Time-Space Tradeoffs of Truncation with Preprocessing

Krzysztof Pietrzak
IST, Klosterneuburg, Austria

Pengxiang Wang
EPFL, Lausanne, Switzerland

Abstract

Truncation of cryptographic outputs is a technique that was recently introduced in Baldimtsi et al. [2]. The general idea is to try out many inputs to some cryptographic algorithm until the output (e.g. a public-key or some hash value) falls into some sparse set and thus can be compressed: by trying out an expected 2^k different inputs one will find an output that starts with k zeros.

Using such truncation one can for example save substantial gas fees on Blockchains where storing values is very expensive. While [2] show that truncation preserves the security of the underlying primitive, they only consider a setting without preprocessing. In this work we show that lower bounds on the time-space tradeoff for inverting random functions and permutations also hold with truncation, except for parameters ranges where the bound fails to hold for "trivial" reasons.

Concretely, it's known that any algorithm that inverts a random function or permutation with range N making T queries and using S bits of auxiliary input must satisfy $S \cdot T \geq N \log N$. This lower bound no longer holds in the truncated setting where one must only invert a challenge from a range of size $N/2^k$, as now one can simply save the replies to all $N/2^k$ challenges, which requires $S = \log N \cdot N/2^k$ bits and allows to invert with $T = 1$ query.

We show that with truncation, whenever S is somewhat smaller than the $\log N \cdot N/2^k$ bits required to store the entire truncated function table, the known $S \cdot T \geq N \log N$ lower bound applies.

2012 ACM Subject Classification Theory of computation → Interactive proof systems; Theory of computation → Cryptographic protocols; Security and privacy → Information-theoretic techniques

Keywords and phrases Time-Space Lower Bounds, Blockchains

Digital Object Identifier 10.4230/LIPIcs.ITC.2025.4

1 Introduction

1.1 Truncation

In [2] Baldimtsi, Chalkias, Chatzigiannis and Kelkar suggested *truncation* as a technique to shorten outputs of cryptographic primitives like hash values, signatures or public keys. The technique applies whenever there's a part of the input that can be chosen arbitrarily, like the randomness in signing and key generation or parts of the input when hashing. The general idea is to simply try out many different inputs until the output lands in some sparse domain, say it starts with Δ zeros, and such an output can be encoded using Δ less bits than a general output.

While the applicability of this technique is limited by the fact that finding an output that starts with Δ zeros will require around 2^Δ invocations of the primitive (so e.g., compressing by 20 bits already requires around a million invocations), in [2] interesting applications are identified where this can make a big difference, including in the context of the Ethereum blockchain, where saving a few bits to be stored on-chain can lead to significant savings. We discuss another application to proofs of space as used in the Chia network in the open problems Section 6.

While truncation puts extra burden on the evaluator (say, a signer), there's no extra cost for the parties that use the output (say, verify the signature). Moreover [2] show that truncation preserves security in the bit security framework of [9]. Informally, a primitive has bit security κ, if every adversary who breaks the security with advantage ϵ must run in time $2^\kappa \cdot \epsilon$.

1.2 Time-Space Tradeoffs

While the preservation of bit security under truncation gives some confidence in this technique, it only considers a setting without precomputation. Unfortunately, truncation can decrease the security of primitives when the adversary is given some auxiliary input. Let us illustrate this considering the one-wayness of a permutation over n bits, which we'll denote with $f : [N] \to [N]$ where $N = 2^n$. A random permutation f does have n bits of bit security, i.e., given a random $y \in [N]$, finding $x, f(x) = y$ will require $N/2$ invocations to f in expectation. But given S bits of advice (that depend on f but not the challenge y), it's possible to invert f using just T invocations whenever

$$S \cdot T \geq N \log(N) \tag{1}$$

The general idea is to compute and store values $x_0, x_T, x_{2T}, \ldots$ where $x_i = f(x_{i-1})$. On challenge y one now applies f until one of the stored $x_{i \cdot T}$ values is hit, and then continues to apply f to $x_{(i-1)T}$ until one hits y. For functions the best space-time tradeoff are somewhat worse. For *random* functions an attack with

$$S^2 \cdot T \in \Theta(N^2 \log(N))$$

is achieved by Hellman tables [7] or rainbow tables [8]. So, any permutation can be inverted with time and space, e.g. $T = S \approx N^{1/2}$ while *random* functions can be inverted with $T = S \approx N^{2/3}$ (for functions that are not random the existing bounds are somewhat worse [5]).

De, Trevisan and Tulsiani [4] (building on work by Yao [11], Gennaro-Trevisan [6] and Wee [10]) prove that the attack as in eq. (1) is basically optimal as any adversary who inverts a *random* permutation or function with a range of size N must satisfy

$$S \cdot T \in \Omega(N) \tag{2}$$

Note that the above is seemingly wrong if $T = 0$ or $S = 0$ as one can invert with no space but N queries or $N \log N$ space and no queries. The proof of the lower bound assumes adversary must always query f on its output, so T is at least 1, and S is assumed to be at least $\log N$ which is required to store the challenge.

The same lower bound applies for random *functions*, though note that in this case it's not matching the upper bound. The actual lower bound is slightly more general showing that $S \cdot T \in \Omega(\epsilon \cdot N)$ must hold for any adversary who inverts with some probability ϵ, but for this introduction we assume ϵ is some constant.

1.3 Time-Space Tradeoffs with Truncation

Now consider the truncated setting, where we have a random function or permutation $f : [N] \to [N]$, but must only invert it on outputs sampled from some truncated range, say where the first Δ bits are set to zero. We'll denote this range with $[\delta N]$ where $\delta = 2^{-\Delta}$.

Now one can store the δN preimages of f on $[\delta N]$ using $S = \delta N \log N$ bits, and using this advice, it's possible to find the preimage $x, f(x) = y$ for any $y \in [\delta N]$ without invoking f at all. As outlined above, this means $T = 1$, but it still contradicts the lower-bound from eq. (2) as $\delta N \log N \notin \Omega(N)$ for $\delta \in o(1/\log N)$.

The main result in this work is Theorem 4, which basically states that the issue just described is the only reason for the lower bound to fail: as long as S is too small to basically store the entire function table of the *truncated* function, the $S \cdot T \in \Omega(N)$ lower bound applies. While we state and prove the bound for functions, it can easily be adapted to permutations. We discuss other truncated primitive in the open problems Section 6.

The technical result implying Theorem 4 is stated in Lemma 5. It uses a so-called compression argument: it shows how an adversary that inverts a random function f can be turned into an encoding for the function table of f, where the length of the encoding depends on the space and time efficiency of the adversary. As the function table of a random function is incompressible, as stated in Fact 1, we can derive a lower bound on the space and time complexity of the adversary.

1.4 The Compression Argument

The starting point for proving the Lemma is a proof of the $S \cdot T \geq N \log N$ lower bound for inverting random functions from [1]. This proof is less elegant than the proof of De et al. from [4], which uses a high level argument about sets, while [1] provide pseudocode of the encoding and argue about the length of its output.

Their encoding is basically Algorithm 1 in this paper for $\delta = 1$. It takes as input an adversary $\mathcal{A}$ who is guaranteed to invert some function $f : [N] \to [N]$ on an ϵ fraction of the range and making no more than T queries.

$\mathcal{A}$ is then invoked on some challenges, and every time $\mathcal{A}$ inverts a challenge we get one entry in the function table "for free". With every invocation, $\mathcal{A}$ can make up to T queries to f, and those queries can later no longer be used as challenges, i.e., we "spoil" up to T of the $\epsilon \cdot N$ challenges with every succesful inversion. Overall we can get $\mathcal{A}$ to invert something in the order of $\epsilon \cdot N/T$ challenges before running out of unspoiled challenges, and thus get an encoding that is around $\epsilon \cdot N/T$ bits shorter than the function table. As a random function is incompressible, this then implies that the advice used by $\mathcal{A}$ must be around $S \gtrapprox \epsilon \cdot N/T$. That's how the $S \cdot T \in \Omega(\epsilon \cdot N)$ lower bound in [1] is proven.

In the truncated setting when $\delta \ll 1$, $\mathcal{A}$ is only guaranteed to invert on an ϵ fraction of the sparse domain $[\delta N]$, and the above argument would only gives us a $S \gtrapprox \delta \cdot \epsilon \cdot N/T$ bound.

A key observation is the fact that we get a $S \gtrapprox \epsilon \cdot N/T$ lower bound even in the truncated case if we assume that not much more than a δ fraction of the queries made by $\mathcal{A}$ map to the sparse $[\delta \cdot N]$ domain, as then each compressed entry only spoils around $\delta \cdot T$ of the $[\epsilon N]$ available challenges.

To prove our lemma we now make a case distinction. If for every compressed value we typically do not spoil more than $T_g \leq 12\delta T$ potential challenges, we use the observation above, so in this case we use basically the same argument as in [1].

In the other case the queries made by $\mathcal{A}$ fall into the sparse $[\delta N]$ set at least 12 times more often than a random query would. Knowing that the output $f(x)$ on a query x is in $[\delta N]$ means we can encode it using just $\log N - \log 1/\delta$ bits. We will encode the positions of the queries made by $\mathcal{A}$ that fall into $[\delta N]$ (there are around $\epsilon \delta N$ such queries) and then encode each output using just $\log N - \log 1/\delta$ bits. As the queries that fall into $[\delta N]$ are sufficiently dense (i.e., 12 times denser than a random query would), encoding those positions uses a bit less per entry than the $\log 1/\delta$ bits we save knowing the entry is in $[\delta N]$. So overall we save $\epsilon\delta N$ bits, and thus for this case can conclude (again using the incompressibility of a random function table) that the space used by $\mathcal{A}$ must be at least $S \gtrapprox \epsilon\delta N$.

2 Notation and Basic Facts

We use brackets like $(x_1, x_2, \ldots)$ or $\{x_1, x_2, \ldots\}$ to denote ordered sets (aka. lists) and unordered sets, respectively. $[N]$ denotes some domain of size N, and for notational convenience we assume $N = 2^n$ is a power of two and identify $[N]$ with $\{0,1\}^n$. For some $\Delta \le n$ and $\delta = 2^{-\Delta}$, we'll denote with $[\delta N]$ some subset of $[N]$ of size δN (the truncated range), say $0^\Delta \| \{0,1\}^{n-\Delta}$, the set of n bits strings that start with Δ zeros, but in principle any subset whose elements can be compressed to (not much more than) $n - \Delta$ bits will do.

For a function $f : [N] \to [M]$ and a set $S \subseteq [N]$, we denote with $f(S)$ the set $\{f(S[1]), \ldots, f(S[|S|])\}$, similarly for a list $L \subseteq [N]$, $f(L)$ is the list $(f(L[1]), \ldots, f(L[|L|]))$.

Randomized adversaries are treated as if they were deterministic. However, this is w.l.o.g. as they are only invoked within encoding/decoding procedures that have access to a shared randomness that can be used to derandomize them.

The following well known fact captures the fact that one cannot compress a random string.

▶ **Fact 1** (from [4]). *For any randomized encoding procedure* $\mathsf{Enc} : \{0,1\}^r \times \{0,1\}^n \to \{0,1\}^m$ *and decoding procedure* $\mathsf{Dec} : \{0,1\}^r \times \{0,1\}^m \to \{0,1\}^n$ *where*

$$\Pr_{x \leftarrow \{0,1\}^n, r \leftarrow U_{|r|}} [\mathsf{Dec}(r, \mathsf{Enc}(r, x)) = x] \ge \delta$$

we have $m \ge n - \log(1/\delta)$

▶ **Fact 2.** *If a set X is at least ϵ dense in Y, i.e., $X \subset Y$, $|X| \ge \epsilon|Y|$, and Y is known, then X can be encoded using $|X| \cdot \log(e/\epsilon)$ additional bits.*

This fact follows from the inequality $\binom{n}{\epsilon n} \le (en/\epsilon n)^{\epsilon n}$, which implies $\log \binom{n}{\epsilon n} \le \epsilon n \log(e/\epsilon)$. Encoding X can be done by identifying which $\epsilon|Y|$ elements to choose from Y.

3 Main Theorem and Lemma

De, Trevisan and Tulsiani [4] show that the simple time-space tradeoff eq. (1) for inverting permutations is basically tight.

▶ **Theorem 3** ([4], as stated in [1]). *Fix some $\epsilon \ge 0$ and an oracle algorithm $\mathcal{A}_{\mathsf{aux}}$ that takes an advice string* aux *of length* $|\mathsf{aux}| = S$ *and makes at most T oracle queries. If with non-neglibile probability for a random function (or permutation) $f : [N] \to [N]$ there exists a string* aux *such that*

$$\Pr_{y \leftarrow [N]} [f(\mathcal{A}^f_{\mathsf{aux}}(y)) = y] \ge \epsilon$$

then

$$T \cdot S \in \Omega(\epsilon N) . \quad (3)$$

In this work we show that this result extends to the case where the challenge comes from a sparse set $[\delta N]$

▶ **Theorem 4** (main). *Fix some $\epsilon \geq 0$ and an oracle algorithm $\mathcal{A}_{\mathsf{aux}}$ that takes an advice string* aux *of length $|\mathsf{aux}| = S$ and makes at most T oracle queries. If with non-neglibile probability for a random function (or permutation) $f : [N] \to [N]$ there exists a string* aux *such that*

$$\Pr_{y \leftarrow [\delta N]}[f(\mathcal{A}^f_{\mathsf{aux}}(y)) = y] \geq \epsilon$$

then either

$$S \in \Omega(\delta\epsilon N) \qquad \text{or} \qquad T \cdot S \in \Omega(\epsilon N)\ . \tag{4}$$

The theorem is proven using a compression argument as stated in the following lemma.

▶ **Lemma 5** (generalized version of a Lemma from [1]). *Let $\mathcal{A}_{\mathsf{aux}}, T, S, \epsilon$ and f be as Theorem 4, and assume $T \leq \delta\epsilon N/40$. There are randomized encoding and decoding procedures* Enc, Dec *such that if $f : [N] \to [N]$ is a function and for some* aux, $|\mathsf{aux}| = S$

$$\Pr_{y \leftarrow [\delta N]}[f(\mathcal{A}^f_{\mathsf{aux}}(y)) = y] \geq \epsilon$$

then

$$\Pr_{r \leftarrow U_{|r|}}[\mathsf{Dec}(r, \mathsf{Enc}(r, \mathsf{aux}, f)) = f] \geq 0.9 \tag{5}$$

and the length of $\mathsf{Enc}(r, \mathsf{aux}, f)$ *is at most*

$$\underbrace{N \log N}_{=|f|} - \frac{\epsilon\delta N}{2T_g} + S + \log(N) \tag{6}$$

Moreover for some $T_g, 1 \leq T_g \leq T$ which is defined by the encoding algorithm: if $T_g \geq 12\delta T$, we can improve the length of the encoding to

$$\underbrace{N \log N}_{=|f|} - \frac{\epsilon\delta N}{2T_g} + S + \log(N) - \delta\epsilon N \tag{7}$$

The T_g above is the average number of f queries that land in the sparse set (i.e., x s.t. $f(x) \in [\delta N]$) that $\mathcal{A}^f_{\mathsf{aux}}$ (when invoked by Enc as defined by Algorithm 1 below) makes for every value it inverts. As $\mathcal{A}$ makes at most T queries per challenge we have $T_g \leq T$. For a random query $x \in [N]$ we have $f(x) \in [\delta N]$ with probability δ, so if T_g is significantly larger than δT, this means that the queries made by $\mathcal{A}_{\mathsf{aux}}$ are special in the sense that they map to $[\delta N]$ much more often than random queries would. We use this crucial observation for a case distinction, deriving the left or the right hand side of eq. (4), depending on whether T_g is below or above $12\delta T$.

4 How Theorem 4 follows from Lemma 5

4.1 Proof of Thm. 4 if $T_g \leq 12\delta T$

If $T_g \leq 12\delta T$, the theorem follows from Lemma 5 using Fact 1 as follows: assume the function table of f in the lemma is chosen uniformly at random (i.e., x in Fact 1 is a uniform $N \log N$ bit string), then the term in eq. (6) can be lower bounded as

$$\underbrace{N \log N}_{=|f|} - \frac{\epsilon\delta N}{2T_g} + S + \log(N) \geq N \log N - \log(1/0.9).$$

Reordering we get

$$S \geq \frac{\epsilon\delta N}{2T_g} - \log(N) - \log(1/0.9)$$

using our assumption that $T_g \leq 12\delta T$

$$S \geq \frac{\epsilon N}{24T} - \log(N) - \log(1/0.9)$$

$$T \cdot S \geq \frac{\epsilon N}{24} - T \cdot \log(N) - T \cdot \log(1/0.9).$$

So $T \cdot S \in \Omega(\epsilon N)$ as claimed (on the rhs of eq. (4) in the Theorem). Note that the extra assumption that $T \leq \epsilon N/40$ in the lemma doesn't matter, as if it's not satisfied the theorem is trivially true.

4.2 Proof of Thm. 4 if $T_g > 12\delta T$

If $T_g > 12\delta T$, the theorem again follows from Lemma 5 using Fact 1, i.e., we again assume the function table of f in the lemma is chosen uniformly at random (i.e., x in Fact 1 is a uniform $N \log N$ bit string), and now the term in eq. (7) can be lower bounded as

$$\underbrace{N \log N}_{=|f|} - \frac{\epsilon\delta N}{2T_g} + S + \log(N) - \delta\epsilon N \geq N \log N - \log(1/0.9).$$

Note that, as in this case we use eq. (7) rather than eq. (6), we have an extra $-\delta\epsilon N$ term on the lhs. Reordering we get

$$S \geq \frac{\epsilon\delta N}{2T_g} - \log(N) - \log(1/0.9) + \delta\epsilon N$$

and thus $S \in \Omega(\delta\epsilon N)$ as claimed.

5 Proof of Lemma 5

We always assume that if $A^f_{\mathsf{aux}}(y)$ outputs some value x, it makes the query $f(x)$ at some point. This is basically w.l.o.g. as we can turn any adversary into one satisfying this by making at most one extra query. If at some point $A^f_{\mathsf{aux}}(y)$ makes an oracle query x where $f(x) = y$, then we also w.l.o.g. assume that right after this query A outputs x and stops.

5.1 The Size of the Encoding

We will now upper bound the size of the encoding of $G, f(Q'), (|q_1|, \ldots, |q_{|G|}|), f([N] - \{G^{-1} \cup Q'\})$ as output in line (15) of the Enc algorithm.

Let $T_g := |B|/|G|$ be the average number of elements we added to the bad set B for every element added to the good set G, then

$$|G| \geq N\epsilon\delta/2T_g\ . \tag{8}$$

To see this we note that when we leave the while loop (see line (8) of the algorithm Enc) it holds that

$$|B| \geq |J|/2 = \epsilon\delta N/2 \text{ so } |G| = |B|/T_g \geq |J|/2T_g = N\epsilon\delta/2T_g \tag{9}$$

Algorithm 1 Enc.

```
1: Input: $\mathcal{A}$, aux, randomness $r$ and a function $f : [N] \to [N]$ to compress (using that
   $\mathcal{A}^f_{\mathsf{aux}}(\cdot)$ inverts $f$ on some $\epsilon$ fraction of $[\delta N]$ )
2: Initialize: $B, G := \emptyset, c := -1$
3: Throughout we identify $[N]$ with $\{0, \ldots, N-1\}$ and $[\delta N]$ with $\{0, \ldots, \delta N - 1\}$
4: Pick a random permutation $\pi : [\delta N] \to [\delta N]$ (using random coins $r$)
5: Let $J := \{y \in [\delta N] \ : \ f(\mathcal{A}^f_{\mathsf{aux}}(y)) = y\}$, $|J| = \epsilon\delta N$   ▷ The set $J \subset [\delta N]$ where $\mathcal{A}$ inverts
6: For $i = 0, \ldots, \delta N - 1$ define $y_i := \pi(i)$.   ▷ Randomize the order
7: For $y \in J$ let $q(y)$ denote all queries made by $A^f(y)$ except the last query (which is $x$ s.t.
   $f(x) = y$).
8: while $|B| < |J|/2 (= \epsilon\delta N/2)$ do   ▷ While the bad set contains less than half of $J$
9:     $c := \min\{c' > c \ : \ y_{c'} \in \{J \setminus B\}\}$   ▷ Increase $c$ to the next $y_c$ in $J \setminus B$
10:    $G := G \cup y_c$   ▷ Add this $y_c$ to good set
11:    $B := B \cup (f(q(y_c)) \cap J)$   ▷ Add spoiled queries to bad set
12: end while
13: Let $G = \{g_1, \ldots, g_{|G|}\}$, $Q = q(g_1), \ldots, q(g_{|G|})$, and define $Q' = (q'_1, \ldots, q'_{|G|}), q'_i \subseteq q(g_i)$
    to contain only the "fresh" queries in $Q$ by deleting all but the first occurrence of every
    element. E.g. if $(q(g_1), q(g_2)) = ((1,2,3,1),(2,4,5,4))$ then $(q'_1, q'_2) = ((1,2,3),(4,5))$.
14: Let $G^{-1} = \{\mathcal{A}^f_{\mathsf{aux}}(y) \ : \ y \in G\}$
15: Output an encoding of (the set) $G$, (the lists) $f(Q'), (|q'_1|, \ldots, |q'_{|G|}|)$, (the remaining
    outputs) $f([N] - \{G^{-1} \cup Q'\})$ and (the advice string) aux.   ▷ We discuss
    how exactly this encoding looks in the proof. The encoding of $f(Q')$ will depend on the
    average number of spoiled queries per challenge $T_g = |B|/|G|$ ($\approx \epsilon\delta N/2|G|$)
```

***G*:** Instead of G we will actually encode the set $\pi^{-1}(G) = \{c_1, \ldots, c_{|G|}\}$, from this encoding Dec (who gets r, and thus knows π) can then reconstruct $G = \pi(\pi^{-1}(G))$. We claim that the elements in $c_1 < c_2 < \ldots < c_{|G|}$ are whp. at least $\epsilon\delta/2$ dense in $[c_{|G|}]$ (equivalently, $c_{|G|} \leq 2|G|/\epsilon\delta$). By Fact 2 we can thus encode $\pi^{-1}(G)$ using $|G|\log(2e/\epsilon\delta) + \log N$ bits (the extra $\log N$ bits are used to encode the size of G which is required so decoding later knows how to parse the encoding). To see that the c_i's are $\epsilon\delta/2$ dense whp. consider line (9) in Enc which states $c := \min\{c' > c \ : \ y_{c'} \in \{J \setminus B\}\}$. If we replace $J \setminus B$ with J, then the c_i's would be whp. close to $\epsilon\delta$ dense in $[N]$ as $J \subset N$ has size $\epsilon\delta N$ and the y_i are uniformly random. As $|B| < |J|/2$, using $J \setminus B$ instead of J will decrease the density by at most a factor 2. If we don't have this density, i.e., $c_{|G|} > 2|G|/\epsilon\delta$, we consider encoding to have failed.

$(|q'_1|, \ldots, |q'_{|G|}|)$: Require $|G|\log T$ bits as each $q'_i \leq q_i \leq T$. A more careful argument (using Fact 2 and that the q'_i are on average at most T_g) requires $|G|\log(eT_g)$ bits.

$f([N] - \{G^{-1} \cup Q'\})$: Requires $(N - |G| - |Q'|)\log N$ bits (using that $G^{-1} \cap Q' = \emptyset$ and $|G^{-1}| = |G|$).

aux: Is S bits long.

$f(Q')$: This is a list of $|Q'|$ elements in $[N]$ and can be encoded using $|Q'|\log N$ bits, but we'll encode it with less if T_g is large.

Algorithm 2 Dec.

```
1: Input: $\mathcal{A}, r$ and the encoding $(G, f(Q'), (|q'_1|, \dots, |q'_{|G|}|), f([N] - \{G^{-1} \cup Q'\}), \mathsf{aux})$.
2: Let $\pi$ be as in Enc.
3: Let $(g_1, \dots, g_{|G|})$ be the elements of $G$ ordered as they were added by Enc (i.e., $\pi^{-1}(g_i) < \pi^{-1}(g_{i+1})$ for all $i$).
4: Invoke $\mathcal{A}^{(\cdot)}_{\mathsf{aux}}(\cdot)$ sequentially on inputs $g_1, \dots, g_{|G|}$ using $f(Q')$ to answer $\mathcal{A}_{\mathsf{aux}}$'s oracle queries.
5: Combine the mapping $G^{-1} \cup Q' \to f(G^{-1} \cup Q')$ (which we learned in the previous step) with $[N] - \{G^{-1} \cup Q'\} \to f([N] - \{G^{-1} \cup Q'\})$ to learn the entire $[N] \to f([N])$
6: Output $f([N])$
```

Summing up we get

$$\begin{aligned} & |\mathsf{Enc}(r, \mathsf{aux}, f)| \\ = \; & \underbrace{|G| \log(2e/\epsilon\delta) + \log N}_{\text{encoding of } G} + \underbrace{|Q'| \log N}_{f(Q')} + \underbrace{|G| \log(eT_g)}_{(|q'_1|, \dots, |q'_{|G|}|)} + \underbrace{(N - |G| - |Q'|) \log N}_{f([N] - \{G^{-1} \cup Q'\})} + \underbrace{S}_{\mathsf{aux}} \\ = \; & |G| \log(2e^2 T_g/\epsilon\delta) + (N - |G| - |Q'|) \log N + |Q'| \log N + S + \log N. \end{aligned}$$

Using the assumption $T_g \le T \le \epsilon\delta N/40$ in the statement of the lemma, which in turn implies $\log(2e^2 T_g/\epsilon\delta) \le \log(N) - 1$, we get

$$\begin{aligned} \le \; & |G|(\log N - 1) + (N - |G| - |Q'|) \log N + |Q'| \log N + S + \log N \\ = \; & (N - |Q'|) \log N + |Q'| \log(N) - |G| + S + \log N \qquad (10) \\ = \; & N \log N - |G| + S + \log N \end{aligned}$$

Plugging in the bound from eq. (8) for $|G|$ we get

$$|\mathsf{Enc}(r, \mathsf{aux}, f)| \le N \log N - \frac{N\delta\epsilon}{2T_g} + S + \log N$$

proving eq. (6) in the Lemma.

5.2 Improved bound if $T_g > 12\delta T$

We'll now prove a bound on the length of the encoding as stated in eq. (7) in the lemma. Here we assume that T_g is large, i.e., $T_g > 12\delta T$. The bound on the length of the encoding improves the expression we got without this assumption, i.e., eq. (6), by $\delta\epsilon N$ bits. We will achieve this by improving the length of the encoding of $f(Q')$ from the trivial $|Q'| \log(N)$ to $|Q'| \log(N) - \delta\epsilon N$ by exploiting the fact that now a large fraction of the elements in $f(Q')$ falls into the sparse set $[\delta N]$, i.e.,

▷ Claim 6. If $T_g > 12\delta T$, $f(Q')$ can be encoded using $|Q'| \log N - \delta\epsilon N$ bits.

With this claim we can improve eq. (10) to $(N - |Q'|) \log N + |Q'| \log(N) - |G| + S + \log N$, which now gives eq. (7) from the Lemma.

It remains to prove the claim. By eq. (8) and eq. (9) $|G| \ge \delta\epsilon N/2T_g$, $|Q'| \le T \cdot |G|$ and $|B| \ge \delta\epsilon N/2$, which implies

$$|B|/|Q'| \ge \delta\epsilon N/2T \cdot |G| \ge T_g/T \ge 12\delta$$

I.e., a 12δ fraction of the queries made during decoding falls into $B \subset [\delta N]$. Using this with Fact 2 we can encode which of the $|B|$ queries from Q' map into B using $|B|\log(e/12\delta)$ bits. For any $x \in B$, we can encode $f(x)$ using $\log(N) - \log(1/\delta)$ bits as $f(x) \in [\delta N]$.

We can now encode $f(Q')$ by first encoding the positions of B in Q', and then $f(B)$ and $f(Q' \setminus B)$ separately, which requires

$$\begin{aligned} & (|Q'| - |B|)\log N + |B|\left(\log(e/10\delta) + \log(N) - \log(1/\delta)\right) \\ = \; & |Q'|\log N + |B|\log(e/12) \\ \le \; & |Q'|\log N - |B| \cdot 2 \\ \le \; & |Q'|\log N - \delta\epsilon N \end{aligned}$$

bits as claimed.

6 Conclusion and Open Problems

In this work we showed that the known time-space tradeoff $S \cdot T \ge N$ for inverting random functions or permutations $f : [N] \to [N]$ also holds for truncated outputs almost up to the point where S is big enough to store the entire truncated function table. This shows that the general idea of truncating cryptographic primitives as suggested in [2] is secure even when preprocessing is considered (as long as the truncated function table is big enough so it can't be stored in practice).

While in this paper we only considered one-wayness of random functions, we believe that our result can be adapted to time-space lower bounds for other primitives, showing the bounds apply also for their truncated analogues. An interesting example considered in [2] is the discrete logarithm problem, for which a $S \cdot T^2 \ge N$ time-space lower bound is known [3].

A likely more challenging but particularly interesting case is the adaption of the "beyond Hellman" proofs of space from [1] to the truncated setting. The key primitive in [1] are functions $[N] \to [N]$ for which a time-space lower bound of $S^k \cdot T \ge N^k$ (for any constant k) can be proven. This improves on the $S \cdot T \ge N$ lower bound for "normal" functions, and the reason this doesn't contradict known upper bounds (by rainbow tables) is the fact that those functions cannot be efficiently evaluated in forward direction (but their entire function table can be computed in time N). A proof that truncation is secure for these functions would allow for better security. Currently those functions are deployed in the proof of space underlying the `chia.net` blockchain, but there's a trade-off that farmers (who are supposed to dedicate space analogous to miners dedicating computation in Bitcoin) can make: by dropping a few bits of every entry, they save space at the cost of having to do some extra computation when computing the proofs[1]. One could use a truncated version of those functions, which would make the initialization of the space for the farmers somewhat more costly, but it would make such "bit dropping" attacks much less attractive.

References

1 Hamza Abusalah, Joël Alwen, Bram Cohen, Danylo Khilko, Krzysztof Pietrzak, and Leonid Reyzin. Beyond hellman's time-memory trade-offs with applications to proofs of space. In Tsuyoshi Takagi and Thomas Peyrin, editors, *ASIACRYPT 2017, Part II*, volume 10625 of *LNCS*, pages 357–379. Springer, Heidelberg, December 2017. `doi:10.1007/978-3-319-70697-9_13`.

[1] `https://github.com/Chia-Network/drplotter`

2 Foteini Baldimtsi, Konstantinos Chalkias, Panagiotis Chatzigiannis, and Mahimna Kelkar. Truncator: Time-space tradeoff of cryptographic primitives. Cryptology ePrint Archive, Paper 2022/1581, 2022. URL: `https://eprint.iacr.org/2022/1581`.

3 Henry Corrigan-Gibbs and Dmitry Kogan. The discrete-logarithm problem with preprocessing. In Jesper Buus Nielsen and Vincent Rijmen, editors, *Advances in Cryptology – EUROCRYPT 2018 – 37th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Tel Aviv, Israel, April 29 – May 3, 2018 Proceedings, Part II*, volume 10821 of *Lecture Notes in Computer Science*, pages 415–447. Springer, 2018. `doi:10.1007/978-3-319-78375-8_14`.

4 Anindya De, Luca Trevisan, and Madhur Tulsiani. Time space tradeoffs for attacks against one-way functions and PRGs. In Tal Rabin, editor, *CRYPTO 2010*, volume 6223 of *LNCS*, pages 649–665. Springer, Heidelberg, August 2010. `doi:10.1007/978-3-642-14623-7_35`.

5 Amos Fiat and Moni Naor. Rigorous time/space tradeoffs for inverting functions. In Cris Koutsougeras and Jeffrey Scott Vitter, editors, *Proceedings of the 23rd Annual ACM Symposium on Theory of Computing, May 5-8, 1991, New Orleans, Louisiana, USA*, pages 534–541. ACM, 1991. `doi:10.1145/103418.103473`.

6 Rosario Gennaro and Luca Trevisan. Lower bounds on the efficiency of generic cryptographic constructions. In *41st FOCS*, pages 305–313. IEEE Computer Society Press, November 2000. `doi:10.1109/SFCS.2000.892119`.

7 Martin E. Hellman. A cryptanalytic time-memory trade-off. *IEEE Trans. Inf. Theory*, 26(4):401–406, 1980. `doi:10.1109/TIT.1980.1056220`.

8 Philippe Oechslin. Making a faster cryptanalytic time-memory trade-off. In Dan Boneh, editor, *Advances in Cryptology – CRYPTO 2003, 23rd Annual International Cryptology Conference, Santa Barbara, California, USA, August 17-21, 2003, Proceedings*, volume 2729 of *Lecture Notes in Computer Science*, pages 617–630. Springer, 2003. `doi:10.1007/978-3-540-45146-4_36`.

9 Shun Watanabe and Kenji Yasunaga. Bit security as computational cost for winning games with high probability. In Mehdi Tibouchi and Huaxiong Wang, editors, *Advances in Cryptology – ASIACRYPT 2021 – 27th International Conference on the Theory and Application of Cryptology and Information Security, Singapore, December 6-10, 2021, Proceedings, Part III*, volume 13092 of *Lecture Notes in Computer Science*, pages 161–188. Springer, 2021. `doi:10.1007/978-3-030-92078-4_6`.

10 Hoeteck Wee. On obfuscating point functions. In Harold N. Gabow and Ronald Fagin, editors, *37th ACM STOC*, pages 523–532. ACM Press, May 2005. `doi:10.1145/1060590.1060669`.

11 Andrew Chi-Chih Yao. Coherent functions and program checkers (extended abstract). In *22nd ACM STOC*, pages 84–94. ACM Press, May 1990. `doi:10.1145/100216.100226`.