# Properties of Password Hashing Functions

An offline attack is a security threat wherein the attacker has gained access to a system’s password file, which is used to verify log in credentials. Hash functions are used as a control mechanism to reduce the risk that such attacks pose by storing passwords as hash codes, from which the original passwords cannot be readily derived[[1]](#footnote-1).

It is important to note that in a system that implements hash functions as a security measure, there may be multiple ‘accepted’ passwords for a single account. This is because authentication is performed through the comparison of hash codes not the passwords that they are generated from. In the event of a hash collision, multiple passwords would generate the same hash code, thus each password could be used in place of any of the other passwords[[2]](#footnote-2). As such a scenario increases system vulnerability to attacks (both online and offline) that attempt to find an accepted password through a series of guesses, hash collisions must be minimized.

Hash functions used for such purposes must be pre-image resistant (one-way)[[3]](#footnote-3). As, in an offline attack, the attacker has managed to obtain a copy of the password file, it is entirely possible for them to have also obtained a copy of the hash function itself. Thus, the function must be designed so that it cannot be reverse-engineered to yield an inverse function, otherwise such a function may be used to derive accepted passwords.

Even if a hash function is pre-image resistant, it is still possible to determine an accepted password through a guided-search or brute-force; that is, entering possible passwords into the hash function until it produces a hash code that is in the password file[[4]](#footnote-4). This poses a significantly greater risk than simply entering guesses into the log in system directly. Firstly, the attacker only needs to find a password that generates any of the hash codes in the entire password file in order to gain entry[[5]](#footnote-5), as opposed to having to find a password that generates the hash code required to log into a specific account. Indeed it is possible to discern approximately 10% of the passwords on a system in only 1010 tries through a guided search[[6]](#footnote-6). Secondly, as the attack is being performed on the attacker’s computer, completely separate to the system itself, it is impossible for the system to artificially limit the number of attempts made in a given length of time[[7]](#footnote-7). The number of possibilities that can be checked is limited only by the processing power of the attacker’s machine, and the speed of the hash function itself. For this reason, hash functions must be slow, as this reduces the number of attempts an attacker can make, which at the very least, slows them down[[8]](#footnote-8). It is important to note that this slowness must be from the operations performed by the hash function, and not from padding, as an attacker would be able to identify and remove padding.

# sdbm Hash Function

Pseudo code derived from C code[[9]](#footnote-9).

BEGIN sdbm

hash = 0

FOR char IN str

ls6 = LEFT SHIFT hash BY 6

ls16 = LEFT SHIFT hash BY 16

hash = char + ls6 + ls16 - hash

ENDWHILE

RETURN hash

END sdbm

hash is the hash code itself. It is an unsigned long. It is unsigned as the algorithm performs bit shifts on it in order to scramble it, were it permitted to possess negative values, the shifts it may result in a negative value that would have to be accounted for, which would complicate the logic significantly. As the hash code is only used for equality comparisons, restricting it to non-negative numbers does not impact its performance at all.

As the contents of all variables are stored as binary, they can be interpreted as a value of a differing type through casting. char is casted as this does not alter its value, it merely interprets it as an integer instead as the algorithm derives the hash code through arithmetic operations, which means that directly processing a character type variable is infeasible.

Bit shifting is performed on the hash code in order to vary the bits with each character, thereby ensuring that strings with the same characters in differing order do not have hash collisions with each other. The actual sizes of each shift are arbitrary, and were chosen because they resulted in the algorithm performing better. The subtraction of the previous iteration’s hash code serves to cause further variation.

1. Kashyap, D. *A Meaningful MD5 Hash Collision Attack* (San Jose: SJSU Scholar Works, 2006) [↑](#footnote-ref-1)
2. "Network Security" Princeton University, accessed October 2, 2015, http://www.cs.princeton.edu/courses/archive/spr11/cos461/docs/rec08-net-security.pdf [↑](#footnote-ref-2)
3. Menezes, A. J., and Paul C. Oorschot. *Handbook of Applied Cryptography*. (Boca Raton: CRC Press, 1997) [↑](#footnote-ref-3)
4. Halderman, J., Waters, B., and Felten, E. *A Convenient Method for Securely Managing Passwords* (Chiba: International World Wide Web Conference Committee, 2005) [↑](#footnote-ref-4)
5. Stallings, W., and Brown, L. *Computer Security Principles and Practice* (Essex: Pearson Education, 2015) [↑](#footnote-ref-5)
6. Mazurek, M., Komanduri, S., Vidas, T., Bauer, L., Christin, N., Cranor, L., Kelley, P., Shay, R., and Ur, B. *Measuring Password Guessability for an Entire University* (Pittsburgh: Carnegie Mellon University, 2013) [↑](#footnote-ref-6)
7. Pfleeger, C., Plfeeger, S., and Margulies, J. *Security in Computing* (New Jersey: Pearson Education, 2015) [↑](#footnote-ref-7)
8. Halderman, J., Waters, B., and Felten, E. *A Convenient Method for Securely Managing Passwords* (Chiba: International World Wide Web Conference Committee, 2005) [↑](#footnote-ref-8)
9. “Hash Functions” York University, accessed October 2, 2015, http://www.cse.yorku.ca/~oz/hash.html [↑](#footnote-ref-9)