Energy Comparison of AES and SHA-1
for Ubiquitous Computing

Jens-Peter Kaps and Berk Sunar⋆

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

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 an-
alyze 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 a few
examples showing 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 severe power constraints
which make it difficult to realize computationally intensive cryptographic func-
tions. Passive RFID tags are powered by the electro magnetic field from 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. 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 for encryption and in CBC-mode to build a secure MAC. TinySec [4] is cipher independent and was tested with RC5 and Skipjack for encryption and CBC-MAC. The authors of [4] are also considering the Advanced Encryption Standard [5].

Many research papers [3,6,7] analyze encryption algorithms for wireless sensor networks exclusively with reference to speed and code size while only a few [8] 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.

2   Cryptographic Functions for Ultra-Low Power Applications

AES was selected by the National Institute of Standards and Technology (NIST) as Federal Information Processing Standard FIPS-197 [5] in 2001. Since then, many hardware implementations have been published. Most of them are optimized for speed and only a few are scalable [9,10] from fast to small. The first ultra-low power implementation was reported in [11] followed by [12] 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

---

[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.

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.

SHA-1 is the most widely used secure hash function and was developed by the National Security Agency. Its security level[2] is considered to be 2^{80}. Recent attacks on SHA-1 [13] 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 [14]. 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.

2.1   Message Authentication Codes

We use HMAC, which is formally described in [15] as $\mathrm{HMAC}_k(x) = \text{SHA-1}((\overline{k} \oplus opad) \,\|\, \text{SHA-1}((\overline{k} \oplus ipad) \,\|\, x))$, to build a message authentication code. It uses a 160-bit key K which is padded with 0's resulting in $\overline{k}$. The terms $\overline{k} \oplus opad$ and $\overline{k} \oplus ipad$ can be precomputed from the 512-bit constants *opad* and *ipad* and $\overline{k}$. Due to the concatenation $((\overline{k} \oplus ipad) \,\|\, x)$ the intermediate hash value of $(\overline{k} \oplus ipad)$ and $(\overline{k} \oplus opad)$ 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 [16] mode to compute authentication codes. This mode is similar to the *Cipher Block Chaining* mode [17,18] in that the result from the previous encryption is XORed with the next plaintext block and encrypted again. 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[3]. CMAC [19] fixes some security deficiencies of CBC-MAC. It uses two subkeys which can be precomputed. The main MAC computation is very similar to CBC-MAC, only the last step is different. However, this would not add much to the complexity of our CBC-MAC implementation.

2.2   Encryption

To some extent, hash functions like SHA-1 can also be used to perform encryption. The best examples are SHACAL [20] and SHACAL-1 [21]. The security of SHACAL was analyzed in [22] and more recently in [23]. SHACAL defines how the compression function of SHA-1 can be used as a 160-bit block cipher with

---

[2] 2^{80} operations have to be made on average to find another input such that the resulting hashes are equal, also called a collision.

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

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

We assume that one 512-bit block of preprocessed data is stored in memory and available to our SHA-1 unit for reading and writing. The operation of our SHA-1 implementation 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 W_t, forwards it to the message digest unit and also stores 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

Most implementations in literature use a 16 stage 32-bit wide shift register for this purpose (512 flip-flops). Our message scheduler is a serial design and needs only one 32-bit register to store a temporary value during computation of the new W_t. It scheduler performs the equation $W_t = ROTL^1 (W_{t-3} \oplus W_{t-8} \oplus W_{t-14} \oplus W_{t-16})$ 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, therefore, each round of SHA-1 takes 5 clock cycles. The necessary address computation 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

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 from [24] 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. Therefore, we view the registers for the working variables as a 5 stage 32-bit wide shift register.

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 K_t, and the current message word 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 and use register e as temporary register. This requires 4 clock cycles per round which is below the need of the message scheduler with 5 clock cycles. Figure 1 shows the block diagram of our implementation of the message digest unit including the round function and the intermediate hash value computation.
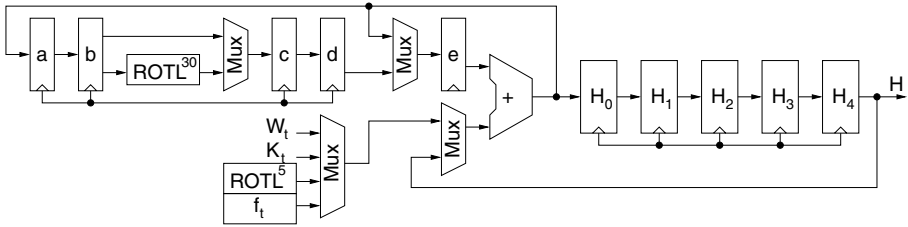
Fig. 1. Proposed Hardware Architecture of the Message Digest Unit

Intermediate Hash Value Computation. During the final stage, 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 H_0, H_1, H_2, H_3, and H_4 in a 5 stage 32-bit wide shift register, similar to our design for the working variables. Computing the final hash value for one input block takes five clock cycles. This 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 is written back to memory. Our 8-bit implementation is inspired by the one reported in [11], however, we restructured the datapath so that the registers are better utilized and the AES computation consumes less clock cycles. In CBC-mode the hash of the previous message block is 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 by 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 is simplified. This is a tradeoff between low area and energy consumption.

The datapath of our implementation is shown in Fig. 2. 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 boxes labeled *Keys* and *Data* are the register files for the 128-bit Round Keys and the 128-bit State Memory respectively. This memory is register based and makes extensive use of clock gating to conserve power.
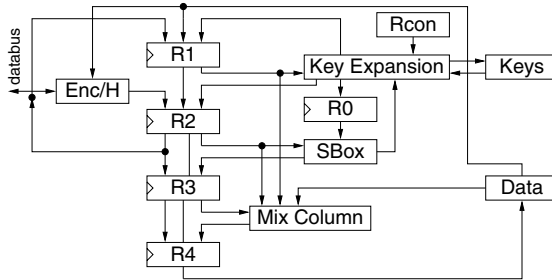

Fig. 2. Block Diagram of our Implementation of the AES Datapath

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.

AddRoundKey–SubBytes–ShiftRows. This stage is run nine times for AES. The round keys for the AddRoundKey operation are computed on the fly. In order not to overwrite an element before its being read, we have to store four elements. leading to our pipeline depth of four: R_1, R_2, R_3, and R_4.

Mix Columns. Feldhofer et. al. [11] 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.

Final AddRoundKey. In this stage we perform the final round key computation and AddRoundKey operation. Then the result is written back to memory.

5   Analysis and Comparison

All our designs were described in VHDL and verified by simulation with ModelSim and test vectors from the respective standards [5,24]. 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. The critical path delay in SHA-1 is more than twice as long as for AES, however, we assume an operating frequency of 500 kHz which is far below their maximum frequency. The total power consumption at 500 kHz 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
Critical Path 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 [11] and [12] 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 [11] consumes an area of 3595 NAND equiv. and needs 1016 clock cycles. The design in [12] 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 a 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 [4] protocol. Table 2 shows the results assuming the TinySec packet format and a payload of 29 bytes.

Message Authentication Codes. 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 nJ and SHA-1 consumes 43.32 nJ. 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. 3 shows the energy consumption for MAC computation

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

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.

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. 4 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.
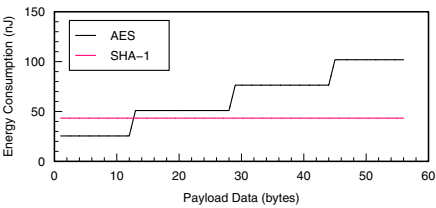

Fig. 3. Energy Consumption of MAC Computation with AES and SHA-1 Depending on Payload Size
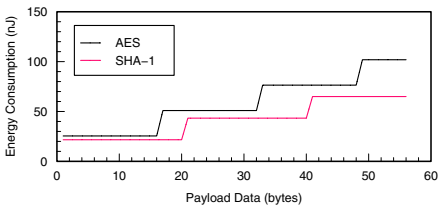

Fig. 4. Energy Consumption of Encryption with AES and SHA-1 Depending on Payload Size
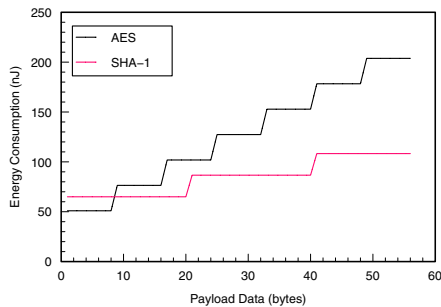

Fig. 5. Energy Consumption of Encryption and MAC Computation with AES and SHA-1 Depending on Payload Size

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. 5).

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, has 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**(1) (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**(5) (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**(5) (2002) 521–534
4. 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
5. National Institute of Standards and Technology (NIST) FIPS Publication 197: Advanced Encryption Standard (AES). (2001)
6. 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
7. 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
8. Prasithsangaree, P., Krishnamurthy, P.: Analysis of energy consumption of RC4 and AES algorithms in wireless LANs. In: GLOBECOM'03. Volume 3., IEEE (2003) 1445–1449
9. Mangard, S., Aigner, M., Dominikus, S.: A highly regular and scalable AES hardware architecture. IEEE Transactions on Computers **52**(4) (2003) 483–491

10. Good, T., Benaissa, M.: AES on FPGA from the fastest to the smallest. In: CHES 2005. Volume 3659 of LNCS., Springer (2005) 427–440
11. Feldhofer, M., Dominikus, S., Wolkerstorfer, J.: Strong authentication for RFID systems using the AES algorithm. In: CHES 2004. Volume 3156 of LNCS., Springer (2004) 357–370
12. Feldhofer, M., Wolkerstorfer, J., Rijmen, V.: AES implementation on a grain of sand. Information Security, IEE Proceedings **152**(1) (2005) 13–20
13. Wang, X., Yin, Y.L., Yu, H.: Collision search attacks on SHA1. Internet (2005)
14. 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: ISC 2002. Volume 2433 of LNCS., Springer-Verlag (2002) 75–89
15. National Institute of Standards and Technology (NIST) FIPS Publication 198: The Keyed-Hash Message Authentication Code (HMAC). (2002)
16. Stinson, D.R.: Cryptography: Theory and Practice. 3 edn. Volume 36 of Discrete Mathematics and its Appications. Chapman & Hall/CRC (2005)
17. National Institute of Standards and Technology (NIST) FIPS Publication 81: DES modes of operation. (1980)
18. National Institute of Standards and Technology (NIST) FIPS Publication 113: Computer Data Authentication. (1985)
19. National Institute of Standards and Technology NIST SP 800-38B: Recommendation for Block Cipher Modes of Operation: The CMAC Mode for Authentication. (2005)
20. Handschuh, H., Naccache, D.: SHACAL. Submission to the NESSIE project, Gemplus, F-92447 Issy-les-Moulineaux, France (2000)
21. Handschuh, H., Naccache, D.: SHACAL: a family of block ciphers. Submission to the NESSIE project, Gemplus, F-92447 Issy-les-Moulineaux, France (2001)
22. Handschuh, H., Knudsen, L.R., Robshaw, M.J.: Analysis of SHA-1 in encryption mode. In: CT-RSA 2001. Volume 2020 of LNCS., Springer Verlag (2001) 70–83
23. Saarinen, M.J.O.: Cryptanalysis of block ciphers based on SHA-1 and MD5. In: FSE 2003. Volume 2887 of LNCS. (2003) 36–44
24. National Institute of Standards and Technology (NIST) FIPS Publication 180-2: Secure Hash Standard (SHS). (2002)