

UMAC for Ultra-low Power Environments

A.L. Huang

Department of Electrical, Electronic and Computer Engineering,
University of Pretoria, 0002 PRETORIA, South Africa
alanh@tuks.co.za

Abstract—UMAC is an authentication algorithm generating Message Authentication Codes (MAC), which are utilized to ensure message integrity and verify user authenticity in a network environment. UMAC was designed for high-end 32-bit/64-bit environment. However, applications such as Ad-Hoc networks or wireless sensor networks (WSN) have resource constrained devices, with limited resources in terms of computational capability and power supply. This paper has adapted UMAC to be implemented in such resource constrained environment. The customized UMAC is implemented on embedded microcontroller and evaluated for its appropriateness for ultra-low power requirements.

I. INTRODUCTION

UMAC is an authentication algorithm using the universal hash function family, NH. NH is a new universal hash function family developed specifically for UMAC [1]. In simplest terms, universal hash functions are collections of hash functions that map messages into short output strings such that the collision (pairs of different inputs with identical outputs) probability of any given pair of messages is small. UMAC allows user to select the underlying cryptographic primitives (e.g. cryptographic hash functions or block ciphers). No new heuristic primitives are developed in UMAC; therefore it is secure as long as the underlying cryptographic primitives are secure. UMAC has been particularly designed to utilize the SIMD (Single Instruction Multiple Data) parallelism of modern processors to achieve high speed. A 64-bit hash code UMAC optimized with MMX (Multimedia extensions) instructions can achieve a speed of more than 1 byte/cycle with messages larger than 256 Kbytes (on a Pentium II machine with MMX) [1]. However, in the case of ultra-low power environment (e.g. wireless sensor node), resources are often extremely restricted. Therefore the standard UMAC implementation is needs to be modified before it can be implemented in such environment.

This paper is organized as following: section II discusses the basic concept behind UMAC and how standard UMAC is practically implemented; section III discusses the UMAC implementation proposed for the ultra-low power environment; section IV shows the results of such implementation and section V gives the summarized findings and conclusions.

II. STANDARD UMAC IMPLEMENTATION

This section explains the basic concept and the standard implementation of the UMAC. The UMAC implementation discussed here is the refined version ([2][8]) as compared to when UMAC standard was first proposed. [1]

Figure 1 shows the functional diagram of the refined UMAC [2].

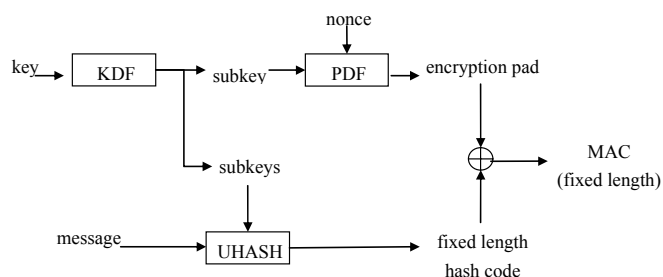


Figure 1. UMAC

A. Key Derivation Function (KDF)

The user selected secret key is expanded into more subkeys using the KDF. The subkeys are used internally by UMAC in UHASH and the pad derivation function (PDF). Block ciphers (e.g. AES) are used in output feed back (OFB) mode to produce the required subkey bits. The OFB mode used in KDF first encrypts a pre-defined value, and then takes the resulting ciphertext output as the next block to be encrypted. This chain of ciphertext outputs is used as the required subkeys. The block cipher in KDF is used as the pseudo random generator (PRG).

B. Pad Derivation Function (PDF)

The PDF is needed to generate the one-time encryption pad to be XORed with the fixed size hash code to produce the MAC. This one-time-pad (also known as otp) is obtained by applying the PDF to a nonce with a subkey generated by the KDF. A block cipher is typically used in the PDF to encrypt the nonce. The resulting ciphertext bytes are used as the one-time-pad.

A non-repeating nonce is needed to ensure that every MAC generated is different even if the data messages are the same. It can be a simple non-secret incrementing counter that is sent with the data message and the appended MAC. To provide protection against replay attacks, the receiver needs to check that no nonce value is used twice. This can be easily achieved when the nonce is a counter.

C. UHASH

UHASH is a keyed hash function, which takes an arbitrary length input data message, and produces as output a fixed length hash code. Figure 2 shows the function diagram of the UHASH.

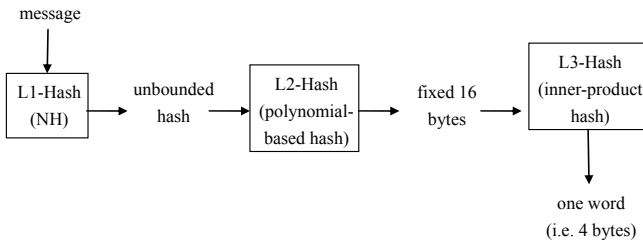


Figure 2. UHASH with word size $w = 32$

UHASH consists of three layers. The first layer is the NH hash function, which is used to compress input messages into strings many times smaller than the input message.

The second layer is a polynomial-based hash function that takes the unbounded (variable length) hash results from NH, and produces a fixed-length 16-byte output (when word size $w = 32$). The polynomial hash function includes prime modulus operations. The security guarantee assured by polynomial hashing degrades linearly with the increasing length of the message being hashed and the prime number value. The prime modulus can be dynamically increased to ensure that the collision probability never grows beyond a certain pre-set bound when hashing a long message.

The third layer is an inner-product hash function that hashes the fixed 16-byte input to a fixed length word (i.e. 4 bytes when $w = 32$). A 36-bit prime modulus operation is used to improve security. Detailed discussions of layer two and layer three implementations are beyond the scope of this paper.

D. NH

The message to be authenticated needs to be represented in words of size w . For illustration purpose, let w be 32 bits. The message is then divided into blocks of n number of words, let $n = 4$ as shown in **Error! Reference source not found.**. This n number of words is the amount of words that the NH hash function will process when it is called. The actual n values range from 32 to 2^{28} bytes, typically $n = 256$ (equivalent to 1024 bytes when $w = 32$ bits) [2].

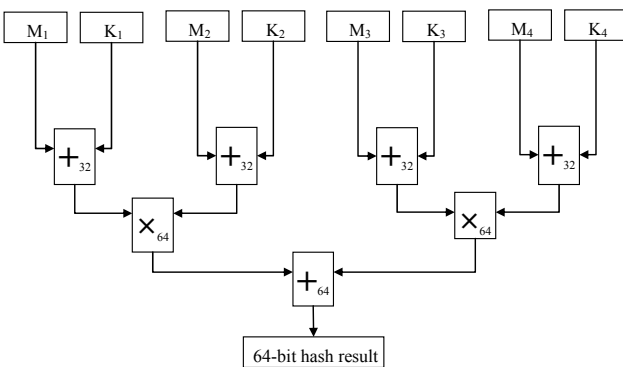


Figure 3. NH hash function with word size $w = 32$ and number of words processed in an NH block $n = 4$.

The message words (M) are processed by the NH hash function as shown in the above figure. The n words subkeys (K) are generated by a pseudo random generator (PRG). With 32-bit words, each message word is added (modulo 32) with the subkey word and then multiplied (modulo 64) with the next word that is also the result of subkey and message word addition. All multiplied (modulo 64) results of an NH

call are then added (modulo 64) to get a 64-bit hash code. Repeated NH hash function calls are performed on all message words at n words per NH function call. The n subkeys remain the same for all NH calls; therefore with the same secret key, subkeys only need to be generated once. The multiple 64-bit hash codes resulting from multiple NH calls are concatenated together as an unbounded (variable length) hash code. Although this hash code is smaller than the original data message, it is still proportional to its size.

Increasing n increases the number of words to be processed in one NH call and results in smaller sized unbounded hash codes after NH calls. This tends to speed up MAC generation on large messages, but requires more memory (for subkeys K_1, \dots, K_n) for processing and could potentially slow the processor by overflowing the processor's cache memory.

NH operations can be optimized using SIMD instructions such as MMX (Multi Media Extensions) instructions. NH calls are heavily used in UMAC, thus by optimizing NH calls one can greatly optimize UMAC.

These three layers (UHASH) are repeated (with some different subkeys) until enough output MAC bytes are produced. For example, with 32-bit word size, UHASH needs to be called twice to obtain a 64-bit MAC.

III. PROPOSED UMAC IMPLEMENTATION

The UMAC implementations in this paper have been greatly modified to be more suitable to a WSN environment. They are implemented from scratch using UMAC internet drafts [2] as guidelines. The embedded microcontroller in which the customized UMAC is implemented is the Texas Instrument MSP430F1232 [6]. It is a 16-bit RISC microcontroller with 8KB flash memory (code memory) and 256 bytes of RAM. The operating clock frequency is 1 MHz (1 MIPS) at 2.2V and 200 μ A.

In the UMAC shown in Figure 1 and Figure 2 from the previous section, the L2-HASH and L3-HASH are used to further reduce the size of the unbounded (variable length) hash code generated by L1-HASH (NH hash function). After the L2-HASH and L3-HASH the output hash code is a fixed size. However, the output hash code size of the NH hash function is proportional to the input message size. Therefore in a closed WSN environment where data message size is typically small, fixed and known, it is possible to customize UMAC to eliminate the need for the L2-HASH and L3-HASH. The UMAC implemented in this paper is similar to Figure 1, except that only the NH hash function is needed in place of UHASH as shown in the figure below.

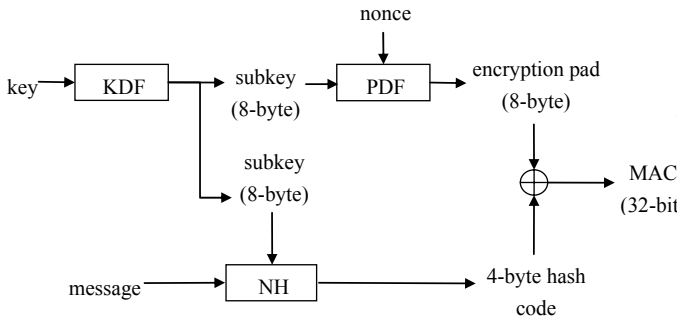


Figure 4. customized UMAC for ultra-low power environment

The customized UMAC in this paper (Figure 4) uses an underlying block cipher with a 64-bit block size. Therefore the KDF (key derivation function) only needs to call the block cipher twice to generate two 8-byte subkeys to be used in the PDF (pad derivation function) and NH hash function (see Figure 5). Note that the subkeys only need to be generated once, and remain the same for the lifetime of the same secret key. One block cipher call in PDF produces 8 bytes of encryption pads to be XORed with a 4-byte (32-bit) hash code. Therefore every PDF call produces enough encryption pads for generating two MACs. The nonce used with PDF is realized with a simple incrementing counter.

The NH hash function is shown in the following figure (Figure 5).

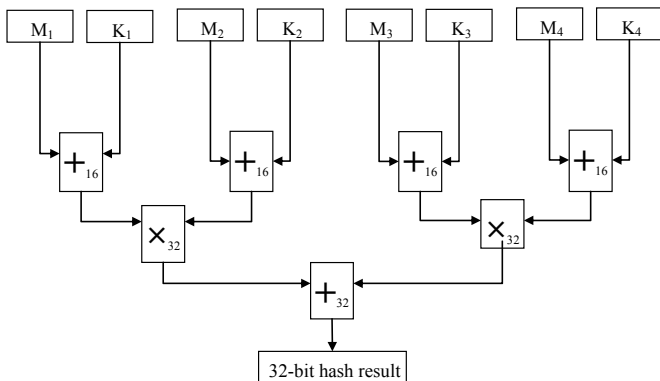


Figure 5. NH hash function with word size $w = 16$ and number of words processed in a NH block $n = 4$

The word size processed in the NH hash function is 16-bit, and the number of words processed in a NH block is equal to 4 words. Therefore the number of bytes processed in one NH hash function call is 8 bytes. Word size is chosen to be 16-bit to achieve a 32-bit hash code after two 16-bit multiplications, and also because the word size of the microcontroller used is also 16-bit.

The data message is divided into 8-byte chunks as input for the NH hash function. The hash result of a NH function call is added to the hash result of the previous NH function call. Therefore, the final hash code will always be of a fixed 4-byte size. For example, to process a 24 byte data message, three NH function calls are required, and all three hash results are added. This method of implementing the NH hash function has also been proposed by Yüksel [3], however, in a hardware implementation only.

The advantage of using such a customized UMAC is clear. After the initial subkeys have been generated, to produce two MACs for any messages afterwards only requires one KDF call (i.e. one block cipher call) and several NH hash calls (depending on the size of the message). Using only one block cipher call to generate every two MACs can save a significant amount of processing time, particularly when the block cipher calls are more expensive than NH function calls. The underlying block cipher of UMAC can also be used to provide encryption that is not provided by UMAC.

IV. RESULTS

UMAC provides only authentication by calculating the MAC (message authentication code). In this paper, UMAC is customized for small size data using XTEA (eXtended Tiny Encryption Algorithm) block cipher as its underlying pseudo random number generator.

A. Power Consumption in MSP430 Microcontroller

Different instructions may require different numbers of clock cycles, resulting in different amounts of energy consumption per cycle. Even different instructions with the same number of clock cycles may consume different amount of energy per cycle because of the nature of the instruction itself. For example an instruction that accesses the main memory (RAM) or registers will consume less energy than an instruction that accesses the flash memory.

However, Law et. al. [4] have shown that the energy per cycle is fairly consistent with the MSP430 microcontroller family with a mean deviation of 6%. Großschadl et. al. [5] have further shown that variable energy consumption per cycle has more influence on high-end microcontrollers and DSPs. For example, Intel's StrongARM SA-1100 has a more complex power management strategy such as the use of conditional clocking trees, which ensures only the presently required units in the microcontroller are clocked and other units remain static; thereby resulting in the difference in energy consumption per cycle.

Therefore it is safe to say that the MSP430F1232 microcontroller used in this paper running at 1 MIPS, 2.2V, and at an average current of 200 μA requires an average power consumption of: $2.2V \times 200\mu A = 440\mu W$.

B. Customized UMAC Results

The customized UMAC-XTEA can be subdivided into three components: KDF, PDF and NH. The key derivation function (KDF) generates two subkeys (total of 16 bytes) needed for both the PDF and the NH hash function. The pad derivation function (PDF) generates a one-time encryption pad to be XORed with the hash code. The NH hash function processes eight-byte blocks and produces four-byte (32-bit) hash codes. Every PDF call invokes an XTEA cipher call, which produces an eight-bytes encryption pad; therefore one PDF call provides encryption pads for generating two MACs.

Table 1 shows the number of CPU cycles needed for the different function calls within UMAC-XTEA. The NH hash function involves 16-bit multiplication operations, which makes the CPU cycle usage depending on the input value to the NH function. The NH function CPU usage below is the average value across several different input values. Furthermore, some MSP430 microcontrollers have built-in hardware multiplier (HW multiplier) (e.g. MSP430F140).

When such microcontrollers are used, the performance of the NH hash function is improved.

Table 1. UMAC-XTEA CPU usage

	KDF	PDF	NH (without HW multiplier)	NH (with HW multiplier)
CPU usage	4679	2300	570	190

The flash memory required for UMAC-XTEA code in the above table is 1333 bytes. This includes code for setting up a simulated 24 byte data packet. If a data packet is of size 24 bytes (3 blocks), then to authenticate such data packet using UMAC-XTEA requires one PDF call and three NH hash function calls (excluding the key setup KDF call). The number of PDF, KDF and NH calls increase linearly as the data size increases.

With a 24 bytes data, the average power consumption for producing a 32-bit MAC is less than 2 μ W. In a ultra-low power wireless sensor network (WSN) environment, packet size is typically less than 50 bytes. It has also been noted that NH function calls typically need a lot fewer CPU cycles than block cipher calls.

V. CONCLUSION

Although UMAC is originally designed for a modern 32/64-bit architecture and for authenticating longer messages, in this paper it has been adapted and optimized for the short message ultra-low power environment, which is typically found in wireless sensor networks (WSN). The customized UMAC is implemented on a resource-constrained microcontroller (MSP430F1232) which is also the same microcontroller used in the TinyMote sensor node [7]. The result shows that the customized UMAC is suitable to be implemented on low-end resource-constrained environment and achieves the ultra-low power requirement.

Although UMAC only provides message authentication, it is however, possible to utilize the underlying block cipher of UMAC to achieve message encryption.

REFERENCES

- [1] J. Black, S. Halevi, H. Krawczyk et. al., "UMAC: Fast and Secure Message Authentication", *Advances in Cryptology - CRYPTO'99*, LNCS 1666, pp. 216. Springer-verlag, 1999.
- [2] T. Krovetz, "UMAC: Message Authentication Code using Universal Hashing" 2000, internet draft, <http://www.ietf.org/internet-drafts/draft-krovetz-umac-07.txt>. Last accessed on April 2005.
- [3] K. Yüksel, "Universal Hashing for Ultra-Low-Power Cryptographic Hardware Applications", Master's thesis, Worcester Polytechnic Institute, May 2004.
- [4] Y. Law, J. Doumen, and P. Hartel, "Benchmarking Block Ciphers for Wireless Sensor Networks (Extended Abstract)", *Proceedings of 1st IEEE International Conference on Mobile Ad Hoc and Sensor Systems*, 2004.
- [5] J. Großschadl, R. Avanzi, E. Savas et. al., "Energy-Efficient Software Implementation of Long Integer Modular Arithmetic", *Workshop on Cryptographic Hardware and Embedded Systems (CHES)*, LNCS 3659, pp. 70-90, Springer-verlag, 2005.
- [6] Texas Instrument MSP430 Microcontroller, <http://www.ti.com>. Last accessed on October 2005.
- [7] S. Mahlke and M. Rötzer, "Energy-Self-Sufficient Wireless Sensor Networks", Technical University of Vienna, Tech. Rep., 2005.
- [8] J. Black, S. Halevi, H. Krawczyk et. al., "Update on UMAC Fast Message Authentication", May 2000, <http://www.cs.ucdavis.edu/~rogaway/umac/update.pdf>. Last accessed on April 2005.

An-Lun (Alan) Huang borned on the 25th of September 1981 in Tainan (Republic of China, Taiwan), matriculated at the Willowridge Highschool in 1999, Pretoria, South Africa. He graduated (Computer Engineering) at the University of Pretoria in 2003. He joined the masters-degree programme at the Centre of Excellence - Telkom at the University of Pretoria in February 2004. In 2004 he also obtained his Honour degree in Electronic engineering.