Energy Comparison of AES and SHA-1 for Ubiquitous Computing

Jens-Peter Kaps and Berk Sunar

{kaps, sunar}@wpi.edu
Department of Electrical & Computer Engineering
Worcester Polytechnic Institute
100 Institute Road, Worcester, MA 01609, U.S.A.

Abstract. Wireless sensor networks and Radio Frequency Identifiers are becoming mainstream applications of ubiquitous computing. They are slowly being integrated into our infrastructure and therefore must incorporate a certain level of security. However, both applications are severely resource constrained. Energy scavenger powered sensor nodes and current RFID tags provide only 20 μW to 50 μW of power to the digital component of their circuits. This makes complex cryptography a luxury. In this paper we present a novel ultra-low power SHA-1 design and an energy efficient ultra-low power AES design. Both consume less than 30 μW of power and can therefore be used to provide the basic security services of encryption and authentication. Furthermore, we analyze their energy consumption based on the TinySec protocol and come to the somewhat surprising result, that SHA-1 based authentication and encryption is more energy efficient than using AES for payload sizes of 17 bytes or larger.

1 Motivation

Not long ago, ubiquitous computing was just a buzzword. Technologies like Radio Frequency Identifiers (RFID) and Wireless Sensor Networks (WSN) are just a few examples that show that embedded and ubiquitous computing has come a long way from hot idea to mass deployment. Computing devices are now embedded into clothing and other products in the form of RFID tags. WSN are still mainly used by researchers but are expected to migrate into mainstream applications like building, health, and environmental monitoring, military target tracking and so on in the near future. Embedded and ubiquitous computing will soon form a crucial part of our infrastructure. Securing this infrastructure is critical.

The most basic security services are privacy, integrity, and authenticity which can be achieved with classic message authentication codes (MAC) and encryption functions. However, WSN nodes and RFID tags impose sever power constraints which make it difficult to realize computationally intensive cryptographic functions. Passive RFID tags are powered by the electro magnetic field from the

the reader and only 20 μW are available to the digital part of the tag[1]. Wireless sensor nodes are currently battery powered but battery replacement poses a major hindrance to scaling wireless sensor networks to thousands of nodes and to deploying them in inaccessible places. We envision that the next generation sensor nodes will be powered by scavengers, which collect energy from environmental sources such as light, radiation, vibration, etc. Micro-Electro-Mechanical Systems (MEMS) based power scavengers can be integrated on chip, which will decrease cost, and can produce up to 8 μW of power [1]. Future MEMS-based scavengers are expected to deliver up to 50 μW continously, enough to power an ultra-low power circuit.

Power consumption in CMOS devices is the sum of leakage power P_{Leak}, which is caused by the leakage current of each gate, and dynamic power P_{Dyn}, caused by switching activity. Therefore, P_{Leak} is proportional to the circuit size and P_{Dyn} is proportional to the clock frequency and switching activity. We observed that at a frequency of 500 kHz, which is used in sensor network implementations [2], leakage power becomes dominant. In order to conserve leakage power we have to reduce the circuit size. A common method to save hardware resources and provide privacy, integrity, and authentication is to use the same cryptographic algorithm for both functions, MAC computation and encryption. SPINS [3] for example, uses RC5 [4] for encryption and in CBC-mode to build a secure MAC. TinySec [5] is cipher independent and was tested with RC5 and Skipjack [6, 7] for encryption and CBC-MAC. The authors of [5] are also considering the Advanced Encryption Standard [8].

Many research papers [3, 9, 10] analyze encryption algorithms for wireless sensor networks exclusively with reference to speed and code size while only a few [11] address the energy consumption of software based implementations. However, the ultra-low power applications we are envisioning, do not provide enough power for running cryptographic algorithms on general purpose microprocessors.

In this paper we are presenting hardware implementations of the advanced encryption standard (AES) and the Secure Hash Algorithm (SHA-1) [12] which are optimized for ultra-low power applications. We are then examining encryption and authentication functions based on AES and SHA-1. To our knowledge this is the first ultra-low power implementation of SHA-1 and the first publication describing the use of SHA-1 for ubiquitous computing on ultra-low power platforms. We are comparing SHA-1 and AES based encryption and authentication functions with respect to their footprint, speed, power and energy consumption. The only paper that compared SHA-1 to AES is [13], however the comparison is only concerned with throughput.

The remainder of the paper is structured as follows. After a short introduction to AES, CBC-MAC, SHA-1 and SHACAL in Section 2, we describe

[1] A simple, passive RFID tag typically consists of an analog and a digital section. The analog section is responsible for powering the tag and wirelessly sending and receiving data. The digital section contains a tiny microcontroller and some memory to store the unique identifier.

our implementations of SHA-1 in Section 3 and AES in Section 4. We present and analyze the results of our implementations in Section 5. The final section (Section 6) concludes our findings.

2 Cryptographic functions for ultra-low power applications

We use AES and SHACAL-1 for encryption and AES in CBC-MAC mode and HMAC [14, 15] with SHA-1 for authentication. Figure 1 shows a top level view of the AES and SHA-1 based encryption and authentication functions.

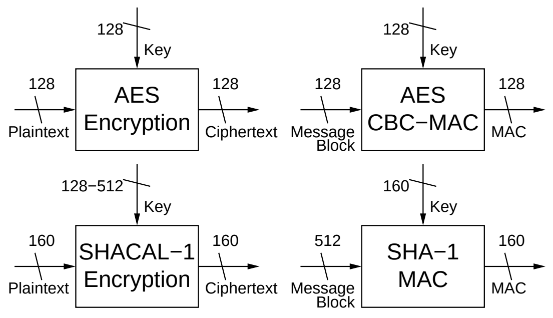

Fig. 1. Encryption and MAC functions based on AES and SHA-1

2.1 AES

AES was selected by the National Institute of Standards and Technology (NIST) as Federal Information Processing Standard FIPS-197 [8] in 2001. Since then, many hardware implementations have been published. Most of them are optimized for speed and only a few are scalable [16, 17] from fast to small. The first ultra-low power implementation was reported in [18] followed by [19] by the same group, both papers analyze the power consumption but not the energy consumption of the circuits. AES is a block cipher with a fixed input size of 128 bits and a key length of either 128 bits, 192 bits, or 256 bits. For our ultra-low power implementation we chose 128 bits. AES applies the same round function ten times to its input, also called State, during encryption. The round function consists of four different transformations: SubBytes, ShiftRows, MixColumns, and AddRoundKey, each changing the State by applying linear, non linear and key dependent transformations.

2.2 SHA-1

SHA-1 is the most widely used secure hash function and was developed by the National Security Agency. Its security level is considered to be 2^{80}, i.e., 2^{80} operations have to be made on average to find another input such that the resulting hashes are equal, also called a collision. Recent attacks on SHA-1 [20] indicate that there might be a potential weakness but no collisions have been found yet. Many implementations of SHA-1 have been reported, most of them optimized for speed. To our knowledge, this is the first ultra-low power implementation of SHA-1. SHA-1 computes a 160-bit hash of messages up to 2^{64} bits in size. Each message needs to be preprocessed by padding the message, appending the message length and splitting it into blocks with a length of 512 bits each. Then the compression function processes each input block and computes intermediate hash values by iterating over simple functions 80 times.

2.3 Message Authentication Codes

SHA-1 can be used to build a message authentication code by introducing a secret 512-bit key K using the secret prefix method SHA-1($K||x$). Due to this concatenation SHA-1 will compute an intermediate hash value of K in the first iteration which can be precomputed. Hence, computation of a MAC requires $\lceil \text{length}(x)/512 \rceil$ operations. However, the secret prefix method is considered insecure [21] even though SHA-1 includes the message length in the hash and TinySec reveals only half of the hash result. We therefore suggest to use HMAC, which is formally described in [22] as

$$\text{HMAC}_k(x) = \text{SHA-1}((\overline{k} \oplus \mathit{opad}) \,||\, \text{SHA-1}((\overline{k} \oplus \mathit{ipad}) \,||\, x)).$$

The 160-bit key K is padded with 0's resulting in $\overline{k}$. The terms $\overline{k} \oplus \mathit{opad}$ and $\overline{k} \oplus \mathit{ipad}$ can be precomputed from the 512-bit constants *opad* and *ipad* and $\overline{k}$. Due to the concatenation $((\overline{k} \oplus \mathit{ipad}) \,||\, x)$ the intermediate hash value of $(\overline{k} \oplus \mathit{ipad})$ and $(\overline{k} \oplus \mathit{opad})$ can be precomputed as well. Hence, computation of a MAC requires $\lceil \text{length}(x)/512 \rceil + 1$ operations of SHA-1.

AES can be used in CBC-MAC [23] mode (see Fig. 2) to compute authentication codes. This mode is similar to the *Cipher Block Chaining* mode [24, 25] in that the result from the previous encryption is XORed with the next plaintext block and encrypted again. The intermediary ciphertexts however are not used in CBC-MAC mode. The computation of a MAC requires $\lceil \text{length}(x)/128 \rceil$ operations. The level of security of AES in this mode is approximately 2^{64}, and not 2^{128} as one might expect, due to the birthday attack[2].

2.4 Encryption

To some extent, hash functions like SHA-1 can also be used to perform encryption. The best examples are SHACAL [26] and SHACAL-1 [27]. The security of

[2] If a sensor node produces one message authentication code per second, than it would produce only 2^{32} in 100 years.

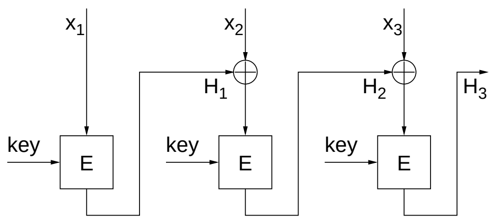

Fig. 2. CBC-MAC – Generating a hash with a block cipher

SHACAL was analyzed in [28] and more recently in [29]. SHACAL defines how the compression function of SHA-1 can be used as a 160-bit block cipher with a 512-bit secret key. Shorter keys can be used by padding the key with zeroes but the minimum key size is 128 bits. AES is a block cipher so its usage for encryption is straight forward.

3 SHA-1 Implementation

The top level block diagram of our SHA-1 implementation is shown in Figure 3. We assume that one 512-bit block of preprocessed data is stored in memory and available to our SHA-1 unit.

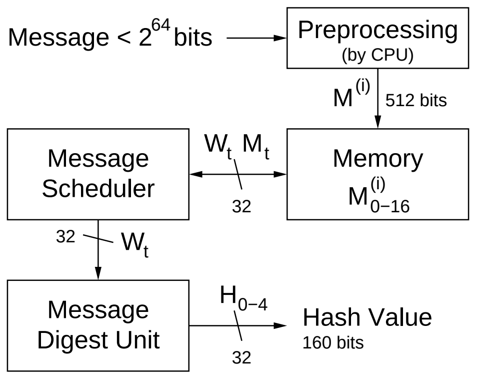

Fig. 3. Top Level Block Diagram of our SHA-1 Implementation

Our SHA-1 implementation incorporates the *Message Scheduler* and the *Message Digest Unit* as well as a memory bus interface and the necessary control logic. The operation is broken down into three stages. The initial stage comprises the first 16 rounds. Here, the message scheduler reads the message block one M_t per round. The next stage is the computation stage which ends with the

80th round. During both stages, the message scheduler computes $\mathrm{W_t}$ and forwards it to the message digest unit and also stores $\mathrm{W_t}$in the external memory. The message digest unit performs the message compression function. The final stage is needed to compute the final hash values from the intermediate hash.

3.1 Message Scheduler

During the computation stage the message scheduler has to compute a new $\mathrm{W_t}$ value in each round based on previously calculated $\mathrm{W_t}$'s. Most implementations in literature use a 16 stage 32-bit wide shift register for this purpose (512 flip-flops). For our ultra-low power implementation we re-use the memory that contains the message assuming that we can overwrite the existing contents. The message scheduler is able to interface with external memory and needs only one 32-bit register to store a temporary value during computation of the new $\mathrm{W_t}$. Figure 4 shows the block diagram of the message scheduler. The control logic, which handles the bus control signals, is not shown for simplicity. The message

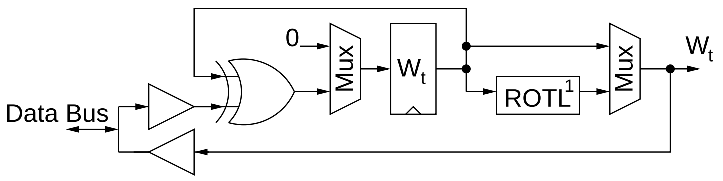

Fig. 4. Block Diagram of the Message Scheduler

scheduler performs the equation

$$W_t = ROTL^1\left(W_{t-3} \oplus W_{t-8} \oplus W_{t-14} \oplus W_{t-16}\right)$$

where $\oplus$ denotes bitwise XOR. Four values have to be read from memory and the result written back to memory in each round. This takes 5 clock cycles in our serial design, therefore, each round of SHA-1 takes 5 clock cycles. The necessary address computation (not shown in Figure 4) is done using dedicated hard wired adders to provide +2, +8 and +13 addition modulo 16 for W_{t-14}, W_{t-8}, and W_{t-3} respectively.

3.2 Message Digest Unit

Figure 5 shows the functional block diagram of the message digest unit as described in the SHA-1 standard [12]. SHA-1 requires five 32-bit working variables (a, b, c, d, e) to which new values are assigned in each round. It can easily be seen that four out of the five words are shifted in each round ($a \rightarrow b, \cdots, d \rightarrow e$) and only determining the new value for a requires computation. The rotation of

$\mathrm{ROTL}^{30}(b) \to c$ can be accomplished by wiring. Therefore, we view the registers for the working variables as a 5 stage 32-bit wide shift register in our ultra-low power SHA-1 implementation.

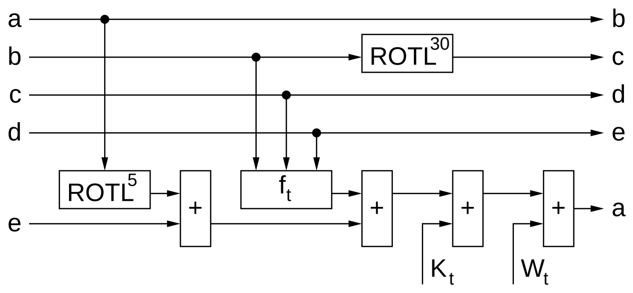

Fig. 5. Functional Block Diagram of the Message Digest Unit

Round Function The round function computes a new value for a and shifts all working variables once per round. The computation for a is a five operand addition modulo 2^{32} where the operands depend on all input words, the round-dependent constant $\mathrm{K_t}$, and the current message word $\mathrm{W_t}$. In order to conserve area and therefore limit the leakage power, we use a single 32-bit adder to perform the four additions. Due to good scheduling of the operations, we can use the register for e also as temporary register for the additions. The four additions and the shift require 4 clock cycles per round which is below the need of the message scheduler with 5 clock cycles to compute the next $\mathrm{W_t}$. Figure 6 shows the block diagram of our implementation of the message digest unit including the round function and the intermediate hash value computation.

Intermediate Hash Value Computation During the final stage, i.e., after the 80^{th} round, the values of the working variables have to be added to the digest of the previous message blocks, or specific initial values for the first message block. This can be done very efficiently without additional multiplexers or adders by arranging all intermediate hash value registers $\mathrm{H_0}$, $\mathrm{H_1}$, $\mathrm{H_2}$, $\mathrm{H_3}$, and $\mathrm{H_4}$ in a 5 stage 32-bit wide shift register, similar to our design for the working variables. Shifting the hash value registers and the working variable registers at the same time and adding the current contents of e to $\mathrm{H_4}$ at each step takes five clock cycles. This again fits into our scheme of 5 clock cycles per round, which leads to a total of 405 clock cycles for the message digest computation of one block.

4 AES Implementation

For our AES implementation we assume that a message block and the private key are stored in memory. The result of the AES computation gets written back

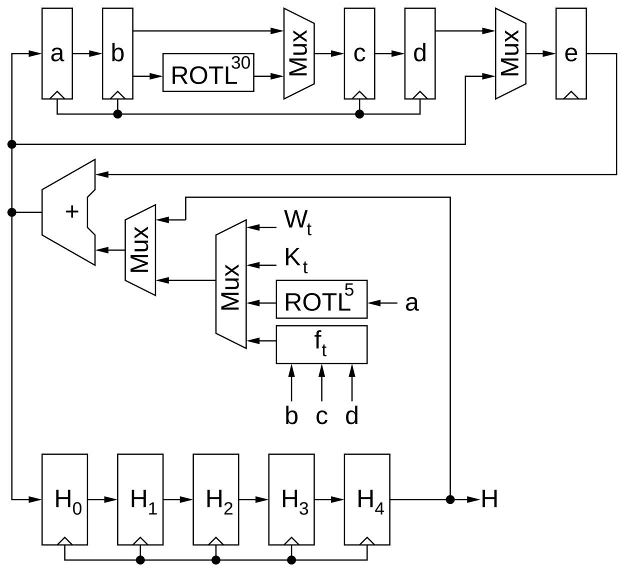

Fig. 6. Proposed Hardware Architecture of the Message Digest Unit

to memory. Our 8 bit implementation is inspired by the one reported in [18], however, we restructured the datapath so that the registers get better utilized and the AES computation consumes less clock cycles. Fig. 7 shows the top level block diagram of our AES implementation. It consists of the Computation Unit, internal memory for key expansion and current state, one S-Box, a unit to compute the round constant Rcon, a control unit and a memory interface.

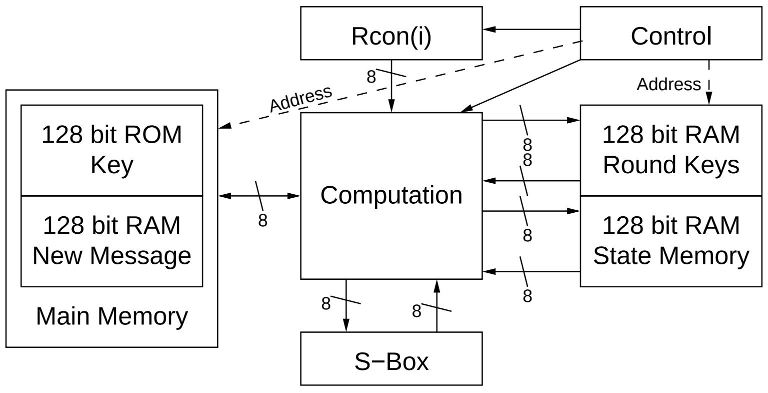

Fig. 7. Top Level Block Diagram of our AES Implementation

In CBC-mode the hash of the previous message block gets XORed with the current message block. Therefore, we can not use the external memory to store the intermediate state as we could for our SHA-1 implementation. The same applies to storing the round keys.

4.1 Datapath

Each AES transformation and the key expansion load their operands in a specific order from the state memory or key memory respectively, and write them back. Some transformations require the storage of temporary results. We streamlined this process be grouping the AES transformations into four stages:

1. Initial AddRoundKey–SubBytes–ShiftRows
2. MixColumns
3. AddRoundKey–SubBytes–ShiftRows
4. FinalAddRoundKey

This grouping enables us to re-use registers and minimize the number of internal memory accesses. It allows us to use a pipelined architecture for stage 1 and 3 which reduces he number of clock cycles by 40 percent. This improvement comes at the cost of only one additional 8 bit register over the minimum possible number or 8 bit registers. Furthermore, the memory addressing scheme gets simplified. This is a tradeoff between low area and energy consumption.

The datapath of our implementation is shown in Fig. 8. It is characterized by the pipelined architecture for stage 1 and 3 as well as the register requirements for stage 2. We used five 8-bit registers, R_0, R_1, R_2, R_3, and R_4. The register R_0 is used exclusively for key storage and is needed to implement the *RotWord* operation. R_2, R_3, and R_4 are used for the state computation. R_1 is used for key computations except during the *MixColumns* operation where it gets reused for state computation. The boxes labeled *Keys* and *Data* are the register files for the Round Keys and the State Memory respectively.

Internal Memory The 128-bit state and the 128-bit round key are stored in internal memory. This memory is register based and makes extensive use of clock gating to conserve power. The state memory has separate write and read addresses so that one value can be read while a value at another address can be written. All stages take advantage of this functionality due to the pipelined architecture. The key memory uses the same address for read and write.

4.2 Message Schedule

Initial AddRoundKey–SubBytes–ShiftRows During this first stage the 128-bit message block and the secret key are read from main memory. If used in CBC-mode, the message is XORed with the previous result. Then the the first AddRoundKey, SubBytes and ShiftRows operations are applied.

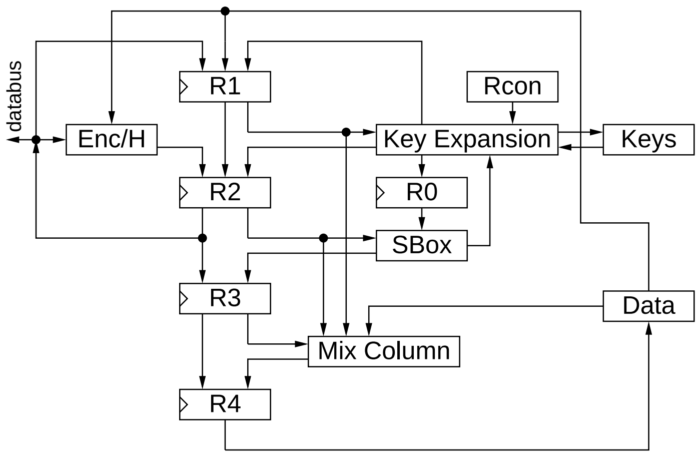

Fig. 8. Block Diagram of our Implementation of the AES Datapath

AddRoundKey–SubBytes–ShiftRows This stage is run nine times for AES. The round keys for the AddRoundKey operation are computed on the fly. This forces us to read data in column order from the state memory. The starting row is immaterial. The read order is $S_{r,0}$, $S_{r,1}$, $S_{r,2}$, $S_{r,3}, \ldots$ which translates to addresses for the state memory. As we merged the AddRoundKey operation and the ShiftRows operation the write order is predetermined. In order not to overwrite an element before its being read, we have to store four elements. Therefore the depth of our pipeline is four: R_1, R_2, R_3, and R_4.

Mix Columns Feldhofer et. al. [18] described a very efficient way for performing the *MixColumns* operation in an 8-bit architecture. It uses the minimum amount of registers needed for this operation. We used the same method, however we use an additional 8-bit register and are now able to reschedule the order of operations. The additional register (R_4) is available from the merging of *AddRoundKey* and *ShiftRows* operation. The new order of operations is shown in Equation 1.

$$\begin{aligned}
S_{3,c} \oplus S_{2,c} \oplus (S_{1,c} \bullet \{03\}) \oplus (S_{0,c} \bullet \{02\}) &= S'_{0,c} \\
S_{0,c} \oplus S_{3,c} \oplus (S_{2,c} \bullet \{03\}) \oplus (S_{1,c} \bullet \{02\}) &= S'_{1,c} \\
S_{1,c} \oplus S_{0,c} \oplus (S_{3,c} \bullet \{03\}) \oplus (S_{2,c} \bullet \{02\}) &= S'_{2,c} \\
S_{2,c} \oplus S_{1,c} \oplus (S_{0,c} \bullet \{03\}) \oplus (S_{3,c} \bullet \{02\}) &= S'_{3,c}
\end{aligned} \tag{1}$$

This order of operations results in the read order: $S_{0,c}$, $S_{1,c}$, $S_{2,c}$, $S_{3,c}$, $S_{0,c}$, $S_{1,c}$, $S_{2,c}, \ldots$. This order of addresses is now very similar to the one needed for the *AddRoundKey* function with row and column addresses swapped. This simplifies the address computation in the control logic.

Final AddRoundKey In this stage we perform the final round key computation and AddRoundKey operation. Then the result is written back to memory. Hence, the cipher can be used for encryption in CBC mode, as well as hash function in CBC-MAC mode.

5 Analysis and Comparison

All our designs were described in VHDL and verified by simulation with ModelSim and test vectors from the respective standards [8, 12]. We synthesized the VHDL code using Synopsys and used ModelSim for further verification and switching activity analysis. Our target library is a 0.13μm, $V_{DD} = 1.2$ V ASIC library from TSMC, which is characterized for power. The final results for power, area, and delay were reported by Synopsys power_compiler at the gate level. We would like to emphasize that our contribution is on the algorithmic and architectural level. Implementing our designs using an ultra-low power ASIC library or a full custom chip design will enable higher energy and power savings. Our results are shown in Table 1. Both designs consume a similar amount of area and power. However, the critical path delay in SHA-1 is more than twice as long as for AES. The critical path in AES includes the S-Box, which is the most complex part of the circuit. The delay of SHA-1 is caused by a 32-bit ripple carry adder. We implemented a carry look ahead adder for SHA-1 in order to reduce the critical path delay, but decided against using it due to the increase in power consumption caused by the additional logic for the carry look ahead. The power consumption of both designs is computed for a 500 kHz clock, which is far below their maximum frequency. The total power consumption of SHA-1 is about 10 % higher than that of AES. Within 534 clock cycles AES can encrypt 128 bits of plaintext. SHA-1 needs 405 clock cycles to compute the hash of 512 bits of data.

Table 1. Results for SHA-1 and AES

	SHA-1	AES
Delay	5.72 ns	2.19 ns
Clock cycles for one operation	405	534
Area (NAND equiv.)	4276	4070
Static Power	23.00 μW	20.23 μW
Dynamic Power (at 500 kHz)	3.74 μW	3.60 μW
Total Power (at 500 kHz)	26.73 μW	23.83 μW

Feldhofer et.al. presented two related AES designs in [18] and [19] consuming 26.9 μW and 4.5 μW respectively with a 100kHz clock. These numbers are difficult to compare with our design as the results for power consumption are highly technology dependent. The encryption only design in [18] consumes an area of 3595 NAND equiv. and needs 1016 clock cycles. The design in [19] needs 3400 NAND equiv. and 1032 clock cycles. Both designs do not support CBC mode which requires extra hardware. It can easily be seen that our implementation uses 20% more hardware resources than their smallest design while using 48% less clock cycles, i.e. it is almost twice as fast. The slight increase in hardware resources leads to large decrease in computation time which reduces the energy consumption while still being an ultra-low power circuit. For a fair comparison of AES and SHA-1, we used the same implementation and optimization techniques with the same ASIC library.

In order to explore the energy consumption of our AES and SHA-1 implementations we focus on the TinySec [5] protocol. Table 2 shows the results assuming the TinySec packet format and a payload of 29 bytes.

Table 2. Energy Results for SHA-1 and AES (29 bytes/packet, 500 kHz)

	MAC		Encryption		Encryption & MAC	
	AES	SHA-1	AES	SHA-1	AES	SHA-1
Energy (nJ)	76.42	43.32	50.95	43.32	127.36	86.64
Power (μW)	23.85	26.74	23.85	26.74	23.85	26.74
Time (ms)	3.20	1.62	2.14	1.62	5.34	3.24
Energy/bit (nJ)	0.33	0.19	0.22	0.19	0.55	0.37

5.1 Message Authentication Codes

Due to the lossy nature of low power wireless transmission it is not feasible to compute a single MAC for multiple packets. Therefore, SPINS [3] computes a MAC for each packet using RC5 [4] in CBC-MAC mode and appends it to the original packet. TinySec defines a packet format for authenticated messages that can carry up to 29 Bytes of payload. The MAC is computed over the payload and the packet header which is four bytes long. Table 2 shows that using AES to compute the MAC over 29+4 bytes consumes 76.42 μJ and SHA-1 consumes 43.32 μJ. Even though SHA-1 consumes 10% more power than AES, the running time of AES is larger by a factor of two, leading to the higher energy consumption. Fig. 9 shows the energy consumption for MAC computation over different payload sizes, each time assuming a four byte overhead. Until the payload reaches 29 bytes AES consumes less or almost equally as much energy as SHA-1. For payloads of 29 bytes or larger AES has to run more than twice while for SHA-1 two iteration are sufficient, due to its longer input size.

5.2 Encryption

Even though TinySec does not specify an encryption only format we still consider it for comparison purposes. We assume that only the payload has to be encrypted and the packet header is transmitted in the clear. Table 2 shows that the difference in Energy consumption between SHA-1 and AES are less dramatic for encryption than for MAC computation. Fig. 10 shows that SHA-1 follows AES closely. This comes from the fact that the input size of AES is 128 bits and of SHA-1 in encryption mode (SHACAL-1) is 160 bits.

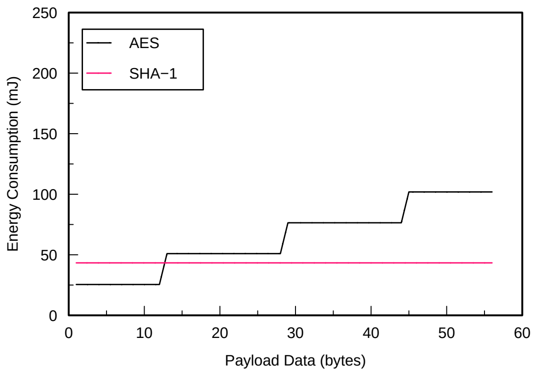

Fig. 9. Energy Consumption of MAC Computation with AES and SHA-1 Depending on Payload Size

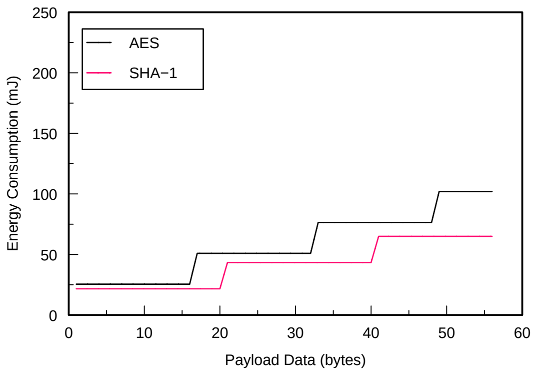

Fig. 10. Energy Consumption of Encryption with AES and SHA-1 Depending on Payload Size

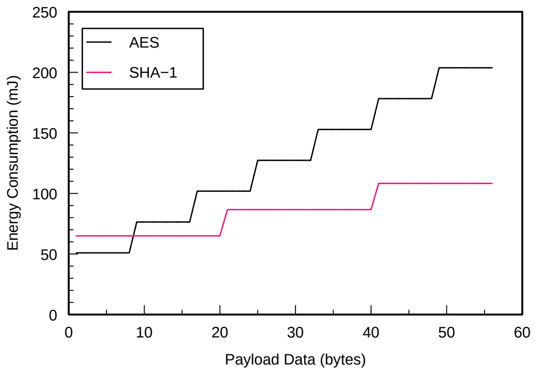

Fig. 11. Energy Consumption of Encryption and MAC Computation with AES and SHA-1 Depending on Payload Size

5.3 Authentication and Encryption

The packet format for Authentication and Encryption specifies a payload of upto 29 bytes and a packet header of eight bytes length. Only the payload has to be encrypted but the MAC is computed over the payload and the message header. Assuming a 29-byte payload, AES consumes almost 1/3 more energy than SHA-1 (see Table 2. For larger payloads the SHA-1 consumes significantly less power (see Fig. 11).

6 Conclusion

This paper presented a novel ultra-low power implementation of SHA-1 and an ultra-low power and low energy AES design. Both circuits consume less than 30 μW of power and could therefore be powered by scavenger circuits. We analyzed the energy consumption of SHA-1 and AES based encryption and message authentication functions. The result of our analysis is that SHA-1 and AES seem to be equally well suited for ultra-low power applications if the payload size is below 17 bytes. For payloads of 17 bytes or above SHA-1 needs significantly fewer iterations than AES and therefore a shorter running time which conserves energy. We want to emphasize that the power consumption of both algorithms is about the same.

References

1. Meininger, S., Mur-Miranda, J., Amirtharajah, R., Chandrakasan, A., Lang, J.: Vibration-to-electric energy conversion. IEEE Transactions on Very Large Scale Integration (VLSI) Systems **9** (2001) 64–76
2. Amirtharajah, R., Chandrakasan, A.P.: Self-powered signal processing using vibration-based power generation. IEEE Journal of Solid-State Circuits **33** (1998) 687–695
3. Perrig, A., Szewczyk, R., Tygar, J.D., Wen, V., Culler, D.E.: SPINS: security protocols for sensor networks. Wireless Networks **8** (2002) 521–534
4. Rivest, R.: The RC5 encryption algorithm. In: Fast Software Encryption. Volume 1008 of Lecture Notes in Computer Science LNCS., Berlin, Springer-Verlag (1995) 86–96
5. Karlof, C., Sastry, N., Wagner, D.: TinySec: A link layer security architecture for wireless sensor networks. In: Second ACM Conference on Embedded Networked Sensor Systems (SenSys 2004), New York, ACM Press (2004) 162–175
6. National Institute of Standards and Technology (NIST) FIPS Publication 185: Escrowed Encryption Standard (EES). (1994)
7. National Institute of Standards and Technology (NIST): Complete SKIPJACK and KEA specification. (1998)
8. National Institute of Standards and Technology (NIST) FIPS Publication 197: Advanced Encryption Standard (AES). (2001)
9. Law, Y., Doumen, J., Hartel, P.: Benchmarking block ciphers for wireless sensor networks. In: IEEE International Conference on Mobile Ad-hoc and Sensor Systems. (2004) 447–456

10. Luo, X., Zheng, K., Pan, Y., Wu, Z.: Encryption algorithms comparisons for wireless networked sensors. In: IEEE International Conference on Systems, Man and Cybernetics. Volume 2. (2004) 1142–1146
11. Prasithsangaree, P., Krishnamurthy, P.: Analysis of energy consumption of RC4 and AES algorithms in wireless LANs. In: IEEE Global Telecommunications Conference, GLOBECOM '03. Volume 3., IEEE (2003) 1445–1449
12. National Institute of Standards and Technology (NIST) FIPS Publication 180-2: Secure Hash Standard (SHS). (2002)
13. Grembowski, T., Lien, R., Gaj, K., Nguyen, N., Bellows, P., Flidr, J., Lehman, T., Schott, B.: Comparative analysis of the hardware implementations of hash functions SHA-1 and SHA-512. In: Information Security, 5th International Conference, ISC 2002. Volume 2433 of Lecture Notes in Computer Science LNCS., Springer-Verlag (2002) 75–89
14. Bellare, M., Canetti, R., Krawczyk, H.: Keying hash functions for message authentication. In: Advances in Cryptology, Crypto '96. Volume 1109 of Lecture Notes in Computer Science (LNCS)., Springer Verlag (1996) 1–15
15. Krawczyk, H., Bellare, M., Canetti, R.: HMAC: Keyed-hashing for message authentication. RFC 2104, Network Working Group (1997)
16. Mangard, S., Aigner, M., Dominikus, S.: A highly regular and scalable AES hardware architecture. IEEE Transactions on Computers **52** (2003) 483–491
17. Good, T., Benaissa, M.: AES on FPGA from the fastest to the smallest. In: Cryptographic Hardware and Embedded Systems - CHES 2005. Volume 3659 of Lecture Notes in Computer Science LNCS., Springer (2005) 427–440
18. Feldhofer, M., Dominikus, S., Wolkerstorfer, J.: Strong authentication for RFID systems using the AES algorithm. In: Proceedings of the 6th international workshop on cryptographic hardware and embedded systems CHES 2004. Volume 3156 of Lecture Notes in Computer Science LNCS., Springer (2004) 357–370
19. Feldhofer, M., Wolkerstorfer, J., Rijmen, V.: AES implementation on a grain of sand. Information Security, IEE Proceedings **152** (2005) 13–20
20. Wang, X., Yin, Y.L., Yu, H.: Collision search attacks on SHA1. Internet (2005)
21. MDx-MAC, from Hash Functions, B.F.M.: Preneel, b. and van oorschot, p. c. In: Advances in Cryptology, Crypto '95. Volume 963 of Lecture Notes in Computer Science (LNCS)., Springer-Verlag (1995) 1–14
22. National Institute of Standards and Technology (NIST) FIPS Publication 198: The Keyed-Hash Message Authentication Code (HMAC). (2002)
23. Stinson, D.R.: Cryptography: Theory and Practice. 3 edn. Volume 36 of Cryptography: Theory and Practice. Chapman & Hall/CRC (2005)
24. National Institute of Standards and Technology (NIST) FIPS Publication 81: DES modes of operation. (1980)
25. National Institute of Standards and Technology (NIST) FIPS Publication 113: Computer Data Authentication. (1985)
26. Handschuh, H., Naccache, D.: SHACAL. Submission to the NESSIE project, Gemplus, F-92447 Issy-les-Moulineaux, France (2000)
27. Handschuh, H., Naccache, D.: SHACAL: a family of block ciphers. Submission to the NESSIE project, Gemplus, F-92447 Issy-les-Moulineaux, France (2001)
28. Handschuh, H., Knudsen, L.R., Robshaw, M.J.: Analysis of SHA-1 in encryption mode. In: Topics in Cryptology CT-RSA 2001. Volume 2020 of Lecture Notes in Computer Science., Springer Verlag (2001) 70–83
29. Saarinen, M.J.O.: Cryptanalysis of block ciphers based on SHA-1 and MD5. In: Fast Software Encryption, 10th International Workshop, FSE 2003. Volume 2887 of Lecture Notes in Computer Science LNCS. (2003) 36–44