On the Effectiveness of Pre-Processing for Cracking Salted Passwords

DRAFT: Not for Distribution[*]

Christine Evangelista[†] Jonathan Katz[‡] Aaron Lowe[§] Jordan Schneider[¶]
Lynesia Taylor[||] Aishwarya Thiruvengadam[‡] Ellen Vitercik[**]

Abstract

De, Trevisan, and Tulsiani [?] show that for every permutation $f : [N] \to [N]$, any algorithm using time T and space S that inverts f on an ϵ fraction of inputs must satisfy the lower bound $TS = \widetilde{\Omega}(\epsilon N)$. We show a $TS = \widetilde{\Omega}(\epsilon N/k)$ trade-off lower bound for the complexity of an inverter for k-to-1 functions. This implies that salt is an effective countermeasure.

We apply our results to the context of a security system where passwords are stored only after they are salted and then hashed using a k-to-1 function. We prove that, even if an adversary knows the salt corresponding to each hashed password, he cannot invert values using time or space less than $ST = \widetilde{\Omega}(\epsilon N/k)$. Finally, we show that this bound is tight when $k = 1$ or $S > \epsilon N/k$. This proves that when either condition holds, salt provides the same security as increased password length.

1 Introduction

Security systems that rely on passwords usually employ a password hash technique, wherein the image of a password under a one-way function is stored, rather than the password in plain-text. This adds security in the event of a data-breach. In order to recover the original passwords, an attacker must find the inverse of a hashed password. As a baseline, during the online phase, an attacker can successively hash possible passwords until he finds a hash that matches the value he wishes to invert. This takes $O(N)$ hash function calls and negligible space. Alternately, an attacker can precompute a table of every possible password and lookup the value he wishes to invert during the online phase, using $O(N)$ space and $\widetilde{O}(1)$ time.

1.1 Previous Works

If the attacker wants to use space and time less than $O(N)$, he can use a *time-memory tradeoff algorithm.* As an example, Hellman [?] presented an algorithm that uses a set of precomputed tables and, with constant probability, returns the inverse of a value hashed under a random function f. The algorithm uses time T and space S such that $TS^2 = N^2$. This is called a time-memory tradeoff because the attacker can trade time for memory or vice versa so long as $TS^2 = N^2$.

[*]Work supported by NSF-CNS-1262805.
[†]Elizabethtown College. **Email:** `evangelistac@etown.edu`.
[‡]Dept. of Computer Science, University of Maryland. **Email:** `{jkatz, aish}@cs.umd.edu`
[§]Haverford University. **Email:** `alowe@haverford.edu`
[¶]Montgomery Blair High School. **Email:** `jor11235@gmail.com`
[||]Spelman College. **Email:** `ltaylo25@scmail.spelman.edu`
[**]Columbia University. **Email:** `emv2126@columbia.edu`

Fiat and Naor [?] extend Hellman's tradeoff to the case where f is not necessarily random, but arbitrary, with a λ probability of collision. They show a $TS^2 = \lambda N^3$ tradeoff holds. Note that for Hellman's random functions, λ is $1/N$, giving Hellman's tradeoff.

While Hellman, Fiat, and Naor provide upper bounds, De et al. [?] provide a lower bound of $ST = \widetilde{\Omega}(\epsilon N)$ for all algorithms that invert an ϵ-fraction of values when f is a permutation.

1.2 Our Contributions

In this paper, we extend De et al.'s methods [?] of deriving a lower bound from the case of inverting permutations to the case of inverting k-to-1 functions and prove that $TS = \widetilde{\Omega}(\epsilon N/k)$. Note that, when $k = 1$ (i.e. f is a permutation), our bound is consistent with De's. We then consider an attacker that knows the salt values of these hashes, and prove the same bound, suggesting that salting k-to-1 hash functions increases security. De et. al. [?] prove that, for $S \geq \frac{\epsilon^2 N}{k}$, their scheme can invert k-to-1 functions with $ST = O(\frac{\epsilon N}{k})$. This shows that, for this range of S, our lower bounds for salted and unsalted k-to-1 functions are tight. Next, we prove that De's lower bound on permutations is tight by presenting an algorithm that uses time and space such that $TS = \widetilde{O}(\epsilon N)$. This algorithm is an extension of one described by De et al. [?] and Fiat and Naor [?]. Finally, we discuss the security of salt in these various contexts.

1.3 Notation

- If X is a integer, [X] is the set of integers from 1 to X.
- We say a function is k-to-1 if every image has a preimage set of cardinality k. Alternatively, exactly k preimages map to a single image.
- If we say an oracle query was made in a set X, we mean that $f(x)$ was called, where $x \in X$.
- The notation $a||b$ is a concatenated with b.
- Any asymptotic bound with a tilde over it (eg. $\widetilde{\Omega}$) indicates that log terms weakening the bound are hidden. Specifically, $\widetilde{O}(g) = O(g \log^c g)$ and $\widetilde{\Omega}(g) = \Omega(\frac{g}{\log^c g})$ for some constant $c \geq 0$.

2 Lower Bound

Let $f : [N] \to [N]$ be k-to-1, so if $y \in [N]$ has a preimage, then its preimage set has cardinality k and $|f([N])| = N/k$. Let A be an algorithm that makes at most T oracle queries such that for every k-to-1 function f, there exists an advice string adv of length at most S such that

$$P[A^f_{adv}(f(x)) \in f^{-1}(f(x))] \geq \epsilon.$$

In this section, we prove that $ST = \widetilde{\Omega}(\epsilon N/k)$. To derive this lower bound, we use a similar method as De et al. use to prove Theorem 8.2 and Corollary 8.2 in [?]. As an overview, De et al. make the argument that, if every permutation can be inverted on an $\epsilon-$fraction of inputs by an oracle algorithm that uses time T and space S, then for every permutation, there exists a randomized encoding scheme that succeeds with constant probability and compresses the permutation. The compression is sufficiently effective to allow the authors to probabilistically derive their lower bound of $ST = \widetilde{\Omega}(\epsilon N)$. We summarize their results below.

Theorem 2.1. *(Theorem 8.2 from De et. al. [?]) Fix an oracle algorithm A which makes at most T oracle queries and which takes an advice string of length S. Fix a parameter ϵ.*

There are randomized encoding and decoding procedures E, D which use shared randomness and such that if f is a permutation and adv is an advice string such that

$$P[A^f_{adv}(f(x)) = x] \geq \epsilon$$

then

$$\underset{r}{P}[D(r, E(r, f)) = f] \geq 0.9$$

and the length of $E(r, f)$ is at most

$$\log N! - \frac{\epsilon N}{100T} + S + O(\log N)$$

bits.

Corollary 2.2. *(Corollary 8.3 from De et. al. [?]) If A is an oracle algorithm that runs in time at most T and such that for every permutation f there is a data structure adv of size $\leq S$ such that*

$$\underset{r}{P}[A^f_{adv}(f(x)) = x] \geq \epsilon,$$

then

$$ST = \widetilde{\Omega}(\epsilon N).$$

As De et al. propose in [?], we say that an inverter succeeds on an $\epsilon-$fraction of inputs if

$$|\{x \in [N] |\ \text{Invert}(f(x)) \in f^{-1}(f(x))\}| = \epsilon N.$$

Since f is k-to-1, another way to think of this is that the inverter successfully finds at least one preimage for $\frac{\epsilon N}{k}$ range values.

Throughout the analysis, we require that $T = \tilde{o}(\epsilon N/k)$. After all, suppose the adversary were allowed the time budget $T = \widetilde{O}(\epsilon N/k)$ when he attempted to invert $f(x) = y$. He could brute force over an $\epsilon/k-$fraction of domain values. Each domain value has a k/N probability of being in the preimage set of y, so the probability that he would find an inverse for y using this brute force attack is ϵ. Therefore, we require that $T = \tilde{o}(\epsilon N/k)$ because otherwise the adversary could simply employ the brute force technique.

To fix notation, we call the endcoding procedure $E : \{0,1\}^n \times \{0,1\}^r \to \{0,1\}^m$ and the decoding procedure $D : \{0,1\}^m \times \{0,1\}^r \to \{0,1\}^n$.

We begin with a simple lemma.

Lemma 2.3. *The number of k-to-1 functions $f : [N] \to [N]$ is*

$$\binom{N}{N/k} \cdot \frac{N!}{k!^{N/k}}.$$

Proof. Consider splitting each of the $N!$ permutations into parts of size k. Each of the N/k parts are counted $k!$ times, so the number of partitions for which we do not care about the order within the partition, but do care about the order of the parts is $N!/k!^{N/k}$. Then for each of these ordered partitions there are $\binom{N}{N/k}$ ways to choose values for each part. □

We are now ready to present our stronger version of De et al.'s Theorem 8.2 in [?].

Theorem 2.4. *Fix an algorithm A which makes at most T oracle queries and which takes an advice string of length S. Fix a parameter $\epsilon \in [0, 1]$.*

If $f : [N] \to [N]$ is a k-to-1 function and adv is an advice string such that

$$P[A^f_{adv}(f(x)) \in f^{-1}(f(x))] \geq \epsilon,$$

then there exist encoding and decoding procedures E, D which use shared randomness such that

$$\underset{r}{P}[D(r, E(r, f)) = f] \geq 0.9$$

and the length of $E(r, f)$ is at most

$$S + O(\log N) + \log\left(\binom{N}{N/k} \cdot \frac{N!}{k!^{N/k}}\right) - \frac{3\epsilon N}{1000kT}.$$

Proof. Recall that there are N/k preimage sets. Using shared randomness r, we generate a random set $\{S_1, \ldots, S_\beta\}$ of the preimage sets such that each set is independently chosen with probability $1/(100T)$. Let

$$R = \cup_{i=1}^{\beta} S_i.$$

Call an element $y \in f(R)$ *good* if:

1. $A_{adv}^f(y) \in f^{-1}(y)$.
2. None of the oracle queries in the computation of $A_{adv}^f(y)$ are in R.

Let G be the set of good elements of $f(R)$. To bound the number of bits used in the encoding scheme, we first derive a lower bound on the cardinality of G.

Lemma 2.5. *With probability at least 0.9,* $|G| \geq 3\epsilon N/(1000kT)$.

Proof. For a fixed algorithm call $A_{adv}^f(y)$:

- For a fixed $x \in [N]$, the probability that $x \in R$ is $1/(100T)$.
- The probability that all T queries made are not in R is minimized when all queries are in different preimage sets.[1] Therefore the probability is at least

$$(1 - 1/(100T))^T \approx e^{-1/100} \geq 1 - 1/100$$

  for large T.
- The probability that there exists a query in R (except possibly $x \in f^{-1}(y)$) is at most

$$1 - (1 - 1/(100T))^T \approx 1 - e^{-1/100} \leq 1/100$$

  for large T.

Let $I = \{x \in [N] \mid A_{adv}^f(f(x)) \in f^{-1}(f(x))\}$. We know that $|I| = \epsilon N$. Since I is partitioned by preimage sets, $|f(I)| = \epsilon N/k$, where $f(I) = \{y \in f([N]) | A_{adv}^f(y) \in f^{-1}(y)\}$. Note that $G \subseteq f(I) \cap f(R)$.

The following two events are independent:

- $y \in f(R)$, which occurs with probability $1/(100T)$.
- During the computation of $A_{adv}^f(y)$, there exists a query in R, which occurs with probability no more than $1/100$.

Therefore, the average number of $y \in f(R)$ such that A inverts y but in doing so makes oracle queries in R is at most $|f(I)|/(10,000T)$. By Markov's inequality, with probability at least 0.95, this quantity is at most $|f(I)|/(500T)$.

Let Y be the sum of independent random variables $Y_1, Y_2, \ldots, Y_{N/k}$ (one for each range element) where $Y_i = 1$ if $y_i \in f(R)$ and A can invert y_i. According to Chernoff's bound, we have that

$$P[Y \geq |f(I)|/(200T)] \geq 1 - \left(\frac{e^{-\delta}}{(1-\delta)^{1-\delta}}\right)^{|f(I)|/(100T)}.$$

Solving for δ, we get that $\delta = 0.5$. Then to get that this probability is at least 0.95, we must have that $2000T \leq |f(I)| = \epsilon N/k$, which is valid since we assume that $T = \tilde{o}(\epsilon N/k)$.

We conclude that $f(R)$ contains at least $|f(I)|/(200T) - |f(I)|/(500T) = 3|f(I_D)|/(1,000T)$ good elements with probability at least 0.9.

□

[1]Since R is the union of preimage sets, if multiple queries $q_1, \ldots, q_n$ are in the same preimage set, they are either all in R or all not in R. Therefore the probability that they all are not in R is $1 - 1/(100T)$, compared to $(1 - 1/(100T))^n$ when they are all elements of different preimage sets.

We now describe the encoding scheme. In the encoding, we include:

1. The advice string adv, using S bits.
2. The size of the set G, using $O(\log N)$ bits. [2]
3. The partition of N into preimage sets, using
$$\log\left(\frac{N!}{(N/k)!k!^{N/k}}\right) \text{ bits.}$$
4. The set $f([N])$, using $\log\binom{N}{|f([N])|}$ bits.
5. The set $f(R)$, using $\log\binom{|f([N])|}{|f(R)|}$ bits.
6. The set G, using $\log\binom{|f(R)|}{|G|}$ bits.
7. A permutation mapping from the preimage sets that partition $[N]-R$ to $f([N]-R)$ using $\log|f([N]-R)|!$ bits.
8. A permutation mapping from the preimage sets that partition $R-f^{-1}(G)$ to $f(R)-G$ using $\log|f(R)-G|!$ bits.

Items (7) and (8) are stored as mappings of preimage sets to their corresponding images. We define an ordering of the preimage sets as follows. Let P_i be the preimage set that contains the lexicographically first element in $[N]\setminus\cup_{j=1}^{i-1}P_j$, i.e. P_1 is the preimage set that contains 1, P_2 is the preimage set that contains the lexicographically first element not in P_1, and so on. Specifically, (7) and (8) denote the permutations $g : [N/k-|f(R)|] \to [N/k-|f(R)|]$ and $h : [|f(R)|-|G|] \to [|f(R)|-|G|]$, respectively. For example, if $g(i) = j$, then we know that the ith partition of $[N]-R$ is mapped to the jth element of $f([N]-R)$. Decoding proceeds as follows:

1. **Initialize an empty table to store the values of f on $[N]$.**
2. **Fill in the values of f restricted to $[N]-R$:** We have encoded $f(R)$ and $f([N])$, so we can derive $f([N])-f(R)$. Using item (7), we can recover the mapping from preimage sets to $f([N])-f(R)$. Using our definition of the preimage set ordering and the partition of preimage sets from (3), we can fill all the rows of our table restricted to $[N]-R\to f([N])-f(R)$.
3. **Fill in the inverses for all $y\in G$:** We must simulate the inversion procedure $A^f_{adv}(y)$. To find an inverse for each $y\in G$, the algorithm only queries elements in $[N]-R$. Since we have filled in the values of f restricted to $[N]-R$ in our table, all oracle queries can be answered. Once we know a single preimage x for some $y\in G$, we can find the entire preimage set $f^{-1}(y)$ from the information in item (3).
4. **Fill in the values of f restricted to $R-f^{-1}(G)$:** We now know the set $f^{-1}(G)$, so we know the set $R-f^{-1}(G)$. We also know the sets G and $f(R)$, so we know the set $f(R)-G$. Therefore, we can determine the values of f restricted to $f : R-f^{-1}(G)\to f(R)-G$ from item (8) and item (3).

.

In total, the length of the encoding is

[2]Both E and D are given r and can derive $|R|$, so it is not needed in the encryption scheme.

$$
\begin{aligned}
&S + O(\log N) \\
&\quad + \log\left[\binom{N}{|f([N])|} \cdot \frac{N!}{(N/k)!k!^{N/k}} \cdot \binom{|f([N])|}{|f(R)|} \cdot |f([N]-R)|! \right.\\
&\quad \left. \cdot \binom{|f(R)|}{|G|} \cdot |f(R)-G|! \right] \\
&= S + O(\log N) \\
&\quad + \log\left[\binom{N}{N/k} \cdot \frac{N!}{(N/k)!k!^{N/k}} \cdot \frac{(N/k)!(N/k-|f(R)|)!}{(N/k-|f(R)|)!|f(R)|!} \right.\\
&\quad \left. \cdot \frac{|f(R)|!(|f(R)|-|G|)!}{(|f(R)|-|G|)!|G|!} \right] \\
&= S + O(\log N) + \log\left[\binom{N}{N/k} \cdot \frac{N!}{k!^{N/k}}\right] - \log(|G|!) \\
&\le S + O(\log N) + \log\left[\binom{N}{N/k} \cdot \frac{N!}{k!^{N/k}}\right] - \frac{3\epsilon N}{1000kT}. \qquad (\log(|G|!) \ge |G| \ge 3\epsilon N/1000kT)
\end{aligned}
$$

□

Corollary 2.6. *Let A be an inversion algorithm that makes at most T oracle queries when inverting a single range element. If for every k-to-1 function f, there exists an advice string adv of length S such that*

$$\underset{x}{P}[A^f_{adv}(f(x)) \in f^{-1}(f(x))] \ge \epsilon,$$

then $ST = \widetilde{\Omega}(\epsilon N/k)$.

Proof. We will use Fact 10.1 from De et al.'s paper to derive this lower bound.

Fact 2.7. *(Fact 10.1 from De et al.) Suppose there is a randomized encoding procedure $Enc : \{0,1\}^n \times \{0,1\}^r \to \{0,1\}^m$ and a decoding procedure $Dec : \{0,1\}^m \times \{0,1\}^r \to \{0,1\}^n$ such that*

$$\underset{r\in U_r}{P}[Dec(Enc(x,r),r) = x] \ge \delta.$$

Then $m \ge n - \log 1/\delta$.

In the context of our proof, n is the number of bits needed to uniquely describe a k-to-1 function from $[N]$ to $[N]$ without using an encryption scheme, so

$$n = \log\left(\binom{N}{N/k} \cdot \frac{N!}{k!^{N/k}}\right).$$

Also, $\delta = 0.9$. Therefore,

$$
\begin{aligned}
m &= S + O(\log N) + \log\left(\binom{N}{N/k} \cdot \frac{N!}{k!^{N/k}}\right) - \frac{3\epsilon N}{1000kT} \\
&\geq \log\left(\binom{N}{N/k} \cdot \frac{N!}{k!^{N/k}}\right) - \log(10/9) \\
&\Rightarrow S + O(\log N) + \log(10/9) \geq \frac{3\epsilon N}{1000kT} \\
&\Rightarrow (S + 0.2)T \geq \frac{3\epsilon N}{1000k} - T \cdot O(\log N) \\
&\Rightarrow (S + 0.2)T \geq \frac{3\epsilon N}{1000k \cdot O(\log N)} - T && (O(\log N) \geq 1) \\
&\Rightarrow 2ST \geq (S + 1.2)T \geq \frac{3\epsilon N}{1000k \cdot O(\log N)} && (S > 1.2) \\
&\Rightarrow ST = \widetilde{\Omega}(\epsilon N/k).
\end{aligned}
$$

□

3 An Application to Salt

As before, let $f : [N] \to [N]$ be k-to-1. We now consider a situation where an adversary is attempting to invert encrypted passwords and the defender has prefixed each password with a string of cryptographic salt before evaluating the concatenation with f. We assume that for each user α with password pw_α and salt s_α, the adversary knows the pair $(s_\alpha, f(s_\alpha||pw_\alpha))$. The adversary is attempting to recover $(s_\alpha||pw_\alpha)$. In this context, we denote the salt dictionary as $[N_s]$ and the password dictionary as $[N_p]$, so $N = N_s N_p$.

We assume the inverter can recover one password in time $T = o(\epsilon N_p)$ with an ϵ probability of success. After all, if $T = \widetilde{O}(\epsilon N_p)$, since the adversary knows the salt s corresponding to a given hash value, he can simply brute force over an ϵ fraction of the $(s||p)$ concatenations as p ranges over $[N_p]$, with an ϵ probability of finding the correct password. In our analysis, we also assume that $N/k > N_p$, or alternatively, $N_s > k$, which is a valid assumption for reasonably large N_s.

The inversion algorithm now takes as input $(s, f(s||p))$ and attempts to recover $(s||p)$. We will prove the lower bound $ST = \widetilde{\Omega}(\epsilon N/k)$ in the case where the adversary knows the salt using a similar randomized encoding scheme argument as in Section 2. However, now we must encode information about the salt corresponding to each of the *good* range elements. Nonetheless, Theorem 2.4 is essentially unchanged when extended to the case of salt. We summarize the modifications necessary in the proof of Theorem 3.1.

Theorem 3.1. *Fix an algorithm A which makes at most T oracle queries and which takes an advice string of length S. Fix a parameter $\epsilon \in [0, 1]$.*

If $f : [N] \to [N]$ is a k-to-1 function and adv is an advice string such that

$$P[A^f_{adv}(s, f(s||x)) \in f^{-1}(f(s||x))] \geq \epsilon,$$

then there exist encoding and decoding procedures E, D which use shared randomness such that

$$\underset{r}{P}[D(r, E(r, f)) = f] \geq 0.9$$

and the length of $E(r, f)$ is at most

$$S + O(\log N) + \log\left(\binom{N}{N/k} \cdot \frac{N!}{k!^{N/k}}\right) - \frac{3\epsilon N}{1000kT}.$$

Proof. We pick our subset of preimage sets as in the proof of Theorem 2.4. Now, we call an element $y \in f(R)$ *good* if, given its corresponding salt s_y:

1. $A^f_{adv}(s_y, y) \in f^{-1}(y)$.
2. None of the oracle queries in the computation of $A^f_{adv}(s_y, y)$ are in R.

The process for computing a lower bound for $|G|$ remains the same as in Section 2 and we conclude that $f(R)$ contains at least $3\epsilon N/(1000kT)$ good elements with probability at least 0.9.

We now describe the encoding scheme, which is almost identical to the scheme presented in Section 2, except for Step (9). In this encoding, we include:

1. The advice string *adv*, using S bits.
2. The size of the set G, using $O(\log N)$ bits. [3]
3. The partition of N into preimage sets, using
$$\log\left(\frac{N!}{(N/k)!k!^{N/k}}\right) \text{ bits.}$$
4. The set $f([N])$, using $\log\binom{N}{f([N])}$ bits.
5. The set $f(R)$, using $\log\binom{f([N])}{|f(R)|}$ bits.
6. The set G, using $\log\binom{|f(R)|}{|G|}$ bits.
7. A permutation mapping from the preimage sets that partition $[N]-R$ to $f([N]-R)$ using $\log|f([N]-R)|!$ bits.
8. A permutation mapping from the preimage sets that partition $R-f^{-1}(G)$ to $f(R)-G$ using $\log(|f(R)|-|G|)!$ bits.
9. The salt corresponding to each G value, using $|G| \cdot \log(N_s)$ bits.

Decoding proceeds as follows:

1. **Initialize an empty table to store the values of f on $[N]$.**
2. **Fill in the values of f restricted to $[N]-R$:** We have encoded $f(R)$ and $f([N])$, so we can derive $f([N])-f(R)$. Using item (7), we can recover the mapping from preimage sets to $f([N])-f(R)$. Using our definition of the preimage set ordering and the partition of preimage sets from (3), we can fill all the rows of our table restricted to $[N]-R \to f([N])-f(R)$.
3. **Fill in the inverses for all $y \in G$:** We must simulate the inversion procedure $A^f_{adv}(s_y, y)$. To find an inverse for each $y \in G$, the algorithm only queries elements in $[N]-R$. Since we have filled in the values of f restricted to $[N]-R$ in our table, all oracle queries can be answered. Once we know a single preimage x for some $y \in G$, we can find the entire preimage set $f^{-1}(y)$ from the information in item (3).
4. **Fill in the values of f restricted to $f: R-f^{-1}(G)$:** We now know the set $f^{-1}(G)$, so we know the set $R-f^{-1}(G)$. We also know the sets G and $f(R)$, so we know the set $f(R)-G$. Therefore, we can determined the values of f restricted to $f: R-f^{-1}(G) \to f(R)-G$ from item (8) and (3).

[3]Both parties know $|R|$, so there is no need to include it in the encryption scheme.

.

In total, the length of the encoding is

$$\begin{aligned}
&S + O(\log N) + |G| \cdot \log N_s \\
&+ \log \left[\binom{N}{|f([N])|} \cdot \frac{N!}{(N/k)!k!^{N/k}} \cdot \binom{|f([N])|}{|f(R)|} \cdot |f([N] - R)|! \right. \\
&\left. \cdot \binom{|f(R)|}{|G|} \cdot |f(R) - G|! \right] \\
=&S + O(\log N) + |G| \cdot \log N_s \\
&+ \log \left(\binom{N}{N/k} \cdot \frac{N!}{(N/k)!k!^{N/k}} \cdot \frac{(N/k)!(N/k - |R|)!}{(N/k - |R|)!|R|!} \cdot \frac{|R|!(|R| - |G|)!}{(|R| - |G|)!|G|!} \right) \\
=&S + O(\log N) + \log \left(\binom{N}{N/k} \cdot \frac{N!}{k!^{N/k}} \right) - \log(|G|!) + |G| \cdot \log N_s \qquad \text{(see footnote}^4\text{)} \\
=&S + O(\log N) + \log \left(\binom{N}{N/k} \cdot \frac{N!}{k!^{N/k}} \right) + |G|(\log N_s - \log |G| + 1)
\end{aligned}$$

To prove the theorem, we must have that $\log N_s - \log |G| + 1 < -1$. To this end, first recall that we assume $T = \tilde{o}(\epsilon N_p)$. Then for large N_p, we can assume that $T < 3\epsilon N_p/4000$, so we have that

$$\begin{aligned}
&4000T < 3\epsilon N_p \\
\Rightarrow\ &\frac{4000T}{3\epsilon N_p} < 1 \\
\Rightarrow\ &\log \left(\frac{4000NT}{3\epsilon N N_p} \right) < 0 \\
\Rightarrow\ &\log \left(\frac{4000N_sT}{3\epsilon N} \right) < 0 \qquad (N_s = N/N_p) \\
\Rightarrow\ &\log \left(\frac{4N_s}{|G|} \right) < \log \left(\frac{4N_s}{1} \frac{1000T}{3\epsilon N} \right) < 0 \qquad (1/|G| \leq 1000T/3\epsilon N) \\
\Rightarrow\ &\log N_s - \log |G| + \log 4 < 0 \\
\Rightarrow\ &\log N_s - \log |G| + 2 < 0
\end{aligned}$$

Therefore, the total length of the encoding is less than

$$\begin{aligned}
&S + O(\log N) + \log \left(\binom{N}{N/k} \cdot \frac{N!}{k!^{N/k}} \right) + |G|(\log N_s - \log |G| + 1) \\
\leq &S + O(\log N) + \log \left(\binom{N}{N/k} \cdot \frac{N!}{k!^{N/k}} \right) - |G| \\
\leq &S + O(\log N) + \log \left(\binom{N}{N/k} \cdot \frac{N!}{k!^{N/k}} \right) - \frac{3\epsilon N}{1000T}.
\end{aligned}$$

□

[4]We use Stirling's approximation, $\log(x!) = x \log x - x + O(\log x)$.

Corollary 3.2. *Let A be an inversion algorithm that makes at most T oracle queries when inverting a single range element. If for every k-to-1 function f, there exists an advice string adv of length S such that*

$$\underset{s,x}{P}[A^f_{adv}(s, f(s||x)) \in f^{-1}(f(s||x))] \geq \epsilon,$$

then $ST = \widetilde{\Omega}(\epsilon N/k)$.

Proof. This proof is exactly the same as the proof of Corollary 2.6. $\square$

4 Tight Lower Bounds for Permutations, with an Application to Salt

4.1 An Algorithm with $ST = \widetilde{O}(\epsilon N)$

If $k = 1$, we claim that the lower bound $ST = \widetilde{\Omega}(\epsilon N)$ is tight. De et al. and Fiat and Naor [**?**, **?**] present an algorithm when $\epsilon = 1$ that uses time T and space S such that $ST = \widetilde{O}(\epsilon N) = \widetilde{O}(N)$. Their algorithm works as follows:

In the preprocessing phase, for each cycle c_i of length greater than T, we store "shortcut elements" $s_{i,j}$ in the cycle spaced T apart. To invert $y = f(x)$, we first check if y is a shortcut element $s_{i,j}$. If it is, we go to the previous shortcut element $s = s_{i,j-1}$ in the cycle c_i and consider

$$f(s), f(f(s)), f^3(s), \ldots$$

until we find at which step $f^n(s) = y$. Then we know the previous step in the calculation, $f^{n-1}(s)$, is x.

If y is not a shortcut element, we simply apply f to y repeatedly until we reach one of two cases. In the first case, we reach a shortcut element $s_{i,j}$, and then we simply repeat the steps above. In the second case, if our points x and y are in a cycle of length less than T, then we find $f^n(y) = y$, so we know $f^{n-1}(y) = x$.

Note that if we store at most N/T elements, then in our computation we have to apply f at most T times, since our shortcut elements are spaced T apart.

In this section, we extend the algorithm for the full range of ϵ values. We do so using a similar method as above, but we carefully select only a fraction of the shortcut elements to store. The algorithm runs as follows:

Preprocessing Phase:

Since f is a permutation, it is composed of disjoint cycles. Let $A = \{c_1, \ldots, c_p\}$ be the set of the cycles ordered from longest to shortest. Let $|c_i|$ be the length of the ith cycle. Let $B = \{c_1, \ldots, c_i\}$ be the first i cycles in A such that $|c_1| + \cdots + |c_i| \leq \epsilon N$ and $|c_1| + \cdots + |c_{i+1}| > \epsilon N$. Finally, let $C = \{c_j \in B \mid |c_j| > T\}$. For each cycle in C, store $\lceil |c_j|/T \rceil$ evenly spaced "shortcut elements" from the cycle. For example, if $c_j = (1, 2, 3, 4, 5)$ and $T = 2$, then we might store the ordered set $\{1, 3, 5\}$. We say that an element $x \in c_j$ for some cycle c_j is *covered* if any of the following properties holds:

- $|c_j| \leq T$.
- x is a stored shortcut element.
- We have stored a shortcut element y preceding x in c_j and a shortcut element z following x in c_j such that $z = f^{(k)}(y)$ where $k \leq T$.

At this point, there are at least

$$\sum_{c_i \in B} |c_i|$$

covered elements. We are now ready to carefully choose the shortcut elements from c_{i+1} that we will store. In order to ensure that we have covered ϵN values, we must cover

$$\beta = \epsilon N - \sum_{c_i \in B} |c_i|$$

more elements. By construction, $\beta \leq |c_{i+1}|$. Let d be a subinterval of c_{i+1} of length β. Choose $\lceil \beta/T \rceil + 1$ evenly spaced shortcut elements in d, including d's boundary points, and store these shortcut elements.

Online Phase:

Let $y = f(x)$ be the input of the algorithm. We use the notation $f(f(x)) = f^{(2)}(x)$. If y is not a stored shortcut element, apply f to y until one of the following scenarios occurs in the cth iteration, for some $c \geq 1$:

1. $c = T$, in which case we say the operation "timed out."
2. $f^{(c)}(y) = y$, in which case we know that $f^{(c-1)}(y) = x$, and the inversion process is complete.
3. $f^{(c)}(y)$ is a stored shortcut element. Let z be the stored shortcut element preceding $f^{(c)}(y)$ from in the same cycle of f. Now, repeatedly apply f to z until one of the following scenarios occurs in the dth iteration, for some d:
   (a) $f^{(d)}(z) = y$, in which case we know that $f^{(d-1)}(z) = x$, and the inversion process is complete.
   (b) $d = T$, and again, the operation timed out.

If the algorithm timed out in either of Cases 1 or 3b, then at least one of the shortcut elements necessary to invert y was not stored in the Preprocessing Phase. In the Preprocessing Phase, we stored enough shortcut elements to cover ϵN elements in intervals between stored shortcut elements. Therefore, we will be able to invert any element with probability at least ϵ.

Now, we show that $ST = \widetilde{O}(\epsilon N)$. First, note that $|C| \leq \epsilon N/T$. Again we assume $T = o(\epsilon N)$.

$$\begin{aligned} S &= \left\lceil \frac{\beta}{T} \right\rceil + 1 + \sum_{c_j \in C} \left\lceil \frac{c_j}{T} \right\rceil \\ &< \frac{\beta}{T} + 2 + \sum_{c_j \in C} \left(\frac{c_j}{T} + 1 \right) \\ &= \frac{\epsilon N}{T} - \frac{1}{T} \sum_{c_j \in C} |c_j| + 2 + \frac{1}{T} \sum_{c_j \in C} |c_j| + |C| \\ &< \frac{2\epsilon N}{T} + 2 < \frac{4\epsilon N}{T}. \end{aligned}$$

Therefore, $ST = \widetilde{O}(\epsilon N)$.

5 Security Benefits of Salt

We claim that, for all k-to-1 functions, where $T = o(\epsilon N_p)$, salt is an effective countermeasure for the defender.

In Section 3 we show that an algorithm attempting to invert a k-to-1 function must follow $ST = \widetilde{\Omega}(\epsilon N/k) = \widetilde{\Omega}(\epsilon N_s N_p/k)$ for $T = o(\epsilon N)$. For fixed k, N_p, and ϵ, this lower bound grows linearly with N_s, and N_s grows exponentially with respect to the ℓ bits of salt. Therefore, the amount of work the attacker does grows exponentially with the amount of work the defender does. This implies that salts provide security for the defender.

Specifically, if a defender cannot control the size of the passwords, he can salt the passwords with ℓ bits each, forcing a precomputation attack to use at least 2^ℓ more space-time. However, we cannot guarantee that the defender prefixing each password with ℓ bits of salt is as effective as each user using ℓ additional bits in his or her password. In other words, if Eve knows the first ℓ bits of a password, and Fred does not, Eve may be able to achieve a better tradeoff. However, both attackers are worse off than if the defender did not salt the passwords. **Note: if the password is weak, a precomputation attack may not be necessary, negating any effect of the salt.**

We will now show a stronger level of security for certain cases. Namely, we show that under certain circumstances, an adversary cannot make an attack with less space or time if he knows the salt corresponding to each password than he can if he is oblivious to the salt. This stronger level of security holds when both of the following two conditions hold:

1. $k = 1$ or $S > \frac{\epsilon^2 N}{k}$ (i.e. if f is not a permutation, then the attacker must be using sufficiently large space).
2. $T < \epsilon N_p$.

First, suppose $k = 1$, so f is a permutation. In Section 4.1 we show Fred, oblivious to the salt, can use an attack that achieves $ST = \widetilde{O}(\epsilon N)$. In Section 3, we show that the best Eve can do, even if she knows the salt corresponding to each password, is $ST = \widetilde{\Omega}(\epsilon N)$ when $T = \widetilde{o}(\epsilon N_p)$. Now suppose that Fred uses the attack outlined in Section 4.1. Since Eve's best case uses the same space-time as Fred's worst case, knowing the salt values cannot possibly help Eve. Therefore, in the context of precomputation attacks where f is a permutation, adding salt is equivalent to increasing the size of the original password.

A similar argument can be made for all k-to-1 functions if the attacker uses space S such that $S > \frac{\epsilon^2 N}{k}$. De et. al. [?] prove that, under this condition on S, an attacker can acheive $ST = \widetilde{O}(\frac{\epsilon N}{k})$. In Sections 2 and 3, we prove a lower bound of $ST = \widetilde{\Omega}(\frac{\epsilon N}{k})$ for attacks on both salted and unsalted systems when $T = \widetilde{o}(\epsilon N_p)$. This means that an attacker who knows the salt value corresponding to each password cannot do better than an attacker who is oblivious to the salt and uses De's scheme. Therefore, additional bits of salt are equivalent to increasing the size of the origional password.

References