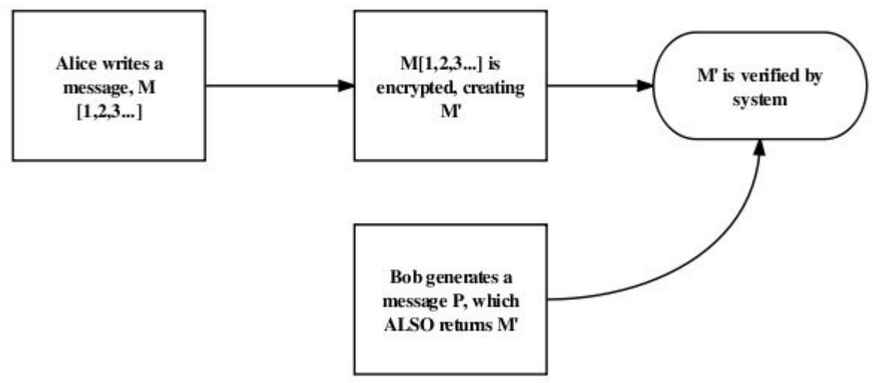
Securing the SHA-1 Algorithm Using Pre-Processing Techniques

Betty Huang

Computer Systems Lab 2009-2010

Abstract

Since 2005, collision factors have been found for the SHA-1 algorithm. This paper aims to provide several mechanisms to improve the applicability of the algorithm for areas that are not able to adopt a new standard readily. Authors such as Yiqun Lisa Yin, Michael Szydlo, and various members of the National Institute of Standards and Technology have written about the possibilty of modifying the original input in order to reduce the likelihood of collisions/pre-image attacks, but none have tested extensively on how much the randomization would increase the efforts of decrpytion. I hope to be able to test a (likely simplified) version of the techniques against several sophisticated collision-based attacks.



Background and Introduction

Most encryption algorithms are designed to be one-way; that is, they prevent reverse engineering to receive the original message inputted by the user. Usually, the process of encryption consists of a pre-image (plaintext), which is then put into a method in order to return a post-image (encrypted text). Then, this document carries a unique “signature,” which is used to verify the validity of the document. Hash functions are designed to be 1-1 (that is, every combination message M[m1, m2, m3] has a unique encrypted text associated with it).

There are two primary types of attacks against these encryption algorithms. First, pre-image processing works in an attempt to retrieve the plaintext from the encrypted text. This is usually difficult and exhaustive for computer systems, and generally infeasible for anyone without extensive resources. Second, collision attacks aims to find areas in which two messages are similar in nature, in order to find another message that matches the encrypted text of the first message. In a visual example:

Discussion

Most of the research and advances in this field have written their code in C, but it would be more convenient to write the code in Python, which has a built in MD5 hash function. Currently, I have preprocessing code written in python for SHA-1, but the tunneling code available is only written in C (and for MD5).If it is possible, I will attempt to recode the method that Klima outlined for the SHA family of algorithms and analyze results from there; as of now, finding collisions for MD5 averages 30.1 seconds on my notebook, which may be improved if the pre-processing technique were to be applied. Unforunately, Klima does not give an indication of what inputs the tunneling program receives, which may be difficult in the future for testing unconvential strings (or particularly long strings)

Results

I expect that the test runs with message preprocessing will require more effort than the test runs that do not use the technique. A time graph of the resultant data could be used as a visual. This project has value to system administrators that do not have the time/resources to implement newer, more secure algorithms, as they will be able to manage their data without worrying as much on the possibility of attacks.