Compact FPGA Implementations of QUAD

David Arditti Côme Berbain Olivier Billet Henri Gilbert

France Telecom R&D
38–40, rue du Gl Leclerc
92794 Issy les Moulineaux Cedex 9 — France
forname.lastname@orange-ftgroup.com

ABSTRACT

QUAD is a stream cipher whose provable security relies on the hardness of solving systems of multivariate quadratic equations. This paper explores FPGA implementations of the stream cipher QUAD and, more specifically, small area ones. Our smallest implementation of QUAD requires only 85 slices (2961 GE) on a Virtex 4 Xilinx FPGA, which makes it not only the smallest provably secure stream cipher, but also a very good competitor among conventional stream ciphers. In particular, we demonstrate an implementation of QUAD's underlying PRNG which results in a 68% improvement over the smallest known AES implementation on FPGA [13].

Categories and Subject Descriptors

B.m [**Hardware**]: Miscellaneous; E.3 [**Data**]: Data Encryption

Keywords

FPGA, RFID, stream cipher, PRNG, forward security, user privacy, hardware implementation.

1. INTRODUCTION

With the development of pervasive devices, highly efficient and/or compact implementations of cryptographic primitives such as pseudo-random number generators and encryption algorithms becomes of special interest. The applications are manyfold, ranging from privacy protection and communication confidentiality to device authentication. Symmetric encryption can be divided into two main types: block ciphers and stream ciphers. Block cipher design may currently be more mature than stream cipher design, but stream ciphers remain of interest because of their efficiency or their compactness and are thus often used in practice. As suggested by the eSTREAM [19] call for stream cipher proposals, at least two profiles seem to be of interest: stream ciphers that are much faster than existing block ciphers, and stream ciphers that require much lower resources for their hardware implementation than existing block ciphers. However about one third of the eSTREAM candidates have already been broken which seems to show that building both a secure and hardware efficient stream cipher is not an easy task.

From a security point of view, a potential advantage of stream ciphers–that is not sufficiently used in practice–is their close connection with the theory of secure pseudo-random number generators (PRNGs) which was developed in the 80's. Several provable secure PRNGs have been proposed. A general construction uses iterations of a one way function and the security proof relates the indistinguishability of the generated sequence to the one wayness of the function. One of the first provably secure PRNGs was introduced by M. Blum and S. Micali in [10] and relates the security of the PRNG to the one-wayness of exponentiation modulo a prime number. Later, L. Blum, M. Blum, and M. Shub proposed in [9] another PRNG exploiting the conjectured intractability of quadratic residuosity modulo Blum integers, and Alexi, Chor, Goldreich and Schnorr proposed in [4] a PRNG construction which security relies on the RSA assumption. Recently [41], R. Steinfeld, J. Pieprzyk, and H. Wang introduced another PRNG based on a new assumption (the so called small solution RSA assumption). Finally, R. Impagliazzo and M. Naor [32] and J.-B. Fischer and J. Stern [42] proposed PRNG constructions respectively relying on the difficulty of the subset sum problem and of the syndrome decoding problem. Despite the theoretical interest of those constructions, none really allows for efficient hardware implementations. Those PRNGs either have a huge internal state (up to 3000 bits), or use a highly expensive one way function (exponentiation modulo a big prime number), or only extract a few bits per iteration.

At Eurocrypt 2006, a new stream cipher with provable security called QUAD was proposed [7]. This cipher has strong security properties since the indistinguishability of the associated PRNG is proved to be reducible to the intractability of the well studied problem of solving multivariate quadratic equations [24]. The smallest version of QUAD recommended in [7] has a small internal state of 160 bits and can extract up to 160 bits per iteration, which makes it attractive for practical use. An additional built-in feature of the stream cipher QUAD, is the natural one-wayness of the function–a randomly chosen multivariate quadratic system: as is shown in the sequel, this provides an additional feature called forward security, which makes QUAD particularly attractive when privacy is a concern as is the case with RFIDs [39].

This is an extended version of the work published in:
ASIACCS'07, March 20-22, 2007, Singapore, ©ACM 2007

We further explain in this paper other properties of QUAD that also perfectly fit the RFID setting. The reader has to keep in mind that this cipher has security proofs attached to it. Hence, a fair comparison should involve the PRNGs cited above [9, 10, 4, 32, 42] or optimised versions of these generators like [26]. Nevertheless, we will show in the following that our implementations of QUAD's PRNG demands far less space than any other provably secure PRNG and is close to the size of standard practical stream cipher implementations. Our smallest implementation requires less than 33% of the number of slices required by the smallest AES FPGA implementation known to date [13], making QUAD a good candidate for radio frequency tags (RFIDs). The purpose of this paper is to show that the strong security requirements satisfied by QUAD are compatible with an efficient hardware implementation.

This paper is divided as follows: Section 3 briefly describes the stream cipher QUAD. Section 4 describes our various FPGA implementations of QUAD's underlying PRNG. Section 5.1 compares the performance of our implementations of QUAD with several provably secure PRNGs as well as with state of the art implementations of the AES.

2. MULTIVARIATE QUADRATIC SYSTEMS

We consider a finite field $\mathrm{GF}(q)$. A multivariate quadratic equation (or equivalently a multivariate quadratic polynomial) in n variables over $\mathrm{GF}(q)$ is a polynomial of degree at most 2 in $\mathrm{GF}(q)[x_1, \ldots, x_n]$ which can be written as

$$Q(x) = \sum_{1 \le i \le j \le n} \alpha_{i,j} x_i x_j + \sum_{1 \le i \le n} \beta_i x_i + \gamma,$$

with coefficients $\alpha_{i,j}$, β_i, and γ in $\mathrm{GF}(q)$. In the particular case $q = 2$, which is the one considered in the sequel, monomials $x_i x_i$ and x_i are equal.

It is easy to see that the set $\mathcal{Q}$ of multivariate quadratic polynomials in n variables is an N-dimensional vector space over $\mathrm{GF}(q)$, where $N = \frac{1}{2}n(n+3)+1$ if $q \neq 2$ and $N = \frac{1}{2}n(n+1)+1$ if $q = 2$. A basis of this vector space is given by the $N-1$ distinct monomial functions of degree one or two, and the non-null constant polynomial. Any element of $\mathcal{Q}$ can be represented by the N-tuple of its $\mathrm{GF}(q)$ coefficients in this basis. Throughout the rest of this paper, by a randomly chosen quadratic polynomial in n unknowns we mean the quadratic polynomial represented in the above basis by a uniformly and independently drawn N-tuple of $\mathrm{GF}(q)$ coefficients.

A multivariate quadratic system S of m quadratic equations in n variables over $\mathrm{GF}(q)$ consist of a set $(Q_1, \ldots, Q_m)$ of m quadratic polynomials in n variables over $\mathrm{GF}(q)$. In the sequel, by a randomly chosen system of m quadratic form in n unknowns, we mean n independently and randomly chosen quadratic polynomials. Such a system is represented by mN coefficients uniformly and independently drawn from $\mathrm{GF}(q)$.

We define the problem of solving multivariate quadratic systems (MQ problem) as follows: given a multivariate quadratic system $S = (Q_1, \ldots, Q_m)$, of m quadratic equations over $\mathrm{GF}(q)$ find a value $x \in \mathrm{GF}(q)^n$, if any, such that $Q_i(x) = 0$ for all $1 \le i \le m$.

Depending on the respective values of n and m, instances of MQ can be either easy or very difficult to solve. For $m = 1$ the number of solutions is known [37] and it is quite easy to find one solution. When m is significantly smaller than n, that is for an underdefined quadratic system, finding a solution is much easier than the exhaustive search on the number of variables [14]. In the opposite situation of an overdefined system ($m > n$) providing m is about $N = \frac{1}{2}n(n+1)+1$ ($q = 2$ case) or about $N = \frac{1}{2}n(n+3)+1$ ($q \neq 2$ case) linearly independent quadratic equations, or more generally when nearly N linearly independent quadratic equations are available, solving an MQ problem is easy by linearization. The total complexity is then only $O(n^6)$. However, for general values of m and n the MQ problem is known to be NP-hard, even when restricted to quadratic equations over GF(2) (see [24, 22]) or over any finite field (see [40]).

Moreover, what makes the MQ problem particularly well suited for cryptographic applications is that it is conjectured to be very difficult not only asymptotically and in worst case, but already for small suitably selected values of m and n and in terms of the average complexity of solving a random instance. The problem seems to be the most difficult when m is close to n. For $m = n$ and $q = 2$ the complexity of the best known solving algorithms is $2^{n-O(\sqrt{(n)})}$ and thus rather close to the 2^n complexity of exhaustive search, and totally out of reach of existing computers for a random instance and values of n larger than 100. Even when $q = 2$, $m = kn$ and $k > 1$ is small enough compared with $\frac{n}{2}$, the best known algorithm such as XL [17] and improved variants of Buchbergers's Gröbner basis computation algorithm such as Faugère's F4 and F5 algorithms [33] are exponential in n for a randomly chosen quadratic system. Much research has been dedicated in the past years to the above problem [16, 15]. Bardet's Ph.D. thesis [6] provides an accurate analysis of the complexity of the most efficient algorithm computing Gröbner basis known to solve a random system of $m = kn$ equations in n unknowns.

3. THE STREAM CIPHER QUAD

This section provides a quick description of QUAD [7]. The stream cipher QUAD is a practical stream cipher with provable security which was introduced by C. Berbain, H. Gilbert, and J. Patarin at Eurocrypt 2006. Its security relies on the hardness of solving randomly generated systems of multivariate quadratic equations over a finite field. The security proof reduces the distinguishability of the keystream generated by QUAD to the MQ problem: QUAD iterates a one way function and extracts a certain number of bits at each iteration. (Iterating a one-way function and extracting a certain amount of hard core bits is a common way to design a provable secure PRNG.)

The keystream generation makes use of two systems S_{it} and S_{out} of multivariate quadratic equations both sharing the same n unknowns over $\mathrm{GF}(q)$, as described on Fig. 1. The first system S_{it} is used to update the internal state and thus contains n equations, whereas the second system S_{out} produces the keystream and contains $m-n$ equations. As explained in [7], the quadratic systems S_{it} and S_{out}, though randomly generated, are both publicly known.

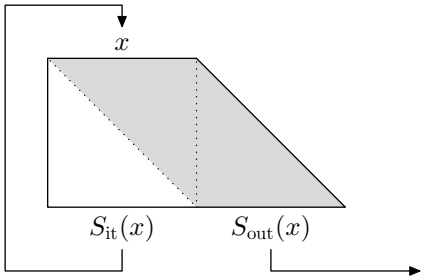

Figure 1: QUAD's internals: S_{it} updates the internal state x while S_{out} filter out the internal state to produce the pseudo random sequence

In our special case of a hardware implementation, it makes sense to focus on the boolean setting $\mathrm{GF}(q) = \mathrm{GF}(2)$. We will restrict our study to the conservative case $m = 2n$, that is both systems S_{it} and S_{out} contain n quadratic equations in the n bits of the internal state. In the sequel and according to Fig. 1, we denote by $x = (x_1, \ldots, x_n)$ the n-bit internal state of the generator.

Hence, keystream generation amounts to iterating the following steps:

- Compute $\big(S_{\text{it}}(x), S_{\text{out}}(x)\big) = \big(Q_1(x), \ldots, Q_{2n}(x)\big)$, with x as the current value of the internal state;
- Output the sequence $S_{\text{out}}(x) = \big(Q_{n+1}(x), \ldots, Q_{2n}(x)\big)$ of n keystream bits;
- Update the internal state x with the sequence of the n first generated bits $S_{\text{it}}(x) = \big(Q_1(x), \ldots, Q_n(x)\big)$.

To turn this secure pseudo-random generator into a stream cipher, an initialization procedure is required which necessitates the introduction of an initialization vector (IV). This step described in [7] is done by reusing the components which actually produce the keystream and thus do not require much more circuitry. (It requires about 10 additional gate equivalents for our smallest proposal and about 100 additional gate equivalents for our medium size proposal.) Moreover, in some contexts, such as in the RFID setting, the IV is not required at all. Consequently we restrict in the following our attention to the problem of implementing QUAD's underlying PRNG alone.

It has been proved in [7] that given a random system and a random initial value, the keystream sequence produced by QUAD is indistinguishable from a random sequence if the problem of inverting a randomly chosen system of multivariate quadratic equations is intractable. However, although equations of the system have to be randomly generated, they does not need to be kept secret. The FPGA designs proposed in the next section take advantage of this simple fact.

In the sequel, we consider three different versions of QUAD: one with only $n = 128$ bits of internal state (and thus with a security level of at most 2^{64}), the smallest recommended version of [7] with $n = 160$ bits of internal state, and finally, a more conservative version with $n = 256$ bits of internal state. In all three cases, both systems S_{it} and S_{out} consist of n quadratic equations in n variables.

4. FPGA DESIGNS

This section presents several designs for implementing QUAD on an FPGA with a variety of sizes and throughputs. Our main focus is on designs with size as small as possible, but we discuss other cases which may still be of interest. Results were obtained using the development environment ISE version 8.1 together with the smallest Xilinx FPGA of the Virtex 4 familly, namely the XCLV25 which uses 90 nm CMOS processes. We used the number of slices to allow a fair comparison of the required area with other designs [13, 44, 38].

The rest of this section is organized as follows: we first outline basic principles shared by our designs in a first paragraph. We then present our lowest area design in a second paragraph, whereas the third paragraph exposes another implementation with a better throughput/area ratio. This allows to select one or the other depending on the targeted applications. Two of the most important usage scenarios are described in Section 5.

4.1 Preliminaries

The most obvious way of implementing QUAD's underlying PRNG is of course to hardwire the two systems of multivariate quadratic equations S_{it} and S_{out} in a straightforward manner. This however, requires a huge amount of space: the naïve technique for describing two randomly chosen systems of n quadratic equations in n unknowns needs $O(n^3)$ coefficients, which necessarily translates into a very large implementation. For instance, an internal state of 160 bits requires 4 Mbits to store the coefficients of the two systems S_{it} and S_{out}.

Our implementations of the PRNG are all based on the same simple idea in order to achieve a low area design. They make use of the very special fact that the two systems of multivariate quadratic equations describing QUAD are *publicly known*; the only secret information concealed to an attacker is the *internal state*. Hence, the only constraint on these systems is that they have to be randomly generated, which means drawn from a large class of systems without specific properties, or behave like randomly chosen systems of quadratic equations with respect to the system solving problem. This section shows how one can take advantage of the fact that these coefficients are publicly known and only have to be chosen in a random fashion. We note that the random generation of the coefficients is not required to be cryptographically secure in this case: coefficients are publicly known and the only point in generating the coefficients randomly is trying to reach one hard instance of the mathematical problem underlying QUAD. In fact, we show that regenerating the coefficients of the systems at each iteration of QUAD's PRNG is enough to transform the need for a huge memory into a tiny generation circuitry. We conjecture that the quality of the randomness used for generating the coefficients does not lower the security of the overall design. Indeed, we carefully checked–as recommended by the authors of QUAD [7]–that all the desirable properties of the set of polynomials obtained this way are present: we en-

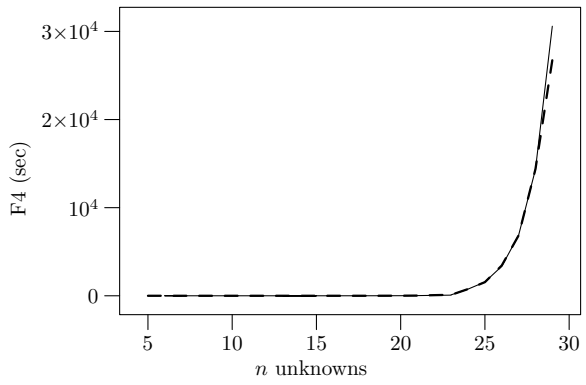

Figure 2: Time required to solve with algorithm F4 from Magma a randomly generated system of n quadratic equations in n variables and time required to solve with F4 a quadratic system of n equations in n variables generated via our proposal for a low area design (dashed curve)

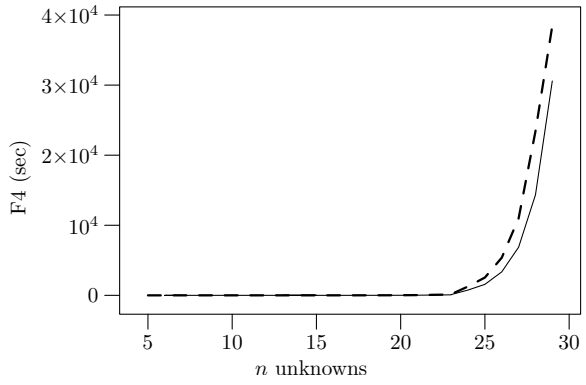

Figure 3: Time required to solve with F4 a randomly generated system of quadratic of n equations in n variables and time required to solve with F4 a quadratic systems of n equations in n variables generated via our proposal for a medium area design (dashed curve)

sured that the polynomials form a linearly independent set and that each of the quadratic polynomials has a high rank. Moreover, we fed the systems generated along our proposals to the F4 system solver [21] of Magma [2] to compare their behavior with respect to the one of quadratic systems generated with the use of a cryptographically secure PRNG (namely, the Blum-Blum-Shub generator [9]). The time required to solve a random instance and the time required to solve systems generated through our proposals match perfectly. This evidence, described by Fig. 2 and Fig. 3, suggests that our proposed system generation does not introduce additional weaknesses into QUAD.

4.2 Lowest Area Design

This paragraph describes the smallest of our designs, which results in a 68% improvement on the smallest known AES design on FPGA [13]. An obvious method for reducing the amount of logic necessary to implement our system of multivariate quadratic equations is to perform the computations

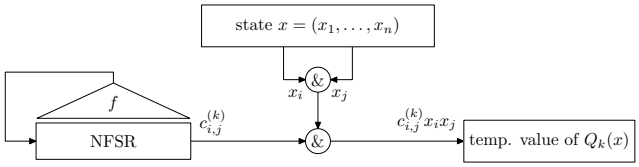

Figure 4: For this very low area design, two main components perform all the computations. The NFSR generates each coefficient of each polynomial in turn. The second component computes the value of the corresponding monomial at the same time. Their combination (a bit product) is accumulated in the temporary output value

sequentially. In our case this amounts to compute each polynomial of the system in turn, just as their plain mathematical expression:

$$Q_k(x_1, \ldots, x_n) = c_{0,0}^{(k)} + c_{1,1}^{(k)} x_1 x_1 + c_{1,2}^{(k)} x_1 x_2 + \cdots \\ \cdots + c_{n-1,n}^{(k)} x_{n-1} x_n + c_{n,n}^{(k)} x_n x_n. \quad (1)$$

To this end, our design makes use of two main components: the first one (a non linear feedback shift register, abbreviated NFSR) generates the coefficients of each monomial of each polynomial in turn and the other one computes the value of the monomials corresponding to the coefficient being generated. More precisely, a new monomial is computed at every clock tick and its contribution is accumulated to the temporary value for the output polynomial being computed. Then, the process starts again and a new polynomial is computed; once all polynomials are computed half of the values is used as keystream and the other half is used to update the internal state, just as shown on Fig. 1.

Our implementation of an iteration of QUAD's PRNG is thus made of two main components as summarized on Fig. 4. One component computes each of the $n(n+1)/2$ monomials in turn, while the other generates the $2n \cdot n(n+1)/2 = n^2(n+1)$ coefficients sequentially.

The first component computing the monomials is implemented as an addressable memory so as to lower its complexity. The second component is implemented as a non linear feedback shift register (NFSR) with a small internal state, so that it only requires a small number of slices but large enough to avoid any repetition in the sequence of the $n^2(n-1)$ coefficients; in practice an internal state of about 32 bits is sufficient and allows for an exhaustive check of the maximality of the NFSR's period. We reused one of Achterbahn's NFSR's [23] with a 31-bit internal state, a period of $2^{31} - 1$ and a feedback function defined as follows:

$$\begin{aligned} f(t_0, t_1, \ldots, t_{30}) &= t_0 + t_3 + t_5 + t_7 + t_{10} + t_{16} + t_{17} \\ &\quad + t_{18} + t_{19} + t_{20} + t_{21} + t_{24} + t_{30} \\ &\quad + t_5 t_{15} + t_{11} t_{18} + t_{16} t_{22} + t_{17} t_{21} \\ &\quad + t_1 t_2 t_{19} + t_1 t_{12} t_{14} t_{17} + t_2 t_5 t_{13} t_{20}. \end{aligned}$$

The monomials and the coefficients are then combined as shown on Fig. 4 to compute the value of each of the $2n$ polynomials in turn. We alternate between the computation of one polynomial of S_{out} (which is then output) and the com-

putation of one polynomial of S_{it} (which will be used to update the internal state) to ensure a constant output rate. Half of the time the value of the polynomial is output, half of the time it is accumulated into a buffer representing the value of S_{it}. When filled, this buffer is used to update the internal state of the cipher for the next iteration. Thus, assuming that $\big(S_{\text{it}}(x), S_{\text{out}}(x)\big) = \big(Q_1(x), \ldots, Q_{2n}(x)\big)$, the polynomials are computed in the following order: $Q_{n+1}(x)$ which is output, $Q_1(x)$ which is put into the buffer, $Q_{n+2}(x)$ which is output, $Q_2(x)$ which is put into the buffer, and so on until all the $2n$ polynomials are computed. At this time, the internal state is updated with the buffer's content.

Table 1: Low area implementation results

Version	**128 bits**	**160 bits**	**256 bits**
Flip/Flops	66	68	68
4 input LUTs	153	169	181
Slices	85	92	97
Gate Equiv. (GE)	2961	3694	4611
Max. Freq. (MHz)	267	244	243
Throughput (Kbps)	16.1	9.5	3.7

This design requires a minimal area since it takes about $2n$ memory bits plus the size of the NFSR to implement it. The drawback is that each iteration of the PRNG requires $O(n^3)$ steps, which results in a somewhat lower throughput compared to practical stream ciphers but a usual throughput when compared to provably secure ciphers. Figures about our implementation are given in Table 1.

4.3 Improving the Throughput/Area Ratio

The implementation proposed in the previous paragraph uses a non-linear feedback shift register (NFSR) to regenerate the coefficients of the polynomials at each iteration. This handles the $n^2(n+1)$ coefficients sequentially, which though allowing a highly compact design, has an obvious impact on the time needed to iterate the PRNG and consequently on the throughput. In this paragraph another implementation is described which offers a better trade-off between size and throughput. Instead of computing the contribution of each monomial to the value of each of the $2n$ polynomials in turn, which takes $O(n^3)$ time, we suggest to compute the contribution of each monomial to the value of $2n$ polynomials simultaneously. For this to happen, one needs to generate $2n$ coefficients at each clock. We show how the use of a finite state machine (FSM) allows for parallel generation of the coefficients while maintaining a highly compact design, thus cutting the overall computation time for every iteration by a factor of $2n$.

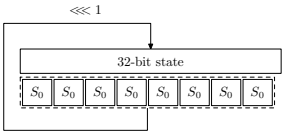

Figure 5: The finite state machine and its update function.

The FSM must have a small internal state as well as a lightweight update function in order to minimize the area. To reach this goal, we chose an internal state of 32 bits together with an update function built around 4-bit non-linear S-boxes. Such S-boxes are convenient since it then requires at most one LUT in the FPGA implementation to compute one bit of the next state. There are examples of 4-bit S-boxes in the literature, see for instance Serpent's S-boxes [8]. The internal state of 32-bits is well suited for our three special cases of interest, namely 128 bits, 160 bits, and 256 bits. As described by Fig. 5, the update function of our FSM consists of the following steps:

- Each 4-bit word of the internal state $\alpha = (\alpha_1, \ldots, \alpha_8)$ of the FSM is mapped through a 4-bit S-box, i.e. the state is updated via: $\alpha = \big(S_0(\alpha_1), \ldots, S_0(\alpha_8)\big)$;
- The state α is rotated by one bit to the left.

Since the FSM has an internal state of 32 bits, there is a need for an expansion from 32 bits to n bits to produce the coefficients for the parallel computation of the value of n polynomials simultaneously. We propose a set of distinct S-boxes to provide this expansion as sketched in Fig. 6. The three cases must be specified separately since they do not require the same expansion factor.

In the case of $n = 128$, we chose an expansion factor of four. Four distinct S-boxes of Serpent S_0, S_1, S_2, and S_3, are used to derive the 128 required coefficients from the internal state α of the FSM as

$$\big(S_0(\alpha_1), \ldots, S(\alpha_8), S_1(\beta_1), \ldots, S_1(\beta_8), \\ S_2(\gamma_1), \ldots, S_2(\gamma_8), S_3(\delta_1), \ldots, S_3(\delta_8)\big),$$

where $\beta = (\alpha \lll 1)$, $\gamma = (\alpha \lll 2)$, and $\delta = (\alpha \lll 3)$.

Since one iteration of the PRNG requires computing $2n$ polynomials, we duplicate the circuitry shown on Fig. 6 to compute both S_{it} and S_{out} at the same time. In the case $n = 256$, four FSMs together with their expansion circuitry have been used. For the case $n = 160$ we chose to use three FSMs with their associated expansion circuitry: one FSM produces four 32-bit words at each clock, while the two others produce only three 32-bit words to obtain the desired 320 bits of coefficients at each clock.

Table 2: QUAD's PRNG implementation with an improved throughput/area ratio

Version	**128 bits**	**160 bits**	**256 bits**
Flip/Flops	350	418	613
4 input LUTs	781	970	1471
Slices	406	509	763
GE	8117	10184	14959
Max. Freq. (MHz)	262	269	260
Throughput (Mbps)	4.1	3.3	2.0

This implementation requires a smaller surface since it takes about $3n$ memory bits plus the size of the FSMs to implement. But it now takes $O(n^2)$ steps to perform an iteration of the PRNG, hence increasing the throughput. Figures for this implementation are given in Table 2. One notices that the throughput decreases with the size of the internal state since it depends on the parameter n just as the functional $1/n$.

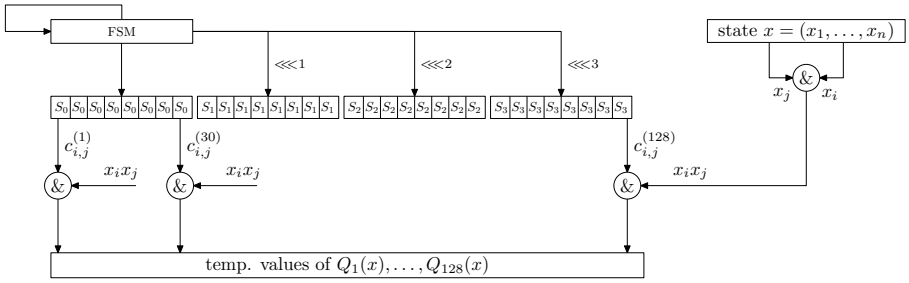

Figure 6: Parallel computation of 128 polynomials. The FSM has an internal state of 32 bits which is expanded through a set of 4-bit S-boxes to simultaneously produce 128 coefficients. All of these coefficients are multiplied by the value of the current monomial $x_i x_j$.

5. CHOSING THE RIGHT DESIGN

We have just presented small area versions of QUAD's underlying PRNG. The characteristics of these small area designs are summarized in Table 3. The smallest version is directly targeted at tiny devices such as highly limited RFIDs. The medium version aims to provide an efficient pseudo-random generator, and we show that its provable security feature do not prevent it to perform as well as most of today's stream ciphers. In the following paragraphs, we review some state of the art solutions in each of these domains and explain how QUAD relates to each of them.

Table 3: Various implementation trade-offs

Implementation	**clocks/it.**	**memory**
QUAD low (sec. 3.2)	$n^2(n+1)$	$\sim \quad n$
QUAD medium (sec. 3.3)	$n(n+1)/2$	$\sim 4n$

5.1 QUAD as an efficient Stream Cipher

Stream ciphers and PRNGs with small footprint have several distinct applications but they are of special interest when power consumption is a bigger constraint than the throughput. This is often the case for instance for mobile applications. This paragraph compare our proposals for the implementation of QUAD with several natural competitors.

Concerning the provable security feature brought by QUAD's design, some of the obvious competitors are: the Blum-Blum-Schub generator, the optimized implementation of the discrete logarithm generator exposed in [30], and the recently proposed generator based on the small solution RSA (SSRSA) assumption [41]. The SSRSA based generator outperforms the discrete logarithm generator, which already outperforms the Blum-Blum-Schub generator due to its ability to output more bits for fewer modular exponentiations. All of them require an internal state of at least 1024 bits (and much more in practice for a good security level) which implies that an FPGA implementation is likely to require more than 10.000 GE. Since we are not aware of any FPGA implementation for these PRNGs, we only refer to the implementation in IBM crypto-processor at 447 MHz with 4 MB of RAM presented in [30], and which achieves a 182 Kbps throughput. Comparisons between such a specific device and FPGAs is difficult, especially when the interest is focused on low area designs.

In order to compare QUAD with other stream ciphers, we need to discuss the implementation of the key and IV setup of QUAD. Let us here recall how it works: the internal state is first set by replicating the key enough time to fill it. Then, the initialization process goes through the following loop: for every bit b_i of the IV in turn, if bit b_i is equal to 1 the internal state is updated with the n-bit output value from S_{it}, and if the bit $b_i$is equal to 0, the internal state is updated with the n-bit output value from S_{out}. After all the k bits of the IV have been processed, the state is updated k times by iterating S_{it} and without producing any keystream.

Since our smallest implementation of QUAD's PRNG is computing one polynomial at a time, we need only one selector in order to either throw the result of the computation or store it in the buffer. This buffer will then be used to update the internal state, depending on the value of the current bit of the IV. The second implementation of QUAD's PRNG is computing in parallel the $2n$ polynomials, and we then need $2n$ extra selectors. In both cases, the number of extra gates required to implement the key and IV setup is small with regard to the total gate count required to implement the PRNG. These implementations of the key and IV setup are very compact but rather slow. Faster implementations can obviously be designed when area is not the critical constraint.

Table 4: Comparing our results with several other FPGA implementations [28, 13, 27]. The clocking frequencies for QUAD are the highest supported by our development board

	Freq. (MHz)	Area (slice)	Thru. (Mbps)	Thru./A. (Kbps/sl)
Trivium [28]	102	40	102	2550
Grain [28]	105	48	105	2187
QUAD low	267	85	0.016	0.2
Hermes8 [28]	45	190	5.6	29.5
Phelix [28]	30	264	3.26	12.3
AES [27]	67	264	2.2	8.3
Sfinks [28]	37	334	7.4	22.1
QUAD med.	262	406	4.1	10.1
AES [13]	60	522	69	132

We now compare our implementations of QUAD with currently available FPGA implementations of state of the art

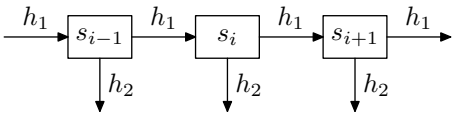

Figure 7: Ohkubo *et al.* [39] hash chain based scheme.

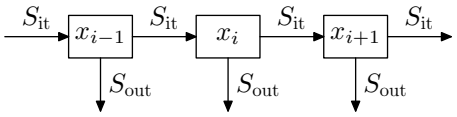

Figure 8: Using QUAD along the lines of [39].

streams ciphers, namely some of the best designs from the eSTREAM project [19]–those are hardware oriented candidates of this project: Grain [29], Trivium [12], Phelix [43], Sfinks [11], and Hermes8 [35]. The figures for these FPGA implementations come from the paper [28]. Of course, the AES may also be used–for instance in counter mode–to produce pseudo-random sequences, and is often the preferred choice to implement a PRNG. The AES implementation on FPGA which seems to best fit our search for compact implementations is the one presented in [13] and [27], and is included in our comparisons.

Table 4 is sorted by area and shows that our implementation of QUAD is competitive with both AES implementations and other stream ciphers.

5.2 QUAD for RFIDs

Several solutions have been proposed to deal with security in RFIDs, sometimes addressing to some degree the privacy concerns. For a state of the art survey of these techniques, the reader should refer to [34]. There are basically three different ways to deal with privacy concerns. A first solution is to provide a mechanism to "Kill" an RFID so as destroy it. This is a somewhat extreme solution since it prohibits the tag and its carrier from taking benefits of any associated services. There is an intermediate solution where the tag is temporarily turned off, and might be turn on whenever a highly occasional service is required, such as after sale support. A second solution is to somehow change the identifier returned whenever the tag is queried. There have been numerous proposals along those lines, such as re-encryption mechanisms, or re-labeling mechanisms to name a few. A third solution is as follows: as there is no way to prevent cheap RFID tags from being tampered with, it is desirable to ensure that, an attacker having opened an RFID tag and accessed its content, there is no way for him to correlate previous identifiers emitted by the tag with this content. This property is called forward secrecy and not only prevents attackers from tracking an RFID tag, but also prevents attackers from obtaining information about this RFID's life, that is about any of the past identifications. It will not prevent, however, an attacker having access to the tag to use it for himself.

This third solution, originally proposed in [39] by M. Ohkubo, K. Suzuki, and S. Kinoshita, offers the most interesting kind of privacy to customers. (In fact, as explained in [5] where a formal adversarial model is described, the scheme [39] is the only one to meet the requirements.) The scheme proposed by [39] uses an initial identifier id as well as two distinct hash functions, h_1 and h_2. The internal state s is set to id upon initialization. Now at each query, the tag outputs $h_2(s)$ and updates its internal state via $s \leftarrow h_1(s)$.

Obviously, this is not an easy task to embed two different hash function designs into an RFID. Moreover, hash function design is a difficult problem as shown by the recent advances in hash function cryptanalysis. On the other hand, the property required by [39] is actually that h_1 and h_2 are one-way functions and that the sequence $h_2(s_i)$ is a pseudo-random sequence: it must be computationally intractable to correlate its outputs, and computationally intractable to run backward.

QUAD perfectly fits these requirements and, as such, is a very good candidate to build forward-secrecy enabled RFIDs. As a cryptographically secure stream cipher, it provides unlinkability of outputs deterring any attacker without physical access to the tag from tracing the RFID's holder. At the same time, the one-wayness of the quadratic system S_{it} used to update QUAD's internal state ensure the forward secrecy: an attacker obtaining access to QUAD's internal state is not able to link this information with any identifier previously issued by the tag. Hence the quadratic system S_{it} used to update QUAD's internal state plays the role of hash function h_1 in Ohkubo, Suzuki, and Kinoshita scheme, while S_{out} plays the role of hash function h_2.

Finally, RFIDs are sometimes more concerned by the area of the design than by the throughput of the underlying pseudo-random generator. In this specific application indeed, one often only has to generate a short identifier (usually about 160 bits) at each authentication.

6. CONCLUSION

In this paper we discussed several implementations of QUAD. Our smallest proposal requires 85 slices (2961 GE), which renders QUAD's pseudo-random generator the smallest one in the field of provable security and a very good competitor to other stream ciphers. Its additional properties makes it especially attractive in the context of RFIDs. Depending on the application, it can be useful to consider our second design, for which the throughput/area ratio is much improved. As we have shown, QUAD accommodates a wide range of trade-offs and its provable security makes it a very good candidate for tailored hardware implementation.

Acknowledgements

The third author wishes to thank Daniel Billet for fruitful discussions, and Matt Robshaw for his careful advices on this work.

7. REFERENCES

[1] Boycott Gillette. `http://www.boycottgillette.com`.

[2] MAGMA: High performance software for Algebra, Number Theory, and Geometry. `http://magma.maths.usyd.edu.au/magma/`.

[3] Akkar, M.-L., Courtois, N. T., Goubin, L., and Duteuil, R. A Fast and Secure Implementation of

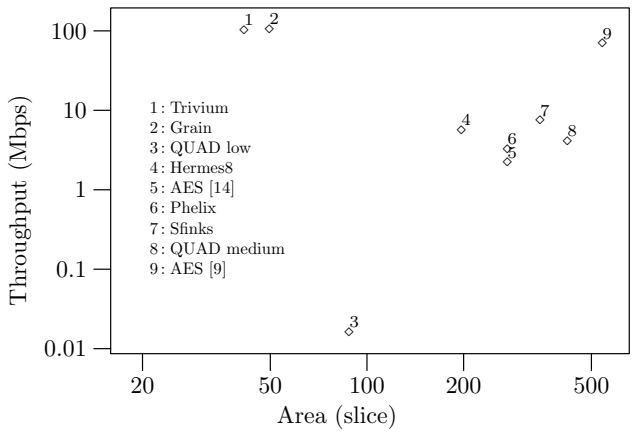

Figure 9: Throughput versus area for FPGA designs of various stream ciphers.

Sflash. In *Public Key Cryptography – PKC 2003* (2003), Y. G. Desmedt, Ed., vol. 2567 of *Lecture Notes in Computer Science*, Springer-Verlag, pp. 267–278.

[4] Alexi, W., Chor, B., Goldreich, O., and Schnorr, C.-P. RSA and Rabin Functions: Certain Parts are as Hard as the Whole. *SIAM J. Comput. 17*, 2 (1988), 194–209.

[5] Avoine, G. Adversarial Model for Radio Frequency Identification. Cryptology ePrint Archive, Report 2005/049, `http://eprint.iacr.org/`, 2005.

[6] Bardet, M. *Étude des systèmes algébriques surdéterminés. Applications aux codes correcteurs et à la cryptographie*. PhD thesis, Université Paris VI, 2004.

[7] Berbain, C., Gilbert, H., and Patarin, J. QUAD: A Practical Stream Cipher with Provable Security. In *Advances in Cryptology – EUROCRYPT 2006* (2006), S. Vaudenay, Ed., Lecture Notes in Computer Science, Springer-Verlag.

[8] Biham, E., Anderson, R. J., and Knudsen, L. R. Serpent: A New Block Cipher Proposal. In *Fast Software Encrytion – FSE '98* (1998), S. Vaudenay, Ed., vol. 1372 of *Lecture Notes in Computer Science*, Springer-Verlag, pp. 222–238.

[9] Blum, L., Blum, M., and Shub, M. A Simple Unpredictable Pseudo-Random Number Generator. *SIAM J. Comput. 15*, 2 (1986), 364–383.

[10] Blum, M., and Micali, S. How to Generate Cryptographically Strong Sequences of Pseudo-Random Bits. *SIAM J. Comput. 13*, 4 (1984), 850–864.

[11] Braeken, A., Lano, J., Mentens, N., Preneel, B., and Verbauwhede, I. SFINKS : A Synchronous Stream Cipher for Restricted Hardware Environments. eSTREAM, ECRYPT Stream Cipher Project, Report 2005/001, 2005. Available at `http://www.ecrypt.eu.org/stream`.

[12] Cannière, C. D., and Preneel, B. Trivium: Specifications. eSTREAM, ECRYPT Stream Cipher Project, Report 2005/001, 2005. Available at `http://www.ecrypt.eu.org/stream`.

[13] Chodowiec, P., and Gaj, K. Very Compact FPGA Implementation of the AES Algorithm. In *Cryptographic Hardware and Embedded Systems – CHES 2003* (2003), C. D. Walter, Çetin Kaya Koç, and C. Paar, Eds., vol. 2779 of *Lecture Notes in Computer Science*, Springer, pp. 319–333.

[14] Courtois, N., Goubin, L., Meier, W., and Tacier, J.-D. Solving Underdefined Systems of Multivariate Quadratic Equations. In *Public Key Cryptography – PKC 2002* (2002), pp. 211–227.

[15] Courtois, N., Klimov, A., Patarin, J., and Shamir, A. Efficient Algorithms for Solving Overdefined Systems of Multivariate Polynomial Equations. In *Advances in Cryptology – EUROCRYPT 2000* (2000), B. Preneel, Ed., vol. 1807 of *Lecture Notes in Computer Science*, Springer-Verlag, pp. 392–407.

[16] Courtois, N., and Patarin, J. About the XL Algorithm over $GF(2)$. In *Topics in Cryptology – CT-RSA 2003* (2003), M. Joye, Ed., vol. 2612 of *Lecture Notes in Computer Science*, Springer-Verlag, pp. 141–157.

[17] Diem, C. The XL-Algorithm and a Conjecture from Commutative Algebra. In *Advances in Cryptology – ASIACRYPT 2004* (2004), P. J. Lee, Ed., vol. 3329 of *Lecture Notes in Computer Science*, Springer-Verlag, pp. 323–337.

[18] ECRYPT. Web page of the eSTREAM project: `http://www.ecrypt.eu.org/stream/`.

[19] ECRYPT. eSTREAM: ECRYPT Stream Cipher Project, IST-2002-507932. Available at `http://www.ecrypt.eu.org/stream/`. Accessed September 29, 2005.

[20] et al., A. RFID Position Statement of Consumer Privacy and Civil Liberties Organizations. `http://www.privacyrights.org/ar/RFIDposition.htm`.

[21] Faugère, J.-C. A New Efficient Algorithm for Computing Gröbner Bases (F4). *Journal of Pure and Applied Algebra 139* (1999), 61–88.

[22] Fraenkel, A. S., and Yesha, Y. Complexity of Solving Algebraic Equations. *Inf. Process. Lett. 10*, 4/5 (1980), 178–179.

[23] Gammel, B. M., Göttfert, R., and Kniffler, O. The Achterbahn Stream Cipher. In *Symmetric Key Encryption Workshop – SKEW 2005*. ECRYPT Stream Cipher Project Report 2005/002.

[24] Garey, M. R., and Johnson, D. S. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W.H. Freeman & Co, 1979, ch. 7.2 Algebraic Equations over $GF(2)$.

[25] Garey, M. R., and Johnson, D. S. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W.H. Freeman & Co, 1979.

[26] Gennaro, R. An Improved Pseudo-random Generator Based on Discrete Log. In *Advances in Cryptology – CRYPTO 2000* (2000), M. Bellare, Ed., vol. 1880 of *Lecture Notes in Computer Science*, Springer-Verlag, pp. 469–481.

[27] Good, T., and Benaissa, M. Aes on fpga from the fastest to the smallest. In *Cryptographic Hardware and Embedded Systems – CHES 2005* (2005), J. R. Rao and B. Sunar, Eds., vol. 3659 of *Lecture Notes in*

Computer Science, Springer, pp. 427–440.

[28] Good, T., Chelton, W., and Benaissa, M. Review of Stream Cipher Candidates from a Low Resource Hardware Perspective. Stream Ciphers Revisited – SASC 2006, Workshop record, 2006.

[29] Hell, M., Johansson, T., and Meier, W. Grain – A Stream Cipher for Constrained Environments. eSTREAM, ECRYPT Stream Cipher Project, Report 2005/001, 2005. Available at `http://www.ecrypt.eu.org/stream`.

[30] Howgrave-Graham, N., Dyer, J. G., and Gennaro, R. Pseudo-random Number Generation on the IBM 4758 Secure Crypto Coprocessor. In *Cryptographic Hardware and Embedded Systems – CHES 2001* (2001), Çetin Kaya Koç, D. Naccache, and C. Paar, Eds., vol. 2162 of *Lecture Notes in Computer Science*, Springer, pp. 93–102.

[31] Imai, H., and Matsumoto, T. Algebraic Methods for Constructing Asymmetric Cryptosystems. In *Applied Algebra, Algebraic Algorithms and Error-Correcting Codes – AAECC 3* (1985), J. Calmet, Ed., vol. 229 of *Lecture Notes in Computer Science*, Springer-Verlag, pp. 108–119.

[32] Impagliazzo, R., and Naor, M. Efficient Cryptographic Schemes Provably as Secure as Subset Sum. *Journal of Cryptology 9*, 4 (1996), 199–216.

[33] Jean-Charles Faugère and Hideki Imai and Mitsuru Kawazoe and Makoto Sugita and Gwénolé Ars. Comparison Between XL and Gröbner Basis Algorithms. In *Advances in Cryptology – ASIACRYPT 2004* (2004), P. J. Lee, Ed., vol. 3329 of *Lecture Notes in Computer Science*, Springer-Verlag, pp. 338–353.

[34] Juels, A. RFID security and privacy: a research survey. *IEEE Journal on Selected Areas in Communications 24* (2006), 381–394.

[35] Kaiser, U. Hermes8. eSTREAM, ECRYPT Stream Cipher Project, Report 2005/001, 2005. Available at `http://www.ecrypt.eu.org/stream`.

[36] Katherine Albrecht, L. M. *How Major Corporations and Government Plan to Track your Every Move with RFID*. 2006.

[37] Lidl, R., and Niederreiter, H. *Finite Fields*. Cambridge University Press, 1997.

[38] Morioka, S., and Satoh, A. An Optimized S-Box Circuit Architecture for Low Power AES Design. In *Cryptographic Hardware and Embedded Systems – CHES 2002* (2002), B. S. J. Kaliski, Ç. K. Koç, and C. Paar, Eds., vol. 2523 of *Lecture Notes in Computer Science*, Springer-Verlag, pp. 172–186.

[39] Ohkubo, M., Suzuki, K., and Kinoshita, S. Cryptographic Approach to "Privacy-Friendly" Tags. In *RFID Privacy Workshop* (MIT, MA, USA, November 2003).

[40] Patarin, J., and Goubin, L. Asymmetric Cryptography with S-Boxes. In *ICICS* (1997), pp. 369–380.

[41] Steinfeld, R., Pieprzyk, J., and Wang, H. On the Provable Security of an Efficient RSA-Based Pseudorandom Generator. In *Advances in Cryptology – ASIACRYPT 2006* (2006), X. Lai, Ed., Lecture Notes in Computer Science, Springer-Verlag.

[42] Stern, J., and Fischer, J.-B. An Efficient Pseudo-Random Generator Provably as Secure as Syndrome Decoding. In *Advances in Cryptology – EUROCRYPT '96* (1996), U. M. Maurer, Ed., vol. 1070 of *Lecture Notes in Computer Science*, Springer-Verlag, pp. 245–255.

[43] Whiting, D., Schneier, B., Lucks, S., and Muller, F. Phelix Fast Encryption and Authentication in a Single Cryptographic Primitive. eSTREAM, ECRYPT Stream Cipher Project, Report 2005/001, 2005. Available at `http://www.ecrypt.eu.org/stream`.

[44] Wolkerstorfer, J., Dominikus, S., and Feldhofer, M. Strong Authentication for RFID Systems Using the AES Algorithm. In *Cryptographic Hardware and Embedded Systems – CHES 2004* (2004), M. Joye and J.-J. Quisquater, Eds., vol. 3156 of *Lecture Notes in Computer Science*, Springer-Verlag, p. 357.