On Negation Complexity of Injections, Surjections and Collision-Resistance in Cryptography

Douglas Miller Adam Scrivener Jesse Stern

Muthuramakrishnan Venkitasubramaniam*

Abstract

Goldreich and Izsak (Theory of Computing, 2012) initiated the research on understanding the role of negations in circuits implementing cryptographic primitives, notably, considering one-way functions and pseudo-random generators. More recently, Guo, Malkin, Oliveira and Rosen (TCC, 2014) determined tight bounds on the minimum number of negations gates (i.e., negation complexity) of a wide variety of cryptographic primitives including pseudo-random functions, error-correcting codes, hardcore-predicates and randomness extractors.

We continue this line of work to establish the following results:

1. First, we determine tight lower bounds on the negation complexity of collision-resistant and target collision-resistant hash-function families.
2. Next, we examine the role of injectivity and surjectivity on the negation complexity of one-way functions. Here we show that,
   (a) Assuming the existence of one-way injections, there exists a monotone one-way injection. Furthermore, we complement our result by showing that, even in the worst-case, there cannot exist a monotone one-way injection with constant stretch.
   (b) Assuming the existence of one-way permutations, there exists a monotone one-way surjection.
3. Finally, we show that there exists list-decodable codes with monotone decoders.

In addition, we observe some interesting corollaries to our results.

*University of Rochester

1 Introduction

A boolean circuit C is *monotone* if it comprises only of fanin-2 AND and OR gates. Monotone circuits have been extensively studied in complexity theory, where one of the fundamental goals is to establish lower bounds on circuit size, and learning theory [17, 10, 5, 1, 6]. In the context of cryptography, Goldreich and Izsak began exploring whether cryptographic primitives can be implemented via monotone circuits [8]. They showed that, assuming the existence of one-way functions, there exists a one-way function implementable by monotone circuits. On the negative side, they proved that psuedo-random generators cannot be implemented using monotone circuits. More recently, Guo, Malkin, Oliveira and Rosen, inspired by a long series of works [12, 7, 15, 2, 3, 16, 13, 4] in complexity theory, initiated the study of the *negation complexity* of realizing cryptographic primitives. Loosely speaking, the *negation complexity* of a primitive is the minimum number of negation gates required in any circuit implementing that primitive.

Markov [12] showed that any boolean function on n-bit inputs can be realized using a circuit with at most $\lceil \log(n+1) \rceil$ negation gates. A more efficient version was proved by Fischer [7], which, in particular, implies that any polynomial-time computable boolean function can be realized using a polynomial-size monotone circuit with $\lceil \log(n+1) \rceil$ negation gates. In essence, the negation complexity of any arbitrary polynomial-time computable function is $O(\log n)$. In [9], quite surprisingly, they show that this is *tight* for various primitives such as pseudorandom functions, error-correcting codes, randomness-extractors and generic hard-core predicates. Interestingly, a new technique was required to establish each lower-bound.

Our first motivating question concerns the negation complexity of collision-resistant hash-functions. Various notions of collision-resistance have been used in cryptographic constructions. Standard hash-functions, such as MD5 and SHA, are referred to as collision-resistant hash-functions (CRH), where the security game of such function families requires an adversary to find a pair of colliding inputs given a function picked uniformly from the family. A weaker form of collision-resistance that can be realized from one-way functions, referred to as universal one-way hash-functions (or target collision-resistant functions (TCR)) [14] require the adversary to produce a target for which it needs to find a collision before seeing the description of the hash-function. Yet another related primitive is second-preimage resistant hash-functions (SPR) where it is infeasible for any adversary to find a collision for a randomly chosen hash-function on a uniformly chosen input. Our first motivating question is:

> *What is the negation complexity of collision-resistance?*

Another interesting result presented in the work of Guo et al. [9] proves the impossibility of monotone one-way permutations.[1] The main result shows that any monotone circuit that implements a permutation must be of the form where the output bits are a permutation of the input bits, in essence, making them easily invertible with probability 1. The result of Guo et al. and Goldreich and Izsak show a gap in what kind of one-way functions are achievable using monotone circuits. Our second motivating question is:

> *Do there exist polynomial-sized one-way surjections or one-way injections with monotone circuit implementations?*

1.1 Our Results

Our first result establishes optimal bounds on the negation complexity of CRHs and TCRs.

[1] A permutation is a length-preserving function that is both *injective* (i.e., one-to-one) and *surjective* (i.e., onto).

Theorem 1 (Informal)**.** *Collision-resistant hash-functions and Target Collision-resistant hash-functions require $\theta(\log n)$ negation gates.*

While we resolve the question for CRHs and TCRs, the problem remains open for Second Pre-Image Resistant functions. Interestingly, since TCRs and SPRs are equivalent,[2] any result on the negation complexity of SPR could reveal something about the negation complexity of universal hash-functions and, consequently, the XOR function.

Our second result explores whether injectivity or surjectivity influences the negation complexity of constructing one-way functions. We answer in the affirmative that, if we relax one of these conditions from a one-way permutation, it is indeed possible to construct monotone one-way functions. More precisely, we prove the following theorems.

Theorem 2 (Informal)**.** *Assume the existence of one-way surjections. Then there exists a (poly-sized) one-way surjection that is computable by a monotone circuit.*

Theorem 3 (Informal)**.** *Assume the existence of one-way injections. Then there exists a (poly-sized) one-way injection that is computable by a monotone circuit.*

We remark that if we start with a one-way permutation $f : \{0,1\}^n \to \{0,1\}^n$, then we can construct a one-way injection $g : \{0,1\}^n \to \{0,1\}^{3n}$ and a one-way surjection $h : \{0,1\}^n \to \{0,1\}^{n-2\log n}$ that are both monotone.

We also complement our injectivity result by showing that there do not exist one-way injections with constant stretch, where stretch refers to the difference in lengths of the output and input. More formally, we prove the following theorem:

Theorem 4 (Informal)**.** *There exists an algorithm that, given oracle access to an injective function $f : \{0,1\}^n \to \{0,1\}^m$, where $m - n = O(1)$, can invert f in polynomial time.*

This can be viewed as a generalization of the result of [9] where they give an algorithm only when $m = n$. We remark that not only can the algorithm invert an arbitrary element in the range of the function, it can also determine whether an element is in the range of the function.

1.2 Our Techniques

We remark that for most of our results we rely on different techniques. We believe that understanding the negation complexity of cryptography is a fundamental problem, and that our work sheds light on how different properties of functions influence the negation complexity of cryptographic primitives. We briefly mention some of our techniques below.

Collision-Resistant Hash-Functions: Establishing that collision-resistant hash-function requires negation gates follows using the pigeon-hole principle. Consider any monotone function with m-bit outputs: In any $m+1$ chain of inputs $x^1, \ldots, x^{m+1}$ (i.e. $x^i \preceq x^{i+1}$)[3] which has strictly increasing Hamming weights there must exist a pair of consecutive inputs that collide. This is because each bit of the output can change at most once in the sequence and there are only m output bits. We can then conclude by using the fact that hash-functions are compressing, i.e. $m < n$ and a chain can be easily constructed by simply flipping input bits one at a time from 0 to 1 starting from 0^n. Following ideas of [9], using a theorem of Markov [12], we also extend this to show that if the collision-resistant function is highly-compressing (namely by polynomially many bits) then it must require $\log n$ negations.

[2] A TCR can be constructed from an SPR by computing a universal hash-function (1-wise independent) on the input before feeding it to the SPR function, namely, masking the inputs with a random key

[3] We write $a \preceq b$, if for any i, i^{th} bit of a is 1 implies that the i^{th} bit of b is 1.

One-Way Monotone Surjections: Our technique for constructing one-way monotone surjections follows the idea of Goldreich and Izsak [8] for constructing one-way monotone functions. Let $\mathsf{ham}(x)$ denote the Hamming weight of the string x. Recall that in their construction, starting from any one-way function $f : \{0,1\}^n \to \{0,1\}^m$, they consider a function f' that behaves as follows:

- On inputs x such that $\mathsf{ham}(x) < \frac{n}{2}$, f' assumes the value 0^m.
- On inputs x such that $\mathsf{ham}(x) = \frac{n}{2}$, f' assumes the value $f(x)$.
- On inputs x such that $\mathsf{ham}(x) > \frac{n}{2}$, f' assumes the value 1^m.

They then show that this function can be implemented using a monotone circuit. Since the middle slice occupies at least $\frac{1}{\sqrt{n}}$ fraction of the total inputs, f' is a weak one-way function. Then a strong monotone one-way function can be obtained via standard parallel repetition.

Suppose we start with a function f that is surjective and apply the construction specified above, the resulting function will no longer be surjective. Instead, we rely on a an error correcting code G with a certain property. More precisely, G maps 'balanced' (i.e., contains the same number of 0s as 1s) strings of length $n + O(\log n)$ to strings of length n surjectively. An example of one such code is the Knuth code [11]. Given such codes, we can augment Goldreich's construction to get a monotone surjective one-way function as follows: We consider a function f' that takes as input $n + O(\log n)$ inputs and first applies G to its input, followed by f on balanced inputs while the rest of the inputs are defined just as before. It follows from the definition that this function is surjective and weak one-way.

One-Way Monotone Injections: The technique for our one-way monotone injection construction can be explained using the same construction as above. Note that, in this construction, inputs of Hamming weight smaller than $n/2$ all map to the same output. Similarly, inputs with Hamming greater than $n/2$ also map to the same value. The main idea here to achieve injection is to get rid of these collisions.

Towards this, we add two blocks of n-bits to the output where the purpose of the first block is to handle inputs of Hamming weight smaller than $n/2$ and the other for those with Hamming weight greater than $n/2$. The result then follows by showing that the following functions can be built using monotone circuits:

- $f_{\mathsf{lower}}(x) = x$ when $\mathsf{ham}(x) < |x|/2$ and $f(x) = 1^n$ otherwise.
- $f_{\mathsf{upper}}(x) = x$ when $\mathsf{ham}(x) > |x|/2$ and $f(x) = 0^n$ otherwise.

A Deterministic Algorithm to Invert Monotone Injection One of our main technical contributions is in proving our lower bound on injections. Here we give an explicit deterministic algorithm that allows inversion of any injections with constant stretch. In fact, we provide a general analysis where the run-time of the algorithm depends on the input length and stretch.

At a high-level, the idea is that given a target $b = f(s)$ of a function $f : \{0,1\}^n \to \{0,1\}^m$, we systematically break f down into a number of different cases, each of which is permutation-like, in the sense that most of the output bits are permutations of the input bits. More specifically, we reduce f to a piecewise function of restrictions of f (over domains where certain input bits are held constant) where fixing one additional bit of the input fixes exactly one bit of the output. At that point we use the correspondence between input and output bits to find pre-images for all

but the stretch bits of y, which are brute-forced. We iterate through the cases until we find one that produces a preimage for y, or we exhaust all cases, in which case no preimage can exist.

The main trick of the proof is the reduction of f into a polynomial (assuming $m - n = O(1)$) number of cases. Essentially, this is done by searching for an input bit that causes more than one output bit to be fixed when set to 1 (or 0), and recursing into both the case where we set it to one and set it to zero. Then, by a combinatorial argument, we show that this will only recurse polynomially many times. In particular, this helps us characterize an important property of functions of super-contstant stretch (that is, $f : \{0,1\}^n \mapsto \{0,1\}^{n+\omega(1)}$) that are potentially hard to solve: they will very often exhibit the worst-case behavior where fixing an input bit to 0 fixes only one output bit to 0, but fixing it to 1 fixes multiple output bits to 1, or vice versa. Functions like this are thus ideal candidates for one-way monotone injections of small stretch.

2 Definitions and Notations

For some $x, y \in \{0,1\}^n$, we write $x \preceq y$ if $x^i \leq y^i$ for all $i \in [n]$. By definition, a Boolean function $f : \{0,1\}^n \to \{0,1\}$ is monotone iff $f(x) \leq f(y)$ whenever $x \preceq y$ and a function $g : \{0,1\}^n \to \{0,1\}^m$ is monotone iff every output bit of g is a monotone Boolean function. If x is a binary string, let $\mathsf{ham}(x)$ be the *Hamming weight* of x, i.e. the amount of 1s that appear in x. A chain $X = (x^1, ..., x^t)$ is a monotone sequence of strings over $\{0,1\}^n$, i.e, $x^i \leq x^{i+1}$ for every $i \in [1, t-1]$. Define an increasing chain as a chain for which all $x^j \neq x^k$ when $j \neq k$.

Let $f : \{0,1\}^n \to \{0,1\}$ be a Boolean function, and let $X = (x^1, x^2, \ldots, x^t)$ be a chain. Then, define $a(f, X)$ to be the largest set of indexes $\{0 \leq i_0, \ldots, i_m \leq n-1\}$ such that $f(x^{i_j}) \neq f(x^{i_{j+1}})$ for every $j \in [0, m-1]$. Furthermore, define $a(f) = max_X(a(f, X))$ where X is a chain to be the *alternating complexity* of f.

The following result was shown by Markov [12]:

Theorem 5. *Let $f : \{0,1\}^n \to \{0,1\}^m$ be a Boolean function computed by a circuit with at most t negations. Then $a(f) \in O(2^t)$.*

2.1 One-Way Functions

Definition 1. *Let $f : \{0,1\}^* \to \{0,1\}^*$ be a polynomial-time computable function. f is (strong)* one-way *if for every* PPT *machine A, there exists a negligible function $\nu(\cdot)$ such that*

$$\Pr[x \leftarrow \{0,1\}^n; y = f(x) : A(1^n, y) \in f^{-1}(f(x))] \leq \nu(n)$$

A function f $\{0,1\}^n \to \{0,1\}^m$ is said to be *injective* if for any $x, y \in \{0,1\}^n$, $x \neq y \implies f(x) \neq f(y)$. f is said to be *surjective* if for any $z \in \{0,1\}^m$, there is an $x \in \{0,1\}^n$ such that $f(x) = z$. f is said to be a weak one-way function if there exists a polynomial p such that for all adversaries A the probability with which it can invert the function is at most $\frac{1}{p(n)}$ for sufficiently large n where the probability is over a randomly chosen input x and the random coins of A.

Definition 2 (Exponentially-hard one-way functions)**.** *Let $f : \{0,1\}^* \to \{0,1\}^*$ be a polynomial-time computable function. f is* exponentially-hard one-way *if there exists some c such that for every probabilistic adversary A that runs in $O(2^{cn})$ time, there exists a negligible function $\nu(\cdot)$ such that*

$$\Pr[x \leftarrow \{0,1\}^n; y = f(x) : A(1^n, y) \in f^{-1}(f(x))] \leq \nu(n)$$

Definition 3 (Regular one-way functions)**.** *Let $f : \{0,1\}^* \to \{0,1\}^*$ be a one-way function. f is regular if there exists a function $\alpha : N \to N$ such that for every $n \in N$ and every $x \in \{0,1\}^n$ we have:*

$$|f^{-1}(f(x))| = \alpha(n)$$

We assume that the regularity $\alpha(\cdot)$ of a function f is not known (i.e. not polynomial-time computable). Without loss of generality, we assume the one-way function is length preserving i.e. $f(\{0,1\}^n) \subseteq \{0,1\}^n$.

2.2 Target Collision-Resistant Hash-Function Families

Definition 4. *Let $\mathcal{G} = \{g_k\}_{k \in \mathcal{K}}$ be a family of functions where each function g_k goes from $\{0,1\}^{n+\ell}$ to $\{0,1\}^n$. We say that $\mathcal{G}$ is a a* Collision-Resistant Hash-Function Family *if (i) the functions g_k are efficiently computable and (ii) for every efficient adversary A , the probability that A succeeds in the following game is negligible in n:*

- *Choose $k \leftarrow \mathcal{K}$*
- *Let $x, x' \leftarrow A(1^n, k)$*
- *A succeeds if $x \neq x'$ and $g_k(x) = g_k(x')$*

Definition 5. *Let $\mathcal{G} = \{g_k\}_{k \in \mathcal{K}}$ be a family of functions where each function g_k goes from $\{0,1\}^{n+\ell}$ to $\{0,1\}^n$. We say that $\mathcal{G}$ is a a* Universal One-Way Hash-Function Family *if (i) the functions g_k are efficiently computable and (ii) for every efficient adversary A , the probability that A succeeds in the following game is negligible in n:*

- *Let $(x, \sigma) \leftarrow A(1^n)$*
- *Choose $k \leftarrow \mathcal{K}$*
- *Let $x' \leftarrow A(\sigma, k)$*
- *A succeeds if $x \neq x'$ and $g_k(x) = g_k(x')$*

Universal One-Way Hash-Function Families [14] as defined above enjoy the property of target collision-resistance. The related notion of *Second Preimage Resistance* follows the same security game with the exception that x and k are uniformly chosen and handed to the adversary instead of allowing the adversary to first choose x before seeing the key. It is well-known how to construct UOWHFs from second preimage resistant families.

2.3 Error-Correcting and Balanced Codes

Let $E : \{0,1\}^n \to \{0,1\}^m$ be a polynomial-time computable function. Given strings $y, y' \in \{0,1\}^m$, define

$$\Delta(y, y') = \frac{\sum_{i=1}^{m} y_i \neq y'_i}{m}$$

We say E is γ-*error-correcting* if for any pair of distinct inputs $x, x' \in \{0,1\}^n$, $\Delta(E(x), E(x')) \geq \gamma$.

Definition 6. *Let $f : \{0,1\}^n \to \{0,1\}^m$ be a polynomial-time computable function. f is a* balanced code *if every element in the image of f is balanced, i.e. $\lceil \frac{m}{2} \rceil$ of its bits are 1 and $\lfloor \frac{m}{2} \rfloor$ of its bits are 0.*

In addition, we define $\mathsf{BAL_n} = \{x \in \{0,1\}^n | x \text{ is balanced}\}$, and O_n to be the strict order on $\mathsf{BAL_n}$ that is inherited from the normal lexicographic order on $\{0,1\}^n$. For example, if $n = 4$, O_n = 0011, 0101, 0110, 1001, 1010, 1100. Finally, if w is a binary string, let w^k represent w whose first k bits are flipped and the rest of the bits remain the same.

We are interested in constructing a surjective, poly-time computable function $G : \mathsf{BAL_n} \to \{0,1\}^{n-2\log n}$ (assuming n is a power of 2 for simplicity). This is known as a decoder for a balanced Knuth Code [11]. For immediate reference we present the code and the proof in Appendix **??**.

Theorem 6. *The following function $G : \mathsf{BAL_n} \to \{0,1\}^{n-2\log n}$ is surjective and computable in polynomial-time.*

```
1: function G(s):
2:     u ← first 2 log n bits of s
3:     if u has balanced Hamming weight then
4:         k ← position of u in O_{2 log n} for BAL_n
5:         w ← last n − 2 log n bits of s
6:         return w^k
7:     else
8:         return 0^{n−2 log n}
```

3 Negation Complexity of Collision-Resistance

In this section, we prove our results regarding the negation complexity of collision-resitant hash-functions. We start with a warm-up Lemma that illustrates the main idea.

Lemma 1. *Let $g : \{0,1\}^n \to \{0,1\}^m$ be any monotone function such that $m < n$. Then there exists an adversary A that can output $x, x' \in \{0,1\}^n$ such that $x \neq x'$ and $g(x) = g(x')$.*

Proof. On a high-level the proof will demonstrate that every increasing chain has a collision. Consider an algorithm A that does the following:

- Choose an arbitrary increasing chain $X = (x^1, ..., x^{n+1})$ over $\{0,1\}^n$ and compute $g(x^i)$ for all i. Suppose there exists an i such that $g(x^i) = g(x^{i+1})$. If so, output (x^i, x^{i+1}). Otherwise output fail.

For all pairs $g(x^i)$ and $g(x^{i+1})$, the montonicity of g and the definition of an increasing chain implies that, either there exists at least one position in the bits of $g(x^i)$ that was a 0 turns to a 1 in $g(x^{i+1})$, or $g(x^i) = g(x^{i+1})$. There are n possible values for i and a maximum of m possible positions that can be flipped from 0 to 1. Furthermore, monotonicity implies that each position can flip at most once. Since $n > m$, by the pigeonhole principle, there must exist at least one such pair that has no bit that flips and thus, for this pair, $g(x^i) = g(x^{i+1})$ and $x^i \neq x^{i+1}$. □

Corollary 1. *There does not exist a family of collision-resistant hash-functions that is computable by monotone circuits.*

Next, we extend the proof to rule out monotone implementations of target collision-resistant functions. Recall that in the security game for target collision-resistance, the adversary needs to pick a target x for which it is required to find a collision before it receives the description of the hash-function.

Theorem 7. *There does not exist a family of target collision-resistant hash-functions that is computable by monotone circuits.*

Assume, for contradiction, that there exists a family of target collision-resistant hash-functions $\{g_k\}_{k\in\mathcal{K}}$ where g_k is a function from $\{0,1\}^n$ to $\{0,1\}^m$ with $m < n$ that can be computed using a monotone circuit of polynomial size. From Lemma 1, we know that for any compressing function, and therefore for any g_k, every increasing chain of inputs has at least one collision.

Pick an arbitrary increasing chain $X = (x^1, ..., x^{n+1})$. Since every function g_k has a collision in this chain, using a standard averaging argument we can conclude that there exists at least one index in $[n]$, say i^*, such that with probability at least $\frac{1}{n}$ over the functions $k \in \mathcal{K}$, $g_k(x^{i^*}) = g_k(x^{i^*+1})$. Consider an adversary A that picks an input uniformly from the chain X and submits it as a target. Upon receiving the function g_k, it outputs (x^i, x^{i+1}) if $g_k(x^i) = g_k(x^{i+1})$. Otherwise, it outputs fail.

It follows that A picks x^{i^*} with probability at least $\frac{1}{n}$ and when this happens, it succeeds in outputting a collision with probability at least $\frac{1}{n}$. Therefore, A succeeds in outputting a collision with non-negligible probability, which is a contradiction.

Next, we strengthen our bound by proving that both CRHs and TCRs are highly non-monotone.

Lemma 2. *Let $g : \{0,1\}^n \to \{0,1\}^m$ be an arbitrary function that can be implemented by a circuit with $t = (1-\epsilon)\log n$ gates. Then there exists a constant $c > 0$ such that, if $m < cn^\epsilon$, then there exists an adversary A that the can output $x, x' \in \{0,1\}^n$ such that $x \neq x'$ and $g(x) = g(x')$.*

Proof. We will rely on Markov's theorem (cf. Theorem 5) that shows that for any boolean function f on n-bit inputs that can be computed by a circuit with at most t gates, $a(f) = O(2^t)$. Recall that $a(f) = max_{\mathcal{X}} a(f, \mathcal{X})$ where $\mathcal{X}$ is any non-decreasing chain over $\{0,1\}^n$ and $a(f, \mathcal{X})$ denotes the number of times the output of f changes when iterating through the chain.

We prove that any increasing chain will have consecutive elements whose outputs are identical under g. Then we can rely precisely on the same adversary as in Lemma 1 that examines the elements in an arbitrary increasing chain and outputs if it finds a collision on any consecutive elements in the chain.

Let $X = (x^1, ..., x^{n+1})$ be an arbitrary increasing chain over $\{0,1\}^n$. As before, we have that every pair of consecutive elements either collide under g or differ by at least one position in the output. By Markov's theorem we know that each position in the output can change at most $c'2^t = c'n^{1-\epsilon}$ times. Hence if $n > mc'n^{1-\epsilon}$ then there must be a collision by the pigeonhole principle. By fixing $c = \frac{1}{c'}$ and observing that $m < \frac{1}{c'}n^\epsilon$, the lemma follows. □

Using the same techniques as before we obtain the following corollary.

Corollary 2. *Let $\{g_k\}_{k\in\mathcal{K}}$ be a family of collision-resistant hash-functions or target collision-resistant functions where g_k is a function from n bits to m bits such that $m < cn^\epsilon$ and computable by a circuit with t gates. Then $t \geq (1-\epsilon)\log n$.*

4 One-Way Monotone Injections

Theorem 8. *If strong one-way permutations exist, then one-way monotone injections exist. In particular, if strong one-way permutations exist, then strong one-way monotone injections exist from n bits to $3n$ bits.*

Proof. Suppose there exists some one-way family of permutations $f_n : \{0,1\}^n \mapsto \{0,1\}^n$ that have a polynomial bound $Q(n)$ on their circuit size. Then, let $T_{i,n} : \{0,1\}^n \mapsto \{0,1\}$ be defined as the function such that $T_i(x)$ is true if and only if x has at least i of its n bits set to 1. First, we show that T_i can be calculated via a polynomial-sized circuit through the following recursive construction: let $T_0(x) = 1$ for all x, and for $i > 0$, let $T_i = \bigvee_{j=1}^{n} \left(x_j \wedge T_{i-1}(x \text{ with bit } j \text{ set to zero})\right)$. Recall that if f_n has circuit size bounded by $Q(n)$, then by De Morgan's laws, we can still construct a monotone circuit $f'_n : \{0,1\}^n \times \{0,1\}^n \mapsto \{0,1\}^n$ with circuit size $\leq Q(n)$ such that $f'_n(x, \neg x) = f_n(x)$ for all x. If we set $H_{i,n}(x) = T_{\lfloor \frac{n}{2} \rfloor, n}(x \text{ with bit } i \text{ set to } 0)$, then note that for x with Hamming weight $\lfloor \frac{n}{2} \rfloor$, $f'_n(x, H_{1,n}(x) \| H_{2,n}(x) \| \cdots \| H_{n,n}(x)) = f_n(x)$. Then, defining $F_n(x) = f'_n(x, H_{1,n}(x) \| H_{2,n}(x) \| \cdots \| H_{n,n}(x))$, consider the following constructed family:

$$g_n(x) = F_n(x) \| \left(x \wedge [T_{\lfloor \frac{n}{2} \rfloor + 1, n}(x)]^n\right) \| \left(x \vee [T_{\lfloor \frac{n}{2} \rfloor, n}(x)]^n\right)$$

Note that the concatenation of the H functions has a polynomial bound in n. Thus, we can construct g_n in a polynomial-size monotone circuit.

Claim 1. *g_n is injective.*

Proof. For inputs x with Hamming weight less than $\lfloor \frac{n}{2} \rfloor$, the last n bits of g_n equals x, and so within this range, there cannot be any collisions. A similar proof shows that there are no collisions among inputs with Hamming weight greater than $\lfloor \frac{n}{2} \rfloor$.

In addition, there are no collisions between inputs of Hamming weight $\lfloor \frac{n}{2} \rfloor$, as F_n is injective.

There are also no collisions between inputs of Hamming weight less than $\lfloor \frac{n}{2} \rfloor$ and those with Hamming weight greater than $\lfloor \frac{n}{2} \rfloor$, because the third section of the latter consists of all 1s, and the third section of the former cannot have Hamming weight greater than or equal to $\lfloor \frac{n}{2} \rfloor$.

Finally, there are no collisions between inputs of Hamming weight less than $\lfloor \frac{n}{2} \rfloor$ and Hamming weight equal to $\lfloor \frac{n}{2} \rfloor$, as the third section of the former cannot have Hamming weight greater than or equal to $\lfloor \frac{n}{2} \rfloor$, and the third section of the latter consists of all 1s. A similar proof shows that there are no collisions between inputs with Hamming weight equal to $\lfloor \frac{n}{2} \rfloor$ and inputs with Hamming weight greater than $\lfloor \frac{n}{2} \rfloor$. □

Now, recall that there are $\Omega(1/\sqrt{n})$ fraction of inputs with Hamming weight $\lfloor \frac{n}{2} \rfloor$. Then, since the first n bits of $g_n = f_n$ for inputs with Hamming weight $\lfloor \frac{n}{2} \rfloor$, the first n bits of $g_n = f_n$ for $\Omega(1/\sqrt{n})$ fraction of inputs. Therefore, g_n is $\Omega(1/\sqrt{n})$ hard. The proof of this is very similar to the proof found in Appendix C, which follows the proof in [8]. g_n can then be amplified by a simple extension to obtain strong one-way injections. □

Corollary 3. *If a one-way injection $f_I : \{0,1\}^n \to \{0,1\}^m$ exists, then a one-way monotone injection $g_I : \{0,1\}^n \to \{0,1\}^{m+2n}$ exists.*

Proof. Since the preceding proof does not require that the one-way function be surjective, the proof is equivalent. Furthermore, the output of the constructed function is m bits concatenated by two sequences of n bits each. □

As a simple corollary, we can construct regular length-preserving functions:

Corollary 4. *If one-way injections exist, regular length-preserving one-way monotone functions exist.*

Proof. By the preceding theorem, we can construct a family of monotone weak one-way functions. $g_n : \{0,1\}^n \mapsto \{0,1\}^{3n}$. Then, construct a family of functions $h_n : \{0,1\}^{3n} \mapsto \{0,1\}^{3n}$ such that

$$h_n(x) = g_n(\text{last n bits of x})$$

Then, h_n is 2^{2n}-regular, as for any element y in the image of h_n, there is exactly one element $x' \in \{0,1\}^n$ such that $g(x') = y$. Then, there are exactly 2^{2n} elements in the domain of h_n whose last n bits are equal to x', thus each of these maps to y. Now, if there was an adversary D and a polynomial q such that h_n can be inverted with probability $1 - A(n) + \frac{1}{q(n)}$, an adversary

$$D'(y) = \text{last } n \text{ bits of } D(y)$$

would invert g_n with probability $1 - A(n) + \frac{1}{q(n)}$ as well, thus contradicting the fact that g_n is weak one-way. Thus, h_n is weak one-way, and can be extended to a strong one-way function. □

Using a complexity leveraging argument, we obtain the following corollary assuming the existence of exponentially hard one-way functions.

Corollary 5. *If strong exponentially hard one-way permutations exist, strong one-way monotone injections exist from n bits to $n + \omega(\log(n))$ bits.*

We need the following lemma to prove our corollary.

Lemma 3. *If $f_n : \{0,1\}^n \mapsto \{0,1\}^{m(n)}$ is sequence of strong exponentially hard one-way functions, then for any function $d \in \omega(\log(n))$, the sequence of functions $g_n : \{0,1\}^{d(n)} \mapsto \{0,1\}^{m(d(n))}$ defined as $g_n(x) = f_{d(n)}(x)$ is strong one-way with respect to poly(n).*

Proof. Suppose that given some f_n exponentially hard (specifically, $O(2^{cn})$ hard) and some $d \in \omega(\log(n))$, there exists some probabilistic adversary A that can invert g_n with non-negligible probability in (assuming n is sufficiently large) poly(n) time. However, this means that A can invert f_n in $Bn^p = B2^{\log(n)p}$ time. Since f_n is exponentially hard, this must be greater than $k2^{cd(n)}$. Since $d \in \omega(\log(n))$, this will be false for large enough n, a contradiction. By contradiction, g_n is a strong one-way function with respect to poly(n). □

Next using the Lemma we establish our Corollary.

Proof. Suppose some f_n is a sequence of strong exponentially hard one-way permutations. Let $d(n) \in \omega(\log(n))$. Now, by the lemma, there exists a strong one-way sequence of permutations g_n on $d(n)/2$ bits that is strong one-way with respect to poly(n). By the previous theorem, this means there exists a strong one-way monotone injection from $d(n)/2$ bits to $1.5 * d(n)$ bits. Append $n - d(n)/2$ input and output bits (each input mapping directly to the corresponding output bit, so as to maintain injectivity) to get a strong one-way monotone injection from n bits to $n + d(n)$ bits, as desired. □

5 One-Way Monotone Surjections

In this section, we provide our construction of one-way monotone surjections. First we require the following definitions:

Definition 7. *If $f : \{0,1\}^n \mapsto \{0,1\}^m$ is a function, let the $k-cut$ of f be the function $f^{(k)} : \{0,1\}^n \mapsto \{0,1\}^m$ such that*

$$f^{(k)}(x) = \begin{cases} 0^m & : \text{ham}(x) < k \\ f(x) & : \text{ham}(x) = k \\ 1^m & : \text{ham}(x) > k \end{cases}$$

Definition 8. *If x is a binary string and k is an integer, the* threshold function *of threshold* k *is the function T_k such that*

$$T_k(x) = \begin{cases} 0 & : ham(x) < k \\ 1 & : ham(x) \geq k \end{cases}$$

Definition 9. *If x is a binary string, let $x^{i\to 0}$ stand for the binary string produced by changing the i^{th} bit in x to 0.*

Proposition 1. *If x is a binary string and ham(x) = k, then $T_k(x^{i\to 0}) = \neg x_i$.*

Proof. If $x_i = 1$, then $x^{i\to 0}$ has Hamming weight $k-1$. Thus, $T_k(x^{i\to 0}) = 0$. If $x_i = 0$, changing the i^{th} bit to 0 does not change x, and so it does not change the Hamming weight. Thus, $T_k(x^{i\to 0}) = 1$. □

The next proposition implicit in [8], states that $f^{(k)}$ is computable by a monotone circuits. For completeness, we provide a proof of the proposition in the appendix.

Proposition 2. *If $f : \{0,1\}^n \mapsto \{0,1\}^m$ is a function computable in polynomial time, then for any k, the $k-cut$ of f, $f^{(k)}$, is computable by a polynomial-sized monotone circuit.*

Proof. Let C be a polynomial-sized circuit for f. Now, modify C to be a circuit with exactly n not-gates represented as directed edges, one leaving each leaf of the circuit. Label these edges $e_1 \ldots e_n$. Such a circuit C' can be constructed that computes the same function as C, and is also polynomial in size.

Now, for each i, replace the edge e_i with the circuit computing $T_k(x^{i\to 0})$. Note that the threshold function can be computed by a monotone circuit polynomial in the size of x. Call this new circuit C^k.

Recall that, when x has Hamming weight k, $T_k(x^{i\to 0}) = \neg x_i$, and thus the threshold function computes the 'not' function, and so C^k computes the same function as C' (and therefore as C) in this special case.

Our final function will be the circuit that computes

$$(T_k(x)^m \ \& \ C^k(x)) \mid T_{k+1}(x)^m$$

where & and | are the bitwise 'and' and bitwise 'or' functions. This circuit is clearly monotone, as C^k is monotone, as is the threshold function, and only 'and' and 'or' gates are required to compute bitwise 'and' and bitwise 'or'. Furthermore, it is polynomial in n, as C^k and the threshold functions are polynomial in n. All that is left to show is that it computes the $k-cut$ of f.

- If $ham(\mathrm{x}) < k$, then $T_k(x)^m = T_{k+1}(x)^m = 0^m$, and so

$$(T_k(x)^m \;\&\; C^k(x)) \mid T_{k+1}(x)^m = (0^m \;\&\; C^k(x)) \mid 0^m = 0^m$$

- If $ham(\mathrm{x}) = k$, then $T_k(x)^m = 1^m$ and $T_{k+1}(x)^m = 0^m$, so

$$(T_k(x)^m \;\&\; C^k(x)) \mid T_{k+1}(x)^m = (1^m \;\&\; C^k(x)) \mid 0^m = C^k(x) \mid 0^m = C^k(x)$$

  Furthermore, since $ham(\mathrm{x}) = k$, we have that $T_k(x^{i\to 0}) = \neg x_i$, thus $C^k(x) = C'(x) = C(x) = f(x)$.

- If $ham(\mathrm{x}) > k$, then $T_k(x)^m = T_{k+1}(x)^m = 1^m$, and so

$$(T_k(x)^m \;\&\; C^k(x)) \mid T_{k+1}(x)^m = (1^m \;\&\; C^k(x)) \mid 1^m = 1^m$$

Therefore, C^k equals $f^{(k)}$.

□

Theorem 9. *If one-way permutations exist, then one-way surjections exist with $O(\log n)$ compression which can be computed by a polynomial-sized monotone circuit.*

Proof. The following function $G : \mathsf{BAL_n} \to \{0,1\}^{\mathrm{n}-2\log \mathrm{n}}$ will be useful in proving this theorem:

```
1: function G(s):
2:     u ← first 2 log n bits of s
3:     if u has even Hamming weight then
4:         k ← position of u in O_{2 log n} for BAL_{2 log n}
5:         w ← last n − 2 log n bits of s
6:         return w^k
7:     else
8:         return 0^{n−2 log n}
```

Claim 2. *$G : \mathsf{BAL_n} \to \{0,1\}^{\mathrm{n}-2\log \mathrm{n}}$ is surjective and computable in polynomial time.*

Proof: First, we prove that G is surjective. Towards proving this, we prove the following Claim.

Claim 3. *Suppose $w \in \{0,1\}^{n-2\log n}$. Then, there exists a $k \in \{0,\ldots,n-2\log n\}$ such that w^k has Hamming weight $\frac{|w|}{2}$.*

Proof. To see this, let $\mathrm{ham}(\mathrm{w}) = l$. Observe that w^0 has Hamming weight l, and $w^{|w|}$ has Hamming weight $|w| - l$. Furthermore, $w^0, w^1, w^2, \ldots$ is a sequence of bit strings such that $\mathrm{ham}(w^i) = \mathrm{ham}(w^{i+1}) \pm 1$. Thus, the Hamming weights of $w^0, w^1, w^2, \ldots, w^{|w|}$ hit every natural number (inclusively) in between $\mathrm{ham}(w^0) = l$ and $\mathrm{ham}(w^{|w|}) = |w| - l$. In this set of numbers lies $\frac{|w|}{2}$. This is because, if $l > \frac{|w|}{2}$, $|w| - l < |w| - \frac{|w|}{2} = \frac{|w|}{2}$, and a similar argument goes for when $l < \frac{|w|}{2}$, and is trivial when $l = \frac{|w|}{2}$. Thus, there is a k such that $\mathrm{ham}(w^k) = \frac{|w|}{2}$. □

Now, to prove that G is surjective, it suffices to show the following:

Claim 4. *If u is the k^{th} string in $\mathsf{BAL}_{2\log \mathrm{n}}$ according to $O_{2\log n}$, $G(uw^k) = w$.*

Proof. First, we show that the k^{th} string in $\mathsf{BAL}_{2\log \mathsf{n}}$ indeed exists. It suffices to show that $|\mathsf{BAL}_{2\log \mathsf{n}}| \geq |\mathsf{w}|$, as we are never required to use $k = |w|$, as if $w^{|w|}$ has even Hamming weight, so does w^0. Thus we only consider $k \in [0, \ldots, |w| - 1]$. First,

$$|\mathsf{BAL}_{2\log \mathsf{n}}| = \binom{2\log \mathsf{n}}{\log \mathsf{n}} \geq \frac{2^{2\log \mathsf{n}-1}}{\sqrt{\log \mathsf{n}}}$$

by Stirling's approximation. Then, we have

$$\frac{2^{2\log n-1}}{\sqrt{\log n}} = \frac{n^2}{2\sqrt{\log n}} \geq \frac{n^2}{2\log n}$$

Finally,

$$\frac{n^2}{2\log n} \geq n - 2\log n$$
$$\iff \frac{\frac{n^2}{2} - n\log n + 2\log^2 n}{\log n} \geq 0$$

And for sufficiently large n, the numerator on the left hand side is greater than 1, and $\frac{1}{\log n} \geq 0$ for $n > 1$.

And so, for sufficiently large n,

$$|\mathsf{BAL}_{2\log \mathsf{n}}| \geq |\mathsf{w}|$$

Claim 5. $uw^k \in \mathsf{BAL}_{\mathsf{n}}$.

Proof. $ham(u) = \log n$ since $2\log n$ is even, and thus

$$ham(uw^k) = ham(u) + ham(w^k) = \frac{2\log n}{2} + \frac{|w^k|}{2} = \frac{2\log n + |w^k|}{2} = \frac{n}{2}$$

□

Then, upon inspection of G, we see that since u has even Hamming weight, G will compute k, the position of u in $O_{2\log n}$. This is the same k as in w^k, by definition of u. Then, G will return $w^{(k)(k)} = w$. □

□

Let G be the function based on Knuth's balanced code [11] described in Section 2.3. Recall that G is computable in polynomial time and surjective (see Appendix **??**). Using this function G, we will now construct the following function, assuming one-way permutations exist. Let $f : \{0,1\}^{n-2\log n} \mapsto \{0,1\}^{n-2\log n}$ be a one-way permutation. Let $f' : \{0,1\}^n \mapsto \{0,1\}^{n-2\log n}$ be the function $(f \circ G)^{(\frac{n}{2})}$.

Claim 6. *f' is monotone and surjective.*

Proof. Given any $x \in \{0,1\}^{n-2\log n}$, there is a $y \in \{0,1\}^{n-2\log n}$ such that $f(y) = x$ (since f is surjective). Then, since G is surjective, there is a $z \in \mathsf{BAL}_{\mathsf{n}}$ such that $G(z) = y$. Thus, $f(G(z)) = x$. Then, since $z \in \mathsf{BAL}_{\mathsf{n}}$, $\mathrm{ham}(z) = \frac{n}{2}$. And so $f'(z) = f(G(z)) = x$. Furthermore, f' is known as the $\frac{n}{2}$-cut of $f \circ G$, and we have already proven that any k-cut of a polynomial-time algorithm has a polynomial-size monotone circuit that computes it. □

It suffices to show that f' is weak one-way, as by a simple extension we can construct a strong one-way function from f'. Again, following [8], we can show that f' is weak.

Lemma 4. *f' is a weak one-way function.*

Assume that adversary D' inverts $f'(U_n)$ with probability at least $1 - A(n) + \epsilon(n)$, where $A(n) = Pr[\text{ham}(U_n) = \frac{n}{2}]$. Now, construct adversary D that, on input $y \in f'(U_n)$, returns $G(D'(y))$.

Claim 7. *For even weight strings in $\{0,1\}^n$, if D' succeeds in inverting f', D will succeed in inverting f.*

Proof. Take $x \in \{0,1\}^n$ of even Hamming weight. Let $y = f'(x)$. Making a simplifying assumption that 0^m and 1^m are not in the range of f, this implies that y is not 0^m or 1^m, as $f' = f \circ G$ in this case.

Now, assume D' outputs $x' \in f'^{-1}(y)$. Since y is neither 0^m nor 1^m, x' has even Hamming weight. Then, $D(y) = G(x')$, and furthermore, $f(G(x')) = y$ because $f(G(x')) = f(x')$ since x' has even Hamming weight. Thus, $D'(y) \in f^{-1}(y)$. □

Claim 8. *D inverts $f(U_n)$ with probability at least $\epsilon(n)$*

Proof.

$$\begin{aligned}
\Pr[D(f(U_n)) \in f^{-1}(f(U_n))] &\geq \Pr[D(f(U_n)) \in f^{-1}(f(U_n)) \wedge \text{ham}(U_n) = n/2] \\
&\geq \Pr[D'(f'(U_n)) \in f'^{-1}(f'(U_n)) \wedge \text{ham}(U_n) = n/2] \\
&\geq \Pr[D'(f'(U_n)) \in f'^{-1}(f'(U_n))] - \Pr[\text{ham}(U_n) \neq n/2] \\
&\geq \epsilon(n).
\end{aligned}$$

□

□

Corollary 6. *If a one-way surjection $f_S : \{0,1\}^{n-2\log n} \to \{0,1\}^m$ exists, then a one-way monotone surjection $g_S : \{0,1\}^n \to \{0,1\}^m$ exists.*

Proof. By replacing the one-way permutation used in the preceding proof with f_S, and using the fact that the injectivity of the permutation is not used in the proof, this corollary follows. □

Corollary 7. *If strong exponentially hard one-way permutations exist, weak one-way monotone surjections exist from n bits to $n - O(\log\log(n))$ bits.*

Proof. Suppose some f_n is a sequence of strong exponentially hard one-way permutations. Let $d \in \omega(\log n)$. Now, via a compexity leveraging argument, we see that there exists a strong one-way permutation g_n on d-bits that is strong one-way with respect to polynomial-time adversaries. By the previous theorem, this means that there exists a monotone weak one-way surjection from d bits to $d - \log d$ bits. Append $n - d$ input and output bits (each input mapping directly to the corresponding output bit, so as to maintain surjectivity) to get a monotone weak one-way surjection from n bits to $n - \log d$ which will be $n - O(\log\log(n))$ as desired by setting d appropriately. □

6 Negation Complexity of Some One-Way Injections

One of our main technical contributions is presented in this section. We show that an arbitrary injective one-way function f from n bits to m bits can be inverted in deterministic time $O(poly(n,m)\binom{m}{m-n}2^{m-n})$.

We say a function f is *perfectly invertible* in P if there is a deterministic polynomial-time adversary that, with only oracle access to any f and given any $x \in \text{codomain}(f)$, outputs the preimage $f^{-1}(x)$ or indicates that x is not in the range of f. Note that this is much stronger than what an adversary normally needs to invert a function in the cryptographic context.

Theorem 10. *There exists an inverter I that, for any $k > 0$, perfectly inverts in P the monotone injections $\{0,1\}^n \mapsto \{0,1\}^m$ in deterministic time $O(poly(n,m)\binom{m}{m-n}2^{m-n})$.*

In order to prove the theorem, we will require the following proposition that follows immediately from an injectivity argument:

Proposition 3. *Given any monotone injection, and any input bit of that function, restricting an input bit to zero (or one) will necessarily restrict at least one output bit to zero (or one). In other words, for any monotone injection $f : \{0,1\}^n \mapsto \{0,1\}^m$, where $f(b_1 b_2 \cdots b_n) = b'_1 b'_2 \cdots b'_m$, then for all $1 \leq t \leq n$, there exists a t_1 such that $b'_{t_1} = 1$ whenever b_t is 1, and a t_0 such that $b'_{t_0} = 0$ whenever b_t is 0.*

Proof. The inverter I consists of two main parts, the reduce method, and the solve method. The adversary begins at the reduce method, and calls the reduce method recursively, until it reaches the base case, at which point the solve method is run.

The reduce method operates on strings in $\{0,1,\star\}^*$, which should be interpreted as follows: the $\star$ bits are considered "free" and may vary throughout the scope of the variables, whereas the 0 and 1 bits are fixed, and will not change. For convenience, we define a couple functions on $\{0,1,\star\}^*$: let $z(y)$ denote the string obtained from y where all positions with a $\star$ in y are replaced with zero, and similarly let $w(y)$ denote the string where all $\star$ bits in y are replaced with ones.

The inverter I operates as follows:

Let x be the input, which has m bits. I calls reduce$(x, \star^n, \star^m)$. beginning a recursive procedure. The procedure reduce$(x, y, \star^m)$ takes three inputs: the target $x \in \{0,1\}^m$ (which is unused, except insofar as it is passed onto the final solve method), the input bit configuration $y \in \{0,1,\star\}^n$ and the output bit configuration, $p \in \{0,1,\star\}^m$. The fixed bits in y should be interpreted as bits which are assumed to be consistent with the preimage of x, and the fixed bits in p should be interpreted as bits that are constant over all possible settings of the free input bits. We will maintain as a recursive invariant in calls to reduce that the difference between the number of free bits in p and the free bits in y is at most $m - n$.

The reduce method is as follows:

- Define f' as the restriction of f such that the domain is the subset of $\{0,1\}^n$ that only disagree with y on its free bits, and the range is the subset of $\{0,1\}^m$ that only disagree with p on its free bits. Note that f' is a monotone injection. For each free bit in y, indexed by t, let $a_t \in \{0,1\}^n$ be the string that has only position t set to one, $0^{t-1}10^{n-t}$, and let b_t be the complement of a_t. Let $r_t = (a_t \vee z(y))$ and $s_t = (b_t \wedge w(y))$ where $\vee$ and $\wedge$ are computed bit-wise. Now, consider $\text{ham}(f'(r_t))$ and $l - \text{ham}(f'(s_t))$. By Proposition 3, both quantities are ≥ 1. If both are $= 1$ for all t, then as a base case, return the result of solve(x, y, p). Otherwise, in the case that one of the quantities is ≥ 2 for some t, branch into two recursive calls: reduce$(x, a_t \vee y, r_t \vee p)$ and reduce$(x, b_t \wedge y, s_t \wedge p)$, where the bitwise $\vee$ and $\wedge$ operations are extended in the natural way, so that $0 \vee \star = \star$, $1 \vee \star = 1$, $\star \vee \star = \star$,

$0 \wedge \star = 0$, $1 \wedge \star = \star$, and $\star \wedge \star = \star$. If either call returns a preimage for x, return that preimage; otherwise, if both branches indicate no solution, return that there is no preimage.

Observe that when the solve method is called from the reduce method, it holds that all of the free bits in the input y, when restricted to 0, set exactly one free bit in the output p to 0, and when restricted to 1, set exactly one free bit in the output p to 1 (since, otherwise we recurse further). Let π, τ be partial functions from free bits in y to the free bits in p such that on input a position in the input indicate which output bit is set to zero and one, respectively. Now, by injectivity, we know that π and τ must themselves be injective, because otherwise, setting two different free input bits would result in the same output.

The solve method is as follows:

- Suppose there are k free input bits. First, discover the π and τ permutations by simply testing each free input bit, and seeing which free output bit corresponds to that input. Consider the images of π and τ, P and T. We know the number of free output bits is limited, so $|P \cup T| \leq k + (m - n)$, and $k = |P| = |T|$. So, $m + k - n \geq |P \cup T| = |P| + |T| - |P \cap T| = 2k - |P \cap T|$, meaning that $k - |P \cap T| \leq (m - n)$. For each bit $b \in P \cap T$, examine the corresponding bit in x; if x has a zero at b, it must be the case that a preimage of x has $T^{-1}(b)$ set to zero, and if x has a one at b, a preimage of x has $P^{-1}(b)$ set to one. Thus, $|P \cap T|$ of the free bits can be deduced, leaving at most $m - n$ remaining free bits. Perform a brute-force search over all possible configurations of these bits, returning a preimage of x if one is found, or indicating no such preimage exists.

It is clear that the solve method runs in $O(m2^{m-n})$ steps. Next, we count the maximum number of times reduce is called. Let $T(i, j)$ be the number of times reduce is called, where i is the number of free input bits and j is the number of free output bits minus the number of free input bits. Note that in the worst case, $T(i, j) = 1 + T(i - 1, j) + T(i - 1, j - 1)$, since in at least one of the paths, more than one outbit bit is set by restricting an input bit. The recursion will necessarily stop as soon as either $i = 0$ or $j = 0$. Equivalently this quantity is bounded by the number of non-backtracking paths between corners on a $n \times (m - n)$ rectangular grid, which is $\binom{m}{m-n}$. Each call takes only $O(nm)$ time, and so the total running time for the whole method is $O\left(nm^2\binom{m}{m-n}2^{m-n}\right)$, which for constant $m - n$ is polynomial in n. □

Corollary 8. *Any monotone injection from $\{0,1\}^n \mapsto \{0,1\}^{n+O(\log n)}$ can be inverted in time $n^{O(\log n)}$*

Recall that Corollary 5 shows that that assuming existence exponential-time hard one-way permutations we can get a monotone injection with $\omega(\log n)$ stretch secure against polynomial-time adversaries. While still a small gap remains, this gives an indication that we cannot significantly improve our results.

Corollary 9. *There is an inverter that, for any constants k, g, perfectly inverts in P any monotone injection $f : \{0,1\}^n \mapsto \{0,1\}^{n+k}$ with g negation gates at the bottom (meaning the negation gates are over only input bits).*

Proof. Suppose such an f existed. Then, consider the following algorithm with oracle access to f: Iterate through every one of the 2^g possible combinations of negated bits. On each combination, define a restriction $f' : \{0,1\}^{n-g} \mapsto \{0,1\}^{n+k}$, which is monotone and injective. Provide this as an oracle to the adversary from the previous theorem and pass along your own challenge input. If you get a result for any chosen combination, return it; otherwise, indicate no solution. □

Corollary 10. *OWIs (one-way injections) with constant stretch and negation gates only at the bottom must have a super-constant number of negation gates.*

Proof. This is the contrapositive of the previous corollary. □

Corollary 11. *OWIs with constant stretch must either have a super-constant number of negation gates, or a negation gate above a subcircuit of superconstant size.*

Proof. If there were only a constant number of negation gates above a circuits of constant size, then by De Morgan's laws, an adversary could convert this into a circuit with a constant number of negation gates at the bottom, which would violate the previous corollary. □

Corollary 12. *OWIs in NC_0 with constant width require a super-constant number of negation gates.*

7 Negation Complexity of List-Decodable Codes

In the work of Guo et al. [9], they show that error-correcting codes are highly non-monotone. Here we extend their result to list-decodable codes.

Theorem 11. *Given any sequence of polynomial-sized circuits f_n that make up a list-encoding function from n bits to $m(n)$ bits that is resistant to $\epsilon m(n)$ bit-flips, the number of negation gates in f_n must be at least* $\log(n) - O(1)$.

In physical reality, the probability of a bit flip over a noisy communication channel is unlikely to decrease as the message size increases; but rather, one would expect it to stay constant or increase. Any reasonable formulation of error-tolerant coding should allow this sort of tolerance, so the fact that almost $\log(n)$ negation gates are required for the encoding part of such a list coding indicates that list encoding is a highly non-monotone problem.

Proof. Let $\epsilon > 0$, and let $f_n : \{0,1\}^n \mapsto \{0,1\}^m$ be some sequence of circuits that make a list-encoding resistant to $\epsilon m(n)$ errors. Now, suppose for contradiction that the number of negation gates in f_n is $\log(n) - b(n)$, where b is unbounded. Since the sequence f_n make up a list-encoding, it follows by definition that for any code s, a decoder must be able to output a polynomial-sized list of outputs that contain all strings within $\epsilon m(n)$ bit flips of $f(s)$. In particular, we have that for some polynomial p, for all n, for all s, $U_n(s) = \{x | \text{ham}(f(x) \oplus f(s)) < \epsilon m(n)\}$ has cardinality $\leq p(n)$. Let d be the degree of p.

Now, fix N large enough so that $n > N$ implies $C(n) = \binom{n/2}{d+1} > p(n)$, $m(n) > \frac{d+1}{\epsilon}$, $l(n) = \left\lfloor \frac{n}{2(d+1)} \right\rfloor > \frac{1}{\epsilon}$, and $n \geq 8(d+1)$. Since b is unbounded, we can find $n > N$ such that $b(n) \geq 2 + \log(d+1) + \log\left(2 + \frac{1}{\epsilon}\right)$. Fix n as this value, and consider the following chain of elements $a_i \in \{0,1\}^n$ for $0 \leq i \leq l(n)$, defined recursively.

The first element of the chain is $a^0 = 0^n$. As inductive invariants, we claim that $\text{ham}(a^i) = i(d+1) < \frac{n}{2}$ and that if $f(a^i) \succ f(a^{i-1})$ then $\text{ham}(f(a^i)) \geq \text{ham}(f(a^{i-1})) + \epsilon m(n)$. In the inductive step, for $0 < i < l(n)$, pick $n/2$ bits that are set to zero in a^{i-1}, $Z^i = \{z^1, z^2, \cdots, z^{n/2}\}$. Let T^i consist of the $C(n)$ subsets of Z^i that have exactly $d+1$ elements, and let $S^i = \left\{a^i \vee \left(\bigvee_{z \in W} e_z\right) | W \in T^i\right\}$, where $e_z \in \{0,1\}^n$ has bit z one and all other bits zero, and $\vee$ refers to a bitwise or. Note S_i consists of $C(n)$ strings above a^{i-1}, each with Hamming weight $\text{ham}(a^{i-1}) + d + 1 = i(d+1)$. Due to the size of S^i, it must contain some element, which we assign to a_i, that not contained in $U_n(a^{i-1})$. If $a^i \succ a^{i-1}$, then since $a^i \notin U_n(a^{i-1})$, $\text{ham}(f(a^i) \oplus f(a^{i-1})) \geq \epsilon m(n)$, it must be the case that $\text{ham}(f(a^i)) \geq \text{ham}(f(a^{i-1})) + \epsilon m(n)$, and so both inductive invariants hold.

Thus, we have constructed the chain $a_0 \prec a_1 \prec \cdots \prec a^{l(n)}$, such that whenever $f(a^{i-1}) \prec f(a^i)$, $\text{ham}(f(a^i)) \geq \text{ham}(f(a^{i-1})) + \epsilon m(n)$. Furthermore, we know due to Markov that since f_n has $\log(n) - b(n)$ negations, there can be at most $2^{\log(n)-b(n)} = \frac{n}{4(d+1)(2+1/\epsilon)}$ times in the chain where $f(a^{i-1}) \not\prec f(a^i)$. By the pigeonhole principle, there must be some subchain $b^1, \cdots, b^k$ with $f(b^1) \prec \cdots \prec f(b^k)$ where $k \geq \frac{l(n)-1}{n/(4(d+1)(2+1/\epsilon))} \geq \frac{n/(2(d+1))-2}{n/(4(d+1))} \left(2 + \frac{1}{\epsilon}\right) \geq \left(2 - \frac{8(d+1)}{n}\right)\left(2 + \frac{1}{\epsilon}\right) \geq 2 + \frac{1}{\epsilon}$. Since $b^1, \cdots, b^k$ is a subchain of $a^1, \cdots, a^{l(n)}$, we have that for each $i > 1$, $\text{ham}(f(b^i)) \geq \text{ham}(f(b^{i-1})) + \epsilon m(n)$, so that $\text{ham}(f(b^k)) \geq \text{ham}(f(b^0)) + (k-1)\epsilon m(n) \geq \epsilon m(n) + m(n)$, which is clearly impossible as the maximum and minimum Hamming weights are $m(n)$ and 0. By contradiction, the claim holds. □

Next, we establish that while the encoder of a list-decodable code cannot be monotone, we prove that the decoder can be made monotone.

Theorem 12. *Suppose there exists a sequence of (possibly probabilistic) circuits C_k of size $S(k)$ on input $\{0,1\}^k$ that outputs a list of $L(k)$ elements, each of which is a string $\{0,1\}^{n(k)}$, for some $n, L, S : \mathbb{N} \mapsto \mathbb{N}$. Then, there exists a polynomial p and a monotone (probabilistic if the original circuit was) circuit M_k of size $S(k)p(k)$ that outputs a list of $\Theta(k)L(k)$ elements, each of which is a string $\{0,1\}^{n(k)}$, such that for all strings y, $M_{|y|}(y)$ contains $C_{|y|}(y)$ as a sublist.*

Proof. We know from an existing result that for some polynomial q, given the sequence of circuits C_k of size $S(k)$, we can construct an equivalent sequence of circuits C'_k of size $S(k)q(k)$, where C'_k has $N(k) = \lceil \log_2(k+1) \rceil$ negation gates. Separate out the negation gates as follows: let

$$a_k(x, \neg b_{k,1}(x), \neg b_{k,2}(x), \cdots, \neg b_{k,N(k)}(x)) = C'_k(x)$$

where a_k is the monotone circuit that takes everything above the negation gate (taking the outputs of the negation gates as additional inputs), and $b_{k,i}$ is the circuit below the i-th negation gate of C'_k (via some arbitrary ordering). Then, construct the desired circuit $M_k(x)$ as the concatenation of $a_k(x, e_1, e_2, \cdots, e_{N(k)})$ for every $e_1, e_2, \cdots, e_{N(k)} \in \{0,1\}$. This only multiplies the size, $S(k)q(k)$ since a_k is a subcircuit of C'_k, by a factor of $2^{N(k)} = \Theta(k)$, meaning the size of M_k is $S(k)p(k)$ for some polynomial p. Also, M_k outputs a list of $\Theta(k)L(k)$ elements (in fact $\Theta(k)$ lists of size $L(k)$). Now, for any input string y, $M_{|y|}(y)$ has a sublist $a_{|y|}(y, \neg b_{|y|,1}(y), \neg b_{|y|,2}(y), \cdots, \neg b_{|y|,N(|y|)}(y))$ which has identical output to $C'_{|y|}(y)$ and thus $C_{|y|}(y)$, proving the claim.

Corollary 13. *If there exists a polynomial-size list decoder for a given sequence of codes $C_k : \Sigma^k \mapsto \Sigma^{n(k)}$ for some $n : \mathbb{N} \mapsto \mathbb{N}$ that outputs a list of size $L(k)$ for some $\mathbb{N} \mapsto \mathbb{N}$, there exists a monotone polynomial-size list decoder for C that outputs a list of size kL_k.*

Corollary 14. *If there exists a probabilistic polynomial-size local list decoder for a given code $C : \Sigma^k \mapsto \Sigma^n$ that outputs a list of size L_k, there exists a monotone probabilistic polynomial-size local list decoder for C that outputs a list of size kL_k.*

□

Acknowledgements

The authors were supported by Google Faculty Research Grant and NSF Awards CNS-1526377 and CNS-1618884. The views expressed are those of the authors and do not reflect the official policy or position of Google, the Department of Defense, the National Science Foundation, or the U.S. Government.

References

[1] Kazuyuki Amano and Akira Maruoka. A superpolynomial lower bound for a circuit computing the clique function with at most (1/6)log log n negation gates. *SIAM J. Comput.*, 35(1):201–216, 2005.

[2] Robert Beals, Tetsuro Nishino, and Keisuke Tanaka. More on the complexity of negation-limited circuits. In *Proceedings of the Twenty-Seventh Annual ACM Symposium on Theory of Computing, 29 May-1 June 1995, Las Vegas, Nevada, USA*, pages 585–595, 1995.

[3] Robert Beals, Tetsuro Nishino, and Keisuke Tanaka. On the complexity of negation-limited boolean networks. *SIAM J. Comput.*, 27(5):1334–1347, 1998.

[4] Eric Blais, Clément L. Canonne, Igor Carboni Oliveira, Rocco A. Servedio, and Li-Yang Tan. Learning circuits with few negations. *CoRR*, abs/1410.8420, 2014.

[5] Avrim Blum, Carl Burch, and John Langford. On learning monotone boolean functions. In *39th Annual Symposium on Foundations of Computer Science, FOCS '98, November 8-11, 1998, Palo Alto, California, USA*, pages 408–415, 1998.

[6] Joshua Buresh-Oppenheim, Valentine Kabanets, and Rahul Santhanam. Uniform hardness amplification in NP via monotone codes. *Electronic Colloquium on Computational Complexity (ECCC)*, 13(154), 2006.

[7] Michael J. Fischer. Hauptvortrag: The complexity of negation-limited networks - A brief survey. In *Automata Theory and Formal Languages, 2nd GI Conference, Kaiserslautern, May 20-23, 1975*, pages 71–82, 1975.

[8] Oded Goldreich and Rani Izsak. Monotone circuits: One-way functions versus pseudorandom generators. *Theory of Computing*, 8(1):231–238, 2012.

[9] Siyao Guo, Tal Malkin, Igor Carboni Oliveira, and Alon Rosen. The power of negations in cryptography. In *Theory of Cryptography - 12th Theory of Cryptography Conference, TCC 2015, Warsaw, Poland, March 23-25, 2015, Proceedings, Part I*, pages 36–65, 2015.

[10] Mauricio Karchmer and Avi Wigderson. Monotone circuits for connectivity require super-logarithmic depth. In *Proceedings of the 20th Annual ACM Symposium on Theory of Computing, May 2-4, 1988, Chicago, Illinois, USA*, pages 539–550, 1988.

[11] Donald E. Knuth. Efficient balanced codes. *IEEE Transactions on Information Theory*, 32(1):51–53, 1986.

[12] A. A. Markov. On the inversion complexity of a system of functions. *J. ACM*, 5(4):331–334, 1958.

[13] Hiroki Morizumi. Limiting negations in non-deterministic circuits. *Theor. Comput. Sci.*, 410(38-40):3988–3994, 2009.

[14] Moni Naor and Moti Yung. Universal one-way hash functions and their cryptographic applications. In *Proceedings of the 21st Annual ACM Symposium on Theory of Computing, May 14-17, 1989, Seattle, Washigton, USA*, pages 33–43, 1989.

[15] Miklos Santha and Christopher B. Wilson. Limiting negations in constant depth circuits. *SIAM J. Comput.*, 22(2):294–302, 1993.

[16] Shao Chin Sung and Keisuke Tanaka. Limiting negations in bounded-depth circuits: An extension of markov's theorem. *Inf. Process. Lett.*, 90(1):15–20, 2004.

[17] Éva Tardos. The gap between monotone and non-monotone circuit complexity is exponential. *Combinatorica*, 8(1):141–142, 1988.

A k-cut of a Function is Monotone

Proposition 4. *If $f : \{0,1\}^n \mapsto \{0,1\}^m$ is a function computable in polynomial-time, then for any k, the $k-$cut of f, $f^{(k)}$, is computable by a polynomial-sized monotone circuit.*

B Missing Proofs from Section 4

Corollary 15. *If strong exponentially hard one-way permutations exist, strong one-way monotone injections exist from n bits to $n + \omega(\log(n))$ bits.*

C Missing Proofs from Section 5

Corollary 16. *If strong exponentially hard one-way permutations exist, weak one-way monotone surjections exist from n bits to $n - \log(\log(n)) - \omega(1)$ bits.*

Proof. Suppose some f_n is a sequence of strong exponentially hard one-way permutations. Let $d(n) \in \omega(1)$. Now, recalling the lemma from the previous section about exponentially hard functions, we see that there exists a strong one-way sequence of permutations g_n on $\log(n)2^{d(n)}$ bits that is strong one-way with respect to $\text{poly}(n)$. By the previous theorem, this means there exists a weak one-way monotone surjection from $\log(n)2^{d(n)}$ bits to $\log(n)2^{d(n)} - \log(\log(n)2^{d(n)})$ bits. Append $n - \log(n)2^{d(n)}$ input and output bits (each input mapping directly to the corresponding output bit, so as to maintain surjectivity) to get a weak one-way monotone surjection from n bits to $n - \log(\log(n)2^{d(n)}) = n - \log(\log(n)) - d(n)$, as desired. □

D Negation Complexity of List-Decodable Codes

Theorem 13. *Given any sequence of polynomial-sized circuits f_n that make up a list-encoding function from n bits to $m(n)$ bits that is resistant to $\epsilon m(n)$ bit-flips, the number of negation gates in f_n must be at least $\log(n) - O(1)$.*

D.1 Some Corollaries to our Injection Lower Bound

Corollary 17. *For any constant c, for any OWI with constant stretch, for all inputs of Hamming weight less than c [assuming n is sufficiently large], at least one negation gate must not trigger, and for all inputs of Hamming weight more than $m - c$, at least one negation gate must trigger.*

Proof. Suppose that for some constant c, all negation gates trigger for some input of Hamming weight c (for large enough n). Then, consider the following algorithm on target x: check all

strings with Hamming weight $\leq c$ via a brute-force search. Otherwise, for each string w with Hamming weight c, guess that w is below the true preimage, and that w has all negations triggered. With non-negligible probability in x, this will happen. Let f_w be the restriction of f to the zero bits of w (the other bits fixed at 1) which has constant width $m - n + c$. Feed this into the inverter in the main theorem to find the inverse of x. The case from the other end follows identically. □

Corollary 18. *Any monotone function f that has a constant number of collisions (in other words, $|\{(x, y)|x \neq y, f(x) = f(y)\}|$ is constant) is perfectly invertible in P.*

Proof. Say there are c collisions. Through the adversary in the main theorem, we can find all such collisions. For each collision (x, y), since $x \neq y$, there is a bit that is 1 for one member of the pair and 0 for the other. First, iterate through all possible 2^c configurations of these bits, then run the adversary of the main theorem on the function restricted to the remaining bits, which still has a constant width of $m - n + c$. □