SitePassword Modulo Bias

Alan H. Karp

July 15, 2024

Abstract

SitePassword hashes a super password, a per site nickname, and a per site userid to produce a site password. The bytes produced by the hash need to be converted to a string of characters selected from the alphabet supported by the web site associated with that user account. In some situations, *modulo bias* reduces the entropy of the site password. This note shows that the effect is small enough to ignore for the range of values SitePassword supports.

1 Introduction

The original SitePassword algorithm used each byte of the hash produced by PBKDF2[1] to compute the i-th character of the site password using $c_i = A[b_i \bmod |A|)]$, where A is an array of characters representing the alphabet, b_i is the byte value, and $|A|$ is the size of the alphabet. If the size of the alphabet evenly divides $2^8 = 256$, every character in the alphabet has the same probability of being selected. If it doesn't, then some characters have a higher probability than others. This *modulo bias* reduces the entropy of the resulting string. This document shows that modulo bias does not have a substantial effect on the guessability of the site password it generates.

2 Recommended Approach

The recommended approach to eliminating modulo bias is *rejection sampling.* Simply ignore any byte value index outside the range of the alphabet. In other words, don't use any $b_i > |A|$. Theoretically, this approach could require an unbounded number of bytes, but the number of bytes needed is not too large in practice.

Unfortunately, while rejection sample makes it harder to guess the site password, it makes it easier to guess your super password. Because the alphabet size is much smaller than the range of values in a byte, SitePassword will need to compute a lot of bytes in order to have a high probability of constructing a password. In particular, the expected number of bytes from the hash needed to select a single character is $E(p) = log(p)/log(1 - |A|/R)$, where p is the desired probability of success. For

[1]Password Based Key Derivation Function

SitePassword's default parameters roughly 3/4 of the byte values will be rejected. As a result, about 30 bytes/character are needed to be able to construct a site password 99.99% of the time. Even that success rate is too low for a tool like SitePassword.

SitePassword must compute all the bytes that might be needed, which reduces the number of iterations that SitePassword can do and still meet the latency requirement. The adversary can be more conservative, gaining as much as a 10x advantage in compute time when guessing your super password. While site passwords that need more bytes won't be useful in guessing your master password, most will.

3 Quantifying Modulo Bias

There will be no need to use extra bytes to construct the site password if modulo bias doesn't weaken the site password very much. One way to quantify that weakening is by looking at the entropy of the site password both with and without modulo bias. Without bias, the entropy of an L-character string picked at random from a uniform distribution is

$$E_0 = L \log_2 |A| \tag{1}$$

The easiest way to understand modulo bias is to repeat the alphabet until the result is has more elements than the largest possible random value R and truncate to size R. Some numbers may appear one more time than others. In those cases, the number of times a character appears is either

$$n_1 = \lfloor R/|A| \rfloor \quad \text{or} \quad n_2 = \lceil R/|A| \rceil, \tag{2}$$

so that $n_1 = n_2$ if $|A|$ divides R or $n_2 = n_1 + 1$ otherwise. The probabilities that one of those characters is chosen is,

$$p_1 = n_1/R \quad \text{or} \quad p_2 = n_2/R. \tag{3}$$

The entropy of selecting a single character from each subset is then

$$E_1 = -(|A| - R \bmod |A|])p_1 \log_2 p_1 \quad \text{or} \quad E_2 = -(R \bmod |A|)p_2 \log_2 p_2. \tag{4}$$

Here (R mod $|A|$) is the number of characters that appear an extra time. Note that if $|A|$ divides R, $E_2 = 0$ and $E_1 = E_0$. The entropy of an L-character site password with modulo bias is $E_b = L(E_1 + E_2)$.

We can now plug in some numbers. SitePassword uses $R = 256$ and allows alphabets in the range $10 \leq |A| \leq 96$. With the default values of $|A| = 62$ and $L = 12$, $E_0 - E_b = .05$ which requires 96% of the 2^{71} guesses needed if there was no modulo bias. The worst case in the allowed range is at $|A| = 96$ where $E_0 - E_b = 0.28$, which translates into 82% as many guesses. The greater loss in guesses is offset by the larger alphabet size leaving the entropy with modulo bias greater than 78 bits. Figure 1 shows the entropy loss, $E_0 - E_b$, in green and the ratio of the number of guesses needed, $2^{-(E_0 - E_b)}$, in purple for $R = 256$ and $L = 12$.

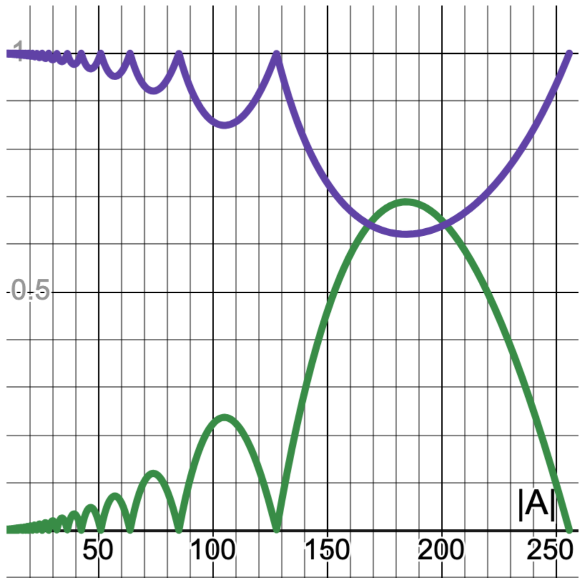

Figure 1: Modulo bias for SitePassword for an L-character site password. The green line is the entropy loss; the purple line the reduction in the number of guesses. Only values of $10 \leq |A| \leq 96$ are relevant to SitePassword.

4 Conclusions

There are cases where a small modulo bias can be a serious problem, such as in choosing a nonce for a stream cypher. That is not the case for SitePassword. Even the most extreme modulo bias reduces the entropy of the site password by less than one bit. Even so, the SitePassword algorithm was changed to use 2 bytes per character, which makes the modulo bias considerably smaller.