Universal Hash Functions for Emerging Ultra-Low-Power Networks

Kaan Yüksel, Jens-Peter Kaps, and Berk Sunar
Electrical & Computer Engineering
Worcester Polytechnic Institute
Worcester, Massachusetts 01609
Email: {kyuksel, kaps, sunar}@wpi.edu

***Abstract*— Message Authentication Codes (MACs) are a valuable tool for ensuring the integrity of messages. MACs may be built around a keyed hash function. In this paper, we propose three variations on NH (a universal hash function explored in UMAC [1]), namely PH, PR and WH. Our main motivation was to prove that universal hash functions can be employed to provide provable security in ultra-low-power applications such as the next generation self-powered sensor networks. The first hash function we propose, i.e. PH, produces a hash of length $2w$ and is shown to be 2^{-w}-almost universal. The other two hash functions, i.e. PR and WH, reach optimality and are proven to be universal hash functions with a much shorter hash length of w. In addition, these schemes are simple enough to allow for efficient constructions. To the best of our knowledge the proposed hash functions are the first ones specifically designed for low-power hardware implementations. We achieved drastic power savings of up to 59% and speedup of up to 7.4 times over NH. Note that the speed improvement and the power reduction are accomplished simultaneously. Our implementation of WH consumes only $11.6\,\mu$W at 500 kHz. It could therefore be integrated into a self-powered device. This enables the use of hash functions in ultra-low-power applications such as "Smart Dust" motes and RFIDs. By virtue of their security and implementation features mentioned above, we believe that the proposed universal hash functions will fill an important gap in cryptographic hardware applications.**

***Index Terms*— Universal hashing, ultra-low-power, self-powered, provable security, message authentication codes**

I. Introduction

WIRELESS sensor networks are a very active topic of research with far reaching applications [2], from monitoring birds on Great Duck Island on the coast of Maine [3] [4], collecting microclimate data in the James San Jacinto Mountains Reserve to several military applications like target tracking [5] and detecting bio-weapons. One type of sensor nodes used for this purpose is "Smart Dust" motes [6]. These are tiny autonomous nodes which contain sensors, some computing power, transceivers, and a power source. They communicate wirelessly and their energy source is extremely limited. Current dust motes still require two AA batteries [7] for operation and are the size of modern pagers. The circuits of newer motes are only 5 square millimeters but they still require batteries. Kahn describes in [6] that tiny batteries for these devices can supply $10\,\mu$W for only one day. Gorder wrote in [7] that the size of the mote depends mainly on the development of new batteries. Some smart dust motes use micro-electromechanical systems (MEMS) for sensing and communicating. Amirtharajah showed in [8] that MEMS could also be used to convert energy from environmental sources, such as light, heat, noise, or vibration into electrical power in order to power a digital system. Devices that harvest power from such sources are commonly referred to as *power scavengers*, and autonomous nodes which use scavengers are called *self-powered*. An implementation of a signal processing unit powered by a large scavenger device that can generate up to $400\,\mu$W is described in [8]. Newer scavengers are based on micro-electromechanical systems (MEMS). They can be integrated into the chip and therefore reduce the cost and size. The scavenger shown in [9] produces around $8\,\mu$W relying solely on ambient vibration.

Smart dust motes are used in distributed sensor networks. The security aspects of these networks have been reviewed by *NAI Labs* in [10]. However, this study focused only on software implementations on current general purpose processors whose energy consumption is far above the amount that can be supplied by a scavenger circuit. Perrig introduced a set of security protocols (SPINS) specifically for sensor networks that are using smart dust sensors [11]. Protecting the integrity of data that is transmitted between nodes is of utmost importance. For example, smart dust motes that are embedded in a bridge could monitor the stress and inform the authorities in case of emergency. Wireless sensors might monitor plant growth, moisture and PH-value on a farm. For this purpose, digital signature schemes have been proposed [12]. However, on low-end computing platforms where processing speed and communication bandwidth is critical, digital signatures may not be the best available choice. Instead, efficient *Message Authentication Codes* (MACs) [13] may be preferable due to their high encryption throughput and short authentication tags. A disadvantage for both digital signature schemes and MACs is that they provide only computational security. This means that an attacker with sufficient computational power may break the scheme. More severely, the lack of a formal security proof makes these schemes vulnerable to possible shortcut attacks.

Universal hash functions, first introduced by Carter and

This material is based upon work supported by the National Science Foundation under Grant No. ANI-0133297.

Wegman [14], provide a unique solution to the aforementioned security problems. Roughly speaking, universal hash functions are collections of hash functions that map messages into short output strings such that the collision probability of any given pair of messages is small. A universal hash-function family can be used to build an unconditionally secure MAC. For this, the communicating parties share a secret and randomly chosen hash function from the universal hash-function family, and a secret encryption key. A message is authenticated by hashing it with the shared secret hash function and then encrypting the resulting hash using the key. Carter and Wegman [15] showed that when the hash-function family is strongly universal, i.e. a stronger version of universal hash functions where messages are mapped into their images in a pairwise independent manner, and the encryption is realized by a one-time pad, the adversary cannot forge the message with probability better than that obtained by choosing a random string for the MAC. The one-time pad encryption and the hash function selection (from the hash function family) require many key bits, which may be too demanding for certain applications. In [16] Brassard observed that combining a universal hash function with a pseudo-random string generator provides a computationally secure message authentication tag with short keys. In this scheme the security of the MAC is dependent on the security of the encryption with the pseudo-random string and the strength of the pseudo-random key used for selecting the hash function.

To our knowledge not much work has been done on improving the performance of universal hashing in hardware. Ramakrishna published a study on the performance of hashing functions in hardware based on universal hashing [17]. However, the main emphasis was on using hash functions for table organization and address translation. In an early work Krawczyk [18] proposed efficient hash functions from a hardware point of view. Considering that a linear feedback shift register (LFSR) can be implemented quite efficiently in hardware, the author's work introduced two constructions: a CRC-based cryptographic hash function, and a construction based on Toeplitz matrix multiplication. The reference gives a sketch for hardware implementation, which includes a key spreader. However, it is difficult to estimate the power consumption of this function from a sketch. There have been no implementations reported so far. In the past decade we have seen many new hash constructions being proposed, constantly improving in speed and collision probability [1], [19]–[23]. For a survey see [24]. However, most of these constructions have targeted efficiency in software implementations, with particular emphasis on matching the instruction set architecture of a particular processor or taking advantage of special instructions made available for multimedia data processing (e.g. Intel's MMX technology). While such high end platforms are essential for everyday computing and communications, in numerous embedded applications space and power limitations prohibit their employment. Smart dust sensor nodes employ a 4-bit or 8-bit low end microprocessor, run the operating system TinyOS [25] and are battery powered. These microprocessors do not provide efficient multiplication or variable rotate / shift instructions [11] which are used by many cryptographic functions. Therefore additional ultra-low power hardware specifically tailored to perform these functions might be useful. In this paper we are also targeting self-powered sensor nodes which do not contain a microprocessor but rather a simple control logic as it is also common in RFID tags [26].

In [1] a new hash function family NH for the UMAC message authentication code was introduced. Although NH was intended for efficient and fast implementation in software we realized that it seems promising for implementation in hardware as well. Therefore, we implemented NH in hardware with an emphasis on low-power and modified it to improve its performance and minimize its power consumption. The resulting three hash functions are shown to be either universal or almost universal, i.e., no security is sacrificed as compared to NH. At the same time, we envision stringent power limitations (less than $20\,\mu$W) in order to make universal hashing practical in systems powered by scavenger units. We present our implementations of these functions and NH and compare the results.

II. Preliminaries

A. Notations

Let $\{0,1\}^*$ represent all binary strings, including the empty string. The set $H = h : A \rightarrow B$, associated with some distribution, is a family of hash functions with domain $A \subseteq \{0,1\}^*$ of size a and range $B \subseteq \{0,1\}^*$ of size b. The set $C \subseteq \{0,1\}^*$ denotes the finite set of key strings. H_K denotes a single hash function chosen from the set of hash functions H according to a random key $K \in C$. In the text we will set $h = H_K$ to denote a hash function h selected randomly from the set H.

The element $M \in A$ stands for a message string to be hashed and is partitioned into blocks as $M = (m_1, \cdots, m_n)$, where n is the number of message blocks of length w. Similarly the key $K \in C$ is partitioned as $K = (k_1, \cdots, k_n)$, where each block k_i has length w. We use the notation $H[n, w]$ to refer to a hash function family where n is the number of message (or key) blocks and w is the number of bits per block.

Let U_w represent the set of nonnegative integers less than 2^w, and P_w represent the set of polynomials over $GF(2)$ of degree less than w. Note that each message block m_i and key block k_i belongs to either U_w, P_w or $GF(2^w)$. Here $GF(2^w)$ denotes the finite field of 2^w elements defined by $GF(2)[x]/(p)$, where p is an irreducible polynomial of degree w over $GF(2)$. Note that, in this setting the bits of a message or key block are associated with the coefficients of a polynomial. Finally, the addition symbol '+' is used to denote both integer and polynomial addition (in a ring or finite field). The meaning should be obvious from the context.

B. Universal Hashing

A universal hash function, as proposed by Carter and Wegman [14], is a mapping from the finite set A with size a to the finite set B with size b. For a given hash function $h \in H$ and for a message pair (M, M') where $M \neq M'$ the following

function is defined: $\delta_h(M, M') = 1$ if $h(M) = h(M')$, and 0 otherwise, that is, the function δ yields 1 when the input message pairs collide. For a given finite set of hash functions $\delta_H(M, M')$ is defined as $\sum_{h \in H} \delta_h(M, M')$, which tells us that $\delta_h(M, M')$ yields the number of functions in H for which M and M' collide. When h is randomly chosen from H and two distinct messages M and M' are given as input, the collision probability is equal to $\delta_h(M, M')/|H|$. We give the definitions of the two classes of universal hash functions used in this paper from [24]:

Definition 1: The set of hash functions $H = h : A \rightarrow B$ is said to be **universal** if for every $M, M' \in A$ where $M \neq M'$,

$$|h \in H : h(M) = h(M')| \ = \delta_H(M, M') = \frac{|H|}{b} \ .$$

Definition 2: The set of hash functions $H = h : A \rightarrow B$ is said to be ϵ**-almost universal** ($\epsilon - AU$) if for every $M, M' \in A$ where $M \neq M'$,

$$|h \in H : h(M) = h(M')| \ = \delta_H(M, M') = \epsilon |H| \ .$$

In this definition ϵ is the upper bound for the probability of collision. Observe that the previous definition might actually be considered as a special case of the latter with ϵ being equal to $1/b$. The smallest possible value for ϵ is $(a-b)/(b(a-1))$.

In the past many universal and almost universal hash families were proposed [1], [19]–[23]. Black et al introduced an almost universal hash function family called NH in [1]. The definition of NH is given below.

Definition 3: ([1]) Given $M = (m_1, \cdots, m_n)$ and $K = (k_1, \cdots, k_n)$, where m_i and $k_i \in U_w$, and for any even $n \geq 2$, NH is computed as follows:

$$\mathsf{NH}_K(M) = \left[\sum_{i=1}^{n/2} ((m_{2i-1} + k_{2i-1}) \bmod 2^w) \right. \\ \left. \cdot ((m_{2i} + k_{2i}) \bmod 2^w) \right] \bmod 2^{2w} \ .$$

In the same paper NH was shown to have a tight bound of 2^{-w} on the collision probability.

III. Our Contribution

We introduce three variations to the NH construction. Each one improves upon the previous one in terms of efficiency, but diverges further from NH:

NH - Polynomial (PH) In this construction NH is redefined with message and key blocks as polynomials over $GF(2)$ instead of integers:

Definition 4: Given $M = (m_1, \cdots, m_n)$ and $K = (k_1, \cdots, k_n)$, where m_i and $k_i \in P_w$, for any even $n \geq 2$, PH is defined as follows:

$$\mathsf{PH}_K(M) = \sum_{i=1}^{n/2} (m_{2i-1} + k_{2i-1})(m_{2i} + k_{2i}) \ .$$

In a hardware implementation this completely eliminates the carry chain and thereby improves all three efficiency metrics (i.e. speed, space, power) simultaneously. That is, due to the elimination of carry propagations, the operable clock frequency (and thus the speed of the hash algorithm) is dramatically increased. Likewise, the area efficiency is improved since the carry network is eliminated. Finally, due to the reduced switching activity, the power consumption is reduced.

NH-Polynomial with Reduction (PR) The size of the authentication tag is a concern for two reasons. The tag needs to be transmitted along with the data therefore the shorter the tag, the less energy will be consumed for its transmission. The energy consumed by transmitting a single bit can be as high as the energy needed to perform the entire hash computation on the node. The energy needed for transmitting the tag is proportional to its bit-length. Secondly, the size of the tag determines the number of flip-flops needed for storing the tag. The original NH as well as PH introduced above require a large number of flip-flops for the double length hash output. In this construction, the storage and transmission requirement is improved by introducing a reduction polynomial of degree matching the block size, hence reducing the size of the authentication tag by half.

Definition 5: Given $M = (m_1, \cdots, m_n)$ and $K = (k_1, \cdots, k_n)$, where m_i and $k_i \in GF(2^w)$, for any even $n \geq 2$, and a polynomial p of degree w irreducible over $GF(2)$, PR is defined as follows:

$$\mathsf{PR}_K(M) = \sum_{i=1}^{n/2} (m_{2i-1} + k_{2i-1})(m_{2i} + k_{2i}) \pmod p \ .$$

Note that the original NH construction eliminates the modular reductions used in the previously proposed hash constructions (e.g. MMH proposed in [20], SQUARE proposed in [23]) since reductions are relatively costly to implement in software. In hardware, however, reductions (especially those with fixed low-weight polynomials) can be implemented quite efficiently.

Weighted NH-Polynomial with Reduction (WH) While processing multiple blocks, it is often necessary to hold the hash value accumulated during the previous iterations in a temporary register. This increases the storage requirement and translates into a larger and slower circuit with higher power consumption. As a remedy we introduce a variation of NH where each processed block is scaled with a power of x. This function is derived from the changes we make to PR which are described in Section V-D.

Definition 6: Given $M = (m_1, \cdots, m_n)$ and $K = (k_1, \cdots, k_n)$, where m_i and $k_i \in GF(2^w)$, for any even $n \geq 2$, and an irreducible polynomial $p \in GF(2^w)$, WH is defined as follows:

$$\mathsf{WH}_K(M) = \sum_{i=1}^{n/2} (m_{2i-1} + k_{2i-1}) \\ \cdot (m_{2i} + k_{2i}) \, x^{(\frac{n}{2} - i)w} \pmod p \ .$$

Due to the scaling with the factor $x^{(\frac{n}{2} - i)w}$, perfect serialization is achieved in the implementation where the new block product is accumulated in the same register holding the

hash of the previously processed blocks. This eliminates the need for an extra temporary register as well as other control components required to implement the data path.

IV. Analysis

In this section we give three theorems establishing the security of the NH variants. The proofs of the first two follow easily from the proof of NH given in [1]. Therefore, we include the proof of Theorem 3 for WH only.

Theorem 1: For any even $n \geq 2$ and $w \geq 1$, $\mathsf{PH}[n, w]$ is 2^{-w}-almost universal on n equal-length strings.

Theorem 2: For any even $n \geq 2$ and $w \geq 1$, $\mathsf{PR}[n, w]$ is universal on n equal-length strings.

Theorem 3: For any even $n \geq 2$ and $w \geq 1$, $\mathsf{WH}[n, w]$ is universal on n equal-length strings.

Proof: For brevity we denote $(m_{2i-1} + k_{2i-1})(m_{2i} + k_{2i}) = mk_{2i}$, $(m'_{2i-1}+k_{2i-1})(m'_{2i}+k_{2i}) = m'k_{2i}$ and so on. Let M, M' be distinct members of the domain A with equal lengths. We are required to show that

$$Pr\left[\mathsf{WH}_K(M) = \mathsf{WH}_K(M')\right] = 2^{-w} \ .$$

Expanding the terms inside the probability expression, we obtain

$$Pr\left[\sum_{i=1}^{n/2} mk_{2i}\left(x^{(\frac{n}{2}-i)w}\right) = \sum_{i=1}^{n/2}(m'k_{2i}\left(x^{(\frac{n}{2}-i)w}\right) \pmod p\right] = 2^{-w} \ . \quad (1)$$

The probability is taken over uniform choices of $(k_1, \ldots, k_n)$ with each $k_i \in GF(2^w)$ and the arithmetic is over $GF(2^w)$. Since M and M' are distinct, $m_i \neq m'_i$ for some $1 \leq i \leq n$. Let $m_{2l} \neq m'_{2l}$. For any choice of $k_1, \ldots, k_{2l-2}, k_{2l}, \ldots, k_n$ having

$$Pr_{k_{2l-1}\in GF(2^w)}\left[\sum_{i=1}^{n/2} mk_{2i}\left(x^{(\frac{n}{2}-i)w}\right) = \sum_{i=1}^{n/2} m'k_{2i}\left(x^{(\frac{n}{2}-i)w}\right) \pmod p\right] = 2^{-w} \quad (2)$$

satisfied for all $1 \leq l \leq n/2$ implies (1). Setting y and z as

$$y = \left[\sum_{i=1}^{l-1} m'k_{2i}x^{(\frac{n}{2}-i)w} - \sum_{i=1}^{l-1} mk_{2i}x^{(\frac{n}{2}-i)w}\right] \pmod p$$

and

$$z = \left[\sum_{i=l+1}^{n/2} m'k_{2i}x^{(\frac{n}{2}-i)w} - \sum_{i=l+1}^{n/2} mk_{2i}x^{(\frac{n}{2}-i)w}\right] \pmod p$$

we rewrite the probability bound in (2) as

$$Pr_{k_{2l-1}}\left[x^{(\frac{n}{2}-l)w}\left[mk_{2l} - m'k_{2l}\right] = y + z \pmod p\right] = 2^{-w} \ .$$

Since $x^{(\frac{n}{2}-l)w}$ is invertible in $GF(2^w)$, the equation inside the probability expression can be rewritten as follows.

$$k_{2l-1}(m_{2l} - m'_{2l}) + m_{2l-1}(m_{2l} + k_{2l}) - m'_{2l-1}(m'_{2l} + k_{2l}) = x^{-(\frac{n}{2}-l)w}(y + z) \pmod p$$

Solving the equation for k_{2l-1}, we end up with the following

$$k_{2l-1} = (m_{2l} - m'_{2l})^{-1}\Big((x^{-(\frac{n}{2}-l)w})(y + z) - m_{2l-1}(m_{2l} + k_{2l}) + m'_{2l-1}(m'_{2l} + k_{2l})\Big) \pmod p \ .$$

Note that $(m_{2l} - m'_{2l})$ is invertible since in the beginning of the proof we assumed that $m_{2l} \neq m'_{2l}$. This proves that for any m_{2l}, m'_{2l} (with $m_{2l} \neq m'_{2l}$) and y, $z \in GF(2^w)$ there exists exactly one $k_{2l-1} \in GF(2^w)$ which causes a collision. Therefore,

$$Pr\left[\mathsf{WH}_K(M) = \mathsf{WH}_K(M')\right] = 2^{-w} \ .$$

□

V. Implementation Details

The power dissipation in CMOS devices can be summarized by the following formula [27]:

$$P = \underbrace{\left(\frac{1}{2} \cdot C \cdot V_{DD}^2 + Q_{se} \cdot V_{DD}\right) \cdot f \cdot N}_{\mathrm{P_{Dynamic}}} + \underbrace{I_{leak} \cdot V_{DD}}_{\mathrm{P_{Leakage}}} \quad (3)$$

The term $\mathrm{P_{Dynamic}}$ represents the power required to charge and discharge circuit nodes as well as the power dissipation during output transitions. The terms C, Q_{se}, and V_{DD} are technology dependent [27]. The switching activity, i.e. the number of gate output transitions per clock cycle, is represented by N and the operating frequency by f. The second term $\mathrm{P_{Leakage}}$ represents the static power dissipation due to the leakage current I_{leak}. The leakage current is directly determined by the number of gates and the fabrication technology. For more information about low-power design see [28]. In order to minimize the power consumption, we designed our CMOS circuits according to the following rules:

- The number of transitions ('0' to '1' and '1' to '0') has to be minimal.
- The circuit size should be minimized.
- Glitches cause unnecessary transitions and therefore should be avoided.

A. NH

The algorithm for NH is described in [1] and is given in this paper in Definition 3. It leads to the simplified block diagram shown in Figure 1. The actual block diagram for the circuit is much more complex and can be found in Figure 2. The message and the key are assumed to be split into n blocks of w bits. Messages that are shorter than a multiple of $2 \cdot w$ are padded. All odd message blocks are applied to input $m1$, all even message blocks to input $m2$. The blocks of the key are applied similarly to $k1$ and $k2$. The final adder accumulates all $n/2$ products.

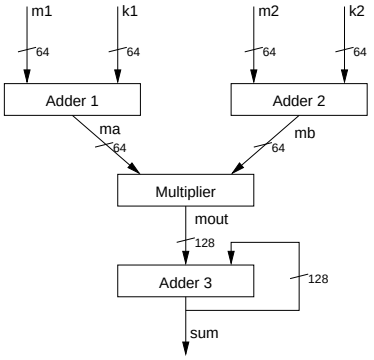

Fig. 1. Simplified Functional diagram for NH

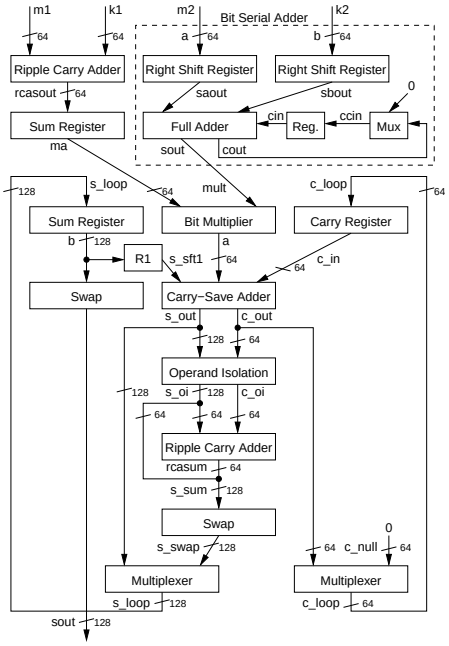

Fig. 2. Block diagram for NH

The output of Adder 1 is $ma = m1 + k1 \bmod 2^w$, the output of Adder 2 is $mb = m2 + k2 \bmod 2^w$. These are integer additions where the carry out is discarded. The multiplication results in $mout = ma \cdot mb$. For each multiplication of two w-bit numbers, w partial products need to be computed and added: $mout = \sum_{j=1}^{w} ma \cdot mb[j] \cdot 2^{j-1}$. As power consumption is our main concern and not speed we chose to implement a bit serial multiplier. It computes one partial product during each clock cycle and adds it to the sum of the previous partial products using the Right Shift Algorithm [29].

A bit serial adder produces one bit of the result with each clock cycle, starting with the LSB and it has minimal glitching. We used a bit serial adder for Adder 2 as its result can directly be used by the bit serial multiplier. However, the multiplicand has to be available immediately. Therefore we used a simple ripple carry adder to implement Adder 1. Its main disadvantage is that it takes a long time until the carries propagate through the adder, causing a lot of glitching and therefore a high power consumption. However, Adder 1 needs to compute a new result only every 64 clock cycles, hence its dynamic power consumption is tolerable. The addition of the partial products is accomplished using a carry-save adder. This adder uses the redundant carry-save notation which results in minimal glitching as the carries are not fully propagated. However, 64 additional flip-flops are required to store the carry bits.

After one multiplication has been computed, its result has to be added to the accumulation of the previous multiplications as indicated by Adder 3 in Figure 1. Rather than having a separate multiplier and adder, in the actual implementation we add the partial products of the next multiplication immediately to the result of the previous additions. This technique stores the result of Adder 3 in the Multiplier thus saving a 128 bit register and a 128 bit multiplexer.

The control logic manages the switching of the multiplexers, loading of the next data set and resetting the carry registers. Due to the iterations of the multiplication, the control logic requires a counter. Traditionally, counters are built using a register and a combinational incrementer. The incrementer requires carry propagations which cause glitching. Furthermore, the delay of the incrementer contributes to the critical path delay of the circuit. Hence optimization of this unit is essential. Instead of an integer counter, we use a linear feedback shift register (LFSR) with 6 flip-flops, enhanced to "count" up to 64. LFSRs have minimal glitching and therefore make power efficient and fast counters.

We use this implementation of the original NH algorithm as a reference for comparison with its variations described below.

B. NH-Polynomial (PH)

The main power consumers in the implementation of NH are the ripple carry adders and flip-flops needed for the multiplier and the bit serial adder. PH is a variation on NH in that it uses polynomials over $GF(2)$ instead of integers. This replaces the costly adders with simple XOR gates which consume significantly less power. Adder 1 is replaced by 64 XOR gates. The bit-serial adder (Adder 2) is replaced by XORs and a 64 bit shift register which reduces the complexity of this unit by 64 flip-flops. The Multiplier and Adder 3 are combined as in NH but the carry-save adders are replaced by XORs. As there are no carries another 64 flip-flops are saved. Just changing NH from using integers to polynomials reduces the number of cells by 65%, the dynamic power consumption by 38% and the leakage power by more than a half.

C. NH-Polynomial with Reduction (PR)

The main difference between PR and PH is that the result is reduced to 64 bits using an irreducible polynomial. In our hardware implementation the multiplication and the reduction are interleaved. This makes the reduction very efficient. Moreover, using low Hamming-weight polynomials the reduction can be achieved with only a few gates and minimal extra delay. However, the Multiplier and Adder 3 can no longer be merged.

Therefore, we are not able to reduce the number of flip-flops in our implementation but we reduced the switching activity as Adder 3 computes a new result only once every 64 clock cycles. The number of cells for this implementation is slightly higher than for PH and thus the leakage power is increased. Due to the reduced switching activity, however, the dynamic power consumption is now 50% less then that of NH.

D. Weighted NH-Polynomial with Reduction (WH)

This design was inspired by the bottlenecks we observed in the implementation of PR. For instance, the Multiplier and Adder 3 (Figure 1) could not be merged as in PH. We removed from PR's implementation a 64 bit register, a 64 bit multiplexer and the XOR gates of Adder 3. The function of the resulting design is characterized by the construction shown in Definition 6. Compared to NH, the removal of the mentioned components reduced the dynamic power consumption by 59%, the leakage power consumption by 66%, and the number of cells by 74%. This dramatic savings become more obvious when the block diagrams for NH in Figure 2 and for WH in Figure 3 are compared.

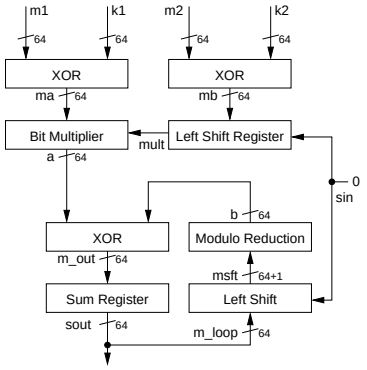

Fig. 3. Block diagram for WH

VI. Implementation Results

For synthesizing our designs we used the Synopsys tools Design Compiler [30] and Power Compiler [31], and the TSMC $0.13\,\mu\mathrm{m}$ ASIC library. The results of the simulation on many input sets were verified with the Maple package [32] for consistency. Table I lists power, area, and delay results of the hash function implementations, synthesized for operation at 100 MHz. The *maximum delay* determines the highest operable frequency.

The dynamic power consumption of WH is $452.3\,\mu\mathrm{W}$ at 100 MHz. This is much higher than our aim of $20\,\mu\mathrm{W}$. The CMOS power formula in Equation 3 shows that the dynamic power consumption is directly proportional to the operating frequency. Hence, the implementations consume $1/200^{th}$ of the dynamic power when clocked at 500 kHz, however, the leakage power remains the same. This lower frequency is used in sensor node implementations [8]. Table II demonstrates that at low speeds the leakage power becomes the limiting factor for ultra-low-power implementations. WH can operate with as little as $11.6\,\mu\mathrm{W}$. This is in the range of the power produced by a MEMS scavenger [9]. We would like to note that we used an ASIC standard cell library to obtain these results. A full custom IC-design would yield even higher power savings.

TABLE I
Comparison of Hash Implementations at 100 MHz

	Dynamic Power		Leakage Power		Number of Cells		Delay/ Speedup	
	μW	%	μW	%		%	ns	x
NH	1093.9	100	28.1	100	1576	100	9.92	1.0
PH	682.7	62	12.1	43	557	35	1.35	7.4
PR	549.9	50	14.0	50	616	39	1.35	7.4
WH	452.3	41	9.4	33	412	26	1.35	7.4

TABLE II
Power Consumption at 500 kHz

	Dynamic		Leakage		Total	
	μW	%	μW	%	μW	%
NH	5.47	100	28.1	100	33.6	100
PH	3.41	62	12.1	43	15.5	46
PR	2.75	50	14.0	50	16.8	50
WH	2.26	41	9.4	33	**11.6**	35

VII. Conclusion

In this paper, we propose three variations on NH (the underlying hash function of UMAC), namely PH, PR and WH. Our main motivation was to prove that universal hash functions can be employed to provide provable security in ultra-low-power applications such as next generation sensor networks. More specifically, hardware implementations of universal hash functions with an emphasis on low-power and reasonable execution speed are considered. We implemented NH for the first time in hardware and presented its simulation results in comparison to those of our newly proposed hash functions.

The first hash function we propose, i.e. PH, produces a hash of length $2w$ and is shown to be 2^{-w}-almost universal. The other two hash functions, i.e. PR and WH, reach optimality and are shown to be universal hash functions with a much shorter hash length of w. Since their combinatorial properties are mathematically proven, there is no need for making cryptographic hardness assumptions and using a safety margin in practical implementations. In addition, these schemes are simple enough to allow for efficient constructions.

To our knowledge the proposed hash functions are the first ones specifically designed for efficient hardware implementations. Designing the new algorithms with efficiency guidelines in mind and applying optimization techniques, we achieved drastic power savings of up to 59% and speedup of up to 7.4 times over NH. Note that the speed improvement and the power reduction are accomplished simultaneously. We also observed that at lower operating frequencies the leakage power becomes the dominant part in the overall power consumption. Our implementation of WH consumes only $11.6\,\mu\mathrm{W}$ at

500 kHz. It could therefore be integrated into a self-powered device. This enables the use of universal hash functions in ultra-low-power applications such as "Smart Dust" motes, RFIDs. By virtue of the security and implementation features mentioned above, we believe that the proposed universal hash functions will fill an important gap in cryptographic hardware applications.

VIII. Future Work

As mentioned in Section VI when clocked at 500 kHz the leakage power is the dominant part of the power consumption. The most effective way to reduce the leakage power is to reduce the circuit size. The circuit size scales with the data path width, i.e. the block size w of the message and the key. As the collision probability is upper bounded by 2^{-w} (see Section IV), reducing the block size w will significantly increase the bound. A possible solution to reduce the circuit size while still preserving the security level is to employ multiple hashing. For this, each message block is hashed multiple times using independent key blocks and the individual hash values are concatenated to form the hash output. For instance, to obtain the collision probability of 2^{-w} with a block size of $w/4$ bits, each message block is hashed 4 times with independent keys. The computed hash outputs are concatenated to form the w bit hash result. The drawback of this method is that it requires 4 times the amount of key material. As a remedy one can employ the well-known Toeplitz approach [1], [18], [33] in which shifted versions of one key rather than independent keys are used. We identify the application of the Toeplitz construction to the proposed hash functions with security analysis and implementation as future work.

References

[1] J. Black, S. Halevi, H. Krawczyk, T. Krovetz, and P. Rogaway, "UMAC: Fast and secure message authentication," in *Advances in Cryptology - CRYPTO '99*, ser. Lecture Notes in Computer Science, vol. 1666. Springer-Verlag, 1999, pp. 216–233.

[2] B. Fulford, "Sensors gone wild," Forbs Global, Oct 2002, http://www.forbes.com/global/2002/1028/076_print.html.

[3] J. Polastre, "Design and implementation of wireless sensor networks for habitat monitoring," Master's Thesis, University of California at Berkeley, Spring 2003.

[4] A. Mainwaring, J. Polastre, R. Szewczyk, D. Culler, and J. Anderson, "Wireless sensor networks for habitat monitoring," in *First ACM Workshop on Wireless Sensor Networks and Applications*, Atlanta, GA, USA., Sep 2002.

[5] R. Burne *et al.*, "Self-organizing cooperative sensor network for remote surveillance: improved target tracking results," in *Proceedings of the SPIE - The International Society for Optical Engineering*, vol. 4232, SPIE. Boston: SPIE-Int. Soc. Opt. Eng, USA, 2001, pp. 313–321.

[6] J. Kahn, K. R. H., and K. Pister, "Next century challenges: mobile networking for "smart dust"," in *Proceedings of the fifth annual ACM/IEEE international conference on Mobile computing and networking.* ACM, 1999, pp. 271–278.

[7] P. Gorder, "Sizing up smart dust," *Computing in Science & Engineering*, vol. 5, no. 6, pp. 6–9, Nov.-Dec. 2003.

[8] R. Amirtharajah and A. P. Chandrakasan, "Self-powered signal processing using vibration-based power generation," *IEEE Journal of Solid-State Circuits*, vol. 33, no. 5, pp. 687–695, May 1998.

[9] S. Meininger, J. O. Mur-Miranda, R. Amirtharajah, A. P. Chandrakasan, and J. H. Lang, "Vibration-to-electric energy conversion," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 9, no. 1, pp. 64–76, Feb 2001.

[10] D. Carman, P. Kruus, and B. Matt, "Constraints and approaches for distributed sensor network security," NAI Labs, Secutity Research Division, Glenwood, MD, Technical Report, Sep 2000.

[11] A. Perrig, R. Szewczyk, J. D. Tygar, V. Wen, and D. E. Culler, "SPINS: security protocols for sensor networks," *Wireless Networks*, vol. 8, no. 5, pp. 521–534, Sep 2002.

[12] W. Diffie and M. E. Hellman, "New Directions in Cryptography," *IEEE Transactions on Information Theory*, vol. IT-22, pp. 644–654, 1976.

[13] G. J. Simmons, Ed., *Contemporary Cryptology*. IEEE Press, 1992.

[14] J. L. Carter and M. Wegman, "Universal classes of hash functions," *Journal of Computer and System Sciences*, vol. 18, pp. 143–154, 1978.

[15] J. Carter and M. Wegman, "New hash functions and their use in authentication and set equality," *Journal of Computer and System Sciences*, vol. 22, pp. 265–279, 1981.

[16] G. Brassard, "On computationally secure authentication tags requiring short secret shared keys," in *Advances in Cryptology - CRYPTO '82*, ser. Lecture Notes in Computer Science, D. Chaum, R. L. Rivest, and A. T. Sherman, Eds. New York: Springer-Verlag, 1983, pp. 79–86.

[17] M. Ramakrishna, E. Fu, and E. Bahcekapili, "A performance study of hashing functions for hardware applications," in *Proceedings of the ICCT '94 International Conference on Computing and Information*, 1994, pp. 1621–1636.

[18] H. Krawczyk, "LFSR-based hashing and authentication," in *Advances in Cryptology - Crypto'94*, ser. Lecture Notes in Computer Science, vol. 839. Springer-Verlag, 1994, pp. 129–139.

[19] V. Shoup, "On fast and provably secure message authentication based on universal hashing," in *Advances in Cryptology - CRYPTO '96*, ser. Lecture Notes in Computer Science, vol. 1109. New York: Springer-Verlag, 1996, pp. 74–85.

[20] S. Halevi and H. Krawczyk, "MMH: Software message authentication in the gbit/second rates," in *4th Workshop on Fast Software Encryption*, ser. Lecture Notes in Computer Science, vol. 1267. Springer, 1997, pp. 172–189.

[21] P. Rogaway, "Bucket hashing and its applications to fast message authentication," in *Advances in Cryptology - CRYPTO '95*, ser. Lecture Notes in Computer Science, vol. 963. New York: Springer-Verlag, 1995, pp. 313–328.

[22] H. Krawczyk, "New hash functions for message authentication," in *EUROCRYPT'95*, ser. Lecture Notes in Computer Science, vol. 921. Springer-Verlag, 1995, pp. 301–310.

[23] M. E. S. Patel and Z. Ramzan, "SQUARE HASH: Fast message authentication via optimized universal hash functions," in *Advances in Cryptology - CRYPTO '99*, ser. Lecture Notes in Computer Science, M. Wiener, Ed., vol. 1666. New York: Springer-Verlag, 1999, pp. 234–251.

[24] W. Nevelsteen and B. Preneel, "Software performance of universal hash functions," in *EUROCRYPT'99*, ser. Lecture Notes in Computer Science, vol. 1592. Berlin: Springer-Verlag, 1999, pp. 24–41.

[25] P. Levis and D. Culler, "Mat: a tiny virtual machine for sensor networks," in *Proceedings of the 10th international conference on architectural support for programming languages and operating systems (ASPLOS-X)*. San Jose, California: ACM Press, 2002, pp. 85–95.

[26] S. Sarma, D. L. Brock, and K. Ashton, "The networked physical world - proposals for engineering the next generation of computing, commerce & automatic identification," MIT: Auto-ID Center," White Paper, Oct 2000.

[27] S. Devadas and S. Malik, "A survey of optimization techniques targeting low power vlsi circuits," in *Proceedings of the 32nd ACM/IEEE Conference on Design Automation*, 1995, pp. 242–247.

[28] J. Rabaey and M. Pedram, *Low Power Design Methodologies*. Norwell, Massachusetts: Kluwer Academic Publishers, 1996.

[29] B. Parhami, *Computer Arithmetic: Algorithms and Hardware Designs*. Oxford University Press, 2000.

[30] *Design Compiler User Guide*, Version 2002.05 ed., Synopsys Inc., Jun 2002.

[31] *Power Compiler User Guide*, Release 2002.05 ed., Synopsys Inc., May 2002.

[32] K. M. Heal, M. L. Hansen, and K. M. Rickard, *Maple V Learning Guide*. New York: Springer Verlag, 1998.

[33] Y. Mansour, N. Nissan, and P. Tiwari, "The computational complexity of universal hashing," in *22nd Annual ACM Symposium on Theory of Computing*. ACM Press, 1990, pp. 235–243.