Fast Dictionary Attacks on Passwords Using Time-Space Tradeoff

Tyler Combs

The University of Oklahoma

tyler.combs@ou.edu

April 14, 2015

Overview

Introduction

- Humans tend to generate passwords that are easy to remember
- Common defense: *composition rules* require passwords to include digits and special characters
- Even with the addition of digits and special characters, Humans are not too random

Introduction

- Consider a string "abababababababab" 16 characters long
- Although the string is long and hard to brute-force, is is not very random
- This can be modeled by the *Kolmogrov conplexity* (K-Complexity)
  - Kolmogrov complexity: The length of the shortest Turing Machine that outputs a string and then haults

Introduction

- Itterate over all strings with K-Complexity $\leq$ some threshold
- Can't be done
  - K-Complexity is uncomputable
  - Human randomness is different than computational randomness
- Solution: Use techniques from natural language processing, such as "Markovian Filters" to generate strings that are phonetically simmilar to the users natural language

Goal

Ultimate goal: given a cipher text c recover the password k such that $H(k) = c$

Time-Space Tradeoff

Full look up table

Time

$O(1)$

Space

$O(|\mathcal{K}|)$

?????

Brute-force attack

Time

$O(|\mathcal{K}|)$

Space

$O(1)$

Time-Space Tradeoff

In 1980 Martin Hellman answes this question by contributing a Time-Space tradeoff for the problem of cryptanalysis.

- Precompute: m “chains” of length t
- Store only the first and last items in each chain

$$\begin{array}{cc} & t \\ m & \begin{bmatrix} k^1_{1,1} \xrightarrow{f_1} \xrightarrow{f_1} \cdots \xrightarrow{f_1} k^1_{1,t} \\ k^1_{m,1} \xrightarrow{f_1} \xrightarrow{f_1} \cdots \xrightarrow{f_1} k^1_{m,t} \end{bmatrix} \end{array}$$

$$\begin{array}{cc} & t \\ m & \begin{bmatrix} k^2_{1,1} \xrightarrow{f_2} \xrightarrow{f_2} \cdots \xrightarrow{f_2} k^2_{1,t} \\ k^2_{m,1} \xrightarrow{f_2} \xrightarrow{f_2} \cdots \xrightarrow{f_2} k^2_{m,t} \end{bmatrix} \end{array}$$

$$\vdots \qquad \qquad \vdots$$

$$\begin{array}{cc} & t \\ m & \begin{bmatrix} k^{t-1}_{1,1} \xrightarrow{f_{t-1}} \xrightarrow{f_{t-1}} \cdots \xrightarrow{f_{t-1}} k^{t-1}_{1,t} \\ k^{t-1}_{m,1} \xrightarrow{f_{t-1}} \xrightarrow{f_{t-1}} \cdots \xrightarrow{f_{t-1}} k^{t-1}_{m,t} \end{bmatrix} \end{array}$$

$$\begin{array}{cc} & t \\ m & \begin{bmatrix} k^t_{1,1} \xrightarrow{f_t} \xrightarrow{f_t} \cdots \xrightarrow{f_t} k^t_{1,t} \\ k^t_{m,1} \xrightarrow{f_t} \xrightarrow{f_t} \cdots \xrightarrow{f_t} k^t_{m,t} \end{bmatrix} \end{array}$$

Chain Generation

The function $f(k)$ maps from one key to another. Where:

$$f(k) = R[H(k)]$$

where

- $H(k)$ is a hash function that maps from key k to ciphertext c
- $R(c)$ is a Reduction function mapping from a ciphertext c back to a value $k \in \mathcal{K}$
- Example reduction function: drop last n characters/bits

$$\begin{array}{cc} & \xrightarrow{\qquad t \qquad} \\ m\downarrow & \begin{bmatrix} k^1_{1,1} \xrightarrow{f_1} \xrightarrow{f_1} \cdots \xrightarrow{f_1} k^1_{1,t} \\ k^1_{m,1} \xrightarrow{f_1} \xrightarrow{f_1} \cdots \xrightarrow{f_1} k^1_{m,t} \end{bmatrix} \end{array}$$

$$\begin{array}{cc} & \xrightarrow{\qquad t \qquad} \\ m\downarrow & \begin{bmatrix} k^2_{1,1} \xrightarrow{f_2} \xrightarrow{f_2} \cdots \xrightarrow{f_2} k^2_{1,t} \\ k^2_{m,1} \xrightarrow{f_2} \xrightarrow{f_2} \cdots \xrightarrow{f_2} k^2_{m,t} \end{bmatrix} \end{array}$$

$$\vdots \qquad\qquad \vdots$$

$$\begin{array}{cc} & \xrightarrow{\qquad t \qquad} \\ m\downarrow & \begin{bmatrix} k^{t-1}_{1,1} \xrightarrow{f_{t-1}} \xrightarrow{f_{t-1}} \cdots \xrightarrow{f_{t-1}} k^{t-1}_{1,t} \\ k^{t-1}_{m,1} \xrightarrow{f_{t-1}} \xrightarrow{f_{t-1}} \cdots \xrightarrow{f_{t-1}} k^{t-1}_{m,t} \end{bmatrix} \end{array}$$

$$\begin{array}{cc} & \xrightarrow{\qquad t \qquad} \\ m\downarrow & \begin{bmatrix} k^t_{1,1} \xrightarrow{f_t} \xrightarrow{f_t} \cdots \xrightarrow{f_t} k^t_{1,t} \\ k^t_{m,1} \xrightarrow{f_t} \xrightarrow{f_t} \cdots \xrightarrow{f_t} k^t_{m,t} \end{bmatrix} \end{array}$$

Chain Generation

Example (Chain Generation)

4-character passwords
6-character password sets

$$pass \rightarrow^{H} FE4gT6 \rightarrow^{R} ofie \rightarrow^{H} FP03u2 \rightarrow^{R} ueyf \cdots \rightarrow^{R} lswq$$

$$\begin{array}{cc} & \xrightarrow{\quad t \quad} \\ m \downarrow & \begin{bmatrix} k^{1}_{1,1} \xrightarrow{f_1} \xrightarrow{f_1} \cdots \xrightarrow{f_1} k^{1}_{1,t} \\ k^{1}_{m,1} \xrightarrow{f_1} \xrightarrow{f_1} \cdots \xrightarrow{f_1} k^{1}_{m,t} \end{bmatrix} \end{array}$$

$$\begin{array}{cc} & \xrightarrow{\quad t \quad} \\ m \downarrow & \begin{bmatrix} k^{2}_{1,1} \xrightarrow{f_2} \xrightarrow{f_2} \cdots \xrightarrow{f_2} k^{2}_{1,t} \\ k^{2}_{m,1} \xrightarrow{f_2} \xrightarrow{f_2} \cdots \xrightarrow{f_2} k^{2}_{m,t} \end{bmatrix} \end{array}$$

$$\vdots \qquad \vdots$$

$$\begin{array}{cc} & \xrightarrow{\quad t \quad} \\ m \downarrow & \begin{bmatrix} k^{t-1}_{1,1} \xrightarrow{f_{t-1}} \xrightarrow{f_{t-1}} \cdots \xrightarrow{f_{t-1}} k^{t-1}_{1,t} \\ k^{t-1}_{m,1} \xrightarrow{f_{t-1}} \xrightarrow{f_{t-1}} \cdots \xrightarrow{f_{t-1}} k^{t-1}_{m,t} \end{bmatrix} \end{array}$$

$$\begin{array}{cc} & \xrightarrow{\quad t \quad} \\ m \downarrow & \begin{bmatrix} k^{t}_{1,1} \xrightarrow{f_t} \xrightarrow{f_t} \cdots \xrightarrow{f_t} k^{t}_{1,t} \\ k^{t}_{m,1} \xrightarrow{f_t} \xrightarrow{f_t} \cdots \xrightarrow{f_t} k^{t}_{m,t} \end{bmatrix} \end{array}$$

Key Recovery

- Given any cipher text c, use the reduction function R to generate a key k_i
- Generate a new chain of length t (where t is the length of chains in the stored table)
- Search the *last* elements of the precomputed table for every key generated from the given ciphertext c

$$\begin{array}{cc} & \xrightarrow{\qquad\qquad t \qquad\qquad} \\ m\big\downarrow & \begin{bmatrix} k^1_{1,1} \xrightarrow{f_1} \xrightarrow{f_1} \dots \xrightarrow{f_1} k^1_{1,t} \\ k^1_{m,1} \xrightarrow{f_1} \xrightarrow{f_1} \dots \xrightarrow{f_1} k^1_{m,t} \end{bmatrix} \\ & \xrightarrow{\qquad\qquad t \qquad\qquad} \\ m\big\downarrow & \begin{bmatrix} k^2_{1,1} \xrightarrow{f_2} \xrightarrow{f_2} \dots \xrightarrow{f_2} k^2_{1,t} \\ k^2_{m,1} \xrightarrow{f_2} \xrightarrow{f_2} \dots \xrightarrow{f_2} k^2_{m,t} \end{bmatrix} \\ & \vdots \qquad\qquad \vdots \\ & \xrightarrow{\qquad\qquad t \qquad\qquad} \\ m\big\downarrow & \begin{bmatrix} k^{t-1}_{1,1} \xrightarrow{f_{t-1}} \xrightarrow{f_{t-1}} \dots \xrightarrow{f_{t-1}} k^{t-1}_{1,t} \\ k^{t-1}_{m,1} \xrightarrow{f_{t-1}} \xrightarrow{f_{t-1}} \dots \xrightarrow{f_{t-1}} k^{t-1}_{m,t} \end{bmatrix} \\ & \xrightarrow{\qquad\qquad t \qquad\qquad} \\ m\big\downarrow & \begin{bmatrix} k^t_{1,1} \xrightarrow{f_t} \xrightarrow{f_t} \dots \xrightarrow{f_t} k^t_{1,t} \\ k^t_{m,1} \xrightarrow{f_t} \xrightarrow{f_t} \dots \xrightarrow{f_t} k^t_{m,t} \end{bmatrix} \end{array}$$

Key Recovery

- Because we found the chain our ciphertext belongs to, we can recover the key that generates the ciphertext
- Time-Space Tradeoff comes from the length of the chains

Example (On Board)

$$\begin{array}{cc} & \xrightarrow{\qquad\qquad t \qquad\qquad} \\ m\big\downarrow & \begin{bmatrix} k^1_{1,1} \xrightarrow{f_1} \xrightarrow{f_1} \cdots \xrightarrow{f_1} k^1_{1,t} \\ \\ k^1_{m,1} \xrightarrow{f_1} \xrightarrow{f_1} \cdots \xrightarrow{f_1} k^1_{m,t} \end{bmatrix} \end{array}$$

$$\begin{array}{cc} & \xrightarrow{\qquad\qquad t \qquad\qquad} \\ m\big\downarrow & \begin{bmatrix} k^2_{1,1} \xrightarrow{f_2} \xrightarrow{f_2} \cdots \xrightarrow{f_2} k^2_{1,t} \\ \\ k^2_{m,1} \xrightarrow{f_2} \xrightarrow{f_2} \cdots \xrightarrow{f_2} k^2_{m,t} \end{bmatrix} \end{array}$$

$$\vdots \qquad\qquad \vdots$$

$$\begin{array}{cc} & \xrightarrow{\qquad\qquad t \qquad\qquad} \\ m\big\downarrow & \begin{bmatrix} k^{t-1}_{1,1} \xrightarrow{f_{t-1}} \xrightarrow{f_{t-1}} \cdots \xrightarrow{f_{t-1}} k^{t-1}_{1,t} \\ \\ k^{t-1}_{m,1} \xrightarrow{f_{t-1}} \xrightarrow{f_{t-1}} \cdots \xrightarrow{f_{t-1}} k^{t-1}_{m,t} \end{bmatrix} \end{array}$$

$$\begin{array}{cc} & \xrightarrow{\qquad\qquad t \qquad\qquad} \\ m\big\downarrow & \begin{bmatrix} k^t_{1,1} \xrightarrow{f_t} \xrightarrow{f_t} \cdots \xrightarrow{f_t} k^t_{1,t} \\ \\ k^t_{m,1} \xrightarrow{f_t} \xrightarrow{f_t} \cdots \xrightarrow{f_t} k^t_{m,t} \end{bmatrix} \end{array}$$

Problem

Chains can merge. The reduction function can produce the same key for two different ciphertexts

- If the merge happens early in the chain generation, the table will not cover as many passwords
- Hard to detect because the chains will still have different end points (Unless the merge happens at the same position in the two chains)

Solution

Rainbow Tables

- Use different related reduction functions R_1 to R_n where n is the length of the chain
- If a collision happens, It has to be at the same position in chain generation and collisions can be detected
- Key recovery changes because we are using a different reduction function each time.
  - assume generate chain starting at position $n-1$ use reduction function R_{n-1} and move backwards through the chain

Smart Dictionary Attacks

- Rainbow tables and the classical space-time tradeoff make **no assumptions** about the keyspace other than its size
- Narayanan and Shmatikov use Markov chains and Regular expressions to make some assumptions about the keyspace. This means that we can search only a "smart" portion of the keyspace.

Markov Chains

- Markov chains are commonly used in natural language processing. Most notably speech recognition systems.
- Markov models have been used to generate passwords for users.
- Given a set of states $S = \{s_1, s_2, \cdots, s_n\}$ there is some probability p_{ij} that denotes the probability of transitioning from state s_i to state s_j
- The probability of transitioning to the next state depends only on the current state

Markov Chains

The order of a Markov chain of order n is defined as:

$$P(X_n = x_n | X_{n-1} = x_{n-1}, X_{n-2} = x_{n-2}, \cdots, X_1 = x_1) =$$
$$P(X_n = x_n | X_{n-1} = x_{n-1}, \cdots, X_{n-m} = x_{n-m})$$

Zero-order Markov Chain:

$$P(X_n = x_n | X_{n-1} = x_{n-1}, X_{n-2} = x_{n-2}, \cdots, X_1 = x_1) =$$
$$P(X_n = x_n)$$

First-order Markov Chain:

$$P(X_n = x_n | X_{n-1} = x_{n-1}, X_{n-2} = x_{n-2}, \cdots, X_1 = x_1) =$$
$$P(X_n = x_n | X_{n-1} = x_{n-1})$$

Zero-order Markov Model

In a zero-order Markov model, each character is generated given its underlying proability distribution. This is based on the frequency of the letter in the users natural language. Formally the zero-order model can be written as:

$$P(\alpha) = \prod_{x \in \alpha} \mathcal{V}(x)$$

where: where

- $P(\cdot)$ is the markovian proability distrubuion
- α is a string of characacters
- $\mathcal{V}(\cdot)$ is the frequency of a letter occuring in English
- x is an individual character

First-order Markov Model

In a First-order Markov model, each ordered pair is assigned a proability and each character is generated by looking at the pervious character. The first-order markov model can be written as:

$$P(x_1 x_2 x_3 \cdots x_n) = \mathcal{V}(x_1) \prod_{i=1}^{n-1} \mathcal{V}(x_{i+1}|x_i)$$

where: where

- $P(\cdot)$ is the markovian proability distrubuion
- x_i are individual characters
- $\mathcal{V}(\cdot)$ is the frequency of a letter or ordered pair occuring in English

Markov Dictionary

A probability distribution is not a dictionary. To create a dictionary, discretize the probabilities into two levels using a threshold θ

Zero-order dictionary

$$\mathcal{D}_{\mathcal{V},\theta} = \{\alpha : \prod_{x \in \alpha} \mathcal{V}(x) \geq \theta\}$$

First-order dictionary

$$\mathcal{D}_{\mathcal{V},\theta} = \{x_1 x_2 \cdots x_n : \mathcal{V}(x_1) \prod_{i=1}^{n-1} \mathcal{V}(x_{i+1} | x_i) \geq \theta\}$$

Markov Dictionary

The zero-order model is better for abrivtations and acrynyms. For example, a user picks their favorite song lyric and the first letter of each word creates their password.

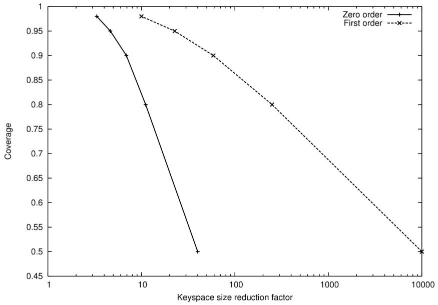

Figure: Convergence vs reduction in Keyspace size ($|\mathcal{K}|$) for 8-character sequences

Deterministic Finite Automaton

- A DFA or Deterministic Finite Automaton is a finite state machine that accepts or rejects a string
- A regular expression can be constructed from a DFA
- Humans are not random with how they use numerals and special characters
    - Numbers tend to be at the end of a password: password1
    - Capital letters are typically at the begining of a password: Password
    - there are typically more lowercase letters in passwords than uppercase letters, numerals, or special characters

Dictionary using a DFA

An improved dictionary is one where strings are both accepted by a Markovian filter and accepted by at least one DFA from some set of DFA's. The updated dictionary is defined as:

$$\mathcal{D}_{\mathcal{V},\theta,\langle M_i \rangle} = \{\alpha : \prod_{x \in \alpha} \mathcal{V}(x) \geq \theta, and \, \exists \, i : M_i accepts \alpha\}$$

where

- A is the set of 26 uppercase characters
- a is the set of 26 lowercase characters
- n is the set if 10 numerals
- s is the set of 5 special characters $\{space, hyphen, underscore, period, comma\}$

Indexing Algorithms

- The goal is to create an algorithm that will efficiently enumerate the passwords in a given password space. Given i as input return the i^{th}
- In the rainbow attack the reduction function maps from ciphertext space to $\{0, 1, \cdots, |\mathcal{K} - 1|\}$
- Composed with a mapping from $\{0, 1, \cdots, |\mathcal{K} - 1|\}$ to a key in $\mathcal{K}$.
- Makes no assumption about keyspace other than its size
- Use the rainbow attack with a "smart" way to choose the keyspace

Dictionary Modification

Modify the dictionary to only consider fixed length strings. This allows for different threshold values θ for each length.

$$\mathcal{D}_{\mathcal{V},\theta,\ell} = \{\alpha : |\alpha| = \ell \mathit{and} \prod_{x \in \alpha} \mathcal{V}(x) \geq \theta\}$$

Discretization

The algorithm also needs to discretize the probability distribution of the strings. First, turn the dictionary into a sum rather than a product.
Transform the product

$$\prod_{x\in\alpha} \mathcal{V}(x) \geq \theta$$

$$\log(\prod_{x\in\alpha} \mathcal{V}(x)) \geq \log(\theta)$$

$$\log(\mathcal{V}(x_1)\mathcal{V}(x_2)\cdots\mathcal{V}(x_n)) \geq \log(\theta)$$

$$\log(\mathcal{V}(x_1)) + \log(\mathcal{V}(x_2)) + \cdots + log(\mathcal{V}(x_n)) \geq \log(\theta)$$

Discretization

To arrive at a discrete version of the modified dictionary:

$$\mathcal{D}_{\mathcal{V},\theta,\ell} = \{\alpha : |\alpha| = \ell and \sum_{x\in\alpha} \mu(x) \geq \lambda\}$$

Where where

- $\mu(x) = \log(\mathcal{V}(x))$
- $\lambda = \log(\theta)$

Discretization

- $\mu(x) = \log(\mathcal{V}(x))$
- Discretize the values of the μ function to the nearest multiple of some μ_0
- Narayanan and Shmatikov use a μ_0 that yields approximately 1000 different discrete values

Partial Dictionary

Define a partial dictionary $\mathcal{D}_{\mathcal{V},\theta,\ell,\theta',\ell'}$ as follows:

- let α be a string such that $|\alpha| = \ell'$
- $\prod_{x \in \alpha} \mathcal{V}(x) = \theta'$

Then

$$\mathcal{D}_{\mathcal{V},\theta,\ell,\theta',\ell'} = \{\beta : \alpha\beta \in \mathcal{D}_{\mathcal{V},\theta,\ell}\}$$

Zero-Order Markovian Dictionary

Precompute the size of a partial dictionary (recursively) and store in a 2D-array of size (ℓ , `num_levels`)

$$|\mathcal{D}_{\mathcal{V},\mathit{threshold},\mathit{total\_length},\mathit{level},\mathit{current\_length}}|$$

Zero-Order Markovian Dictionary

```
partial_size1(current_length, level)
{
    if level >= threshold: return 0
    if total_length = current_length: return 1
    sum = 0
    for each char in alphabet
        sum = sum + partial_size1(current_length+1,
                                  level+mu(char))
    return sum
}
```

Complexity linear in the product of total length, number of characters in alphabet, and number of levels

Zero-Order Markovian Dictionary

- For cryptanalysis, given an index i produce the corresponding key k in the dictionary $\mathcal{D}$
- Use precomputed partial size to determine the first character by looking up a value from the precomputed matrix.
  - Adjust the index to a new index relative to the first character
  - Adjust the threshold based on the frequency of the first character

Zero-Order Markovian Dictionary

Initally call `get_key1(0,0)`

```
get_key1(current_length, index, level)
{
    if total_length = current_length: return ""
    sum = 0
    for each char in alphabet
        new_level = level + mu(char)
        // looked up from precomputed array
        size = partial_size1[
            current_length+1][new_level]
        if sum + size > index
            return char + get_key1(
                current_length+1,
                index-sum, new_level)
        sum = sum + size
```

First-Order Markovian Dictionary

Same as zero order only in get_key, we need to keep track of the last character

DFA Dictionary

Similar to zero-order Markov dictionary except instead of a threshold and levels, we have states and transitions

Any Keyspace $\mathcal{K}$

Assume that we have a superspace $\mathcal{K}' \supset \mathcal{K}$ ad we need to decide that given $\alpha \in \mathcal{K}'$ if $\alpha \in \mathcal{K}$

Split K' into m bins of size t and precompute the number of members in each bin that are in K

Given an index, quickly figuew out what bin it falls into and interate over all keys in that bin and test each one for membership

$O(|K'|)$ precomputation time ... storage and index

Hybrid Markovian/DFA

Multiple Keyspaces

Experiment

- Measure coverage of rainbow attack vs hybrid attack
- 142 real user passwords
- 6-Character alphanumeric sequences for the rainbow attack ($|\mathcal{K}| = 36^6 \approx 2 * 10^9$)
- 70 regular expressions

Results

Category	Count	Rainbow	Hybrid
Length at most 5	63	29	63
Length 6	21	10	17
Length 7	18	0	10
Length 8, A* or a*	9	0	6
Others	31	0	0
Total	142	39(27.5%)	96(67.6%)
only length ≥ 6	79	10(12.7%)	33(41.8%)

Figure: Passwords recovered in Hybrid attack vs. Rainbow attack

Conclusions

- One of many attacks targeting human weakness
- Some possible defences against dictionary attacks for human memorable passwords
  - Graphical passwords
  - Biometric information
- Are these actually safer?

Analysis