Hash Algorithms for 8051-based Sensornodes

Manuel Koschuch, Matthias Hudler and Michael Krüger
Competence Centre for IT-Security, FH Campus Wien, University of Applied Science, Favoritenstrasse 226, 1100 Vienna, Austria

Keywords: Efficient Implementation, Hash Algorithms, Sensor Networks, Sensor Nodes, SHA-1, SHA-3, Tiger Hash.

Abstract: Wireless Sensors Networks are still an emerging technology. Their special architecture allows for unique applications that would be impossible, or at least very difficult, to implement using other technologies. But the wireless data transmission between the single nodes poses new challenges from a security point of view: the single messages have to be secured against eavesdropping and manipulation, as well as the individual nodes have to be secured against capture and extraction of their secret key. Cryptographic hash functions are an integral part of most cryptographic network protocols, whether they are used for signatures or message integrity. In this position paper, we describe a preliminary performance evaluation of three very different hash-functions on a Texas Instruments CC2530 sensor node, based on an 8051 microcontroller: Tiger, representing a hash designed for 64-bit architectures, the current standard SHA-1, and Grøstl, a SHA-3 finalist. Our preliminary results indicate that even without any major optimizations hash algorithms that were clearly not designed to run on constrained devices can be adapted to these environments with quite acceptable results, thereby giving designers of sensor network security protocols new implementation options.

1 INTRODUCTION

Wireless Sensor Networks (WSNs) are an ever emerging field of technology. The usage of a huge number of tiny sensor nodes, distributed across a large area and communicating their sensor readings in a per-hop fashion to a centralized base-station for processing allows for efficient measurement of environmental conditions, power plant or factory surveillance, traffic monitoring, or new applications in personal health care and ubiquitous computing.

But the distributed nature of WSNs also requires the implementation of security measures against eavesdropping on or manipulating the communication over the air interface. Single sensor nodes can also very easily and almost undetectable be removed from the network, so that an attacker can try to read out any secret keys stored in the node. Traditional approaches using asymmetric cryptography are usually assumed to be too resource intensive for the small, 8-Bit sensor nodes, while symmetric cryptography has the drawback of lacking authentication and the necessity to distribute the same key across a (potentially large) number of nodes.

But regardless of the actual cryptographic technique used, cryptographic hash-functions play a major role in almost every implementation of a secure communication scheme. Whether they are used for signature generation, key derivation, or HMAC calculations, an efficient and secure way of hashing usually quite small amounts of data is needed.

There is already a huge number of publications dealing with security protocols for WSNs, some of them also heavily relying on hash-functions. But when it comes to evaluating the performance of different hashes on small, constrained devices like 8-bit microcontrollers, or whether there are viable alternatives to the ubiquitous SHA-1, there is a much lower number of research results available.

In this position paper we try to make a first, deliberately rough, approach of trying to compare the performance of different hash-functions, also some that have never been designed with embedded devices in mind, on a Texas Instruments CC2530 Sensor node, using an 8051 microcontroller with 32MHz clock frequency. Our goal is to determine whether is is sensible to also consider non-standard, but nevertheless secure hash-functions when designing a custom security protocol for a WSN, or if the existing standards also satisfy the special requirements of embedded devices.

The remainder of this position paper is structured

Koschuch M., Hudler M. and Krüger M..
Hash Algorithms for 8051-based Sensornodes.
DOI: 10.5220/0004062400650068
In *Proceedings of the International Conference on Data Communication Networking, e-Business and Optical Communication Systems* (DCNET-2012), pages 65-68
ISBN: 978-989-8565-23-5

as follows: in Section 2 we give an overview of the properties of a hash-function that are required to consider that function *secure* as well as a short description of the functions selected for this first test. Section 3 details our modifications to the hash functions to make them work on the target device and first results, while finally Section 4 describes the next steps planned.

2 HASH FUNCTIONS

A general hash-function has the property of mapping arbitrary length input to fixed length output, and doing so efficiently. So the first two properties of every hash-function are

- Compression and
- Efficiency

These two properties do not suffice for a *cryptographic* hash-function, where in addition the following characteristics must hold:

- Preimage Resistance
- Second-Preimage Resistance and
- Collision Resistance

For the following discussion, let $H(x) = y$ denote the hash-function H applied to the arbitrary length input x, resulting in the fixed length output y.

Preimage resistance means that given y it should be unfeasible to determine x.

Second-Preimage resistance means that given the tuple (x,y) it should be infeasible to find $x' \neq x$ such that $H(x') = y$.

Finally, collision resistance as the strongest property requires that it should be infeasible to find x and x' with $x \neq x'$ such that $H(x) = H(x')$.

The strongest attack against a given hash-function with n-bit output length is to find two distinct values yielding the same hash, which, due to the birthday paradox, results in an effort of $2^{n/2}$. With 2^{80} being the approximate practical complexity limit today, this translates into a minimal output length of the hash-function of 160 bit.

SHA-1 fulfills this requirement, as well as all the final candidates of the SHA-3 competition. For the purpose of this work, we selected three different hash-functions and discussed a fourth one.

2.1 Tiger

The *Tiger* hash-function (Anderson and Biham, 1996) was designed for 64-bit processors, and serves as more or less a "worst-case" design for an 8-bit microcontroller. It consists of (at least) 3 passes, with 8 rounds each, utilizing 4 S-Boxes with a combined size of 8 Kbytes. It operates on 64-byte input blocks and generates a 192-bit digest. Internally only additions, subtractions, multiplications and xor's of 64-bit values are used, in addition to the S-Box lookups. This structure allows for a very efficient implementation on machines which are able to handle 64 bit natively, but poses a huge challenge when ported to an 8-bit microcontroller.

We chose this algorithm mainly as a "proof-of-concept" to find out whether porting of such a hash-function with clearly different design goals in mind results in at least decent performance or is even possible at all.

2.2 SHA-1

The Secure-Hash-Algorithm (SHA) family is still (since 1995, 1993 when one also counts SHA-0) the current standard for hash algorithms. Currently there are basically 2 different versions used, SHA-1 with an output size of 160 bit, and SHA-2 with an output size of 224, 256, 384 or 512 bit, respectively (NIST, 2002).

Currently there are no known published attacks against full SHA-2, yet there do exists attacks against SHA-1 (Manuel, 2011). In this work, we use a SHA-1 implementation as representation for the still most widely deployed hash-function.

SHA-1 takes 64-byte input blocks, and produces a 160-bit message digest. It is designed with 32-bit machines in mind and consists of 80 rounds, with the round function changing every 20 rounds. Internally, only logical operations and circular shifts are used.

2.3 AES-based Hashes

The usual approach of designing a hash-function is to use some function block with compression property, whose output looks as random as possible, and repeatedly apply this function to the input until all message blocks are processed. Since a block cipher can also be seen as a compression function (given n-bit block size and a k-bit key, the cipher maps this $(n+k)$-bit Input to an n-bit output), using the current standard for symmetric cryptography as a hash bulding block seems like a natural choice.

The problem when simply using the Advanced Encryption Standard (AES) with the standard construction of a hash-function lies in its block-size: while Rijndael, the underlying algorithm for AES, supports larger blocks, AES is fixed to 128-bit block

size. This results in a 128-bit digest, too small to be considered secure against simple brute-force attacks.

In (Bos et al., 2011) several techniques to increase this digest are proposed, some of them we are currently evaluating on the sensor node. Using an AES-based hash would allow the usage of the existing AES co-processor on the node, in this way hopefully mitigating the additional complexity required to extend the digest size.

2.4 SHA-3

The 6-year long competition to find the new standard hash algorithm SHA-3 is expected to be finished by the end of 2012. Currently, there are five finalists left: BLAKE (Jean-Philippe Aumasson and Luca Henzen and Willi Meier Raphael C.-W. Phan, 2010), Grøstl (Gauravaram, P. and Knudsen, L. R. and Matusiewicz, K. and Mendel, F. and Rechberger, C. and Schlffer, M. and and Thomsen, S. S., 2011), JH (Wu, H., 2011), Keccak (Bertoni, G. and Daemen, J. and Peeters, M. and Assche, G. v., 2011), and Skein (Ferguson, N. and Lucks, S. and Whiting, B. S. D. and Bellare, M. and Kohno, T. and Walker, J. C. J., 2008). All of these functions have a digest size of at least 224 bits.

We chose Grøstl as an example candidate for a first comparison, although an evaluation of the remaining 4 candidates is also planned. Grøstl operates on a 64-byte input, producing a 256 or 512-bit message digest. Internally, it's structure is very similar to AES, even using the same S-Box as defined for Rijndael, and also using similar approaches operating on a two-dimensional state. The individual round operations are repeated 10 or 14 times, for 256 or 512 bit digest size, respectively.

3 IMPLEMENTATION AND PRELIMINARY RESULTS

Tiger, SHA-1 and Grøstl-256 were implemented on a Texas Instruments Zig-Bee compliant CC2530 Sensornode, featuring an 8051 microcontroller core clocked at 32MHz. As compiler, the IAR C compiler for 8051, version 8.10.1, was used.

As a first test, no assembler optimizations were performed, the code was entirely written in C.

In the case of SHA-1, the reference implementation given in RFC3174 (Eastlake, D. and Jones, P., 2001) was slightly modified, some functions were exchanged by macros for better speed.

Tiger was entirely ported to 32-bit datatypes that are supported in IAR declaring them as *long long*. All basic operation on the 64-bit values from the the Tiger reference implementations were rewritten to operations on 32-bit datatypes. We also performed an additional split and tried to use 16-bit datataypes instead, but due to the resulting increase in codesize, the 16-bit version ran slower then the 32-bit one.

For Grøstl we also used the reference implementation from the project homepage. Again, slight modifications were performed to adapt the code to the 8051 platform.

We then used the Tiger reference test vectors as input to all three hash-functions and timed the number of cycles needed for processing a single block (that is, 64 Bytes or 512 Bits, respectively, for all hash-functions analyzed). Table 1 gives an overview of our results, while Table 2 compares the code sizes for the three implementations.

Our results show that, barring any optimizations but the most obvious ones, even a hash-function designed for 64-bit machines like Tiger can achieve acceptable performance on an 8051 microcontroller. The bad results for Grøstl are surprising yet will be optimized during the next steps, at least on an ATmega163 Grøstl has already been shown to perform very favorably. Again, our focus in this first round was only on a comparison of unoptimized versions of the algorithms, ported straightforward from the reference implementations.

Table 1: Number of cycles per 64-byte input block.

Hash	cycles/block
Tiger	193,944
SHA-1	561,589
Grøstl-256	2,813,496

Table 2: Code size after compilation , optimized for *speed*.

Hash	Code Size in kB
Tiger	85.332
SHA-1	58.936
Grøstl-256	89.880

4 OUTLOOK

We performed a preliminary performance analysis of three hash functions from different domains (one designed for 64-bit machines, the current standard, and one SHA-3 finalist) on a TI CC2530 sensor node, using an 8051 microcontroller. Our main goal was to answer the question whether a more or less arbitrarily selected, secure cryptographic hash-function can be implemented on a sensor node, heavily constrained in terms of available memory and processing power,

without extensive optimizations and still achieve acceptable performance.

The first results are promising, although there is still a lot of work to be done: in addition to implementing the remaining four SHA-3 finalists, we currently investigate the use of the sensornodes' AES coprocessor to implement and speed up AES-based hash constructions, with the final goals of identifying an efficient, secure cryptographic hash-function apart from SHA-1 that can be used as a primitive building block of a security framework for wireless sensor networks and giving a general indication how much effort has to be put into optimizing a chosen hash-function for constrained environments.

REFERENCES

Anderson, R. and Biham, E. (1996). Tiger: A fast new hash function. In *Fast Software Encryption, Third International Workshop Proceedings*, pages 89–97. Springer-Verlag.

Bertoni, G. and Daemen, J. and Peeters, M. and Assche, G. v. (2011). The keccak reference.

Bos, J. W., Özen, O., and Stam, M. (2011). Efficient hashing using the aes instruction set. In *Proceedings of the 13th international conference on Cryptographic hardware and embedded systems, CHES'11*, Lecture Notes in Computer Science, pages 507–522. Springer-Verlag.

Eastlake, D. and Jones, P. (2001). Rfc3174 - us secure hash algorithm 1 (sha1).

Ferguson, N. and Lucks, S. and Whiting, B. S. D. and Bellare, M. and Kohno, T. and Walker, J. C. J. (2008). The skein hash function family.

Gauravaram, P. and Knudsen, L. R. and Matusiewicz, K. and Mendel, F. and Rechberger, C. and Schlffer, M. and and Thomsen, S. S. (2011). Grøstl - a sha-3 candidate.

Jean-Philippe Aumasson and Luca Henzen and Willi Meier Raphael C.-W. Phan (2010). Sha-3 proposal blake.

Manuel, S. (2011). Classification and generation of disturbance vectors for collision attacks against sha-1. *Des. Codes Cryptography*, 59(1-3):247–263.

NIST (2002). Secure hash signature standard (shs) (fips pub 180-2). Technical report, National Institute of Standards and Technology.

Wu, H. (2011). The hash function jh.