Salted Hashes for Message Authentication – Proof of concept on Tiny Embedded Systems

René Romann, Ralf Salomon
University of Rostock
Institute of Applied Microelectronics and Computer Engineering
18051 Rostock, Germany
Email: {rene.romann}, {ralf.salomon}@uni-rostock.de

***Abstract*—Intelligent embedded systems become more and more widespread. Especially in the field of smart environments, such as smart homes, the systems are communicating with each other. If wireless communication is used, security becomes important. This paper explores to what extent *salted hashes* might be used on tiny embedded systems to provide message authentication. To this end, this paper uses two very different microcontrollers for calculating salted hases using SHA-1 and SHA-256. The execution times vary between 2.5 and 160 milliseconds, which is fast enough to provide user responses in time.**

***Index Terms*—smart home, security, remote control, home automation**

I. Introduction

Research on intelligent embedded systems is receiving increasing attention in the area of smart appliances. The term “smart appliances” refers to rather small but still “intelligent” devices that are equipped with at least some sensors, actuators, with low to moderate processing capabilities, and a network interface of some sort. A collection of smart appliances constitutes an *ensemble*, if they cooperate with each other in order to help users accomplish their goals in an unobtrusive way that does not require significant configuration activities [1] [2]. Thus, the research focus is more on *parsimonious design*, self-organization and self-configuration mechanisms, intention recognition, high-precision localization, and fusing heterogenious platforms, rather than being on processing speed, memory capacity, and data throughput. Typical smart appliances include light sources, switches, window blinds, beamers, presentation laptops, heaters, as well as garage and front doors.

For doing specific research, several projects resort to a simplified smart home, which contains a small number of light sources and switches, as can be seen in Fig. 1 . The employed communication devices might be wireless, such as WiFi, Bluetooth, infrared, ultra sonic, and the like. Furthermore, a small communication and administration server might be part of the entire system. Even tough being of a rather educational nature, such a setup is still challenging enough for most of the research issues mentioned above.

Given that the appliances communicate with each other wirelessly and given that the required configuration efforts should be as low as possible, such a setup is a first-class target for security attacks. After recording some of the wirelessly exchanged packages, for example, an intruder might assume the control of the smart home. The consequences can range from being annoying, e.g., light sources are switched on and off, to serious, e.g., doors are opened by non-authorized persons. Thus, the employment of appropriate security mechanisms is essential. Commonly used mechanisms include data encryption along with a key infrastructure, near-field communication for key distribution, and digital signatures. The choice of these mechanisms, however, have to fit the system’s constraints, the potential security attacks, as well as the users’ requirements. Section II elaborates a little more on these issues.

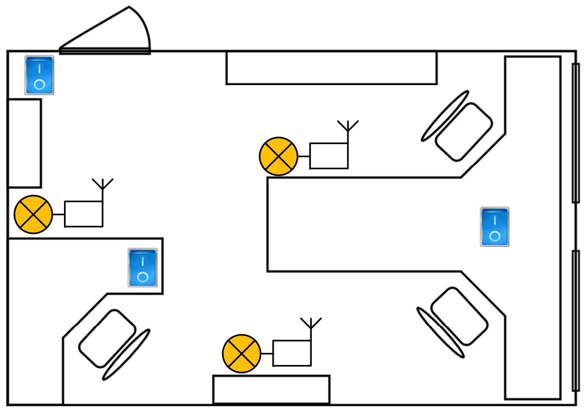
Figure 1. Smart office with multiple lamps and controls

It is well known that for simple devices, asymmetric encryption algorithms are computationally way too demanding. Therefore, those devices have to resort to symmetric cryptography. One of the commonly used methods is known as salted hashes: a salt is a *secret* that is shared between the two communication partners. This salt is used to calculate a message’s hash value in order to provide its authenticity. Section III describes this algorithm and its setup in more detail.

The algorithms described in Section III have been implemented on two low-cost devices, a CC1010 microcontroller evaluation board and a MSP430 microcontroller, to prove whether they may be used as lightweight alternative to ensure security on those resource-constraint devices. All the technical details including the devices relevant specification are summarized in Section IV. Consequently, the results of the performed case study are presented in Section V. Finally,

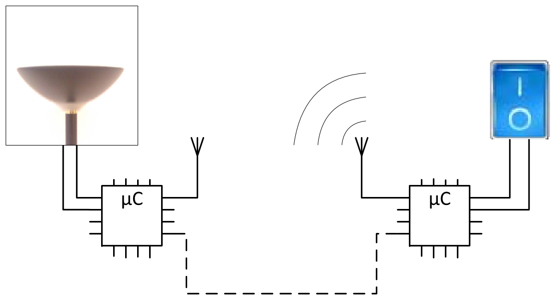

Figure 2. Simplified smart home containing one lamp and one switch with their processing units and their communication connection (wireless or wired)

Section VI concludes this paper with a brief discussion.

II. System Constraints and Potential Security Attacks

As a simple example of a smart home, Fig. 2 shows two devices, a light source and a control button. Those two devices contain some sort of processing unit and a communication interface (wireless or wired). The processing unit generates and evaluates messages. Those messages are transmitted from one device to the other using their communication interfaces.

One of the constraints is the physical dimension of the processing unit and the communication interface. The physical dimension should be kept minimal to integrate the processing unit and the communication interface into already existing device housings. With mobile battery-powered devices, the power usage is another relevant aspect. The less power the system uses, the longer the battery will last. Therefore, microcontrollers are used as processing units. They provide small physical dimensions together with a low power consumption. However, they only provide a limited amount of computational resources and memory.

Other constraints result from the users' wishes. They ask for comfort, which means mobile control, easy integration, and simple configuration of new devices. Furthermore, the time between the button press and its actuator response should be minimal. All of these user requirements add additional difficulties to the constrains listed before. Mobile controls can be realized using wireless communication. But this asks for proper security mechanisms.

Without security mechanisms, an attacker could easily send messages to the devices inside the smart home and thus assume their control. Also, if using security, an attacker could gain access with a brute force attack. The consequences of such attacks may be annoying if lamps are switched on and off, but can also be serious if doors can be opened by unauthorized persons.

To overcome this problems, messages have to be authenticated in a way that manipulations can be detected. The same applies to messages that are recorded and replayed.

One widely known approach is to use encryption. This includes the use of the Advanced Encryption Standard (AES) [3], RSA [4], or Elliptic Curve Cryptography (ECC) [5]. Whereas some microcontrollers have built-in hardware support for AES encryption, others do not. Even if the microcontrollers had such AES support in hardware, they would be limited to their specific features (e.g., algorithm and key length) without being easily upgradeable. As history shows, key lengths and algorithms proven to be safe at a specific date have become outdated later on. Because of this, AES hardware support may not be a choice for long-term development, and is therefore not part of this case study. Other research [6] has shown that the footprint of RSA is 19 kilobytes, which is quite high. Furthermore encryption takes about 20 seconds, which is too long in many application sectors. Similar results can be found for the ECC algorithm. However, the execution time of ECC is as short as 2.8 seconds. The times given are way too long to fit the users requirements of a fast responding system. Therefore, these algorithms are not suitable on those devices.

Another approach is the usage of digital signatures. In contrast to message encryption, only a "fingerprint" of the message is calculated. A hash function, such as the Secure Hash Algorithm (SHA) 1, calculates the hash ("fingerprint") for the message. Afterwards this hash is encrypted using some sort of asymmetric encryption. One combination is the use of SHA-1 and RSA [7]. Recent research [8] has shown that the execution time of a digital signature using a 1024 bit RSA key may be as low as two or three seconds, which is quite faster than those times for a complete encryption. But this is still too slow to meet the users' requirements of "immediate" system response.

It is obvious that the time required for creating a digital signature comprises the time for hashing the message and the time required for encrypting the calculated value. Another approach, which does not require encryption at all, is known as message authentication code (MAC). These message authentication codes use a cryptographic hash function (such as SHA-1 [9] or the younger SHA-2 family) in conjunction with a secret key that is added to the message before hashing. This secret key is usually referred to as "salt" which gave those algorithms the name "salted hashes". Because salted hashes do not involve encryption techniques at all, their memory footprint and their processing time should be smaller compared to digital signatures mentioned above. One of those algorithms commonly used is the Hashed-key Message Authentication (HMAC) algorithm [10] which is used by the Transport Layer Security (TLS) protocol [11].

Section III gives a short introduction of how salted hashes work, whereas Sec. IV contains details about a proof-of-concept implementation.

III. Salted Hashes

Hash functions are one-way functions that compress a message of arbitrary length to a fixed-sized fingerprint. The SHA family is a widely-used option. It comprises different hash functions, such as the SHA-1 [9] and the SHA-2 family [12].

The calculation done can be described as following, but may differ from hash function to hash function. All hash functions

```
procedure HASH($message, messagelength$)
    $inter \leftarrow 0xA3$                    ▷ 0xA3 is the start vector
    $index \leftarrow 0$
    while $index < messagelength$ do
        $inter \leftarrow (inter + message[index]) mod 256$
        $index \leftarrow index + 1$
    end while
    return $inter$
end procedure
```

Figure 3. Pseudocode for a simple hash function

have in common that they process the message blockwise. Similar to a recursion, the result of the hash calculation of a block depends on the result of the previous block. This is done by using an intermediate result memory that contains a special initialization value before processing the first block. After all blocks have been processed, this result memory contains the calculated hash value.

Figure 3 shows pseudocode of how a very simple hash function might look like. At first, the intermediate memory is initialized with a special value. Then every block of the message is hashed (inner while block). When all blocks have been processed, the result is returned to the caller.

Although Fig. 3 presents a very simple example, the pseudocode shows how most hash functions work. Within this paper, more complex hash functions such as SHA-1 and SHA-256 are used. They differ from the given pseudocode especially within the compression function (Fig. 3, line 4) as well as their start vectors (Fig. 3, line 1). Another difference is the block size. Whereas the sample code uses bytewise compression, the more complex hash functions usually have multi byte block sizes.

To add authenticity, a secret key S, commonly referred as "salt", is added to the message M before being hashed. Thus the result H^* of the hash calculation depends on both the message and the secret. One of the simplest possibilities to add the key to the message is to concatenate both.

If only two communication partners share a secret key, each one can be sure that the other one sent the message, if the validation of the salted hash succeeds. Other devices or attackers are not able to compute a valid hash due to the fact that they are not knowing the secret key.

To ensure that the secret key is only known by the two communication partners, the key has to be exchanged out-of-band. In an experimental setup, this may be done by static linking at setup time, but later on, it should be replaced by dynamic out-of-band setup methods, such as near-field communication or infrared LED blinking patterns.

The secret key should be 128 bits long. Even if a fast PC can compute 2^{40} hashes per second, it takes 2^{88} seconds (309 * 10^{24} years) to compute every possible hash for a given message with an unknown secret.

```
procedure TESTBENCH
    for each hash function do
        for each input vector do
            $outputpin \leftarrow 'high'$
            $value \leftarrow$ HASH($input\,vector, length$)
            $outputpin \leftarrow 'low'$
            if $value = precomputed\,value$ then
                print Identical
            else
                print Error
            end if
        end for
    end for
end procedure
```

Figure 4. Pseudocode for a simple hash function

IV. METHODS

In order to estimate whether or not salted hashes are suitable on resource-limited devices, these methods have been implemented on a microcontroller.

This proof-of-concept is done within a testbench, which calculates salted hashes for different key and message combinations. This testbench is written in C, and therefore easily portable to different microcontrollers. It uses five different input vectors (containing hash and message) to the hash function and validates the result against a pre-computed value to validate the correctness of the algorithm on that specific microcontroller. Figure 4 shows the pseudocode for this testbench.

For testing purposes, two different microcontrollers are used. One is a Chipcon CC1010 mounted to an evaluation module. It is clocked with 14.7 MHz and has 32 kilobytes of flash memory. In addition, it contains a wireless interface for 868 MHz ISM band [13]. This is a rather old model, but similar models are still in use.

The second microcontroller is a Texas Instruments MSP430F2274, which is mounted on a eZ430-RF2480 test board [14]. This microcontroller is clocked at 16 Mhz and has 32 kilobytes of flash memory [15]. In addition, it is connected to a CC2480 chip, which provides a ZigBee interface [16].

The hash algorithms tested for this proof-of-concept include the widespread SHA-1 algorithm and its successor SHA-256. For both algorithms, C source-code is given within their corresponding RFC documents [9] [12]. Using the Keil μVision 4 software with its C51 compiler for the CC1010, both implementations take about 4.8 kilobytes of flash memory. For the MSP430, the Texas Instruments Code Composer Studio is used. Similarly to the CC1010 implementation, the MSP430 implementation takes roughly 5 kilobytes of memory.

To measure the algorithms' execution times, an output pin on the microcontroller is pulled to high level just before the hash calculation starts. Right afterwards the hash function call, this pin is turned out back to low level. To capture the signals on that pin, an Agilent DS0X3034A oscilloscope is used. A

Table I
PROCESSING TIMES FOR DIFFERENT INPUT VECTORS AND HASH FUNCTIONS

Input Vector	Processing Time [ms]			
	CC1010		MSP430	
Length [bytes]	SHA-1	SHA-2	SHA-1	SHA-2
33	85	155	2.5	7
43	85	155	2.5	7
44	85	155	2.5	7
59	170	310	5	14
59	170	310	5	14

serial connection to a PC is also established for starting the testbench and validating the test results.

V. RESULTS

Table I shows the results of the tests. As can be seen, the times for calculating a salted hash for a single input vector depend on the length of that vector. The longer the input vector containing the key and the message, the longer the calculation takes. Due to the blocksize of the used algorithms, times are equivalent if the input vector lengths are nearly equal.

Both SHA-1 and SHA-256 use a blocksize of 64 bytes (512 bits) whereas the last block contains 64 bits of length information for the hashed message. This means that the last block can only contain 56 bytes of data instead of 64. The two vectors with 59 bytes exceed this 56 byte limit for the last block. Therefore another block is added. Therefore the processing time is doubled.

As can be seen in Table I, the calculation on the CC1010 takes about 85 milliseconds and 155 milliseconds using the SHA-1 and SHA-256, respectively, for every block of the input vector regardless of the message and the secret key. The MSP430 performs much faster with 2.5 milliseconds and 7 milliseconds for the SHA-1 and the SHA-256, respectively, which is roughly 20 to 30 times faster than the CC1010.

This speedup of a factor of 20 to 30 between the CC1010 and the MSP430, which are both operated at nearly the same clock frequency, is a result of the internal architecture. Whilst CC1010 is an 8-bit microcontroller with an 8051 architecture, the MSP430 uses a 16-bit RISC architecture. Therefore CC1010 has to process at least four instructions for a 32-bit-wide integer operation that the MSP430 can achieve with only two instructions. Additional instructions may be executed, especially for moving content between different processor registers, which further increases the processing time.

With the proposed 128 bit secret key (see, also, Sec. III), the payload can contain 40 bytes of data without more than one block to be processed. So, if the message to be sent is shorter than 40 bytes (which seems to be quite enough for a lamp switching operation), the time for calculating or validating a hash is below a maximum of 155 milliseconds, which is fast enough for a user to be interpreted as the lamp turning on "instantly".

Referring to Table II, this proof-of-concept shows that salted hashes are able to provide message authentication on tiny embedded systems with less processing time and memory footprint than the alternative approaches. Compared to a digital signature using ECC, computing time is lowered by nearly 90 % even if using the "slower" SHA-256 algorithm. Also the memory footprint is reduced by nearly 75 % compared to digital signatures.

Table II
COMPARISON OF DIFFERENT APPROACHES TO ENSURE MESSAGE AUTHENTICITY

Algorithm	Footprint [kilobytes]	Time [ms] Generation / Validation
asymmetric encryption (RSA) [6]	19	21500 / 790
digital signature (RSA + SHA) [8]	21	2000 / 2430
digital signature (ECC) [6]	19	2850 / 1350
Salted Hash (using SHA-1)	**5**	**2.5 / 2.5 (MSP430) 85 / 85 (CC1010)**
Salted Hash (using SHA-256)	**5**	**7 / 7 (MSP430) 155 / 155 (CC1010)**

VI. DISCUSSION

This paper has shown that *salted hashes* are an option to provide message authentication on a $1 : 1$ communication channel. Even on resource limited devices, such as the CC1010 or MSP430 microcontrollers, the SHA-1 and SHA-256 algorithms can be implemented using only 5 kilobytes of program memory. The calculation times of one block vary by the hash function; SHA-1 consumes 85 milliseconds per block, whereas SHA-256 consumes 155 milliseconds per block on the CC1010. Using the MSP430 microcontroller, the performance can be further improved to 2.5 milliseconds per block for the SHA-1 and 7 milliseconds per block for the SHA-256. Both are fast enough for providing real-time in smart home environments.

Future research will be dedicated into implementing the salted hashes into a framework for secure smart home application control. This includes additional device support, wireless transmission of commands, and the out-of-band key transmission for the system setup.

ACKNOWLEDGEMENT

Special thanks are due to the entire MuSAMA graduate school including all the PhD students as well as the supervisors for providing a very stimulating environment. This work was supported in part by the DFG graduate school 1424.

REFERENCES

[1] M. Weiser, "The computer for the 21st century," *Scientific American*, vol. 265, no. 3, pp. 94–104, Sep. 1991. [Online]. Available: http://www.nature.com/scientificamerican/journal/v265/n3/pdf/scientificamerican0991-94.pdf

[2] T. Kirste, *Smart Environments*. Berlin, Germany: Springer-Verlag, Aug. 2006, ch. 17, pp. 321–337.

[3] National Institute of Standards and Technology, "NIST FIPS 197: Advanced Encryption Standard (AES)," Nov. 2001. [Online]. Available: http://csrc.nist.gov/publications/fips/fips197/fips-197.pdf

[4] R. L. Rivest, A. Shamir, and L. Adleman, “A method for obtaining digital signatures and public-key cryptosystems,” *Commun. ACM*, vol. 21, no. 2, pp. 120–126, Feb. 1978. [Online]. Available: http://doi.acm.org/10.1145/359340.359342

[5] D. McGrew, K. Igoe, and M. Salter, “IETF RFC 6090: Fundamental Elliptic Curve Cryptography Algorithms,” Feb. 2011. [Online]. Available: http://tools.ietf.org/html/rfc6090

[6] H. Wang and Q. Li, “Efficient implementation of public key cryptosystems on mote sensors (short paper),” in *Proceedings of the 8th International Conference on Information and Communications Security*, ser. ICICS’06. Berlin, Heidelberg: Springer-Verlag, 2006, pp. 519–528. [Online]. Available: http://dx.doi.org/10.1007/11935308_37

[7] J. Jonsson and B. Kaliski, “IETF RFC 3447: Public-Key Cryptography Standards (PKCS) #1: RSA Cryptography Specifications Version 2.1,” Feb. 2003. [Online]. Available: http://tools.ietf.org/html/rfc3447

[8] J. Ayuso, L. Marin, A. J. Jara, and A. F. G. Skarmeta, “Optimization of Public Key Cryptography (RSA and ECC) for 16-bits Devices based on 6LoWPAN,” in *1st International Workshop on the Security of the Internet of Things, Tokyo, Japan*, 2010.

[9] D. Eastake and P. Jones, “IETF RFC 3174: US Secure Hash Algorithm 1 (SHA1),” Sep. 2001. [Online]. Available: https://tools.ietf.org/html/rfc3174

[10] H. Krawczyk, M. Bellare, and R. Canetti, “IETF RFC 2104: HMAC: Keyed-Hashing for Message Authentication,” Feb. 1997. [Online]. Available: http://tools.ietf.org/html/rfc2104

[11] T. Dierks and E. Rescorla, “IETF RFC 4346: The Transport Layer Security (TLS) Protocol - Version 1.1,” Apr. 2006. [Online]. Available: http://tools.ietf.org/html/rfc4346

[12] D. Eastlake 3rd and T. Hansen, “IETF RFC 4634: US Secure Hash Algorithms (SHA and HMAC-SHA),” Jul. 2006. [Online]. Available: http://tools.ietf.org/html/rfc4634

[13] Chipcon Products from Texas Instruments. (2006, Feb.) User Manual: CC1010DK Development Kit. [Online]. Available: http://www.ti.com/lit/ug/swru055/swru055.pdf

[14] Texas Instruments. (2008, Apr.) eZ430-RF2480 Demonstration Kit - User’s Guide. [Online]. Available: http://www.ti.com/lit/ug/swru151a/swru151a.pdf

[15] ——. (2012, Aug.) MSP430F22x2, MSP430F22x4 Mixed Signal Microcontroller Datasheet. [Online]. Available: http://www.ti.com.cn/cn/lit/ds/symlink/msp430f2274.pdf

[16] ——. (2011, Jun.) CC2480 Datasheet. [Online]. Available: http://www.ti.com/lit/ds/symlink/cc2480a1.pdf