Inference Attacks on a Hash Function
Master’s Project Proposal

Benjamin W. Bloom
Department of Computer Science
Rochester Institute of Technology
Rochester, NY, USA

January 18, 2010

1 Overview of Cryptographic Hash Functions

A hash function takes a message of arbitrary length, and produces a fixed length output from that message. Cryptographic hash functions have several additional constraints on top of that broad definition. They must be collision resistant, as well as first and second preimage resistant. Collision resistance means that finding two random messages such that they both hash to the same fixed length output should be computationally infeasible. Preimage resistance means that the algorithm must not be readily invertible. In other words, it should be computationally infeasible to find a message that produces a given hash, when provided with only the hash. Second-preimage resistance means that given a message and an output, it should be infeasible to find a second message which hashes to the same output as the first.

A “weak” hash function is one that is one that does not meet one or more of the criteria for a Cryptographic hash function. Such algorithms are often used in applications where this kind of security is not required, such as computing keys for hash tables. A “strong” hash function is one that meets all three criteria for a Cryptographic hash function. These will be the focus of this project, and will be discussed in detail later on.

The first and second preimage attacks expect to take $2^n$ hash function evaluations to successfully find a preimage. A collision attack operates on the birthday attack principle, which states that in a group of twenty-three random people, the probability that two of them share the same birthday is fifty percent. The birthday attack to find a collision in a hash function states that it takes $2^{n/2}$ evaluations of the hash function in order to expect to find two messages that hash to the same value.

Cryptographic hash functions are widely used for applications varying from password exchange protocols to data integrity to digital signatures. If a hash function that is used in one of these applications can be shown to be weak, then it is no longer fit for use, and the security of the application is compromised.

Consider a digital signature. This is generally a form of public key cryptographic security. A message is hashed and the resulting hash is encrypted with a Public or Private key (depending on which party is sending the message). When the recipient gets the message, they can hash the message using the same hash algorithm, and decrypt the encrypted hash using the Public or Private key (whichever one was not used for encrypting). If the locally computed hash and the
decrypted value are the same, then the message is accepted as authentic. If the hash function used for this purpose is not cryptographically secure, then an attacker can substitute a different message which hashes to the same thing as the original message, and keep the signature, and the recipient will accept the attacker’s message as authentic.

These types of applications are the motivation for trying to break a hash function. The successful break of the function will rule it out as a good candidate for a secure hash algorithm. The inability to break the algorithm does not mean it cannot be broken, but does instill a level of confidence in the security of the algorithm that was not there before examination.

2 Summary of Project

2.1 Inference

My project will use a method called inference in order to try to break a hash function. Inference uses information about a function’s behavior in order to deduce the value of variables in the function. For example, when dealing with the exclusive-or function, if the two inputs are \(a\) and \(b\), and the output is \(c\), and it is known that \(a = 1\) and \(c = 1\), then through inference, it can be deduced that \(b = 0\), based on knowledge about XOR’s behavior. Furthermore, if one of the variables is known, and the other two unknown, then even though specific values cannot be determined, something can still be said about them, which is that if \(b = 0\) and \(a\) and \(c\) are unknown, then \(a = c\), even though it is unclear whether they are equal to 0 or 1.

2.2 CubeHash

My project will focus on CubeHash. This is a hash algorithm developed by Daniel J. Bernstein, and is a candidate in the NIST competition to select SHA-3. This algorithm is a very good candidate to attack with an inference attack, as it features a very simple round function, and the only mathematical operations are addition and exclusive-or. It also uses rotation and swap operators, which do not add mathematical complexity to an inference attack. CubeHash is currently one of fourteen candidate algorithms that has made it through the first two rounds of the NIST competition. There are many published attacks on weakened versions of CubeHash, but none on the full-strength versions.

2.3 Project Goal

I hope to be able to show that an inference attack will do better than a brute-force attack on a hash function, by analyzing the solve time of the inference attack versus the solve time of a brute force attack on various weaker versions of the algorithm. I would like to provide an inference attack as an idea for ways to solve computationally heavy problems. This attack is similar to a SAT solver, in that it has a huge number of variables, and will try to find an assignment that satisfies the function.

3 Hypothesis

Displaying an Inference Attack on a hash function that can provide a preimage in a faster time than a brute force is a novel contribution, especially since the fastest attacks on a Cryptographic Hash Function should be a brute force attack. The ability to show that a hash function is breakable faster than brute force will be a great step in showing that it should not be used for applications which Cryptographic hash functions are normally used. Conversely, if a thorough investigation using this type of attack cannot show that the hash function is breakable in less time than a brute force attack, that helps to give support to the fact that the function is a viable Cryptographic hash function.
4 Versus Existing Work

There are many types of different attacks on Cryptographic Hash Functions that have been published. Some of these are generic attacks that apply to most Cryptographic Hash Functions [2]. Other attacks are specific to CubeHash itself, and a paper by Aumasson, et al. shows several of these [1], including some that take an enormous memory pool in order to execute. Another attack which takes an infeasibly large memory pool is shown by Khovratovich[5]. These papers also describe attacks which look specifically at CubeHash’s underlying round function, and try to exploit weaknesses in it. This is the same style of attack that will be shown in this project, just done differently, using different mechanics.

The existing work that has been done on CubeHash which involves Inference Attacks has been done by Bloom and Janis[3]. This is also reflected in the paper by Bloom and Kaminsky[4] which focuses on Single Block attacks and Statistical Tests, rather than Inference itself.

5 System Architecture

The system is set up to attack any version of the function, since the three parameters \( r \) (the number of rounds of mixing), \( b \) (the amount of input bytes to mix per round), and \( h \) (the size of the hash output) can be varied (within specified limits), the best approach is to be able to attack any of them, though I will primarily focus on large values of \( b \) and \( h \). There is a basic Inference engine which can handle Exclusive Or (XOR) and Addition variable links. The idea is to link up all of the variables that depend directly upon each other using these XOR and Addition objects, and then start setting variables to certain values and see how quickly a preimage can be resolved.

There will probably only be a handful of classes in the final system, these Inference objects, and a driver system, that performs variable-assignment backtracking when certain combinations of variable values are found to not work together to produce the desired result.

6 Principal Deliverables

- Technical Report of System Performance
- Examples of preimages
- System Code
- User Manual
- Design Documentation

References


7 Schedule

- System Code Completion: Friday, March 19, 2010
• Finished gathering data and analysis: Monday, March 29, 2010

• Finished Technical Report writeup: Monday, April 5, 2010

• Finished Documentation and User Manual: Wednesday, April 12, 2010

• Target Defense Date: Friday, April 23, 2010

8 Current Status

Since this project is an expansion of a previous course project, a decent chunk of the system is written, although it might require quite a bit of refactoring in order to do exactly what I have specified in this proposal.