Mathematics of the SHA-1 Algorithm

Brendan Schlaman

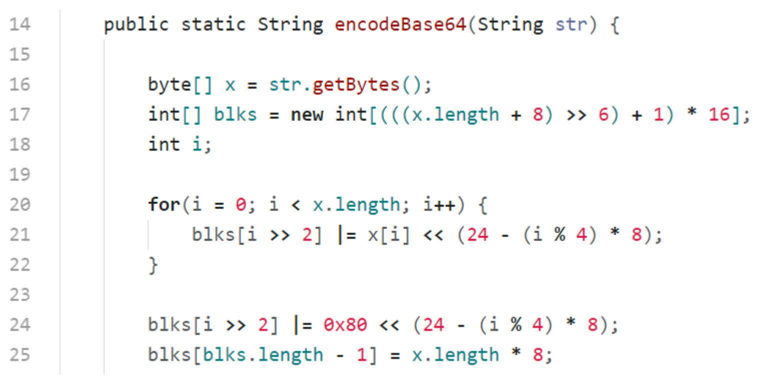
December 2018

Abstract

The purpose of this document is to explain the SHA-1 Hashing Algorithm and it's security to someone with a moderate understanding of mathematics and computer science. The code is first stepped through, and then the security of the algorithm is evaluated. The version of the source code examined here was converted to Java by Russell Beattie and Sam Ruby, and can be found at
github.com/opendatakit/javarosa/blob/master/src/org/javarosa/core/util/SHA1.java

1 SHA-1 Algorithm Stepthrough

1.1 Message is converted to bytes



```
14  public static String encodeBase64(String str) {
15
16      byte[] x = str.getBytes();
17      int[] blks = new int[(((x.length + 8) >> 6) + 1) * 16];
18      int i;
19
20      for(i = 0; i < x.length; i++) {
21          blks[i >> 2] |= x[i] << (24 - (i % 4) * 8);
22      }
23
24      blks[i >> 2] |= 0x80 << (24 - (i % 4) * 8);
25      blks[blks.length - 1] = x.length * 8;
```

Line 61 first makes an array of bytes. This array is the length of the original string (in characters), where each character is represented by a 7 bit number, following the ASCII standard. For example,

$$"A" = 66 = 0b1000010$$

1.2 Message is padded

SHA1 uses particular padding standards in order to ensure that the final message length is a multiple of 512 bits, e.g. that $message.length = 512k, \{k \in \mathbb{N}\}$.
The following standard is used:

1. The bit 1 is added to the end of the message.
2. A 64 bit representation of the size of the message is generated.
   (NOTE: the Java Program that we are following uses a 32 bit representation. This means that it can only handle messages of length at most $2^{32} - 1$ bits).
3. An amount of 0 bits are added in between the 1 and the 64 bit size representation such that the final length is a multiple of 512 bits.

The message then takes the form:

$$\textit{message} \mid 1 \mid k \text{ 0 bits } (0 \le k < 512) \mid \text{64 bit length representation}$$

Example message lengths and resultant padding:

440 bits	1	0000000	64 bit length representation
448 bits	1	0 x 511	64 bit length representation
448 bits	1	0 x 511	64 bit length representation

2 BC Security Analysis

As far as we know, it is impossible to undo a hashing algorithm, even when you are able to see the source code. In fact, all major hashing algorithms are completely open source (this is done purposefully to show that there is nothing magic going on "behind the scenes"). However, the impossibility of reverse-engineering a hash is not proven; we're just *really* sure it's true. So sure, in fact, that banks secure your transactions with it.

This reduces the type of attack that can be made on hash encrypted data to some variant of brute force: by finding two different input strings that yields the same hash. Since the output of SHA1 is 160 bits long, one might assume that 2^160 "guesses" are expected before a collision, but the real number is actually much lower, on the order of 2^80 (luckily still far beyond the capabilities of all computers on earth combined). The reason for this is commonly referred to the *Birthday Problem*. The problem is as follows: *Given n people in a room, what is the probability that two or more will share birthdays?* Or, stated in our terms: *How many randomly chosen days are expected before a "birthday collision?"*

We start with an equation to represent the probability that k people will have *distinct* birthdays (given a perfectly random distribution of possible birthdays). We will refer to a new person entering the room as a "draw".

$$P_{kdistinctdraws} = \frac{365}{365} \times \frac{365}{365} \times \frac{365}{365} \times \frac{365}{365} \times \frac{365}{365} \dots = \frac{k!}{k^n(k-n)!}$$

where n is the number of options (in this case, 365. In the case of SHA1, 2^160). The opposite of k distinct birthdays is at least one collision after n birthdays, i.e:

$$P_{collision} = 1 - P_{kdistinctdraws} = 1 - \frac{n!}{n^k(n-k)!}$$

This is all well and good, but we are looking for *the expected number of birthdays before a collision.* Before we get there, we will momentarily take a side step to develop an equation for expected value that is useful to us. Recall from probability theory:

$$\langle x \rangle = \sum_i x_i P_i$$

If we restrict the domain of x to non-negative integers,

$$\langle x \rangle = \sum_{k=1}^{\infty} nP(x = k)$$

$$= P(x = 1) + 2P(x = 2) + 3P(x = 3) + 4P(x = 4) + \dots$$

When we expand this sum vertically, we notice:

$$\begin{aligned}
P(x > 0) &= P(x = 1) + P(x = 2) + P(x = 3) + P(x = 4) + \dots \\
P(x > 1) &= \qquad\qquad\quad P(x = 2) + P(x = 3) + P(x = 4) + \dots \\
P(x > 2) &= \qquad\qquad\qquad\qquad\qquad\; P(x = 3) + P(x = 4) + \dots \\
&\vdots \qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\quad \vdots \\
P(x > k) &= \qquad\qquad\qquad\qquad\qquad\qquad\quad P(x = k+1) + \dots
\end{aligned}$$

Hence,

$$\langle x \rangle = \sum_{k=0}^{\infty} P(x > k)$$

Since k is an integer, $P(x > 0) = P(x \geq 1)$, so we write

$$\langle x \rangle = \sum_{k=1}^{\infty} P(x \geq k)$$

At this point, it is helpful to translate this new probability expression to English. $P(x \geq k)$ *is the probability that number of draws before a collision is greater than or equal to the k (or current) draw. That is, a collision hasn't happened yet.* Let's work with this expression a bit.

$$P(x \geq k) = \frac{n!}{n^k(n-k)!} = \prod_{j=0}^{k-1}(1 - \frac{j}{n}) = P_{k\ distinct\ draws}$$

Borrowing from calculus, we know that $e^y \approx 1 + y, \{|y| \ll 1\}$ where, in our case, $y = -\frac{j}{n}$. Using some basic algebra:

$$\prod_{j=0}^{k-1}(1 - \frac{j}{n}) = \prod_{j=0}^{k-1} e^{-\frac{j}{n}} = e^{-(\frac{1+2+3+\ldots+(k-1)}{n})} = e^{-(\frac{k(k-1)}{2n})}$$

Since in our problem $k \ll n$, $k-1 \simeq k$. So therefore, we have

$$P(x \geq k) \approx e^{-(\frac{k^2}{2n})}$$

In fact, $a = e^{-(\frac{k(k-1)}{2n})}$ overestimates the probability, and $e^{-(\frac{k^2}{2n})}$ underestimates a. This will do well for our birthday problem, but let's now turn to much larger n, as is the case with hashing. For large n:

$$\langle x \rangle \approx \int_0^n e^{-(\frac{t^2}{2n})} dt$$

NOTE: We are integrating from 0 instead of 1 (like in the sum) because it will make the integral easier to solve. We are overestimating by the term $\int_0^1 e^{-(\frac{t^2}{2n})} dt \approx 1$, which is negligible at this scale. Substituting $\frac{t^2}{2n} = s^2$,

$$\begin{aligned}
\langle x \rangle &\approx \sqrt{2n} \int_0^{\frac{n}{\sqrt{2n}}} e^{-s^2} ds \\
&= \sqrt{2n} \left[\int_0^{\infty} e^{-s^2} ds - \int_{\sqrt{\frac{n}{2}}}^{\infty} e^{-s^2} ds \right] \\
&\approx \sqrt{2n} \left[\frac{\sqrt{\pi}}{2} - 0 \right] \\
&= \sqrt{\frac{n\pi}{2}} \\
&\approx 1.25\sqrt{n}
\end{aligned}$$

Finally, we have our answer. So... how did we do? Let's try it for the birthday problem. $1.25\sqrt{365} \approx 23.9$. Subtracting off the 1 that we overestimated by, we get 22.9 (or 22 full people expected before a collision), which is the correct (and surprising) answer! That means that given 23 people with a random distribution of birthdays, the probability is *greater than 50%* that two or more people will share birthdays. This also means that for a 160 bit long hash, the expected number of input guesses before finding a collision is $1.25\sqrt{2^{160}} \approx 2^{80}$. And in fact, in February 2017, a team of researches did just that! Using some clever tricks, they were able to achieve a collision very early, with only around 2^{64} guesses.

So what about *my* hashing algorithm? The result is only 12 bits long. This means that one can expect to find a collision with $1.25\sqrt{2^{12}} = 80$ guesses! Not very secure indeed!