Transferable State Attack on Iterated Hashing Functions

July 26, 2012

Brian Wallace

Ballast Security

Abstract

In this paper I will describe an attack on the iterated use of hashing functions used as

key stretching algorithms where the state of a hash can be transferred to the next hash function.

Transferable State Attack on Iterated Hashing Functions

This attack was developed while developing a tool to derive the master password of the Password Safe application. This application uses the key stretching algorithm defined by section 4.1 of the paper Secure Applications of Low-Entropy Keys by John Kelsey, Bruce Schneier, Chris Hall, and David Wagner with SHA-256 as the hashing function. There has been some question as to whether this is an attack on the algorithm they defined or on the SHA-256 algorithm.

The Algorithm

For those unfamiliar with this algorithm, I will supply an overview of the algorithm. Its purpose is to turn a string of bytes that by themselves are a weak cryptographic key, combine them with a salt, then run them through a hashing function together. After that, the results of that are put through a hashing function in an iterative manner.

This algorithm establishes an increased computation time for the result, making it useful in situations where the key derivation process needs to lengthened. These situations include password based encryption such as in the Password Safe application.

The General Weakness

For those unfamiliar with this algorithm, I will supply an overview of the algorithm. Its purpose is to turn a string of bytes that by themselves are a weak cryptographic key, combine them with a salt, then run them through a hashing function together. After that, the results of that are put through a hashing function in an iterative manner.

This algorithm establishes an increased computation time for the result, making it useful in situations where the key derivation process needs to lengthened. These situations include password based encryption such as in the Password Safe application.

Weakness in Iterated SHA-256

In the application Password Safe, which is supported by Bruce Schneier, this algorithm is used with SHA-256. I had begun devising the attack before knowing that this algorithm was created by someone so highly respected. It wasn't until I tried benchmarking my attack against the Password Safe cracking implementation in the Jumbo patch of John the Ripper that I realized I had broken new ground in attacking these hashes.

Using a single hash and a dictionary attack with John the Ripper, it computed 1463 hashes per second(Password Safe hash with 2048 iterations of SHA-256) while my implementation computed 1611.53 hashes per second. That is 90.78% of the computation time. The optimization is even more prominent as higher iteration counts are used.

The optimization used here is as I had described above for the general weakness. The state is reused in the next iteration from the previous iteration allowing for seamless integration of iterations, then ending with a comparison to the "desired state", which is the final hash partially reversed to its states. I am continuing to add to these optimizations, as this weakness opens up lots of room for different optimizations.

Implementation

Normally when implementing an iterated hash function, you would do something like this.

```
SHA256_t sha;
sha.Update((unsigned char *)"password", 8);
sha.Update((unsigned char *)"salt", 4);
sha.Finalize(hash);
for(int x = 0; x < iterations; x++)
{
      SHA256_t sha1;
      sha1.Update(hash, 32);
      sha1.Finalize(hash);
}
```

But in my implementation, it gets simplified a bit from the outside.

```
      SHA256_t sha;
      sha.Update((unsigned char *)"password", 8);
      sha.Update((unsigned char *)"test", 4);
      sha.IterativeFinalize(hash, iterations);
```

Its under the hood that it starts to get more intricate. It does hash the first hash completely except that it never turns it into an actual hash, but instead uses the state in a call from the IterativeFinalize function. Normally, most of the actual work in a hashing function is done in its "transform" section. Here is the implementation of SHA256's transform function that I'm using.

```
inline void SHA256_t::sha256_transform(uint *state, uint * data)
{
     uint word00, word01, word02, word03, word04, word05, word06,
word07;
      uint word08, word09, word10, word11, word12, word13, word14,
word15;
      uint temp0,temp1,temp2,temp3,temp4,temp5,temp6,temp7;

      temp0 = state[0]; temp1 = state[1];
      temp2 = state[2]; temp3 = state[3];
      temp4 = state[4]; temp5 = state[5];
      temp6 = state[6]; temp7 = state[7];

      //First Iteration
      temp7 += ROTXOR2( temp4 ) + CHOICE( temp4, temp5, temp6 ) +
0x428a2f98 + ( (word00 = data[0]) );
      temp3 += temp7;
      temp7 += ROTXOR1( temp0 ) + MAJORITY( temp0, temp1, temp2 );

      temp6 += ROTXOR2( temp3 ) + CHOICE( temp3, temp4, temp5 ) +
0x71374491 + ( (word01 = data[1]) );
      temp2 += temp6;
      temp6 += ROTXOR1( temp7 ) + MAJORITY( temp7, temp0, temp1 );

      temp5 += ROTXOR2( temp2 ) + CHOICE( temp2, temp3, temp4 ) +
0xb5c0fbcf + ( (word02 = data[2]) );
      temp1 += temp5;
      temp5 += ROTXOR1( temp6 ) + MAJORITY( temp6, temp7, temp0 );

      temp4 += ROTXOR2( temp1 ) + CHOICE( temp1, temp2, temp3 ) +
0xe9b5dba5 + ( (word03 = data[3]) );
      temp0 += temp4;
```

```
      temp4 += ROTXOR1( temp5 ) + MAJORITY( temp5, temp6, temp7 );

      temp3 += ROTXOR2( temp0 ) + CHOICE( temp0, temp1, temp2 ) +
0x3956c25b + ( (word04 = data[4]) );
      temp7 += temp3;
      temp3 += ROTXOR1( temp4 ) + MAJORITY( temp4, temp5, temp6 );

      temp2 += ROTXOR2( temp7 ) + CHOICE( temp7, temp0, temp1 ) +
0x59f111f1 + ( (word05 = data[5]) );
      temp6 += temp2;
      temp2 += ROTXOR1( temp3 ) + MAJORITY( temp3, temp4, temp5 );

      temp1 += ROTXOR2( temp6 ) + CHOICE( temp6, temp7, temp0 ) +
0x923f82a4 + ( (word06 = data[6]) );
      temp5 += temp1;
      temp1 += ROTXOR1( temp2 ) + MAJORITY( temp2, temp3, temp4 );

      temp0 += ROTXOR2( temp5 ) + CHOICE( temp5, temp6, temp7 ) +
0xab1c5ed5 + ( (word07 = data[7]) );
      temp4 += temp0;
      temp0 += ROTXOR1( temp1 ) + MAJORITY( temp1, temp2, temp3 );

      temp7 += ROTXOR2( temp4 ) + CHOICE( temp4, temp5, temp6 ) +
0xd807aa98 + ( (word08 = data[8]) );
      temp3 += temp7;
      temp7 += ROTXOR1( temp0 ) + MAJORITY( temp0, temp1, temp2 );

      temp6 += ROTXOR2( temp3 ) + CHOICE( temp3, temp4, temp5 ) +
0x12835b01 + ( (word09 = data[9]) );
      temp2 += temp6;
      temp6 += ROTXOR1( temp7 ) + MAJORITY( temp7, temp0, temp1 );

      temp5 += ROTXOR2( temp2 ) + CHOICE( temp2, temp3, temp4 ) +
0x243185be + ( (word10 = data[10]) );
      temp1 += temp5;
      temp5 += ROTXOR1( temp6 ) + MAJORITY( temp6, temp7, temp0 );

      temp4 += ROTXOR2( temp1 ) + CHOICE( temp1, temp2, temp3 ) +
0x550c7dc3 + ( (word11 = data[11]) );
      temp0 += temp4;
      temp4 += ROTXOR1( temp5 ) + MAJORITY( temp5, temp6, temp7 );

      temp3 += ROTXOR2( temp0 ) + CHOICE( temp0, temp1, temp2 ) +
0x72be5d74 + ( (word12 = data[12]) );
      temp7 += temp3;
      temp3 += ROTXOR1( temp4 ) + MAJORITY( temp4, temp5, temp6 );

      temp2 += ROTXOR2( temp7 ) + CHOICE( temp7, temp0, temp1 ) +
0x80deb1fe + ( (word13 = data[13]) );
      temp6 += temp2;
      temp2 += ROTXOR1( temp3 ) + MAJORITY( temp3, temp4, temp5 );
```

```
      temp1 += ROTXOR2( temp6 ) + CHOICE( temp6, temp7, temp0 ) +
0x9bdc06a7 + ( (word14 = data[14]) );
      temp5 += temp1;
      temp1 += ROTXOR1( temp2 ) + MAJORITY( temp2, temp3, temp4 );

      temp0 += ROTXOR2( temp5 ) + CHOICE( temp5, temp6, temp7 ) +
0xc19bf174 + ( (word15 = data[15]) );
      temp4 += temp0;
      temp0 += ROTXOR1( temp1 ) + MAJORITY( temp1, temp2, temp3 );



      temp7 += ROTXOR2( temp4 ) + CHOICE( temp4, temp5, temp6 ) +
0xe49b69c1 + ( (word00 += ROTXOR4( word14 ) + word09 + ROTXOR3(
word01 ) ) );
      temp3 += temp7;
      temp7 += ROTXOR1( temp0 ) + MAJORITY( temp0, temp1, temp2 );

      temp6 += ROTXOR2( temp3 ) + CHOICE( temp3, temp4, temp5 ) +
0xefbe4786 + ( (word01 += ROTXOR4( word15 ) + word10 + ROTXOR3(
word02 ) ) );
      temp2 += temp6;
      temp6 += ROTXOR1( temp7 ) + MAJORITY( temp7, temp0, temp1 );

      temp5 += ROTXOR2( temp2 ) + CHOICE( temp2, temp3, temp4 ) +
0x0fc19dc6 + ( (word02 += ROTXOR4( word00 ) + word11 + ROTXOR3(
word03 ) ) );
      temp1 += temp5;
      temp5 += ROTXOR1( temp6 ) + MAJORITY( temp6, temp7, temp0 );

      temp4 += ROTXOR2( temp1 ) + CHOICE( temp1, temp2, temp3 ) +
0x240ca1cc + ( (word03 += ROTXOR4( word01 ) + word12 + ROTXOR3(
word04 ) ) );
      temp0 += temp4;
      temp4 += ROTXOR1( temp5 ) + MAJORITY( temp5, temp6, temp7 );

      temp3 += ROTXOR2( temp0 ) + CHOICE( temp0, temp1, temp2 ) +
0x2de92c6f + ( (word04 += ROTXOR4( word02 ) + word13 + ROTXOR3(
word05 ) ) );
      temp7 += temp3;
      temp3 += ROTXOR1( temp4 ) + MAJORITY( temp4, temp5, temp6 );

      temp2 += ROTXOR2( temp7 ) + CHOICE( temp7, temp0, temp1 ) +
0x4a7484aa + ( (word05 += ROTXOR4( word03 ) + word14 + ROTXOR3(
word06 ) ) );
      temp6 += temp2;
      temp2 += ROTXOR1( temp3 ) + MAJORITY( temp3, temp4, temp5 );

      temp1 += ROTXOR2( temp6 ) + CHOICE( temp6, temp7, temp0 ) +
0x5cb0a9dc + ( (word06 += ROTXOR4( word04 ) + word15 + ROTXOR3(
```

```
word07 ) ) );
      temp5 += temp1;
      temp1 += ROTXOR1( temp2 ) + MAJORITY( temp2, temp3, temp4 );

      temp0 += ROTXOR2( temp5 ) + CHOICE( temp5, temp6, temp7 ) +
0x76f988da + ( (word07 += ROTXOR4( word05 ) + word00 + ROTXOR3(
word08 ) ) );
      temp4 += temp0;
      temp0 += ROTXOR1( temp1 ) + MAJORITY( temp1, temp2, temp3 );

      temp7 += ROTXOR2( temp4 ) + CHOICE( temp4, temp5, temp6 ) +
0x983e5152 + ( (word08 += ROTXOR4( word06 ) + word01 + ROTXOR3(
word09 ) ) );
      temp3 += temp7;
      temp7 += ROTXOR1( temp0 ) + MAJORITY( temp0, temp1, temp2 );

      temp6 += ROTXOR2( temp3 ) + CHOICE( temp3, temp4, temp5 ) +
0xa831c66d + ( (word09 += ROTXOR4( word07 ) + word02 + ROTXOR3(
word10 ) ) );
      temp2 += temp6;
      temp6 += ROTXOR1( temp7 ) + MAJORITY( temp7, temp0, temp1 );

      temp5 += ROTXOR2( temp2 ) + CHOICE( temp2, temp3, temp4 ) +
0xb00327c8 + ( (word10 += ROTXOR4( word08 ) + word03 + ROTXOR3(
word11 ) ) );
      temp1 += temp5;
      temp5 += ROTXOR1( temp6 ) + MAJORITY( temp6, temp7, temp0 );

      temp4 += ROTXOR2( temp1 ) + CHOICE( temp1, temp2, temp3 ) +
0xbf597fc7 + ( (word11 += ROTXOR4( word09 ) + word04 + ROTXOR3(
word12 ) ) );
      temp0 += temp4;
      temp4 += ROTXOR1( temp5 ) + MAJORITY( temp5, temp6, temp7 );

      temp3 += ROTXOR2( temp0 ) + CHOICE( temp0, temp1, temp2 ) +
0xc6e00bf3 + ( (word12 += ROTXOR4( word10 ) + word05 + ROTXOR3(
word13 ) ) );
      temp7 += temp3;
      temp3 += ROTXOR1( temp4 ) + MAJORITY( temp4, temp5, temp6 );

      temp2 += ROTXOR2( temp7 ) + CHOICE( temp7, temp0, temp1 ) +
0xd5a79147 + ( (word13 += ROTXOR4( word11 ) + word06 + ROTXOR3(
word14 ) ) );
      temp6 += temp2;
      temp2 += ROTXOR1( temp3 ) + MAJORITY( temp3, temp4, temp5 );

      temp1 += ROTXOR2( temp6 ) + CHOICE( temp6, temp7, temp0 ) +
0x06ca6351 + ( (word14 += ROTXOR4( word12 ) + word07 + ROTXOR3(
word15 ) ) );
      temp5 += temp1;
      temp1 += ROTXOR1( temp2 ) + MAJORITY( temp2, temp3, temp4 );
```

```
     temp0 += ROTXOR2( temp5 ) + CHOICE( temp5, temp6, temp7 ) +
0x14292967 + ( (word15 += ROTXOR4( word13 ) + word08 + ROTXOR3(
word00 ) ) );
     temp4 += temp0;
     temp0 += ROTXOR1( temp1 ) + MAJORITY( temp1, temp2, temp3 );




     temp7 += ROTXOR2( temp4 ) + CHOICE( temp4, temp5, temp6 ) +
0x27b70a85 + ( (word00 += ROTXOR4( word14 ) + word09 + ROTXOR3(
word01 ) ) );
     temp3 += temp7;
     temp7 += ROTXOR1( temp0 ) + MAJORITY( temp0, temp1, temp2 );

     temp6 += ROTXOR2( temp3 ) + CHOICE( temp3, temp4, temp5 ) +
0x2e1b2138 + ( (word01 += ROTXOR4( word15 ) + word10 + ROTXOR3(
word02 ) ) );
     temp2 += temp6;
     temp6 += ROTXOR1( temp7 ) + MAJORITY( temp7, temp0, temp1 );

     temp5 += ROTXOR2( temp2 ) + CHOICE( temp2, temp3, temp4 ) +
0x4d2c6dfc + ( (word02 += ROTXOR4( word00 ) + word11 + ROTXOR3(
word03 ) ) );
     temp1 += temp5;
     temp5 += ROTXOR1( temp6 ) + MAJORITY( temp6, temp7, temp0 );

     temp4 += ROTXOR2( temp1 ) + CHOICE( temp1, temp2, temp3 ) +
0x53380d13 + ( (word03 += ROTXOR4( word01 ) + word12 + ROTXOR3(
word04 ) ) );
     temp0 += temp4;
     temp4 += ROTXOR1( temp5 ) + MAJORITY( temp5, temp6, temp7 );

     temp3 += ROTXOR2( temp0 ) + CHOICE( temp0, temp1, temp2 ) +
0x650a7354 + ( (word04 += ROTXOR4( word02 ) + word13 + ROTXOR3(
word05 ) ) );
     temp7 += temp3;
     temp3 += ROTXOR1( temp4 ) + MAJORITY( temp4, temp5, temp6 );

     temp2 += ROTXOR2( temp7 ) + CHOICE( temp7, temp0, temp1 ) +
0x766a0abb + ( (word05 += ROTXOR4( word03 ) + word14 + ROTXOR3(
word06 ) ) );
     temp6 += temp2;
     temp2 += ROTXOR1( temp3 ) + MAJORITY( temp3, temp4, temp5 );

     temp1 += ROTXOR2( temp6 ) + CHOICE( temp6, temp7, temp0 ) +
0x81c2c92e + ( (word06 += ROTXOR4( word04 ) + word15 + ROTXOR3(
word07 ) ) );
     temp5 += temp1;
     temp1 += ROTXOR1( temp2 ) + MAJORITY( temp2, temp3, temp4 );
```

```
     temp0 += ROTXOR2( temp5 ) + CHOICE( temp5, temp6, temp7 ) +
0x92722c85 + ( (word07 += ROTXOR4( word05 ) + word00 + ROTXOR3(
word08 ) ) );
     temp4 += temp0;
     temp0 += ROTXOR1( temp1 ) + MAJORITY( temp1, temp2, temp3 );

     temp7 += ROTXOR2( temp4 ) + CHOICE( temp4, temp5, temp6 ) +
0xa2bfe8a1 + ( (word08 += ROTXOR4( word06 ) + word01 + ROTXOR3(
word09 ) ) );
     temp3 += temp7;
     temp7 += ROTXOR1( temp0 ) + MAJORITY( temp0, temp1, temp2 );

     temp6 += ROTXOR2( temp3 ) + CHOICE( temp3, temp4, temp5 ) +
0xa81a664b + ( (word09 += ROTXOR4( word07 ) + word02 + ROTXOR3(
word10 ) ) );
     temp2 += temp6;
     temp6 += ROTXOR1( temp7 ) + MAJORITY( temp7, temp0, temp1 );

     temp5 += ROTXOR2( temp2 ) + CHOICE( temp2, temp3, temp4 ) +
0xc24b8b70 + ( (word10 += ROTXOR4( word08 ) + word03 + ROTXOR3(
word11 ) ) );
     temp1 += temp5;
     temp5 += ROTXOR1( temp6 ) + MAJORITY( temp6, temp7, temp0 );

     temp4 += ROTXOR2( temp1 ) + CHOICE( temp1, temp2, temp3 ) +
0xc76c51a3 + ( (word11 += ROTXOR4( word09 ) + word04 + ROTXOR3(
word12 ) ) );
     temp0 += temp4;
     temp4 += ROTXOR1( temp5 ) + MAJORITY( temp5, temp6, temp7 );

     temp3 += ROTXOR2( temp0 ) + CHOICE( temp0, temp1, temp2 ) +
0xd192e819 + ( (word12 += ROTXOR4( word10 ) + word05 + ROTXOR3(
word13 ) ) );
     temp7 += temp3;
     temp3 += ROTXOR1( temp4 ) + MAJORITY( temp4, temp5, temp6 );

     temp2 += ROTXOR2( temp7 ) + CHOICE( temp7, temp0, temp1 ) +
0xd6990624 + ( (word13 += ROTXOR4( word11 ) + word06 + ROTXOR3(
word14 ) ) );
     temp6 += temp2;
     temp2 += ROTXOR1( temp3 ) + MAJORITY( temp3, temp4, temp5 );

     temp1 += ROTXOR2( temp6 ) + CHOICE( temp6, temp7, temp0 ) +
0xf40e3585 + ( (word14 += ROTXOR4( word12 ) + word07 + ROTXOR3(
word15 ) ) );
     temp5 += temp1;
     temp1 += ROTXOR1( temp2 ) + MAJORITY( temp2, temp3, temp4 );

     temp0 += ROTXOR2( temp5 ) + CHOICE( temp5, temp6, temp7 ) +
0x106aa070 + ( (word15 += ROTXOR4( word13 ) + word08 + ROTXOR3(
```

```
word00 ) ) );
      temp4 += temp0;
      temp0 += ROTXOR1( temp1 ) + MAJORITY( temp1, temp2, temp3 );




      temp7 += ROTXOR2( temp4 ) + CHOICE( temp4, temp5, temp6 ) +
0x19a4c116 + ( (word00 += ROTXOR4( word14 ) + word09 + ROTXOR3(
word01 ) ) );
      temp3 += temp7;
      temp7 += ROTXOR1( temp0 ) + MAJORITY( temp0, temp1, temp2 );

      temp6 += ROTXOR2( temp3 ) + CHOICE( temp3, temp4, temp5 ) +
0x1e376c08 + ( (word01 += ROTXOR4( word15 ) + word10 + ROTXOR3(
word02 ) ) );
      temp2 += temp6;
      temp6 += ROTXOR1( temp7 ) + MAJORITY( temp7, temp0, temp1 );

      temp5 += ROTXOR2( temp2 ) + CHOICE( temp2, temp3, temp4 ) +
0x2748774c + ( (word02 += ROTXOR4( word00 ) + word11 + ROTXOR3(
word03 ) ) );
      temp1 += temp5;
      temp5 += ROTXOR1( temp6 ) + MAJORITY( temp6, temp7, temp0 );

      temp4 += ROTXOR2( temp1 ) + CHOICE( temp1, temp2, temp3 ) +
0x34b0bcb5 + ( (word03 += ROTXOR4( word01 ) + word12 + ROTXOR3(
word04 ) ) );
      temp0 += temp4;
      temp4 += ROTXOR1( temp5 ) + MAJORITY( temp5, temp6, temp7 );

      temp3 += ROTXOR2( temp0 ) + CHOICE( temp0, temp1, temp2 ) +
0x391c0cb3 + ( (word04 += ROTXOR4( word02 ) + word13 + ROTXOR3(
word05 ) ) );
      temp7 += temp3;
      temp3 += ROTXOR1( temp4 ) + MAJORITY( temp4, temp5, temp6 );

      temp2 += ROTXOR2( temp7 ) + CHOICE( temp7, temp0, temp1 ) +
0x4ed8aa4a + ( (word05 += ROTXOR4( word03 ) + word14 + ROTXOR3(
word06 ) ) );
      temp6 += temp2;
      temp2 += ROTXOR1( temp3 ) + MAJORITY( temp3, temp4, temp5 );

      temp1 += ROTXOR2( temp6 ) + CHOICE( temp6, temp7, temp0 ) +
0x5b9cca4f + ( (word06 += ROTXOR4( word04 ) + word15 + ROTXOR3(
word07 ) ) );
      temp5 += temp1;
      temp1 += ROTXOR1( temp2 ) + MAJORITY( temp2, temp3, temp4 );

      temp0 += ROTXOR2( temp5 ) + CHOICE( temp5, temp6, temp7 ) +
0x682e6ff3 + ( (word07 += ROTXOR4( word05 ) + word00 + ROTXOR3(
```

```
word08 ) ) );
      temp4 += temp0;
      temp0 += ROTXOR1( temp1 ) + MAJORITY( temp1, temp2, temp3 );

      temp7 += ROTXOR2( temp4 ) + CHOICE( temp4, temp5, temp6 ) +
0x748f82ee + ( (word08 += ROTXOR4( word06 ) + word01 + ROTXOR3(
word09 ) ) );
      temp3 += temp7;
      temp7 += ROTXOR1( temp0 ) + MAJORITY( temp0, temp1, temp2 );

      temp6 += ROTXOR2( temp3 ) + CHOICE( temp3, temp4, temp5 ) +
0x78a5636f + ( (word09 += ROTXOR4( word07 ) + word02 + ROTXOR3(
word10 ) ) );
      temp2 += temp6;
      temp6 += ROTXOR1( temp7 ) + MAJORITY( temp7, temp0, temp1 );

      temp5 += ROTXOR2( temp2 ) + CHOICE( temp2, temp3, temp4 ) +
0x84c87814 + ( (word10 += ROTXOR4( word08 ) + word03 + ROTXOR3(
word11 ) ) );
      temp1 += temp5;
      temp5 += ROTXOR1( temp6 ) + MAJORITY( temp6, temp7, temp0 );

      temp4 += ROTXOR2( temp1 ) + CHOICE( temp1, temp2, temp3 ) +
0x8cc70208 + ( (word11 += ROTXOR4( word09 ) + word04 + ROTXOR3(
word12 ) ) );
      temp0 += temp4;
      temp4 += ROTXOR1( temp5 ) + MAJORITY( temp5, temp6, temp7 );

      temp3 += ROTXOR2( temp0 ) + CHOICE( temp0, temp1, temp2 ) +
0x90befffa + ( (word12 += ROTXOR4( word10 ) + word05 + ROTXOR3(
word13 ) ) );
      temp7 += temp3;
      temp3 += ROTXOR1( temp4 ) + MAJORITY( temp4, temp5, temp6 );

      temp2 += ROTXOR2( temp7 ) + CHOICE( temp7, temp0, temp1 ) +
0xa4506ceb + ( (word13 += ROTXOR4( word11 ) + word06 + ROTXOR3(
word14 ) ) );
      temp6 += temp2;
      temp2 += ROTXOR1( temp3 ) + MAJORITY( temp3, temp4, temp5 );

      temp1 += ROTXOR2( temp6 ) + CHOICE( temp6, temp7, temp0 ) +
0xbef9a3f7 + ( (word14 += ROTXOR4( word12 ) + word07 + ROTXOR3(
word15 ) ) );
      temp5 += temp1;
      temp1 += ROTXOR1( temp2 ) + MAJORITY( temp2, temp3, temp4 );
      temp0 += ROTXOR2( temp5 ) + CHOICE( temp5, temp6, temp7 ) +
0xc67178f2 + ( (word15 += ROTXOR4( word13 ) + word08 + ROTXOR3(
word00 ) ) );
      temp4 += temp0;
      temp0 += ROTXOR1( temp1 ) + MAJORITY( temp1, temp2, temp3 );
```

```
      state[0] += temp0;
      state[1] += temp1;
      state[2] += temp2;
      state[3] += temp3;
      state[4] += temp4;
      state[5] += temp5;
      state[6] += temp6;
      state[7] += temp7;
}
```

Now for most developers, reading that is a bit daunting. Just to note, it includes a couple of functions that are actually left as #defines to speed things up.

Now here is one way to speed up hashing. This does not include the optimizations I made for cracking the hash(reversing of another hash and using the state of that hash for comparison. For that code, you can see it at https://github.com/bwall/SafeCracker/blob/master/CrackPwSafe/src/SHA256.cpp

```
inline void SHA256_t::sha256_transform_i(uint Iterations)
{
    uint word00,word01,word02,word03,word04,word05,word06,word07;
     uint word08,word09,word10,word11,word12,word13,word14,word15;
     uint temp0, temp1, temp2, temp3, temp4, temp5, temp6, temp7;

     for(uint counter = 0; counter < Iterations; counter++)
     {
           temp0 = 0x6a09e667UL;
           temp1 = 0xbb67ae85UL;
           temp2 = 0x3c6ef372UL;
           temp3 = 0xa54ff53aUL;
           temp4 = 0x510e527fUL;
           temp5 = 0x9b05688cUL;
           temp6 = 0x1f83d9abUL;
           temp7 = 0x5be0cd19UL;

           temp7 += ROTXOR2( temp4 ) + CHOICE( temp4, temp5, temp6 )
+ 0x428a2f98 + ( (word00 = state[0]) );
           temp3 += temp7;
           temp7 += ROTXOR1( temp0 ) + MAJORITY( temp0, temp1, temp2
);

           temp6 += ROTXOR2( temp3 ) + CHOICE( temp3, temp4, temp5 )
+ 0x71374491 + ( (word01 = state[1]) );
           temp2 += temp6;
```

```
            temp6 += ROTXOR1( temp7 ) + MAJORITY( temp7, temp0, temp1
);

            temp5 += ROTXOR2( temp2 ) + CHOICE( temp2, temp3, temp4 )
+ 0xb5c0fbcf + ( (word02 = state[2]) );
            temp1 += temp5;
            temp5 += ROTXOR1( temp6 ) + MAJORITY( temp6, temp7, temp0
);

            temp4 += ROTXOR2( temp1 ) + CHOICE( temp1, temp2, temp3 )
+ 0xe9b5dba5 + ( (word03 = state[3]) );
            temp0 += temp4;
            temp4 += ROTXOR1( temp5 ) + MAJORITY( temp5, temp6, temp7
);

            temp3 += ROTXOR2( temp0 ) + CHOICE( temp0, temp1, temp2 )
+ 0x3956c25b + ( (word04 = state[4]) );
            temp7 += temp3;
            temp3 += ROTXOR1( temp4 ) + MAJORITY( temp4, temp5, temp6
);

            temp2 += ROTXOR2( temp7 ) + CHOICE( temp7, temp0, temp1 )
+ 0x59f111f1 + ( (word05 = state[5]) );
            temp6 += temp2;
            temp2 += ROTXOR1( temp3 ) + MAJORITY( temp3, temp4, temp5
);

            temp1 += ROTXOR2( temp6 ) + CHOICE( temp6, temp7, temp0 )
+ 0x923f82a4 + ( (word06 = state[6]) );
            temp5 += temp1;
            temp1 += ROTXOR1( temp2 ) + MAJORITY( temp2, temp3, temp4
);

            temp0 += ROTXOR2( temp5 ) + CHOICE( temp5, temp6, temp7 )
+ 0xab1c5ed5 + ( (word07 = state[7]) );
            temp4 += temp0;
            temp0 += ROTXOR1( temp1 ) + MAJORITY( temp1, temp2, temp3
);

            temp7 += ROTXOR2( temp4 ) + CHOICE( temp4, temp5, temp6 )
+ 0xd807aa98 + ( (word08 = 0x80000000U) );
            temp3 += temp7;
            temp7 += ROTXOR1( temp0 ) + MAJORITY( temp0, temp1, temp2
);

            temp6 += ROTXOR2( temp3 ) + CHOICE( temp3, temp4, temp5 )
+ 0x12835b01 + ( (word09 = 0) );
            temp2 += temp6;
            temp6 += ROTXOR1( temp7 ) + MAJORITY( temp7, temp0, temp1
);
```

```
            temp5 += ROTXOR2( temp2 ) + CHOICE( temp2, temp3, temp4 )
+ 0x243185be + ( (word10 = 0) );
            temp1 += temp5;
            temp5 += ROTXOR1( temp6 ) + MAJORITY( temp6, temp7, temp0
);

            temp4 += ROTXOR2( temp1 ) + CHOICE( temp1, temp2, temp3 )
+ 0x550c7dc3 + ( (word11 = 0) );
            temp0 += temp4;
            temp4 += ROTXOR1( temp5 ) + MAJORITY( temp5, temp6, temp7
);

            temp3 += ROTXOR2( temp0 ) + CHOICE( temp0, temp1, temp2 )
+ 0x72be5d74 + ( (word12 = 0) );
            temp7 += temp3;
            temp3 += ROTXOR1( temp4 ) + MAJORITY( temp4, temp5, temp6
);

            temp2 += ROTXOR2( temp7 ) + CHOICE( temp7, temp0, temp1 )
+ 0x80deb1fe + ( (word13 = 0) );
            temp6 += temp2;
            temp2 += ROTXOR1( temp3 ) + MAJORITY( temp3, temp4, temp5
);

            temp1 += ROTXOR2( temp6 ) + CHOICE( temp6, temp7, temp0 )
+ 0x9bdc06a7 + ( (word14 = 0) );
            temp5 += temp1;
            temp1 += ROTXOR1( temp2 ) + MAJORITY( temp2, temp3, temp4
);

            temp0 += ROTXOR2( temp5 ) + CHOICE( temp5, temp6, temp7 )
+ 0xc19bf174 + ( (word15 = 256) );
            temp4 += temp0;
            temp0 += ROTXOR1( temp1 ) + MAJORITY( temp1, temp2, temp3
);



            temp7 += ROTXOR2( temp4 ) + CHOICE( temp4, temp5, temp6 )
+ 0xe49b69c1 + ( (word00 += ROTXOR4( word14 ) + word09 + ROTXOR3(
word01 ) ) );
            temp3 += temp7;
            temp7 += ROTXOR1( temp0 ) + MAJORITY( temp0, temp1, temp2
);

            temp6 += ROTXOR2( temp3 ) + CHOICE( temp3, temp4, temp5 )
+ 0xefbe4786 + ( (word01 += ROTXOR4( word15 ) + word10 + ROTXOR3(
word02 ) ) );
            temp2 += temp6;
            temp6 += ROTXOR1( temp7 ) + MAJORITY( temp7, temp0, temp1
);
```

```
            temp5 += ROTXOR2( temp2 ) + CHOICE( temp2, temp3, temp4 )
+ 0x0fc19dc6 + ( (word02 += ROTXOR4( word00 ) + word11 + ROTXOR3(
word03 ) ) );
            temp1 += temp5;
            temp5 += ROTXOR1( temp6 ) + MAJORITY( temp6, temp7, temp0
);

            temp4 += ROTXOR2( temp1 ) + CHOICE( temp1, temp2, temp3 )
+ 0x240ca1cc + ( (word03 += ROTXOR4( word01 ) + word12 + ROTXOR3(
word04 ) ) );
            temp0 += temp4;
            temp4 += ROTXOR1( temp5 ) + MAJORITY( temp5, temp6, temp7
);

            temp3 += ROTXOR2( temp0 ) + CHOICE( temp0, temp1, temp2 )
+ 0x2de92c6f + ( (word04 += ROTXOR4( word02 ) + word13 + ROTXOR3(
word05 ) ) );
            temp7 += temp3;
            temp3 += ROTXOR1( temp4 ) + MAJORITY( temp4, temp5, temp6
);

            temp2 += ROTXOR2( temp7 ) + CHOICE( temp7, temp0, temp1 )
+ 0x4a7484aa + ( (word05 += ROTXOR4( word03 ) + word14 + ROTXOR3(
word06 ) ) );
            temp6 += temp2;
            temp2 += ROTXOR1( temp3 ) + MAJORITY( temp3, temp4, temp5
);

            temp1 += ROTXOR2( temp6 ) + CHOICE( temp6, temp7, temp0 )
+ 0x5cb0a9dc + ( (word06 += ROTXOR4( word04 ) + word15 + ROTXOR3(
word07 ) ) );
            temp5 += temp1;
            temp1 += ROTXOR1( temp2 ) + MAJORITY( temp2, temp3, temp4
);

            temp0 += ROTXOR2( temp5 ) + CHOICE( temp5, temp6, temp7 )
+ 0x76f988da + ( (word07 += ROTXOR4( word05 ) + word00 + ROTXOR3(
word08 ) ) );
            temp4 += temp0;
            temp0 += ROTXOR1( temp1 ) + MAJORITY( temp1, temp2, temp3
);

            temp7 += ROTXOR2( temp4 ) + CHOICE( temp4, temp5, temp6 )
+ 0x983e5152 + ( (word08 += ROTXOR4( word06 ) + word01 + ROTXOR3(
word09 ) ) );
            temp3 += temp7;
            temp7 += ROTXOR1( temp0 ) + MAJORITY( temp0, temp1, temp2
);

            temp6 += ROTXOR2( temp3 ) + CHOICE( temp3, temp4, temp5 )
```

```
+ 0xa831c66d + ( (word09 += ROTXOR4( word07 ) + word02 + ROTXOR3(
word10 ) ) );
            temp2 += temp6;
            temp6 += ROTXOR1( temp7 ) + MAJORITY( temp7, temp0, temp1
);

            temp5 += ROTXOR2( temp2 ) + CHOICE( temp2, temp3, temp4 )
+ 0xb00327c8 + ( (word10 += ROTXOR4( word08 ) + word03 + ROTXOR3(
word11 ) ) );
            temp1 += temp5;
            temp5 += ROTXOR1( temp6 ) + MAJORITY( temp6, temp7, temp0
);

            temp4 += ROTXOR2( temp1 ) + CHOICE( temp1, temp2, temp3 )
+ 0xbf597fc7 + ( (word11 += ROTXOR4( word09 ) + word04 + ROTXOR3(
word12 ) ) );
            temp0 += temp4;
            temp4 += ROTXOR1( temp5 ) + MAJORITY( temp5, temp6, temp7
);

            temp3 += ROTXOR2( temp0 ) + CHOICE( temp0, temp1, temp2 )
+ 0xc6e00bf3 + ( (word12 += ROTXOR4( word10 ) + word05 + ROTXOR3(
word13 ) ) );
            temp7 += temp3;
            temp3 += ROTXOR1( temp4 ) + MAJORITY( temp4, temp5, temp6
);

            temp2 += ROTXOR2( temp7 ) + CHOICE( temp7, temp0, temp1 )
+ 0xd5a79147 + ( (word13 += ROTXOR4( word11 ) + word06 + ROTXOR3(
word14 ) ) );
            temp6 += temp2;
            temp2 += ROTXOR1( temp3 ) + MAJORITY( temp3, temp4, temp5
);

            temp1 += ROTXOR2( temp6 ) + CHOICE( temp6, temp7, temp0 )
+ 0x06ca6351 + ( (word14 += ROTXOR4( word12 ) + word07 + ROTXOR3(
word15 ) ) );
            temp5 += temp1;
            temp1 += ROTXOR1( temp2 ) + MAJORITY( temp2, temp3, temp4
);

            temp0 += ROTXOR2( temp5 ) + CHOICE( temp5, temp6, temp7 )
+ 0x14292967 + ( (word15 += ROTXOR4( word13 ) + word08 + ROTXOR3(
word00 ) ) );
            temp4 += temp0;
            temp0 += ROTXOR1( temp1 ) + MAJORITY( temp1, temp2, temp3
);
```

```
            temp7 += ROTXOR2( temp4 ) + CHOICE( temp4, temp5, temp6 )
+ 0x27b70a85 + ( (word00 += ROTXOR4( word14 ) + word09 + ROTXOR3(
word01 ) ) );
            temp3 += temp7;
            temp7 += ROTXOR1( temp0 ) + MAJORITY( temp0, temp1, temp2
);

            temp6 += ROTXOR2( temp3 ) + CHOICE( temp3, temp4, temp5 )
+ 0x2e1b2138 + ( (word01 += ROTXOR4( word15 ) + word10 + ROTXOR3(
word02 ) ) );
            temp2 += temp6;
            temp6 += ROTXOR1( temp7 ) + MAJORITY( temp7, temp0, temp1
);

            temp5 += ROTXOR2( temp2 ) + CHOICE( temp2, temp3, temp4 )
+ 0x4d2c6dfc + ( (word02 += ROTXOR4( word00 ) + word11 + ROTXOR3(
word03 ) ) );
            temp1 += temp5;
            temp5 += ROTXOR1( temp6 ) + MAJORITY( temp6, temp7, temp0
);

            temp4 += ROTXOR2( temp1 ) + CHOICE( temp1, temp2, temp3 )
+ 0x53380d13 + ( (word03 += ROTXOR4( word01 ) + word12 + ROTXOR3(
word04 ) ) );
            temp0 += temp4;
            temp4 += ROTXOR1( temp5 ) + MAJORITY( temp5, temp6, temp7
);

            temp3 += ROTXOR2( temp0 ) + CHOICE( temp0, temp1, temp2 )
+ 0x650a7354 + ( (word04 += ROTXOR4( word02 ) + word13 + ROTXOR3(
word05 ) ) );
            temp7 += temp3;
            temp3 += ROTXOR1( temp4 ) + MAJORITY( temp4, temp5, temp6
);

            temp2 += ROTXOR2( temp7 ) + CHOICE( temp7, temp0, temp1 )
+ 0x766a0abb + ( (word05 += ROTXOR4( word03 ) + word14 + ROTXOR3(
word06 ) ) );
            temp6 += temp2;
            temp2 += ROTXOR1( temp3 ) + MAJORITY( temp3, temp4, temp5
);

            temp1 += ROTXOR2( temp6 ) + CHOICE( temp6, temp7, temp0 )
+ 0x81c2c92e + ( (word06 += ROTXOR4( word04 ) + word15 + ROTXOR3(
word07 ) ) );
            temp5 += temp1;
            temp1 += ROTXOR1( temp2 ) + MAJORITY( temp2, temp3, temp4
);

            temp0 += ROTXOR2( temp5 ) + CHOICE( temp5, temp6, temp7 )
+ 0x92722c85 + ( (word07 += ROTXOR4( word05 ) + word00 + ROTXOR3(
```

```
word08 ) ) );
            temp4 += temp0;
            temp0 += ROTXOR1( temp1 ) + MAJORITY( temp1, temp2, temp3
);

            temp7 += ROTXOR2( temp4 ) + CHOICE( temp4, temp5, temp6 )
+ 0xa2bfe8a1 + ( (word08 += ROTXOR4( word06 ) + word01 + ROTXOR3(
word09 ) ) );
            temp3 += temp7;
            temp7 += ROTXOR1( temp0 ) + MAJORITY( temp0, temp1, temp2
);

            temp6 += ROTXOR2( temp3 ) + CHOICE( temp3, temp4, temp5 )
+ 0xa81a664b + ( (word09 += ROTXOR4( word07 ) + word02 + ROTXOR3(
word10 ) ) );
            temp2 += temp6;
            temp6 += ROTXOR1( temp7 ) + MAJORITY( temp7, temp0, temp1
);

            temp5 += ROTXOR2( temp2 ) + CHOICE( temp2, temp3, temp4 )
+ 0xc24b8b70 + ( (word10 += ROTXOR4( word08 ) + word03 + ROTXOR3(
word11 ) ) );
            temp1 += temp5;
            temp5 += ROTXOR1( temp6 ) + MAJORITY( temp6, temp7, temp0
);

            temp4 += ROTXOR2( temp1 ) + CHOICE( temp1, temp2, temp3 )
+ 0xc76c51a3 + ( (word11 += ROTXOR4( word09 ) + word04 + ROTXOR3(
word12 ) ) );
            temp0 += temp4;
            temp4 += ROTXOR1( temp5 ) + MAJORITY( temp5, temp6, temp7
);

            temp3 += ROTXOR2( temp0 ) + CHOICE( temp0, temp1, temp2 )
+ 0xd192e819 + ( (word12 += ROTXOR4( word10 ) + word05 + ROTXOR3(
word13 ) ) );
            temp7 += temp3;
            temp3 += ROTXOR1( temp4 ) + MAJORITY( temp4, temp5, temp6
);

            temp2 += ROTXOR2( temp7 ) + CHOICE( temp7, temp0, temp1 )
+ 0xd6990624 + ( (word13 += ROTXOR4( word11 ) + word06 + ROTXOR3(
word14 ) ) );
            temp6 += temp2;
            temp2 += ROTXOR1( temp3 ) + MAJORITY( temp3, temp4, temp5
);

            temp1 += ROTXOR2( temp6 ) + CHOICE( temp6, temp7, temp0 )
+ 0xf40e3585 + ( (word14 += ROTXOR4( word12 ) + word07 + ROTXOR3(
word15 ) ) );
            temp5 += temp1;
```

```
            temp1 += ROTXOR1( temp2 ) + MAJORITY( temp2, temp3, temp4
);

            temp0 += ROTXOR2( temp5 ) + CHOICE( temp5, temp6, temp7 )
+ 0x106aa070 + ( (word15 += ROTXOR4( word13 ) + word08 + ROTXOR3(
word00 ) ) );
            temp4 += temp0;
            temp0 += ROTXOR1( temp1 ) + MAJORITY( temp1, temp2, temp3
);




            temp7 += ROTXOR2( temp4 ) + CHOICE( temp4, temp5, temp6 )
+ 0x19a4c116 + ( (word00 += ROTXOR4( word14 ) + word09 + ROTXOR3(
word01 ) ) );
            temp3 += temp7;
            temp7 += ROTXOR1( temp0 ) + MAJORITY( temp0, temp1, temp2
);

            temp6 += ROTXOR2( temp3 ) + CHOICE( temp3, temp4, temp5 )
+ 0x1e376c08 + ( (word01 += ROTXOR4( word15 ) + word10 + ROTXOR3(
word02 ) ) );
            temp2 += temp6;
            temp6 += ROTXOR1( temp7 ) + MAJORITY( temp7, temp0, temp1
);

            temp5 += ROTXOR2( temp2 ) + CHOICE( temp2, temp3, temp4 )
+ 0x2748774c + ( (word02 += ROTXOR4( word00 ) + word11 + ROTXOR3(
word03 ) ) );
            temp1 += temp5;
            temp5 += ROTXOR1( temp6 ) + MAJORITY( temp6, temp7, temp0
);

            temp4 += ROTXOR2( temp1 ) + CHOICE( temp1, temp2, temp3 )
+ 0x34b0bcb5 + ( (word03 += ROTXOR4( word01 ) + word12 + ROTXOR3(
word04 ) ) );
            temp0 += temp4;
            temp4 += ROTXOR1( temp5 ) + MAJORITY( temp5, temp6, temp7
);

            temp3 += ROTXOR2( temp0 ) + CHOICE( temp0, temp1, temp2 )
+ 0x391c0cb3 + ( (word04 += ROTXOR4( word02 ) + word13 + ROTXOR3(
word05 ) ) );
            temp7 += temp3;
            temp3 += ROTXOR1( temp4 ) + MAJORITY( temp4, temp5, temp6
);

            temp2 += ROTXOR2( temp7 ) + CHOICE( temp7, temp0, temp1 )
+ 0x4ed8aa4a + ( (word05 += ROTXOR4( word03 ) + word14 + ROTXOR3(
word06 ) ) );
```

```
            temp6 += temp2;
            temp2 += ROTXOR1( temp3 ) + MAJORITY( temp3, temp4, temp5
);

            temp1 += ROTXOR2( temp6 ) + CHOICE( temp6, temp7, temp0 )
+ 0x5b9cca4f + ( (word06 += ROTXOR4( word04 ) + word15 + ROTXOR3(
word07 ) ) );
            temp5 += temp1;
            temp1 += ROTXOR1( temp2 ) + MAJORITY( temp2, temp3, temp4
);

            temp0 += ROTXOR2( temp5 ) + CHOICE( temp5, temp6, temp7 )
+ 0x682e6ff3 + ( (word07 += ROTXOR4( word05 ) + word00 + ROTXOR3(
word08 ) ) );
            temp4 += temp0;
            temp0 += ROTXOR1( temp1 ) + MAJORITY( temp1, temp2, temp3
);

            temp7 += ROTXOR2( temp4 ) + CHOICE( temp4, temp5, temp6 )
+ 0x748f82ee + ( (word08 += ROTXOR4( word06 ) + word01 + ROTXOR3(
word09 ) ) );
            temp3 += temp7;
            temp7 += ROTXOR1( temp0 ) + MAJORITY( temp0, temp1, temp2
);

            temp6 += ROTXOR2( temp3 ) + CHOICE( temp3, temp4, temp5 )
+ 0x78a5636f + ( (word09 += ROTXOR4( word07 ) + word02 + ROTXOR3(
word10 ) ) );
            temp2 += temp6;
            temp6 += ROTXOR1( temp7 ) + MAJORITY( temp7, temp0, temp1
);

            temp5 += ROTXOR2( temp2 ) + CHOICE( temp2, temp3, temp4 )
+ 0x84c87814 + ( (word10 += ROTXOR4( word08 ) + word03 + ROTXOR3(
word11 ) ) );
            temp1 += temp5;
            temp5 += ROTXOR1( temp6 ) + MAJORITY( temp6, temp7, temp0
);

            temp4 += ROTXOR2( temp1 ) + CHOICE( temp1, temp2, temp3 )
+ 0x8cc70208 + ( (word11 += ROTXOR4( word09 ) + word04 + ROTXOR3(
word12 ) ) );
            temp0 += temp4;
            temp4 += ROTXOR1( temp5 ) + MAJORITY( temp5, temp6, temp7
);

            temp3 += ROTXOR2( temp0 ) + CHOICE( temp0, temp1, temp2 )
+ 0x90befffa + ( (word12 += ROTXOR4( word10 ) + word05 + ROTXOR3(
word13 ) ) );
            temp7 += temp3;
            temp3 += ROTXOR1( temp4 ) + MAJORITY( temp4, temp5, temp6
```

```
);

            temp2 += ROTXOR2( temp7 ) + CHOICE( temp7, temp0, temp1 )
+ 0xa4506ceb + ( (word13 += ROTXOR4( word11 ) + word06 + ROTXOR3(
word14 ) ) );
            temp6 += temp2;
            temp2 += ROTXOR1( temp3 ) + MAJORITY( temp3, temp4, temp5
);

            temp1 += ROTXOR2( temp6 ) + CHOICE( temp6, temp7, temp0 )
+ 0xbef9a3f7 + ( (word14 += ROTXOR4( word12 ) + word07 + ROTXOR3(
word15 ) ) );
            temp5 += temp1;
            temp1 += ROTXOR1( temp2 ) + MAJORITY( temp2, temp3, temp4
);

            temp0 += ROTXOR2( temp5 ) + CHOICE( temp5, temp6, temp7 )
+ 0xc67178f2 + ( (word15 += ROTXOR4( word13 ) + word08 + ROTXOR3(
word00 ) ) );
            temp4 += temp0;
            temp0 += ROTXOR1( temp1 ) + MAJORITY( temp1, temp2, temp3
);

            state[0] = 0x6a09e667UL + temp0;
            state[1] = 0xbb67ae85UL + temp1;
            state[2] = 0x3c6ef372UL + temp2;
            state[3] = 0xa54ff53aUL + temp3;
            state[4] = 0x510e527fUL + temp4;
            state[5] = 0x9b05688cUL + temp5;
            state[6] = 0x1f83d9abUL + temp6;
            state[7] = 0x5be0cd19UL + temp7;
      }
}
```

As you can see, word00-word07 are initialized with the previous state unsigned integers. Directly transferring the state from the previous hash to the next hash input, essentially linking the hashes directly into each other. There are 2 other “word” variables that are set with non-zero static values, and thats just for SHA-256’s data block formatting. The following is a benchmark of the two implementations against each other.

```
Hashing SHA-256 with 134217728 iterations.
Original: (68.333000 seconds) 1964171.469129 hashes per second.
Optimized: (55.732000 seconds) 2408270.455035 hashes per second.
Completed in 81.5594% of the time
```

So that's all nice. I made something faster than another thing I wrote. Let's see how it stands up next to another application that does the same thing. For instance, let's use John the Ripper with the Jumbo patch.

```
bwall@devBox:/pentest/passwords/john$ time john --wordlist=test.lst
hashLoaded 1 password hash (Password Safe SHA-256 [32/64])
guesses: 0  time: 0:00:00:17 43.82% (ETA: Fri Jul 27 11:21:58 2012)
c/s: 1462  trying: bear
guesses: 0  time: 0:00:00:38 DONE (Fri Jul 27 11:21:59 2012)  c/s:
1466  trying: sss

real  0m38.771s
user  0m38.638s
sys   0m0.004s
```

Now lets compare that to safe-cracker running on the same machine.

```
bwall@devBox:~/Projects/pwcrack/CrackPwSafe/bin/Release$ time ./safe-
cracker ~/Projects/pwsafe.psafe3 /pentest/passwords/john/test.lst

Hashes(10000) Per Second(6.09366s): 1641.05
Hashes(20000) Per Second(12.1719s): 1643.13
Hashes(30000) Per Second(18.2494s): 1643.89
Hashes(40000) Per Second(24.3281s): 1644.19
Hashes(50000) Per Second(30.4068s): 1644.37

real  0m34.659s
user  0m34.682s
sys   0m0.000s
```

Not only is it faster, but it also does the extraction too. I do plan on contributing to the John the Ripper project soon with this optimization.

References

John Kelsey, B. S. (2012). *Secure Applications of Low-Entropy Keys.* Retrieved July 26, 2012, from Bruce Schneier: http://www.schneier.com/paper-low-entropy.pdf

Wallace, Brian. (2012, July 21). *Attack Source Code*. Retrieved July 26, 2012, from SafeCracker: https://github.com/bwall/SafeCracker/blob/master/