Birthday Attacks on Various Hash Functions

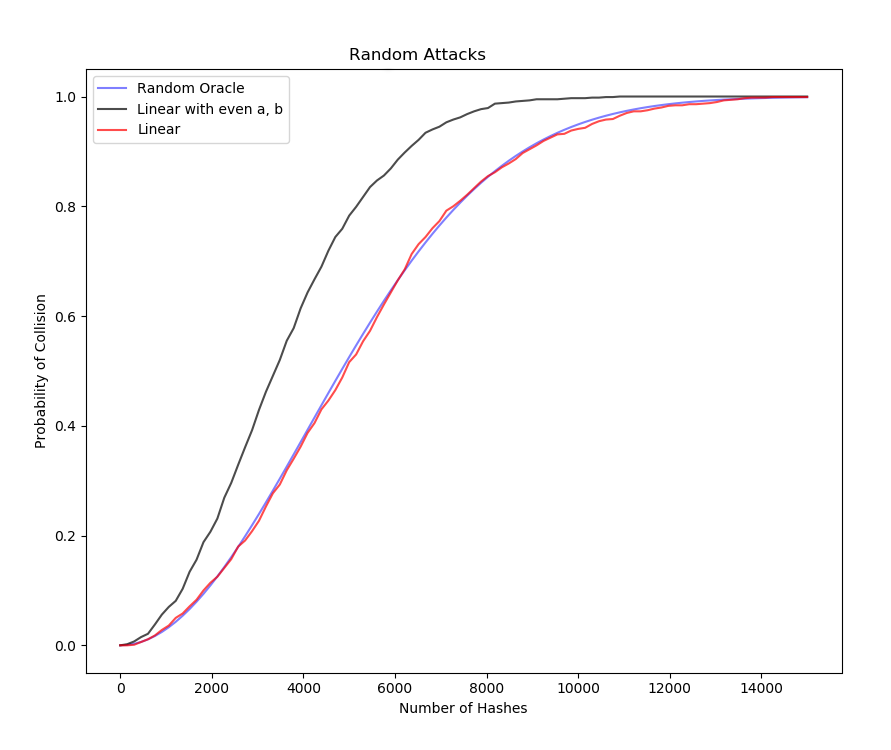
By Dane Zieman

**OVERVIEW**

The purpose of this experiment is to examine how traits such as preimage resistance or collision resistance affect hash functions in practice. In this experiment various hash functions were subjected to birthday attacks and their performance was measured. The hash functions used in this project were MD5 and two simple hash functions invented for this experiment. Additionally, these hash functions were compared to the random oracle model. The purpose of this choice of hash functions was to compare a modern, robust hash function to two simple, less robust hash functions and compare how various types of attacks work against these types of hash functions.

**EXPERIMENT DESIGN**

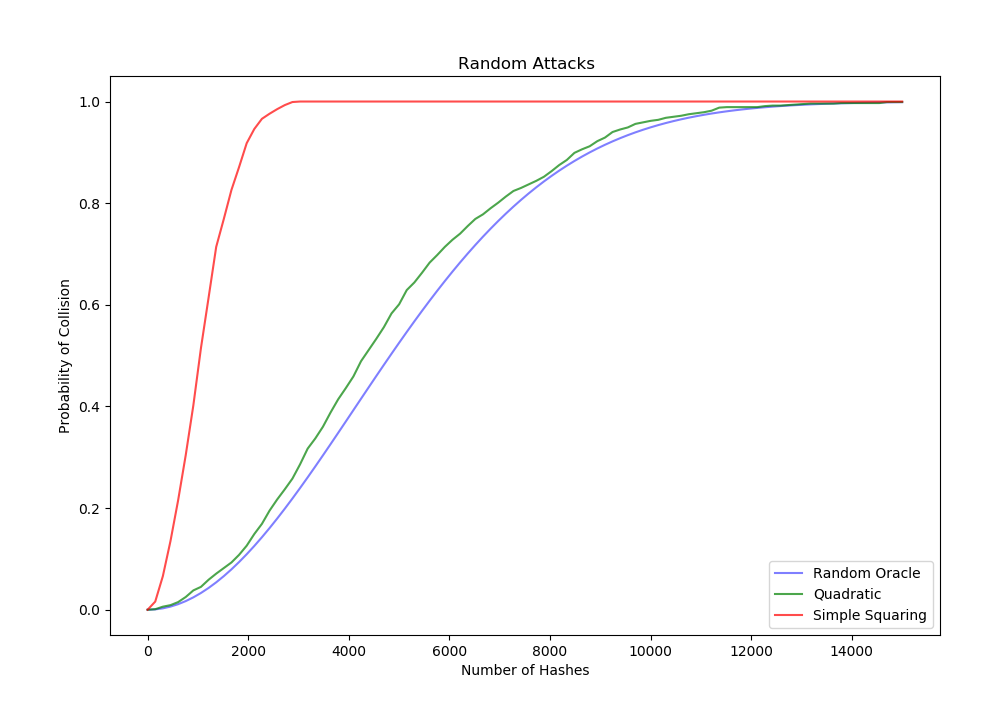
The first step of this experiment was to design a program that could perform birthday attacks on hash functions and record how long it takes to get a collision. This was accomplished by writing a program that can be found at https://github.com/dane8373/MCS425HashFunctions. Next, the two simple hash functions needed to be designed. These hash functions needed to be designed such that they would have similar resistance to random birthday attacks as the random oracle model hash function would. The original goal was to one of these functions be a simple a\*x + b mod n linear hash function, and have the other one be a a\*x2 mod n quadratic hash function. However, these functions needed to be refined slightly in order to achieve similar resistance against random birthday attacks as the random oracle model. For example, the linear function would not work with a randomly chosen a and b, as the case where a and b are both even would leave the hash function more vulnerable to random birthday attacks. This is because if a and b are both even, then the hash value will also always be even, and thus only half of the output space will ever be used. This problem was remedied by choosing a and b to be large prime numbers, and the difference in resistance to random birthday attacks for these two variants of the linear hash function are shown in Figure 1 below.



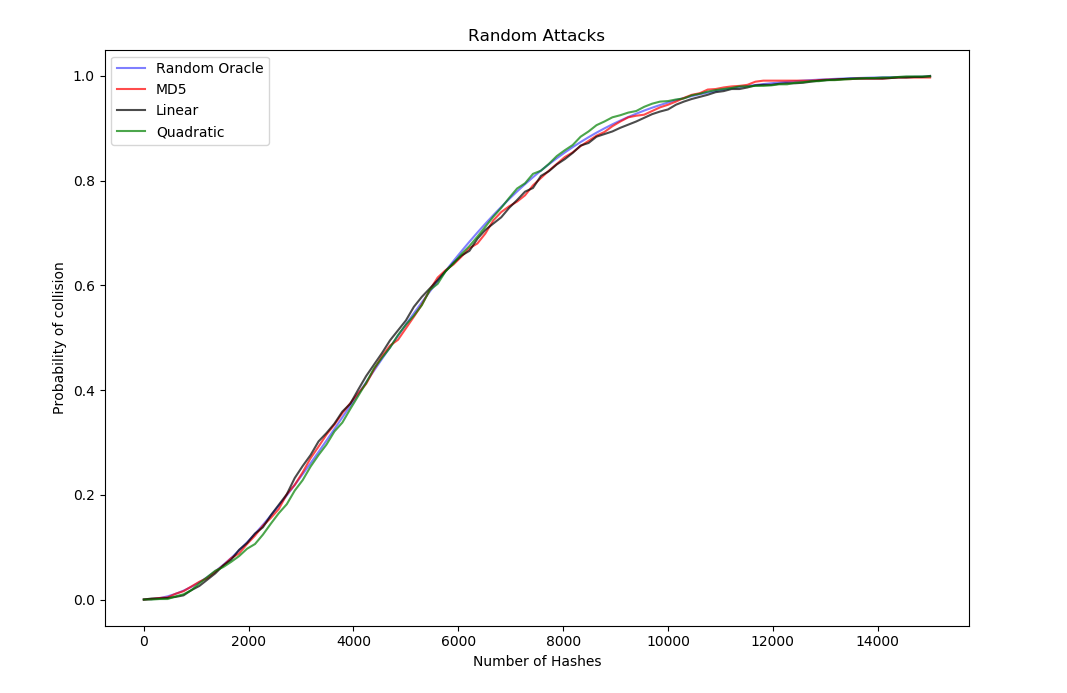
**Figure 1:** Probability of collision forLinear hash function (a\*x + b mod n) with different choice for a and b (1000 tests)

Likewise for the quadratic function, a simple a\*x2 mod n hash function did not have strong resistance to random birthday attacks and needed to be modified. Instead of a simple a\*x2 mod n, the following quadratic equation was used for the quadratic hash function.

This was also chosen with a and b both as large prime numbers. The x%2 term was added to ensure both odd and even numbers would get mapped to by the hash function. Figure 2 shows the performance of these two functions against random attacks

 **Figure 2:** Probability of collision the two quadratic hash functions. (1000 tests)

Once these two hash functions were created, these two functions along with MD5 were tested against random attacks, the results are shown below in Figure 3.



**Figure 3:** Probability of collision forLinear, Quadratic, and MD5 hash functions against random attacks (1000 tests).

As evident by Figure 3, all of the has functions show good resistance to random attacks. For the main experiments, these hash functions were subjected to the following more specialized attacks

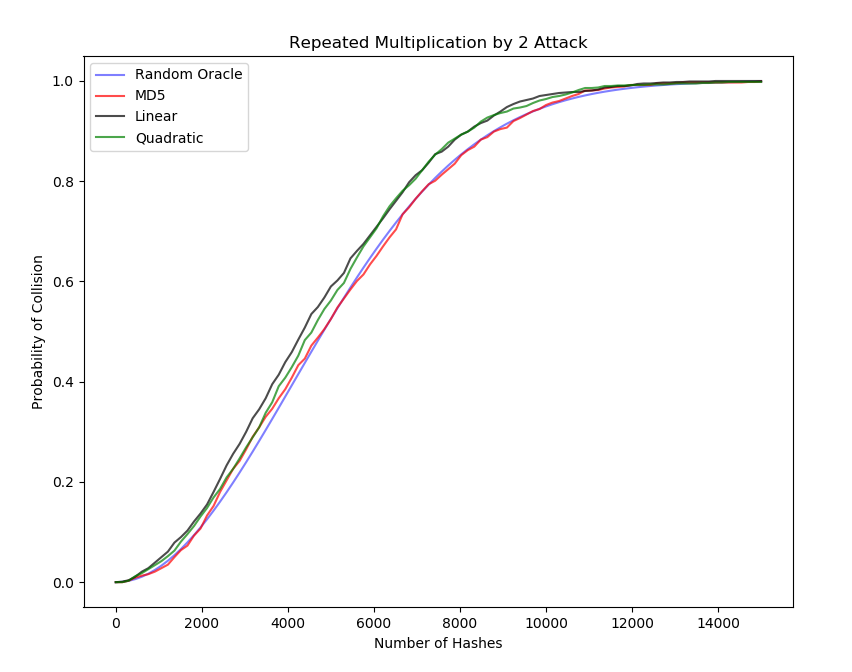
1. Repeated Multiplication by 2 attack: Repeatedly multiply a seed number by 2 and hash it. Pick a new seed once the resultant is greater than 64 bits.
2. Evens only attack: Only hash even inputs
3. Additive probing: Start with a seed number, and add this seed to itself every iteration of the test
4. Square of old Hash: Start with a random number for the first input, then for every subsequent iteration take the square of the previous output as the input for the next iteration.

These attacks were chosen because they were hypothesized to have different effects on the simplistic hash functions. For each of these tests, the collision detector was run 1000 times, and the probability of a collision for a given number of hashes was calculated. For each of the tests, the input size was 64 bits, and the output size was 24 bits. These numbers were chosen as it allowed the 1000 tests to be performed in a reasonable time on a standard computer (around 5 minutes per attack).

**EXPERIMENTAL RESULTS**:

*Repeated multiplication by 2 attack:*

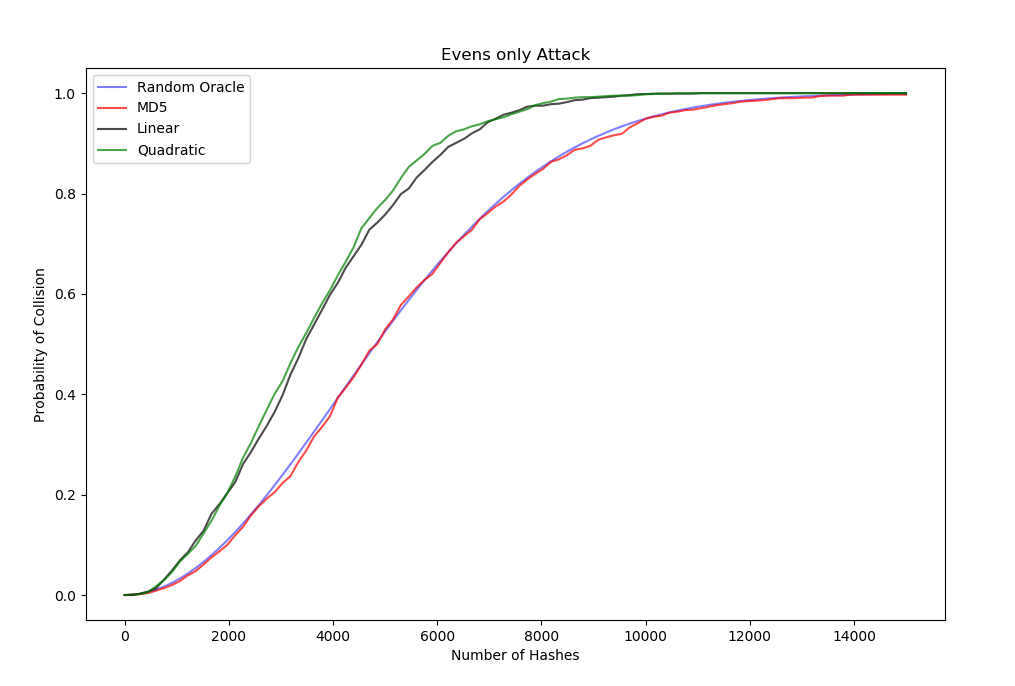
Figure 4 shows the results of the repeated multiplication by 2 attack for the three hash functions

 **Figure 4:** Probability of collision forLinear, Quadratic, and MD5 hash functions against repeated multiplication by 2 attacks (1000 tests).

For this type of attack, the linear and quadratic model performed slightly worse than the MD5 hash function did, the MD5 model was not affected at all by this type of attack. This attack had an effect on the linear and quadratic hash functions map numbers that are close to each other to hashes that are also relatively close to each other. The repeated multiplication by 2 ensures that the input numbers aren’t too close together, making collisions more likely

*Evens Only attack*

Figure 5 shows the performance of the hash functions against an evens only attack



**Figure 4:** Probability of collision forLinear, Quadratic, and MD5 hash functions against evens only attacks (1000 tests).

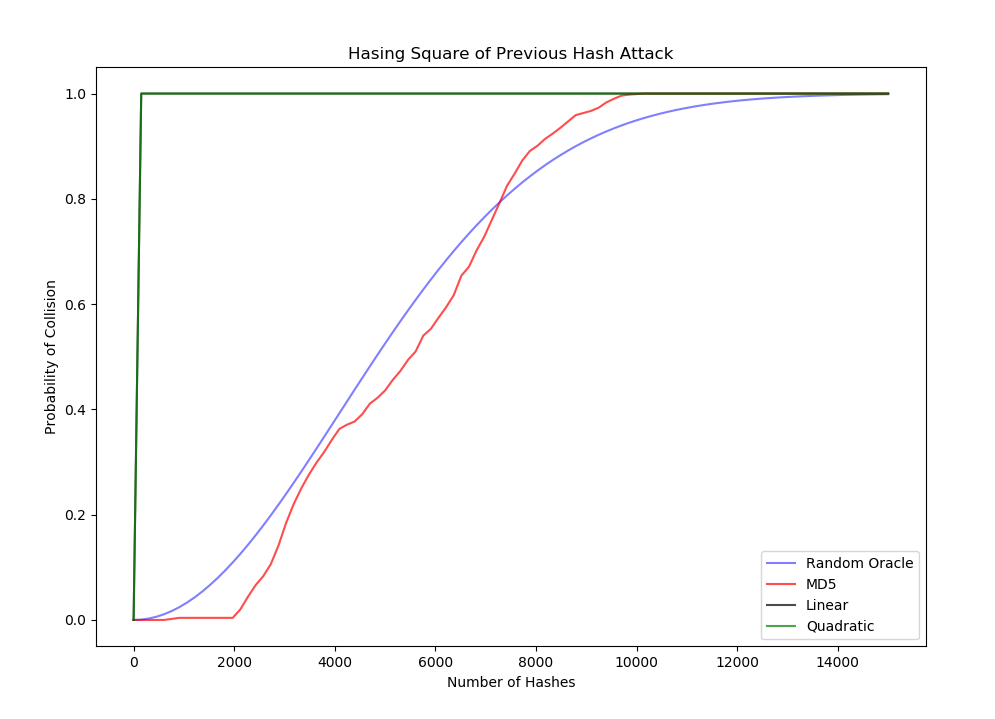
For this type of attack, the linear and quadratic hash functions performed significantly worse than the MD5 model, which was unaffected by this type of attack. This type of attack was very effective against the linear and quadratic hash functions because the parity of the input determines the parity of the output. Therefore, by only trying one parity of numbers against these hash functions, the output space is effectively halved.

*Additive probing attack*

For this attack, the MD5 model was unaffected, performing roughly the same as it does for random attacks. The linear and quadratic functions had fewer collisions under this attack than even the random oracle model. In fact, not even one collision was found in a reasonable amount of computing time. This fact, however, isn’t necessarily good news for these hash functions. If an attacker were to perform an attack like this on a hash function, and notice that they are receiving significantly fewer collisions than the random oracle model, then they would gain information about how the hash function is functioning. Namely, they would realize the property that close numbers get mapped to close hashes, even without knowing the type of hash function that is being used. Using this knowledge, and attacker would be better prepared to construct as successful attack for this hash function.

*Square of previous hash attack*

Figure 5 shows the performance of these hash functions against the square of previous hash attack



For this attack, The linear and quadratic hash functions had collisions almost immediately, it took an average of 20 attacks for them to have a collision. Even the MD5 hash function was effected by this type of attack. This makes sense for the linear and quadratic attack, these hash functions essentially perform modular multiplications and additions. By repeatedly squaring the output of the hash function, it is effectively multiplying and adding the same number mod 2^24, and eventually these additions and multiplications will lead to multiply by a multiplicative inverse or adding an additive inverse.

**DISCUSSION**

Despite the simple hash functions having good performance against random attacks, they showed weakness to the targeted attacks. The evens only, Repeated multiplication by 2 and square of previous hash attack were all able to find collisions relatively quickly. This is because these hash functions do not have good preimage resistance, collision resistance or second preimage resistance. Given an input value to these hash functions, it is rather easy to find another input that maps to the same value (for example, for the a\*x + b mod n function, for any input t the number t + 2^24 will map to the same hash as t. Likewise for the squaring function, -t would have the same hash as t). Additionally, even if an attacker does not know the exact details of the hash function, they can run various attacks, observe the performance and make inferences about how the numbers are being generated (if the hash function does not exhibit strong preimage resistance).

**CONCLUSION**

When examining a hash function, it is important to examine more than uniform dispersion of the hash values. As demonstrated here, if the hash function does not have strong collision resistance or preimage resistance, then the hash function will succumb to various types of attacks. Further, even if an attack does not generate collisions, it may leak information about how the hash function works if the hash function generates fewer collisions than the random oracle model would predict.