$ parallel queries to oracle F. Let $\tilde{\sigma} := \sigma \cup \{(x_i, F^{T/3}(x_i)) \mid i \in [P]\}$ be the set containing all pairs computed during preprocessing and computed now.

Now for $i \in [T/3]$, the adversary $\mathcal{A}_1$ iteratively keeps computing $F^{T/3+i}(x)$ and checking whether $(F^{T/3+i}(x), F^{T2/3+i}(x))$ is contained in $\tilde{\sigma}$. If, for some $i \in [T/3]$ such a pair is found, the adversary skips from $F^{T/3+i}(x)$ to $F^{T2/3+i}(x)$ and computes $F^{T}(x)$ from there on honestly. If no such pair is found in these $T/3$ steps (and the evaluation did not end up in a cycle in the functional graph), then $\mathcal{A}_1$ outputs $\perp$ and aborts. If the adversary is successful, it is clear that $\mathcal{A}_1$ makes $2T/3$ sequential queries to F.

The analysis of the adversary's success probability is basically the same as in the previous proof. Let hit be the event that for some value in $z \in \{F^{T/3+1}(x), \ldots, F^{2T/3}(x)\}$, it holds that $(z, F^{T/3}(z)) \in \tilde{\sigma}$. To win the sequentiality security experiment, the adversary needs

$$\Pr[\mathsf{hit}] > 1/2,$$

where the probability is taken over the random choices of $\mathcal{A}_0$, $\mathcal{A}_1$, and the choice of the challenge x.

$$\begin{aligned}\Pr\left[\neg\mathsf{hit}\right] &= \left(1 - \frac{(T/3)}{N}\right)^{(3N/T-P)+P} = \left(1 - \frac{(T/3)}{N}\right)^{3N/T} \leq \left(e^{-T/3}\right)^{3/T} \\ &= e^{-1} < 1/2.\end{aligned}$$

Thus it follows that the adversary $\mathcal{A}$ will be successful with probability strictly greater than half. □