Cryptographic Applications of Random Variables

Austin Chase Minor

November 24, 2016

1 Introduction

You are a budding cryptanalyst and have noticed a security flaw in your companies login system. You find that matching certain keys to values gives you access to peoples account. You have no way of determining which key/value pairs match. Also, you only have so many retry counts. Your job is to determine the number of key/value matches you will get given a certain retry count. This is a mathematical hard problem to determine the exact probability since as a key/value is matched the number of keys decrease and thus the probability increases for a match. So remembering the theory of random variables, you code the problem statement and simulate to find the average number of matches for a given retry count.

We will start with the mathematical description of the problem. From there, we will give the problem statement. After running the test, we will analyze the problem and give a conclusion.

2 Mathematical Description

Let A, B be a set of keys and values respectively.
Let $f : A \to B$ be the function relating keys to values that you are trying to discover.
Let $\phi : A \times B \to 0, 1$ be a truth function representing $true = 1$ if $f(a) = b; a \in A, b \in B$ and $false = 0$ otherwise.
Let $k \in \mathbb{N}$ represent the number of retry counts allowed (the number of tries for each individual $b \in B$ to the ϕ function.

3 Problem Statement

Through some thought, it can be shown that the best way to go about trying potential values is to try each $b \in B$ with every $a \in Ak$ times removing a as they are matched. This guarantees a minimum of k matches. Furthermore, it maximizes the shared information between $b's$ resulting in a higher probability. The Matlab code implementing this is below.

By repeating this trial over k several times, we can generate a sample of the actual probability. This allows us to estimate the mean and standard deviation along with seeing the general shape of the function. The Matlab code for this along with the corresponding graphs are below.

```
PRECISION = 100;
RETRY_COUNT = 10000;

freq = zeros(PRECISION, PRECISION);
match_num = 0;

%Graph for each k as
for j = 1:PRECISION
  for i = 1:RETRY_COUNT
    match_num = match(PRECISION,j);
    freq(j, match_num) = freq(j, match_num) + 1;
  end
end

%variation and standard deviation
%removes nonzero values of freq for stddev and var
temp = zeros(PRECISION, RETRY_COUNT);
for i = 1:PRECISION
  for j = 1:PRECISION
    if freq(i,j) != 0
      %t is [prev array, array of size number of matches with vals of match number]
      t = [temp(i,temp(i,:) != 0) ones(1,freq(i,j))*j];
      %pads array with zeros to help with previous assignment
      temp(i,:) = padarray(t, [0 (RETRY_COUNT - size(t,2))], 'post');
    end
  end
end

variance = var(temp');
std_deviation = std(temp');
mean_val = mean(temp');

figure
surf(freq);
figure
plot(1:PRECISION, variance, 1:PRECISION, std_deviation);
figure
plot(1:PRECISION, mean_val);
```

Listing 1: CryptoSim Main

```
function successes = match(sz, k)
  A = B = 1:sz;
  A = randomize_array(A);
  count = 0;

  for i = 1:sz
    for j = 1:k
      if A(i) == B(j)
        B(j) = [];
        count = count + 1;
        break;
      end
    end
  end
  successes = count;
end
```

Listing 2: Match

```
function B = randomize_array(A)
  sz = size(A);
  sz = sz(2);
  for i = 1:sz
    r = floor(sz * rand(1)) + 1;
    temp = A(i);
    A(i) = A(r);
    A(r) = temp;
  end
  B = A;
end
```

Listing 3: Randomize Array

4 Analysis of Problem

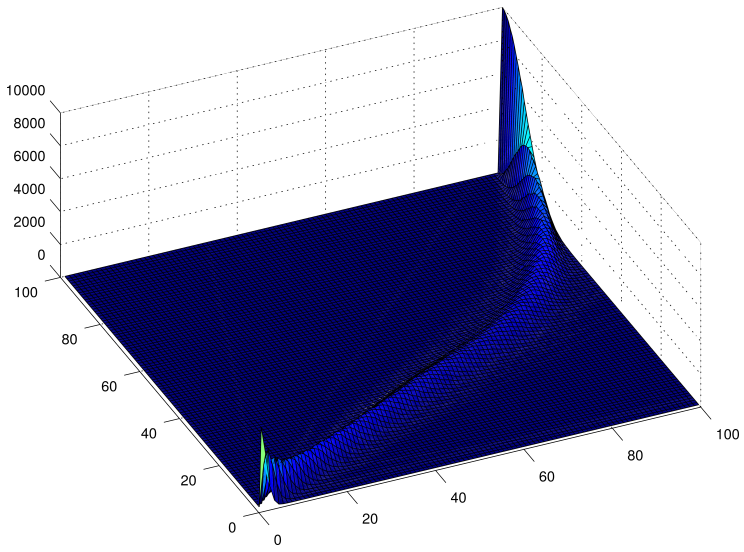

Figure 1: 3D Image of the match frequences and numbers

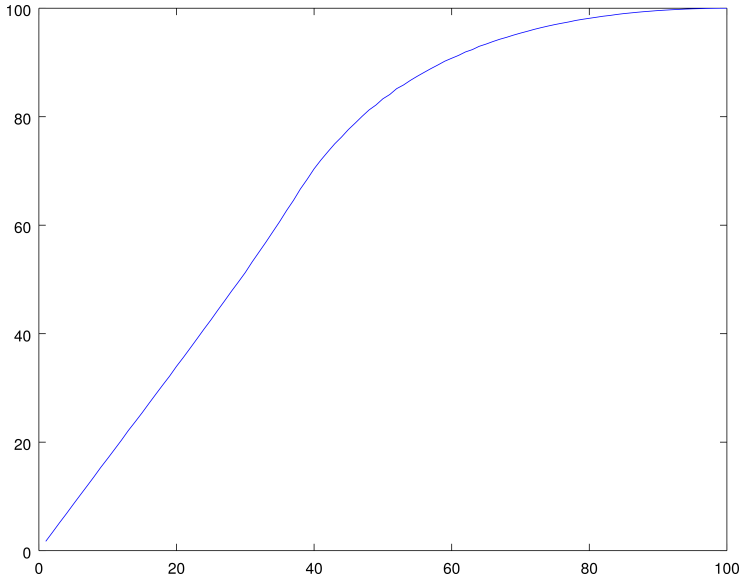

Figure 2: Average mathches over k

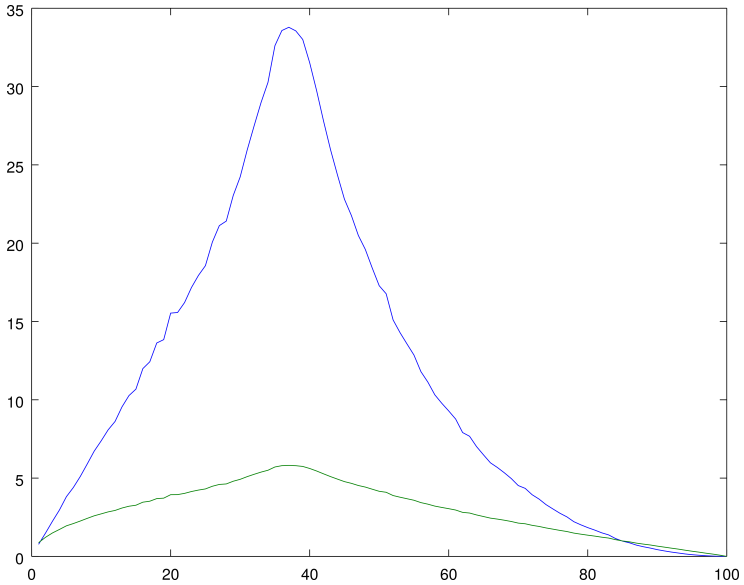

Figure 3: Standard Deviation and Variance over k

It is logical to assume that at a certain point k their would be a tipping point. I.E. their would be a point where the number of matches being found would reduce the number left to find in a vicious cycle resulting in complete matches at a lower number. However, Figure 2 shows that this indeed is not what happens. Instead the graph increases linearly then levels off. This is surprising and means that in theory this problem is not as insecure as it may seem at first glance. Other than that, the mean is not very interesting. It shows that the number of matches to k is between (1:1 and 2:1).

Overall, the matches you receive are fairly consistent. This is shown by the standard deviation in Figure 3. At max, the standard deviation is 5-6. This occurs around 38. We theorize that this is due to the leveling off of the graph in Figure 2 for high k. The low k, we theorize, is due to the lack of matches contributing to more matches, this phenomenon being described above. Thus, an attacker would be able to consistently break the system at the rate given by Figure 2 (the mean).

5 Conclusion

We discovered that this problem is far from being as cryptographically insecure as at first glance. The growth rate does not hit a tipping point. Rather, it actually levels off instead. However, this merely moves it down from high security problem to a medium security problem because the growth rate is still too close to (2:1).