ESSENCE: Change Log

Jason Worth Martin

January 12, 2009

Abstract

This document describes changes from the original NIST submission for ESSENCE. No changes have been made to the algorithm or the source code. The only changes are clarifications of the specification and corrections to typographical errors and exposition.

1 Changes to ESSENCE: A Family of Cryptographic Hashing Algorithms

1. In the introduction, the last paragraph has been rewritten to reflect code performance as of January 2009.

   Original Version: All of these design constraints do force a performance trade off. For example, we have written naive C code implementations of the E permutation with and without look-up tables. The implementation without the look-up tables runs at approximately 160 processor cycles/byte (on an x86_64 Linux platform), while the implementation with look-up tables executes at approximately 53 cycles/byte on the same platform. (We expect that highly tuned assembly code would perform better, but at the time of this writing that has not been implemented.) So, there is certainly a trade-off for resistance to timing attacks. However, that trade-off is not severe for ESSENCE, and the existence of demonstrated real-world successful attacks against the timing-sensitive implementations of otherwise secure cryptosystems leads us to believe that it is necessary to have an algorithm which can have reasonable implementations that are resistant to side-channel attacks.

   Corrected Version: All of these design constraints do force a performance trade off, and a serial implementation of ESSENCE runs much slower than a similar implementation of any of the SHA-2 family of hashing algorithms. However, the design of ESSENCE emphasizes security through a simple algorithm structure while allowing for performance through parallel implementations. A serial version of ESSENCE restricted to only 32-bit general purpose registers runs at over 176 cycles per byte on a single core of an Intel Core 2 processor. However, a parallel version running on a quad-core Intel Core 2 processor and using assembly code to access 128-bit vector registers is capable of running at nearly 12 cycles per byte.

2. In Figure 2 the line of pseudo-code `r1 := r1` was corrected to `r2 := r1`.

3. In Figure 5 the line of pseudo-code `r1 := r1` was corrected to `r2 := r1`.

4. In section 3.3 the aside reminding the reader of the definition of a companion matrix had the indices reversed for the polynomial coefficients in the matrix and ignored the fact that the leading coefficient is 1. This has been corrected.

   Original Version: Recall that the companion matrix of a polynomial $f(x) = b_n x^n + b_{n-1}x^{n-1} + \cdots + b_1 x + b_0$ is the $n \times n$ matrix

$$C_f = \begin{pmatrix} 0 & 1 & 0 & 0 & \dots & 0 \\ 0 & 0 & 1 & 0 & \dots & 0 \\ & & & \ddots & & \\ & & & & & \\ 0 & 0 & 0 & 0 & \dots & 1 \\ -b_{n-1} & -b_{n-2} & -b_{n-3} & -b_{n-4} & \dots & -b_0 \end{pmatrix}.$$

   Corrected Version: Recall that the companion matrix of a polynomial $f(x) = x^n + b_{n-1}x^{n-1} + \cdots + b_1 x + b_0$ is the $n \times n$ matrix

$$C_f = \begin{pmatrix} 0 & 1 & 0 & 0 & \dots & 0 \\ 0 & 0 & 1 & 0 & \dots & 0 \\ & & & \ddots & & \\ & & & & & \\ 0 & 0 & 0 & 0 & \dots & 1 \\ -b_0 & -b_1 & -b_2 & -b_3 & \dots & -b_{n-1} \end{pmatrix}.$$

5. Meltem Sonmez Turan pointed out an error in section 3.2.1, the criteria for the F function. Criteria 2 incorrectly listed the maximal linear complexity possible for a sequence generated by an n-bit shift register as 2^n. The correct upper bound is $2^n - 1$. The shift register used in ESSENCE with the given F function has linear complexity 255, not 256, and the connecting polynomial is $\sum_{k=0}^{255} x^k$, not $x^{256} + 1$ as was originally written.

   Original Version: Given any sequence, the linear complexity of the sequence is the length of the smallest linear feedback shift register (LFSR) which can generate the sequence. The maximal linear complexity for a shift register sequence generated by an n-bit register is 2^n.

The function F, used in the configuration shown in Figure 3 with I set to zero produces a shift register sequence with maximal linear complexity (i.e. its linear complexity is 256 with the connecting polynomial $x^{256} + 1$).

Corrected Version: Given any sequence, the linear complexity of the sequence is the length of the smallest linear feedback shift register (LFSR) which can generate the sequence. The maximal linear complexity for a shift register sequence generated by an n-bit register is $2^n - 1$.

The function F, used in the configuration shown in Figure 3 with I set to zero produces a shift register sequence with maximal linear complexity. Its linear complexity is 255 with connecting polynomial $\sum_{k=0}^{255} x^k$.

6. Nicky Mouha pointed out an error in Figure 7: Inefficient L_{64} Implementation in C. The line

   ```
   temp = (temp << 1) & P_64;
   ```

   should be

   ```
   temp = (temp << 1) ^ P_64;
   ```

   This has been corrected.

7. Added an Acknowledgment section to thank individuals who have pointed out errors.

2 Changes to ESSENCE: A Candidate Hashing Algorithm for the NIST Competition

1. Inserted section 3 labeled “Some Overloaded Definitions”. This new section gives a clearer explanation of some of the terms used in the specification documents. In particular, it explains the differences between the overloaded usages of the word “block”. Note that the insertion of this section has increased the section numbers of all following sections by one.

2. In section 7, “Running Hash”, a statement to clarify what is computed for a hash of a message with zero length was added. In particular, the description of bytes 0-7 of the final block hash been changed as follows:

Original Version: An unsigned 64-bit integer representing the number of complete Merkle-Damgård blocks processed.

Corrected Version: An unsigned 64-bit integer representing the number of complete Merkle-Damgård blocks processed. If ESSENCE is requested to hash a message of zero length, then all bits in this field will be set to ones.

3. In section 7, “Running Hash”, added subsection 7.1, “Resistance to Length Extension Attacks” to clarify how ESSENCE is resistant to length extension attacks.