FPGA Implementation of SQUASH

François Gosset[1] François-Xavier Standaert[2]
Jean-Jacques Quisquater
Université catholique de Louvain
Crypto Group, 3 Place du Levant
1348 Louvain-la-Neuve, Belgium
francois.gosset@uclouvain.be fstandae@uclouvain.be
jean-jacques.quisquater@uclouvain.be

Abstract

Passive RFID tags are devices with very poor computational capability. However an increasing number of applications require authentication of the tag. For this purpose, a simple solution is to use a challenge-response protocol. For example, the reader can send a random challenge R and the tag responds $H(R \oplus S)$, where S is a secret known by the reader and H is a hash function. Then, the reader can check whether the tag knows S. SQUASH, introduced by Adi Shamir in February 2008 [1], is a new hash function designed for this task. In this article, we describe an FPGA implementation of this algorithm minimizing the resources.

SQUASH is based on the one-way function $c = m^2 \mod n$ coming from the Rabin cryptosystem [2]. To make it secure, the binary length of n must be at least 1000 bits long [8]. In [1], the author suggests using a 64-bit non-linear feedback shift register to generate m, a not yet factorized Mersenne's number $(2^x - 1)$ as modulus n, and to send out the bits of c without storing them. This process avoids storing three 1000-bit long numbers. The multiplications are achieved by on-the-fly convolutions, sending each bit as soon as it is computed. Consequently, the only needed memory aims at storing the carry of the previous steps in the convolution. For the output, a window of 32 or 64 (or more) bits in c is used. It yields a hash function with inputs of 64 bits that is scalable in output.

In this paper, we propose implementations designed in order to minimize the resources, possibly at the cost of an increased execution time. The target device is a Xilinx Virtex-4 XC4VLX200-10 FPGA. The algorithm recommended by Adi Shamir has 64 bits in input and 32 in output, and $n = 2^{1277} - 1$. To reduce the hardware cost, we minimized the number of registers in the implementation data and control part. On the XC4VLX200, the design results require 377 slices. The full execution time to produce 32 bits is 63,250 clock cycles at 222 MHz, so we reach a throughput of 112,300 bits per second. We also implemented the algorithm with other size numbers. For 128 bits in input and 64 in output, we get 619 slices and 104,114 clock cycles at 206 MHz. In general, the length of the output influences the execution time while the length of the input influences the number of registers and slices.

[1]Supported by the Walloon region (Project E.USER, Belgium)
[2]Supported by the FNRS (Belgium)

1 Introduction

In the last few years, RFID tags have been introduced in an increasing number of applications. They can now be found in supermarket antithefts, warehouse inventory control, public transportation passes, pet identification, secure passports, etc. With the multiplication and specialization of their uses, RFID tags need to be secured. Classical cryptography, however, is generally unsuitable for low-cost tags. Indeed, their computational capability is weak (typically 1,000 to 10,000 gates shared between their application and the security) and their memory does not exceed several hundreds of bits [1]. In low cost applications, the storage of a single 1024-bit RSA key is already a real challenge. As a consequence, the design of new cryptographic primitives facing strong implementation constraints is an important research direction.

Classical RFID applications require the possibility for the tag to be able to interactively authenticate itself securely to a reader. A simple challenge-response protocol can be used for this purpose. This protocol implies that, when challenged by the reader, the tag is able to prove that it knows a secret S without revealing it.

Namely, the reader first sends the challenge R to the tag. Then the tag computes $H(R \oplus S)$ and sends it to the reader. Finally, the reader computes $H(R \oplus S)$ and compares it with the response of the tag. The secret is not revealed if it is computationally hard to extract S knowing $H(R \oplus S)$ and R. Computationally hard means that it is impossible to compute with current technologies. Such a scenario can be performed with hash functions. Most of them (like SHA-1) are designed to be one-way and collision resistant, although the collision resistant aspect is not important in the present application. In February 2008, Adi Shamir introduced a new hash function, called SQUASH, to perform this challenge-response protocol [1]. Due to its low-cost objective, it was expected to provide an efficient alternative for lightweight hardware implementations.

The goal of this work is to present an FPGA implementation of SQUASH, as a first step in the assessment of the hardware cost of this new hash function. FPGA's were chosen because they make it possible to quickly implement a hardware design and because they are comparison point, in terms of frequencies or hardware resources.

In this paper, we first describe the SQUASH algorithm. Then, we discuss the selection of its parameters. Finally, we present its implementation results in a recent FPGA family of devices.

2 The SQUASH algorithm

2.1 From Rabin's scheme...

Rabin encryption scheme is a public-key cryptosystem [2]. The public key n is generated from two prime numbers p and q. The couple p, q is the secret key. If m is the message that must be encrypted, the ciphertext is $c = m^2 \mod n$. The security of this scheme is related to the difficulty of factorization; so, to make it secure, the binary length of n must be at least 1000 bits long [8]. The function $c = m^2 \mod n$ is a good one way function, but not a collision resistant one. Indeed, m, $-m$, $m + n$,... have the same response c. Because it has been studied for more than 30 years, Rabin's scheme is a good candidate for the construction of a secure one way function. However, it requires storing m, n and c, which means roughly 3000 bits in memory. Moreover, a modular square must be computed, which is usually too expensive for small devices like RFIDs. Therefore, the goal of SQUASH is to reduce these resources.

2.2 ... to SQUASH!

SQUASH simplifies Rabin's scheme by reducing the size of the operands, and by simplifying and approximating the modular square operation. In this section, we use 64-bit length input and 32-bit length output as suggested in [1]. However, it is not a requirement of the algorithm and we implemented SQUASH with different sizes, as described in section 3.1.

Do not store big numbers. Rabin's scheme requires computing $c = m^2 \mod n$ where c, m and n are at least 1000 bit long. SQUASH avoids storing them. First of all, m is not stored but generated from the 64-bit long input of SQUASH (in the example of the challenge-response protocol explained in the introduction, the input is simply $R \oplus S$). The input is used as a seed for a reversible nonlinear feedback shift register (NLFSR). It is easy to generate successive bits of m when they are needed in the forward and backward direction: $m = \text{NLFSR}(R \oplus S)$. To avoid storing n, SQUASH uses universal moduli, which have a simple and efficient binary representation, i.e. the Mersenne numbers that have the from $2^k - 1$. A good candidate is $2^{1277} - 1$, a composite number with no known factors. Finally, the output of SQUASH is a 32-bit length number. So, SQUASH uses only a short window in the Rabin ciphertext as output hash. This window is computed in the center of c, starting at c_{616} with a length of 32 bits. This avoids storing the 3000 bits for m, n and c.

Compute the multiplication by on-the-fly convolution. The multiplication is computed by on-the-fly convolutions. They require only an 12-bit register for the carry generated by the previous steps in the convolution, and a 12-bit adder.

Approximate the value of the window. The bits of the window are not computed exactly, but approximated. To compute the exact least significant bit (LSB) of the window, all the previous bits should have to be computed. However, SQUASH extends the window with only 16 additional bits, called guard bits. SQUASH computes the value of the extended window assuming that the carry into its LSB was zero. It is sufficient because the carry into each bit position in the computation of m^2 can be at most 11-bit long. Then, if we add 16 guard bits to the computed window, we have only a small probability of less than 1/32 of computing an incorrect carry into the 17^{th} bit we compute.

Modular reduction $\mod n = 2^k - 1$. The use of Mersenne number as a modulus gives an advantage for the computation of the modular reduction. Let g_1 be the upper half of m^2 and g_0 the lower half of m^2; so $m^2 = g_1 2^k + g_0$. If the modulus n is $2^k - 1$, the modular reduction is very easy and requires only an addition: $m^2 = g_1 + g_0 = c \mod n$ since $2^k = 1 \mod n$. To compute the bit j in the lower half of m^2, we have to sum (over the integers, not modulo 2) all the products $m_v.m_{j-v}$ for $v = 0, 1, ..., j$ and add to the carry from the computation of the previous bit this sum. To compute the bit $j + k$ in the upper half of m^2, we have to sum (over the integers, not modulo 2) all the products $m_v.m_{j-v+k}$ for $v = j + 1, .., k - 1$ and add the carry from the computation of the previous bit to this sum. And to compute the bit j in $c = m^2 \mod n$, where $n = 2^k - 1$, we have to add bits j and $j + k$ of m^2, along with their carries.

The final SQUASH algorithm is:

Set $c = 0$, $m = \text{NLFSR}(R \oplus S)$
for $j = 600$ to $j = 647$
  $c = \sum_{v=0}^{1276} m_v.m_{(j-v \mod k)} + c$
  $c_j = c_0$
  $c = \lfloor \frac{c}{2} \rfloor$
Output the 32 bits $c_{616}, ..., c_{647}$

3 Implementation

3.1 The parameters

In SQUASH, different parameters have to be fixed. They are the modulus $n = 2^k - 1$, the nonlinear feedback shift register, the size of the inputs and the size of the outputs.

The modulus n. As described in section 2.2, we chose $n = 2^{1277} - 1$, as suggested in [1], but it is not the only interesting Mersenne number $2^k - 1$. Alternatives like $2^{1061} - 1$ and $2^{1237} - 1$, both composite numbers not yet factorized, could have been used.

The NLFSR. Another important parameter to fix is the form of the nonlinear feedback shift register. A *linear* shift register cannot be used because a polynomial-time attack exists[3] when the short mixed value $R \oplus S$ is expanded by a linear feedback shift register, and then squared modulo $n = pq$. For the implementations, we chose a nonlinear feedback shift register with a feedback function of this form:

$$f(x^1, ..., x^n) = g(x^1, ..., x^n) \oplus \overline{x^1}.\overline{x^2}...\overline{x^{n-1}},$$

where $g(x^1, ..., x^n)$ is a feedback polynomial of the same-size maximum-length *linear* shift register, and $\overline{x}$ is NOT(x). A NLFSR with this feedback function $f(x^1, ..., x^n)$ is a maximum-length NLFSR [3].

The size of the inputs and the outputs. In [1], Shamir suggests to use a 64 bit-length input and a 32 bit-length output, but we chose to be more flexible on the size of the inputs and the outputs. The scalable behavior of the output is very simple to get. We just have to compute more bits in the window of c. It does not require more hardware resources (except the size of the counter that counts the number of bits in the window of c computed), just more computation time. We implemented SQUASH with output sizes of 32, 64 and 128 bits. Smaller outputs would result in a too weak security level and larger outputs are probably not necessary for RFID applications.

Changing the size of the inputs implies more changes in the architecture. Indeed, the construction of the nonlinear feedback shift register directly depends on the size of the inputs. However, the remainder of the architecture is the same because the goal of the nonlinear shift register is to produce the number to be squared. We implemented SQUASH with two input sizes: 64 and 128 bits. This choice was guided by the same considerations as the size of the outputs: smaller inputs would give unacceptable security levels (in the case of a 32-bit length input, the complete response of the hash function can be tabulated with only 4.3×10^9 entries) and bigger inputs would make no sense in the context of RFID.

[3] Developed by Serge Vaudenay in a private communication [1].

3.2 Architecture

The operative part of SQUASH includes on the one hand the nonlinear feedback shift registers and on the other hand the 12-bit adder. The nonlinear feedback shift registers provide the bits of m used for the multiplication, and the 12-bit adder computes the sums used for the multiplication and the reduction (see section 2.2). The 1-bit multiplier is just the logic operator AND.

The multiplications are computed by convolution: $c_j = \sum_{v=0}^{1276} m_v.m_{(j-v \mod 1277)} +$ carry. To achieve this convolution, we use two nonlinear feedback shift registers. The first NLFSR has to provide the bits of m in the forward direction, from m_0 to m_{1276}; and the second NLFSR in the backward direction, from m_{1276} to m_0. When the first NLFSR delivers m_{1276}, the next delivered bit must be m_0. In the same way, when the second NLFSR delivers m_0, the next delivered bit must be m_{1276}. To implement this jump in the succession of the bits of m avoiding an unreasonable increase of the execution time, we use a third register. It stores the state m_0 or m_{1276} and provides the state required by one or the other shift register. This is illustrated by Figure 1.

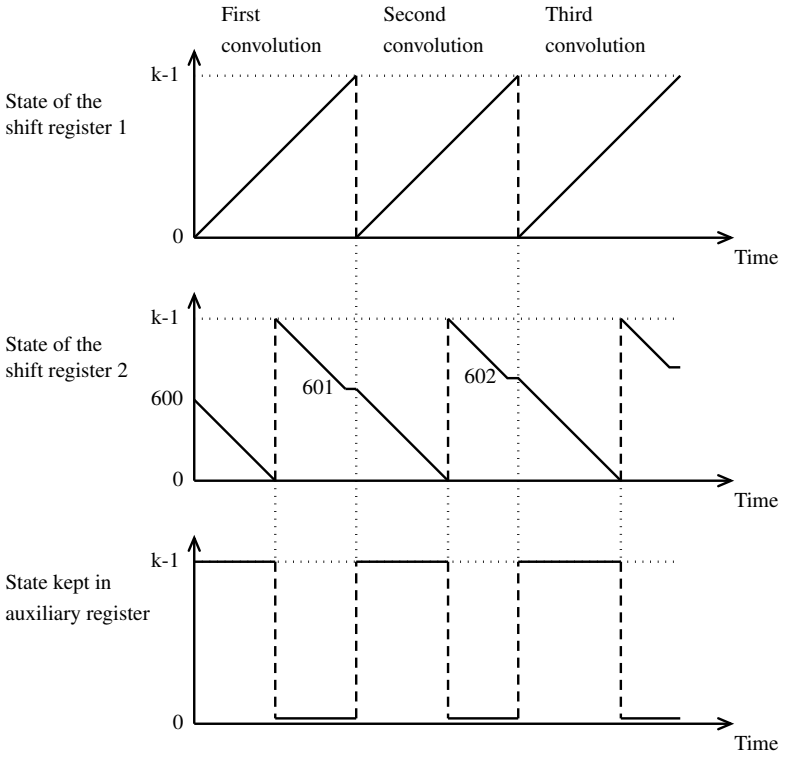

Figure 1: The state of the registers during the convolutions.

Shift register 1 starts at m_0. Shift register 2 starts at m_{600}. The auxiliary register is in the state m_{1276}. When the computation of the convolution begins, the first shift register increases and the second one decreases. When shift register 2 reaches m_0, the auxiliary register provides the state m_{1276} and stores the state m_0 given by the second shift register. This state m_0 will be given at the first shift register, and so on.

For the first convolution, the first shift register must give m_0, the second one m_{600} and the auxiliary register must be at m_{1276}. However, at the beginning of the execution, the only available state is m_0 (remember that m is generated by the nonlinear feedback shift register, whose seed is $R \oplus S$). An initialization phase is consequently required:

1. Load $R \oplus S$ in the three registers.

2. Shift the first nonlinear feedback shift register 1276 times forward. Then, it will be at the state m_{1276}.

3. Load m_{1276} in the second nonlinear shift register, and swap the contents of the first nonlinear feedback shift register and the auxiliary register. Now, the shift register 1 has the state m_0 and the two other registers the state m_{1276}.

4. Shift the nonlinear feedback shift register 2 backward to get m_{600}. Then, the first shift register has m_0, the second m_{600} and the auxiliary register m_{1276}.

5. The initialization phase is finished. The second shift register can go on shifting backward, and the first one starts shifting forward, as shown in figure 1.

To perform the addition, a 12-bit adder is used (Figure 2). The adder adds the products $m_i.m_{j-v}$ comming from the NLFSR's. The counter counts the additions in the convolution $\sum_{v=0}^{1276} m_v.m_{(j-v \mod 1277)} + \text{carry}$. When a new convolution starts, the least significant bit of the result of the previous convolution can be sent and the right-shifted result of the previous convolution is the new carry into the new convolution.

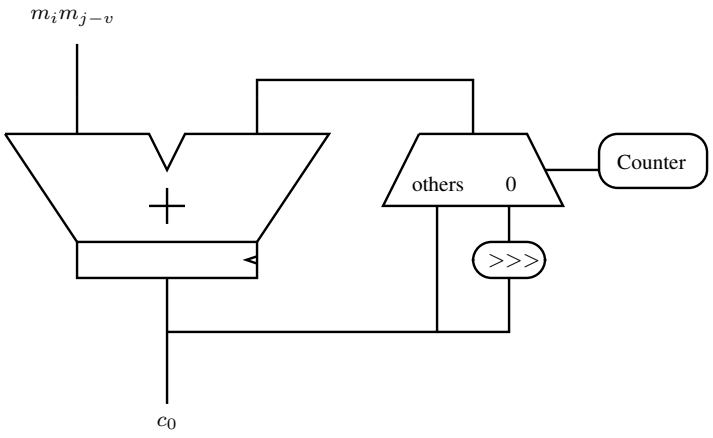

Figure 2: Architecture of the adder.

Finally, the control part of the algorithm is based on counters. Each nonlinear feedback shift register has a counter memorizing the state in which they are. Simple tests on those counters give the load and the shift instructions for the registers.

4 Results

Implementation results were achieved for the XC4VLX200 Xilinx Virtex4 LX FPGA, with the lowest speedgrade. ISE 9.1i was used for synthesis and place & route while test/debug was performed with Modelsim SE 6.1d. SQUASH was implemented with different parameters. The sizes of the inputs are 64 or 128 bits (it means that the secret S and the challenge R have to be 64- or 128-bit long), and the sizes of the outputs are 32, 64 or 128 bits. Tables 1 and 2 present our implementation results from which the following observations can be highlighted:

1. First, the size of the outputs only influences the execution time. Indeed, to get one more bit in output, SQUASH just needs to compute one more convolution, so 1277 clock cycles. Moreover, for each full execution of SQUASH, a fixed

initialization phase[4] of 22,386 clock cycles is required, regardless of the size of the inputs and the outputs. The bit rate in table 1 includes this initialization phase.

2. By contrast, the size of the inputs influences the hardware cost. On a Xilinx Virtex4 FPGA, SQUASH requires approximately 400 slices when the input is 64-bit long and 700 slices when it is 128-bit long. The difference comes from the larger shift register used when the input is 128-bit long. Table 2 also mentions the number of gates, but these numbers strongly relate to the target FPGA. Therefore, they do not represent the number of gates needed to make an ASIC like a real RFID tag. In such a device, we can reasonably expect that there will be fewer gates. So, this number can be used as an upper bound of the number of gates, to compare with other implementations. SQUASH requires between 6,000 and 7,000 gates for a 64-bit long input, and between 11,000 and 13,000 gates for a 128-bit long input.

Input size # bits	Output size # bits	Frequency MHz	# clock cycles (full execution)	Execution time ms	Bit rate bit/s
64	32	222	63,250	0.285	112,300
64	64	217	104,114	0.479	133,400
64	128	208	185,842	0.893	143,300
128	32	222	63,250	0.285	112,300
128	64	212	104,114	0.491	130,300
128	128	206	185,842	0.902	141,900

Table 1: Timing results.

Input size # bits	Output size # bits	# registers	# slices	# gates	Frequency MHz
64	32	252	377	6,303	222
64	64	254	378	6,328	217
64	128	258	425	7,089	208
128	32	451	613	11,237	222
128	64	452	619	11,293	212
128	128	469	756	12,920	206

Table 2: Hardware resources needed.

Finally, these results can be compared with the hardware costs of standard cryptographic algorithms. In [5], a very small hardware implementation of the Advanced Encryption Standard AES is presented. It is optimized for low-resource requirements, like RFID tags and requires 3400 gates. In [6], a strong symmetric authentication using AES is presented. This implementation requires 3600 gates. Compared with those two implementations, SQUASH shows a higher implementation cost. But it is to be traded with its interesting security properties. Additionally, ASIC implementations of SQUASH would certainly give rise to lower gate counts. As a matter of fact, the present FPGA implementation is a only first prototype purposed to obtain intuitions on the performances of SQUASH.

[4]The initialization phase includes the initialization of the shift registers (section 3.2) and the computation of the 16 guard bits (section 2.2).

5 Conclusion

This work presents an FPGA implementation of the SQUASH algorithm. The motivation was to assess hardware cost of this algorithm designed for low-cost RFID tags. This kind of device has between 1,000 and 10,000 gates. We can reasonably devote one quarter of those resources to the security; it means 250 to 2,500 gates for the biggest tags. Our implementation of SQUASH requires at most 6,000 gates. It is an upper bound, because this number represents the gates used in the FPGA (as discussed in section 4). Currently, it seems too much for low-cost RFIDs, but it gives some encouraging perspectives. On the one hand, we can reasonably expect doing better with an ASIC implementation. On the other hand, SQUASH relies on a well investigated mathematical problem, which consequently gives better confidence in its security. Indeed, although SQUASH itself is not provably secure, it is based on a construction (SQUASH without the NLFSR) that is proved to be as secure as Rabin's scheme. In [1], Shamir argues that the NLFSR is probably a good choice to complete the construction. We followed this suggestion. However, it should be noted that another function could be used with limited impact on the hardware cost if the NLFSR exhibits weaknesses.

References

[1] Adi Shamir, *SQUASH - A New MAC With Provable Security Properties for Highly Constrained Devices Sush As RFID Tags*, Proc. Fast Software Encryption - FSE 2008, Lausanne, Switzerland, February 2008

[2] Michael Rabin, *Digitalized Signatures and Public-Key Functions as Intractable as Factorization*, Massachusetts Intitute of Technology, Laboratory for Computer Science, TR-212, January 1979

[3] Alfred J. Menezes, Paul C. van Oorschot and Scott A. Vanstone, *Handbook of Applied Cryptography*, CRC Press, August 2001, pp. 202-203.

[4] Xilinx, Inc., *Virtex-4 User Guide*, available at www.xilinx.com.

[5] Martin Feldhofer, Johannes Wolkerstorfer and Vincent Rijmen, *AES Implementation on a Grain of Sand*, IEE Proc., vol. 152, no. 1, Oct. 2005, pp. 13-20.

[6] Martin Feldhofer, Sandra Dominikus and Johannes Wolkerstorfer, *Strong Authentication for RFID Systems Using the AES Algorithm*, Proc. Workshop on Cryptographic Hardware and Embedded Systems - CHES 2004, Cambridge (Boston), Ma., USA, Springer, vol. 3156, August 2004, pp. 357-370.

[7] Thomas Eisenbarth, Sandeep Kumar, Christof Paar, Axel Poschmann and Leif Uhsadel, *A Survey of Lightweight-Cryptography Implementations*, IEEE Design & Test of Computers, vol. 24, no. 6, November 2007, pp. 522-533.

[8] Steve Babbage, Dario Catalano, Carlos Cid, Louis Granboulan, Tanja Lange, Arjen Lenstra, Phong Nguyen, Christof Paar, Jan Pelzl, Thomas Pornin, Bart Preneel, Vincent Rijmen, Matt Robshaw, Andy Rupp, Nigel Smart and Michael Ward, *ECRYPT Yearly Report on Algorithms and Keysizes (2006)*, D.SPA.21 Rev. 1.1, January 2007.