# The GLUON family: a lightweight Hash function family based on FCSRs

**Abstract.** Since the beginning of the SHA3 competition, the cryptographic community has seen the emergence of a new kind of primitives: the lightweight cryptographic hash functions. At the time writing this article, two representatives of this category have been published: QUARK [7] and PHOTON [18] designed to match RFID constraints.
In this paper, we propose a third representative of this category which is called GLUON. It is based on the sponge construction model [11] as QUARK and PHOTON and inspired by two stream ciphers F-FCSR-v3 [4] and X-FCSR-v2 [10]. From the generic definition of our lightweight hash function, we derive three different instances according the required security level that must be reached.
For example, our lightest instance (GLUON-128/8) dedicated to 64-bit security level fits in 2071 gate-equivalents which stays competitive when compared with the parallel implementation of U-QUARK. The software performances are good for GLUON-224/32, our heaviest instance.
**Keywords:** lightweight hash function, FCSRs, sponge functions.

## Introduction

The last five years have seen the emergence of new challenging tasks that consist in designing lightweight primitives dedicated to very constrained environments such as sensors or RFID tags. Among those proposals, many block ciphers such as PRESENT [13], HIGH [20], mCrypton [23] or KATAN & KTANTAN [15] have been specially designed to fit with a very compact hardware implementation.

The same kind of works concerning lightweight hash functions has just been initiated with two existing standalone proposals: QUARK [7] and PHOTON [18] both based on sponge constructions [11]. The need for such proposals comes first from the embedded system community (RFID and sensor) and second from the lack of lightweight SHA3 finalists [26]. Indeed, all the SHA3 finalists requires more than 12000 GE for a 128-bit security level. Some previous proposals such as SQUASH [29] or ARMADILLO [8] have been done but show their respective weaknesses [28, 1]. The first step in the design of lightweight hash function dates from the use of the block cipher PRESENT in the Davies-Meyer mode of operation. However, and as already noticed in [7, 18], sponge constructions allow a better ratio between internal state size and security level, better that for traditional modes of operations even if hashing small messages is not really efficient due to the squeezing step.

Following the recent proposals for lightweight hash functions taking their name from small particles[1], we propose a new lightweight hash function family

[1] see: http://www.131002.net/data/talks/quarks_rump.pdf

called GLUON based on a sponge construction. This new family is based on a particular Feedback with Carry Shift Register (FCSR), elementary building block well studied during the two last decades. Even if the hardware size of such a primitive is a little bit heavier than the basic building blocks used in QUARK and PHOTON, we think that the well-known design principles of FCSRs could be considered as a strength of our proposed design. Moreover, the software performances of our design are also good.

This paper is organized as follows: in Section 1, we recall the definitions of sponge constructions and of word ring FCSRs. In Section 2, we describe the underlying function that composes the sponge construction based on a word ring FCSR. In Section 3, we give some insights about our chosen design whereas in Section 4 we sum up all the security observations we made concerning the $f$ function. In Section 5, we provide performance results for hardware and software implementations. Finally, Section 6 concludes this paper.

# 1 Background

## 1.1 Sponge constructions

Sponge constructions have been proposed by Bertoni et al. in [11] as a new way of building hash functions from a fixed function or a fixed permutation. A sponge construction as defined in Fig. 1 has a *rate* $r$ which corresponds with the block length, a *capacity* $cp$ and an *output length* $N$. The *width* of its internal state $b$ is defined as $r + cp$ that must be greater than $N$. Of course, we could set $N = cp$ for non null $r$.

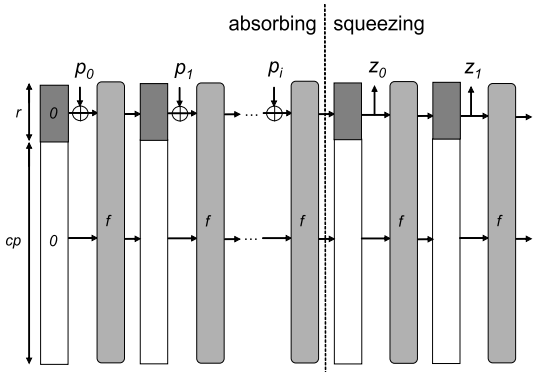


**Fig. 1.** The sponge construction.

Given an initial state, the sponge construction processes a message $M$ of length in words of size $r$ bits $|M|$ as follows:

1. **Initialization:** The message is padded by appending a '1' bit and sufficiently many zeros to reach a length multiple of $r$.
2. **Absorbing phase:** The $r$-bit message blocks are xored into $r$ bits of the state interleaved with applications of the function $f$.
3. **Squeezing phase:** Some $r$ bits of the state are returned as output, interleaved with applications of the function $f$, until $N$ bits are returned.

### 1.2 FCSR automata in word ring representation

Roughly speaking, a FCSR as defined in [17, 22] consists of a binary main register and of a carry register but contrary to LFSRs the performed operations are no more xors over $\mathbb{F}_2$ but additions with carry in the set of 2-adic integers $\mathbb{Z}_2$ (i.e. the set of power series: $\sum_{i=0}^{\infty} s_i 2^i$, $s_i \in \{0,1\}$). Each cell of the main register produces a sequence $S = (s_n)_{n\in\mathbb{N}}$ that is eventually periodic if and only if there exist two numbers $p$ and $q$ in $\mathbb{Z}$, $q$ odd, such that $s = \sum_{i=0}^{\infty} s_i 2^i = p/q$. This sequence is strictly periodic if and only if $pq \leq 0$ and $|p| \leq |q|$. The period of $S$ is the order of 2 modulo $q$, i.e., the smallest integer $P$ such that $2^P \equiv 1 \pmod{q}$. The period satisfies $P \leq |q| - 1$. If $q$ is prime and if $P = |q| - 1$, the sequence $S$ is called an $\ell$-sequence. $\ell$-sequences have many proved properties that could be compared to the ones of $m$-sequences: known period, good statistical properties, fast generation, etc.

F. Arnault et al. have studied in [4] and in [10] efficient hardware and software FCSRs using matrix representations. They give the following definition:

**Definition 1.** *A (diversified or ring) FCSR is an automaton composed of a main shift register of $n$ binary cells $m = (m_0, \ldots, m_{n-1})$, and a carry register of $n$ integer cells $c = (c_0, \ldots, c_{n-1})$. It is updated using the following relations:*

$$\begin{cases} m(t+1) = Tm(t) + c(t) & \bmod 2 \\ c(t+1) = Tm(t) + c(t) & \div 2 \end{cases} \tag{1}$$

*where $T$ is a $n \times n$ matrix with coefficients 0 or 1 in $\mathbb{Z}$, called transition matrix.*

*Note that $\div 2$ is the traditional expression: $X \div 2 = \frac{X - (X \bmod 2)}{2}$.*

They also prove the following property:

**Theorem 1 ([4] Theorem 1).** *The series $\boldsymbol{M}_i(t)$ observed in the cells of the main register are 2-adic expansion of $p_i/q$ with $p_i \in \mathbb{Z}$ and with $q = \det(I - 2T)$.*

The $T$ transition matrix completely defines the ring FCSR as shown in Theorem 1. Moreover and as shown in [4], ring FCSRs have better hardware implementations than classical Galois or Fibonacci FCSRs. Moreover, the diffusion speed (which is faster than in Galois/Fibonacci FCSRs) is related to the diameter $d$ of the transition graph. This diameter is the maximal distance between two cells of the main register. In other words, $d$ is the distance after which all the cells of the main register have been influenced by at least one other cell through the feedbacks. It corresponds to the minimal number of clocks required to have all the cells of the main register influenced by at least one other cell. $d$ should

be small for better diffusion. Thus, in the case of a ring FCSR, the diffusion of differences is also improved because these diffusions could be also computed as the diameter $d$ of the graph associated to the transition matrix $T$. Typically this value is close to $n/4$ for a ring FCSR instead of $n$ in the Galois or Fibonacci cases.

In [10], Berger et al. describe word ring FCSRs that are efficient both in hardware and in software. Those FCSRs are completely determined by the choice of the matrix $T$. Contrary to ring FCSRs, word ring FCSRs act on words of size $r$ bits depending on the targeted architecture. Classically, $r$ could be equal to 8, 16, 32 or 64 bits. Then, in the generic description of a word ring FCSR, the associated matrix $T$ is defined on $r$-bit words. The main register of this kind of FCSR could thus be represented as $w$ $r$-bit words $m_0, \cdots, m_{w-1}$ with feedback words $c_0, \cdots, c_{w-1}$. The deduced $T$ matrix is of size $w \times w$. For example, the following $T$ matrix represents a word ring FCSR acting on $r = 8$ bits words:

$$T = \begin{pmatrix} 0 & I & 0 & SL^3 & 0 \\ 0 & 0 & I & 0 & SR^2 \\ 0 & 0 & SL^1 & I & 0 \\ SR^3 & 0 & 0 & 0 & I \\ I & 0 & 0 & 0 & 0 \end{pmatrix}$$

where $I$ is the $r \times r$ identity matrix, the $SL^a$ operation is the left shift at 8-bit level by $a$ bits, $SR^b$ is the right shift operation at 8-bit level by $b$ bits (two other operations could also be used: $RL^d$, the rotation on the left by $d$ bits and $RR^e$, the rotation on the right by $e$ bits).

This matrix defines the associated word ring FCSR described in Figure 2 with $n = 40$ and $r = 8$.

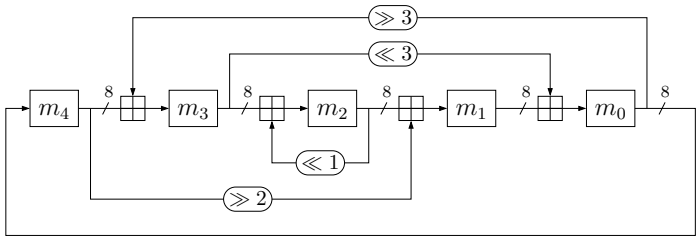


**Fig. 2.** A FCSR with efficient software design. The pluses in the boxes represent eight parallel binary adders with carry.

The corresponding $q$ value could be directly computed using the formula given in Theorem 1 and is equal to $-1497813390989$. This number is prime and has a maximal order. Thus the corresponding FCSR produces $\ell$-sequences, and is not only efficient in software due to the word oriented structure, it is also efficient in hardware (here only 24 binary additions are required) because the

intrinsic word ring nature of this FCSR representation limits the number of required gates. Moreover, the diameter of the associated graph is equal for this example to 13 ($d = 13$) which means that a complete diffusion of all the cells is achieved after $d + 4 = 17$ clocks.

## 2 Description of the GLUON Hash family

The GLUON family is based on a sponge construction where the $f$ function calls a filtered FCSR. The filtered FCSR is directly inspired from the F-FCSR-v3 hardware stream cipher [4] and by the X-FCSR-v2 software stream cipher [10]. All the proposed instances of the GLUON family varies according the version but all versions are based on "ring-word FCSRs" to be efficient not only in hardware but also in software. The general structure of the GLUON family is described on Fig. 3. The elementary building blocks are the following ones:

- The content of the word ring FCSR of size $w$ words of $r$ bits is denoted $m(t) = (m_0(t), \cdots, m_{w-1}(t))$ for the main register where $w$ is the length in words of the considered FCSR and $c(t) = (c_0(t), \cdots, c_{w-1}(t))$ for the carry register with $a$ active memories (i.e. $a$ internal feedbacks). The FCSR is defined by its associated matrix $T$ as seen in the previous section.
- A filter $FI$ is also defined to filter the content of the main register $m(t)$. It is xor-linear to break the 2-adic structure of the automaton. As done in [4], it consists in xoring together some shifted version of the words of the main register that have active carries. More precisely, let $\mathcal{F} = \{m_{f_0}, \cdots, m_{f_{\ell-1}}\}$ be the set of all the words $m_i$ that have a feedback. Then, $r$ bits of output are: $\forall\ 0 \leq i < r,\ FI = \bigoplus_{j \equiv i \mod r} (m_{f_j})_{<<<k}$ with $k$ a value in the set $[-r/2, +r/2]$. This linear filter is only used at the end of the computation of the function to extract the output from the state of the FCSR.

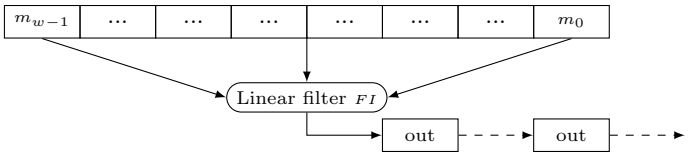


**Fig. 3.** General view of the $f$ function of the GLUON family.

The size of the input/output of $f$ is $b = (w - 1) \times r$ (instead of $w \times r$ as expected because a particular word (the word $w - 1$) of the main register is not used as input of the function).

### 2.1 Details of the $f$ function

$f$ processes a $b$-bit input in three steps:

**Initialization:** For an input $s = (s_0, \cdots, s_{b-1})$, $f$ initializes its internal state as follows:
- The $w-1$ first words of $m$ are initialized with the $(w-1) \times r$ input bits.
- The last word of $m$ is initialized with the all-one string of length $r$.
- The carry register $c$ is initialized with the all-zero string.

**State Update:** From the internal state $(m(t), c(t))$, the FCSR is clocked $d+4$ times using its internal transition function:

$$m(t+1) = Tm(t) + c(t) \mod 2$$
$$c(t+1) = Tm(t) + c(t) \div 2$$

**Computation of the output:** The FCSR is then clocked $w-1$ times. At each iteration, a $r$-bit word is extracted with the linear filter $F$ in order to obtain the $(w-1) \times r$ bits of output.

### 2.2 The sponge construction deduced from the $f$ function

The rate of the associated sponge construction is $r$, its capacity is $cp = (w-2) \times r$. The size of $f$ is $b = r + cp$. Let us now analyze in few words how this $f$ function fits in the sponge model. The external part of the $f$ function is composed of the first $r$-bit word of the input whereas the internal part consists of the following $(w-2)$ $r$-bit words of the input. In other words, during the absorbing steps, at each call of $f$, a message word $p_i$ of length $r$ bits is xored with the first $r$-bit word of the FCSR main register. Then, this first $r$-bit word is output when the squeezing step begins.

### 2.3 Proposed instances

As done for Quark and `PHOTON`, we propose 3 different instances of our GLUON family functions that we will denote GLUON-$N/r$ (the same that for Quark). From the sizes parameters, we use the same algorithm than the one proposed in [10] to compute the different $T$ matrices: we randomly pick matrices $T$ such as $\log_2(q) \geq n$, $q = \det(I - 2T)$ is prime, the order of 2 modulo $q$ is equal to $|q| - 1$ and is maximal to ensure that the outputs are composed of $\ell$-sequences. Moreover, $q$ has been chosen such that the diameter $d$ is sufficiently small to ensure a quick diffusion and with a number of carries close to $wr/2$. The filters are randomly chosen but guaranteeing that at least half of the picked words have a carry.

- *64-bit security level:* $r = 8$, $cp = 128$, $b = 136$, $N = 128$ leading to a FCSR composed of 18 8-bit words and with about 70 carries. The complete description of the Matrix $T$ and of the filter $FI$ is given in Appendix A.
- *80-bit security level:* $r = 16$, $cp = 160$, $b = 176$, $N = 160$ leading to a FCSR composed of 12 16-bit words and with about 90 carries. The complete description of the Matrix $T$ and of the filter $FI$ is given in Appendix B.

– *112-bit security level:* $r = 32$, $cp = 224$, $b = 256$, $N = 224$ leading to a FCSR composed of 9 32-bit words and with about 130 carries. The complete description of the Matrix $T$ and of the filter $FI$ is given in Appendix C.

As done for PHOTON, we could also add some other instances with as possible output sizes 80 and 256. In those cases, the corresponding $r$ values are respectively 8 and 32.

## 3 Design Rationale

### 3.1 Flat Sponge Claim

Sponge construction is a recent model for iterated hash functions and random number generation developed in [11] and in [12]. The general security claim is done in the indifferentiability framework introduced by Maurer, Renner and Holenstein [24]. In [11], the authors prove that the success probability of differentiating a sponge construction calling a random permutation or transformation from a random oracle is upper bounded by $1 - \exp(-Q^2 2^{-(cp+1)})$ with $Q$ the number of calls to $f$ or its inverse when possible.

From this particular bound, the authors of [11] deduce a simplification of the proof model which considers from the previous result only the worst-case success probability. This simplification called the *flat sponge claim* makes the security of a concrete function seen as a random sponge depend on the capacity of the random sponge. More precisely, the collision resistance of a sponge construction under the flat sponge claim is $2^{\min(cp,N)/2}$, the (second) preimage resistance is $2^{\min(cp/2,N)}$.

Thus, we design our function using this flat sponge argument which allows to minimize the internal state (looking at the indifferentiability in the random oracle model) of the used function when compared with other traditional hash constructions such as the Merkle-Damgård one for example. In fact, sponge constructions lead to achieve the highest security level that could be obtained for a hash construction.

### 3.2 Choice of the $f$ function

As already mentioned for the hash function QUARK [7], the design choice of $f$ comes from this simple idea: from a stream cipher with an internal state of size $n$, one can construct a function from $\{0,1\}^b$ into itself as follows:

– The $b$-bit input is padded to an initial state of size $n$ bits,
– The stream cipher is initialized as usual,
– The first $b$ output bits compose the output of the $f$ function.

Under the hypothesis that the stream-cipher is "perfect", the $f$ function looks like a random function. In other words, finding a bias in the function $f$ is equivalent to find a weakness of the stream-cipher.

**F-FCSR-v3 and X-FCSR-v2.** We have based our $f$ design on a mix between two FCSR based stream ciphers which are F-FCSR-v3 [4] and X-FCSR-v2 [10]. The first one is dedicated to hardware and uses a linear filter as done here whereas the second one introduces the word ring oriented structure of a FCSR which is efficient both in hardware and in software. This kind of automata is the building block of our $f$ design.

Since the first proposal of a stream cipher based on a FCSR in 2005 in [2], the security of such designs has been carefully studied through the eStream call for stream ciphers [27]. One of the first proposals called F-FCSR-v2 based on a Galois FCSR was however successfully attacked by M. Hell and T. Johansson in 2008 in [19]. This attack exploits a particular dependence between the feedback bit and all the carry bits. This attack is also efficient against the first version of X-FCSR (X-FCSR-v1) as shown in [30]. Those particular attacks lead to the modification of the original proposals into the new versions F-FCSR-v3 and X-FCSR-v2 based on new matrix representations of the FCSR leading to discard the two previous attacks as detailed in Section 4. Since their publications, two years ago, no attack has been exhibited against the two new stream ciphers which are based on strong security arguments. This leads to have a relative level of confidence concerning those two particular primitives, particularly if we look at the absence of existing distinguishers against the building blocks.

Moreover, those designs are simple and relatively efficient both in software and hardware. Thus, these simple arguments concerning the efficiency of F-FCSR-v3 and of X-FCSR-v2 and the security level claimed lead to naturally consider them as possible instance for a compression function in the sponge model.

**Quality of the $f$ function.** The good statistical properties (period, balanced sequences and so on) of the underlying building blocks come from the 2-adic properties and are equivalent over $\mathbb{Z}_2$ to the ones of LFSRs over $\mathbb{F}_2$. We will see now how the design choices made here are efficient to prevent classical attacks.

First, the number of clocks in $f$ is $d+4$ to ensure a complete diffusion of the message block into all the words of the FCSR. Even if a difference is introduced in the message block, this difference will influence all the output blocks. This is due to the diameter definition which clearly improves the diffusion speed in a word ring representation. In other words, the diffusion is complete after $d+4$ clocks.

Let us consider a FCSR with connection integer $q$ which produces $\ell$-sequences, i.e. such that the order of 2 modulo $q$ is $P = |q| - 1$. After a few iterations from an initial state, the automaton is in a periodic sequence of states of length $P$. The average number of required iterations to be in such a state is experimentally less than $log_2(n)$, where $n$ is the size of the main register (see [5] for more details). Thus, during the application of $f$ such states are always reach because for all our cases $log_2(n) < d$. Those particular periodic states are all on a (the) main cycle of size $P$, generally the number of periodic state is close to $2^n$ (equal to $2^{(w+1)r}$ in our design). This leads to consider a function $f$ which is really

close to a permutation from $\{0,1\}^b$ into itself because the surjective part of the construction is really limited once the function $f$ acts on the main cycle. In other words, we exploit here the fact that the transition function of a FCSR becomes a permutation on its periodic states.

So, our $f$ function could be considered as an application from $\{0,1\}^b$ into the set of periodic states, i.e. into a set of size $\{0,1\}^n$ that has a behavior close to a permutation leading to a very low collision probability. Indeed, each possible input value conducts to a particular part of the main cycle. The entering function is thus a quasi-permutation on its cycles with a very small loose of entropy. As soon as the main cycle is reached, $f$ does not anymore loose entropy even after several iterations. The entropy lost that could be considered here comes from the extraction step. Thus, in summary, the design of $f$ prevents any entropy loss and the behavior of $f$ is close to the behavior of a random function.

However, the generic word ring FCSR structure brings with it a particular problem: there is on the graph representation of the FCSR two particular fixed points which are the all-zero point and the all-one point. Thus, to prevent attacks producing collisions on two different messages $M$ and $M'$ which differ on a certain number of all-zero blocks at the beginning (i.e. $M = 0||0||m_0$ and $M' = 0||m_0$ for example which lead to the same output results), we introduce the all-one bit constant in a word of the main register to avoid the all-zero point. The all-one point is discarded by the initialization of the carry register to the all-zero word at each $f$ application. These two states can never be reached by design.

## 4 Security Analysis

As for QUARK and `PHOTON`, we follow the *hermetic sponge strategy*, which consists in adopting the sponge construction and building an underlying function $f$ that should not have any structural distinguishers. The indifferentiability proof of the sponge construction shows that any non-generic attack on a GLUON hash function implies a distinguisher for its function $f$ (but the inverse is not necessarily true). This reduces the security of the hash function to the security of $f$. As already explained in Section 3.1, the indifferentiability proof of the sponge construction ensures an expected complexity against any differentiating attack such as collision or (second) preimage attacks. Thus, as GLUON follows the hermetic sponge strategy, we directly apply those results and analyze, in this section, the $f$ function.

**Collision attack and preimage attack.** If one tries to inverse the $f$ function from the outputs, he will face a combinatorial explosion, for a given $m_{k+1}$ bit value of the main register, the values $c_k$ and $m_k$ producing $m_{k+1}$ are not unique because contrary to the Galois or Fibonacci FCSRs, zeroing the content of the carry register no more guarantees to be on the main cycle of the FCSR (see [5] for more details). Whereas in this case, the knowledge of a part of the carry register and of the main register guarantees the possible inversion of those values, this is no more the case for a word ring FCSR (see [4] for more details). One

point of the state graph could have two preimages. So, the inversion of the $f$ function could not be easily done. Thus, due to the combinatorial explosion, the (second) preimage search has a complexity of $2^{3w*r}$, the combined size of the two registers.

The complexity of the collision search for the $f$ function could be directly deduced also from the previous remark noticing that the direct collisions search is to find $p$ and $p'$ such that $p \equiv p' \mod q$. So if we write again this equation at binary level for the contents of the main register and of the carry register, we are facing an instance of the subset sum problem, with a complexity equals to $2^{w*r/2}$ (if the carries are zeroes) or $2^{3w*r/2}$ (in the general case).

**Cube attacks and Cube testers.** Recently, new kind of attacks called "Cube attacks" have appeared (the most recent reference in this area is [6]). Those attacks could be efficient as soon as "Cube testers" show their efficiency in simplifying a part of the ANF of a function. Thus, those attacks could be applied against functions which have particular weaknesses in their algebraic structure. As shown in [9] for the case of a Galois FCSR, such a particular structure (with a low degree component) does not exist in a Galois FCSR. This is the same thing for a word ring FCSR and thus for the $f$ function chosen here. Indeed, for example with a Galois FCSR of size 16 after 7 clocks, the number of monomials in the output algebraic equations is 125420 with a degree of at least 10 considering only the variables of the main register as unknowns. The number of monomials of the obtained system becomes huge with a high degree as mentioned in [9] as soon as at least 10 clocks are performed. We thus conjecture that it discards the potential use of cube testers and renders cube attacks harmless as soon as at least 10 clocks are performed.

**Linear and differential attacks.** Linear attacks have been performed against Galois FCSRs using a method called LFSRization by Hell and Johansson in [19] and against Fibonacci FCSRs by Fischer et al. in [16]. The first attack relies on the existence of correlations between the carries and the feedback values. More precisely, the probability that the feedback bit is equal to 0 during $t$ consecutive clocks is $2^{-t}$ for a Galois FCSR allowing with this probability to linearize the FCSR (i.e. if the feedbacks are forced to 0 during $t$ clocks the behavior of the FCSR becomes linear). With a word ring FCSR, this probability becomes $2^{-t\Delta u}$ where $u$ is the number of bits of the main register controlling a feedback. Thus, for example, with a word ring FCSR of size 128 bits with around 70 feedbacks leads to a probability of $2^{-70}$ to control at the null value the feedback bits during one clock. The attack presented in [16] is also discarded because it only concerns stream ciphers where the underlying building block is a Fibonacci FCSR which is not the case here.

Differential attacks stay a very powerful tool to find inner collisions in compression functions as shown with MD5 and SHA-1 [31]. The main idea relies on the non-ideal behavior of difference propagations through the compression function. The resistance of F-FCSR stream ciphers against differential attacks

have been proven after some changes in F-FCSR-H v2 due to a slow diffusion of differences presented in [21]. More precisely, in [21], the authors show that a difference introduced in some cells of the FCSR automaton remains localized as long as this difference does not reach the feedback end of the register. From this remark, they deduce an attack on the $IV$ process of F-FCSR-H v1. Thus, in [3], The designers prevent this attack from happening by redesigning the $IV$ setup and increasing the number of initial clocks to be sure that a sufficient diffusion of differences occurs. In the new ring and word ring design, the diffusion criterion given by the diameter $d$ allows to determine the minimal value from which a sufficient diffusion level occurs. This is why, the number of clocks of the $f$ function is $d + 4$: we are sure that a minimal diffusion (in general and for differences in particular) is reached. In [21], the authors especially focus on one bit difference due to the structure of the FCSR, this is the same case for a word ring FCSR, the word structure being just a representation. So, after $d$ clocks and for the best differential case, a one bit difference becomes a two bits difference, after $d + 1$ clocks, this difference is spread in average into many cells becoming in average at least a four bits difference at different places and so on. So, we conjecture that the number of clocks equal to $d + 4$ is sufficient to correctly spread in average differences in several cells.

**Slide distinguishers.** As mentioned in Section 3.2, slide distinguishers could be built on the two particular fix points of the graph of a FCSR which are the all-zero point transformed into itself and the all-one point also transformed into itself. The all-one point could not be reached for $f$ when used inside the sponge construction due to the way the sponge construction builds its internal states. However, the all-zero point could be reached for $f$ if a particular initial value is not given to a particular word of the main register of the FCSR. This is why, to discard slide distinguishers that make use of block messages equal to zero, a particular word of the main register is initialized to the all-one point.

In conclusion, we could say that since the beginning of FCSR study in the cryptographic context 6 years ago, all the original awkwardnesses leading to conventional attacks or to more tricky attacks have been discarded along the design process. This is why, we think that cryptanalyzing ring FCSRs passes through creating new attacks exploiting really original sorts of relations that are no more linear, differential or algebraic.

## 5 Performances

This section details our experimental results. We describe the tools used and compare our results to the existing works.

### 5.1 Hardware performances

This section reports our hardware implementation of the GLUON instances. For simulation, we used the *ModelSim PE* simulator [25], version *10,0a*. For

synthesis, we used the ASIC synthetisor *Cadence RTL* version *RC10.1.101* (32-bits version) [14]. Our implementation consists into two components: the function $f$ and a register $R$ used to store the intermediate results.

We now detail the optimizations made for GLUON and discuss our implementation choices.

**Loading the FCSR in parallel** There is two ways to load data into the main register of a ring FCSR. All parts (*i.e.* $m_0..m_i$) can be loaded in parallel by inserting multiplexors before each $m_i$. This solution allows fast execution as the entire main register is loaded in one step. The other solution is to load the main register serially. Such a solution is usual for the classical representations of an FCSR, *i.e.* Galois or Fibonacci. The data are injected by one side of the register ($m_{N-1}$). A data $m_i$ reached its correct position after $i$-th steps of the FCSR. However, all the feedbacks of the FCSR must be inactivated using multiplexors. For the Galois or the Fibonacci representation a single multiplexor is enough. For a ring-FCSR, more multiplexors are needed: one per feedback. As a consequence, the cost of a parallel solution is very close to the cost of the serial one. We keep the parallel version.

**Storage** The different GLUON instances require to store the output of the filter at different time. These values are stored directly in the register $R$ to save memory.

**Optimization of the adders** Adders are used in the FCSR. In our design, adders are implemented as in [4], that is to say: $c = (a.b) \oplus (a \oplus b) \oplus c$ and $s = (a \oplus b) \oplus c$ where . denotes the binary AND. The synthesizer optimizes the design by factorizing the common sub-expression $a \oplus b$. An interesting solution to implement adders is the one used in PRESENT [13], but we did not implement adders this way because it is too costly in our case.

| Hash function | Security | | Block | Area | Lat. | Thr. |
|---|---|---|---|---|---|---|
| | Pre. | Coll. | [bits] | [GE] | [cycles] | kbps |
| GLUON-64 | 128 | 64 | 8 | 2071 | 66 | 12.12 |
| GLUON-80 | 160 | 80 | 16 | 2799.3 | 50 | 32 |
| GLUON-112 | 224 | 112 | 32 | 4724 | 55 | 58.18 |
| U-QUARK×8 | 128 | 64 | 8 | 2392 | 68 | 11.76 |
| D-QUARK×8 | 160 | 80 | 16 | 2819 | 88 | 18.18 |
| S-QUARK×16 | 224 | 112 | 32 | 4640 | 64 | 50.00 |
| PHOTON-80 | 160 | 80 | 16 | 1168 | 132 | 12.15 |

**Table 1.** Performances of GLUON with previous works.

**Results** Results are reported in Table 1. As we can see, our proposals compete well in terms of area with Quark proposals: they are 13.4% smaller for the 64-bit version and similar for the 80-bit and 112-bit versions. However, it seems that the greater the size is, the greater the gap between our results and those of Quark. The throughput is similar for the 64-bit version, worst for the 80-bits version and better for the last one. Compared to PHOTON (we only give the results for 80 bits as it is the provided version for which can compare), the area is clearly not in our favor but our throughput is better.

We did not include power consumption because the power consumption strongly depends on the technology used and cannot be compared between different technologies in a fair manner. In addition, simulated power results strongly depend on the simulation method used, and the effort spent.

### 5.2 Software performances

This section reports our software implementations. Table 2 gives the software performances we obtained on the variants detailed in Section 2.3. The processor used for the benchmarks is an Intel Core 2 Duo clocked at 2.66 GHz. Note that our implementation is not optimized for a particular processor, *e.g.* it is not optimized for running on a 64-bit processor. Results are good and better than the QUARK versions we benchmarked on the same machine. As expected, PHOTON is better (we only give performance comparison for the 80 bits version as it is the only provided version with which we can compare with).

| | |
|---|---|
| GLUON-64 | 9577 |
| U-QUARK | 43373 |

| | |
|---|---|
| GLUON-80 | 2643 |
| D-QUARK | 35103 |
| PHOTON-80 | 1243 |

| | |
|---|---|
| GLUON-112 | 683 |
| S-QUARK | 25142 |

**Table 2.** Software performances in cycles per byte of the GLUON variants for long messages, compared to QUARK versions and PHOTON when possible

## 6 Conclusion

After the lightweight hash proposals Quark and `PHOTON`, the family of lightweight particles expands with the GLUON family and three particular instances . GLUON-128/8, GLUON-160/16 and GLUON-224/32. Even if the software and hardware performances of GLUON are worst than the ones of `PHOTON`, there are comparable when targeting hardware to the parallelized versions of Quark. We think that our design is relevant and of real interest because the basic building blocks have been well-studied since twenty years.

We do not develop here (as done for `PHOTON`) the case where the squeezing step is reduced and produces more output blocks at each step. An initial study concerning this aspect shows that we could transform GLUON-128/8 into a

more challenging version fitting with the requirements of such a modified version. The idea is to directly output, during the squeezing step, one 8-bit word at each clock without waiting for $d + 4$ clocks and without changing the input of the $f$ function at each $f$ call. Of course, due to its simplicity, this version is more risky but the reached security level could be compared to the one of F-FCSR-v3.

## References


1. Mohamed Ahmed Abdelraheem, Céline Blondeau, María Naya-Plasencia, Marion Videau, and Erik Zenner. Cryptanalysis of armadillo2. Cryptology ePrint Archive, Report 2011/160, 2011. `http://eprint.iacr.org/`.
2. François Arnault and Thierry P. Berger. F-FCSR: design of a new class of stream ciphers. In *Fast Software Encryption - FSE 2005*, LNCS 3557, pages 83–97. Springer, 2005.
3. François Arnault, Thierry P. Berger, and Cédric Lauradoux. Update on F-FCSR Stream Cipher. ECRYPT - Network of Excellence in Cryptology, Call for stream Cipher Primitives - Phase 2 2006. `http://www.ecrypt.eu.org/stream/`.
4. François Arnault, Thierry P. Berger, Cédric Lauradoux, Marine Minier, and Benjamin Pousse. A new approach for fcsrs. In *Selected Areas in Cryptography - SAC 2009*, volume 5867 of *LNCS*, pages 433–448. Springer, 2009.
5. François Arnault, Thierry P. Berger, and Marine Minier. Some Results on FCSR Automata With Applications to the Security of FCSR-Based Pseudorandom Generators. *IEEE Transactions on Information Theory*, 54(2):836–840, 2008.
6. Jean-Philippe Aumasson, Itai Dinur, Willi Meier, and Adi Shamir. Cube testers and key recovery attacks on reduced-round md6 and trivium. In *Fast Software Encryption - FSE 2009*, volume 5665 of *LNCS*, pages 1–22. Springer, 2009.
7. Jean-Philippe Aumasson, Luca Henzen, Willi Meier, and María Naya-Plasencia. Quark: A lightweight hash. In *Cryptographic Hardware and Embedded Systems, CHES 2010*, volume 6225 of *LNCS*, pages 1–15. Springer, 2010.
8. Stéphane Badel, Nilay Dagtekin, Jorge Nakahara, Khaled Ouafi, Nicolas Reffé, Pouyan Sepehrdad, Petr Susil, and Serge Vaudenay. Armadillo: A multi-purpose cryptographic primitive dedicated to hardware. In *Cryptographic Hardware and Embedded Systems, CHES 2010*, volume 6225 of *LNCS*, pages 398–412. Springer, 2010.
9. Thierry P. Berger and Marine Minier. Two algebraic attacks against the f-fcsrs using the iv mode. In *Progress in Cryptology - INDOCRYPT 2005*, volume 3797 of *LNCS*, pages 143–154. Springer, 2005.
10. Thierry P. Berger, Marine Minier, and Benjamin Pousse. Software oriented stream ciphers based upon fcsrs in diversified mode. In *Progress in Cryptology - INDOCRYPT 2009*, volume 5922 of *LNCS*, pages 119–135. Springer, 2009.
11. Guido Bertoni, Joan Daemen, Michael Peeters, and Gilles Van Assche. On the indifferentiability of the sponge construction. In *Advances in Cryptology - EUROCRYPT 2008*, volume 4965 of *LNCS*, pages 181–197, 2008.
12. Guido Bertoni, Joan Daemen, Michael Peeters, and Gilles Van Assche. Sponge-based pseudo-random number generators. In *Cryptographic Hardware and Embedded Systems, CHES 2010*, volume 6225 of *LNCS*, pages 33–47. Springer, 2010.
13. Andrey Bogdanov, Lars R. Knudsen, Gregor Leander, Christof Paar, Axel Poschmann, Matthew J. B. Robshaw, Yannick Seurin, and C. Vikkelsoe. PRESENT: An Ultra-Lightweight Block Cipher. In *Cryptographic Hardware and Embedded Systems - CHES 2007*, LNCS 4727, pages 450–466. Springer, 2007.

14. Cadence. Encounter rtl compiler. http://www.cadence.com/products/ld/rtl_compiler.
15. Christophe De Cannière, Orr Dunkelman, and Miroslav Knezevic. KATAN and KTANTAN - A Family of Small and Efficient Hardware-Oriented Block Ciphers. In *Cryptographic Hardware and Embedded Systems - CHES 2009*, LNCS 5747, pages 272–288. Springer, 2009.
16. Simon Fischer, Willi Meier, and Dirk Stegemann. Equivalent Representations of the F-FCSR Keystream Generator. In *ECRYPT Network of Excellence - SASC Workshop*, pages 87–94, 2008. Available at `http://www.ecrypt.eu.org/stvl/sasc2008/`.
17. Mark Goresky and Andrew Klapper. Periodicity and distribution properties of combined fcsr sequences. In *Sequences and Their Applications - SETA 2006*, volume 4086 of *LNCS*, pages 334–341. Springer, 2006.
18. Jian Guo, Thomas Peyrin, and Axel Poschmann. The photon family of lightweight hash functions. In *CRYTPO 2011*, volume to appear of *LNCS*. Springer, 2011.
19. Martin Hell and Thomas Johansson. Breaking the F-FCSR-H stream cipher in real time. In *Advances in Cryptology - ASIACRYPT 2008*, volume 5350 of *LNCS*, pages 557–569. Springer, 2008.
20. Deukjo Hong, Jaechul Sung, Seokhie Hong, Jongin Lim, Sangjin Lee, Bonseok Koo, Changhoon Lee, Donghoon Chang, Jaesang Lee, Kitae Jeong, Hyun Kim, Jongsung Kim, and Seongtaek Chee. HIGHT: A New Block Cipher Suitable for Low-Resource Device. In *Cryptographic Hardware and Embedded Systems - CHES 2006*, LNCS 4249, pages 46–59. Springer, 2006.
21. Éliane Jaulmes and Frédéric Muller. Cryptanalysis of the f-fcsr stream cipher family. In *Selected Areas in Cryptography - SAC 2005*, volume 3897 of *LNCS*, pages 20–35. Springer, 2005.
22. Andrew Klapper and Mark Goresky. 2-adic shift registers. In *Fast Software Encryption - FSE'93*, LNCS 809, pages 174–178. Springer, 1993.
23. Chae Hoon Lim and Tymur Korkishko. mCrypton - A Lightweight Block Cipher for Security of Low-Cost RFID Tags and Sensors. In *Workshop on Information Security Applications - WISA 2005*, LNCS 3786, pages 243–258. Springer Verlqg, 2005.
24. Ueli M. Maurer, Renato Renner, and Clemens Holenstein. Indifferentiability, impossibility results on reductions, and applications to the random oracle methodology. In *Theory of Cryptography, First Theory of Cryptography Conference, TCC 2004*, volume 2951 of *LNCS*, pages 21–39. Springer, 2004.
25. ModelSim. Modelsim pe - simulation and debug. `http://model.com/content/modelsim-pe-simulation-and-debug`.
26. National Institute of Standards and Technology. Announcing Request for Candidate Algorithm Nominations for a NewCryptographic Hash Algorithm (SHA-3) Family. *Federal Register*, 27(212):62212–62220, November 2007. Available: `http://csrc.nist.gov/groups/ST/hash/documents/FR_Notice_Nov07.pdf`(2008/10/17).
27. Network of Excellence in Cryptology ECRYPT. Call for stream cipher primitives. `http://www.ecrypt.eu.org/stream/`.
28. Khaled Ouafi and Serge Vaudenay. Smashing squash-0. In *Advances in Cryptology - EUROCRYPT 2009*, volume 5479 of *LNCS*, pages 300–312. Springer, 2009.
29. Adi Shamir. Squash - a new mac with provable security properties for highly constrained devices such as rfid tags. In *Fast Software Encryption - FSE 2008*, volume 5086 of *LNCS*, pages 144–157. Springer, 2008.
30. Paul Stankovski, Martin Hell, and Thomas Johansson. An efficient state recovery attack on X-FCSR-256. In *Fast Software Encryption - FSE 2009*, volume 5665 of *LNCS*, pages 23–37. Springer, 2009.

31. Xiaoyun Wang and Hongbo Yu. How to break md5 and other hash functions. In *Advances in Cryptology - EUROCRYPT 2005*, volume 3494 of *LNCS*, pages 19–35. Springer, 2005.

# A Matrix $T$ and $FI$ function for the 64-bit security level GLUON version

**Matrix $T$:** Parameters: $w = 18$ blocks of $r = 8$ bits. The value of $q$ is equal to:

$$-22902874254594039368339911884338634654967949$$

```
> SL;
[0 1 0 0 0 0 0 0]
[0 0 1 0 0 0 0 0]
[0 0 0 1 0 0 0 0]
[0 0 0 0 1 0 0 0]
[0 0 0 0 0 1 0 0]
[0 0 0 0 0 0 1 0]
[0 0 0 0 0 0 0 1]
[0 0 0 0 0 0 0 0]
```

```
> SR;
[0 0 0 0 0 0 0 0]
[1 0 0 0 0 0 0 0]
[0 1 0 0 0 0 0 0]
[0 0 1 0 0 0 0 0]
[0 0 0 1 0 0 0 0]
[0 0 0 0 1 0 0 0]
[0 0 0 0 0 1 0 0]
[0 0 0 0 0 0 1 0]
```

$$T = \begin{pmatrix}
0 & I & 0 & 0 & 0\,0\,0\,0 & 0 & 0 & 0 & 0 & SR^3 & 0 & 0 & 0 & 0\,0 \\
0 & 0 & I & 0 & 0\,0\,0\,0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0\,0 \\
0 & 0 & 0 & I & 0\,0\,0\,0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0\,0 \\
0 & 0 & 0 & 0 & I\,0\,0\,0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & SL^6 & 0\,0 \\
0 & 0 & 0 & 0 & 0\,I\,0\,0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0\,0 \\
0 & 0 & 0 & 0 & 0\,0\,I\,0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0\,0 \\
0 & 0 & 0 & 0 & 0\,0\,0\,I & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0\,0 \\
0 & 0 & 0 & 0 & 0\,0\,0\,0 & I & 0 & 0 & SR^3 & 0 & 0 & 0 & 0 & 0\,0 \\
0 & 0 & 0 & 0 & 0\,0\,0\,0 & 0 & I & 0 & 0 & 0 & 0 & SR^3 & 0 & 0\,0 \\
0 & 0 & 0 & 0 & 0\,0\,0\,0 & 0 & 0 & I & 0 & 0 & 0 & 0 & 0 & 0\,0 \\
0 & 0 & 0 & 0 & 0\,0\,0\,0 & 0 & 0 & SL & I & 0 & 0 & 0 & 0 & 0\,0 \\
0 & 0 & 0 & 0 & 0\,0\,0\,0 & 0 & 0 & 0 & 0 & I & 0 & 0 & 0 & 0\,0 \\
0 & 0 & 0 & SL^6 & 0\,0\,0\,0 & 0 & 0 & 0 & 0 & 0 & I & 0 & 0 & 0\,0 \\
0 & 0 & 0 & 0 & 0\,0\,0\,0 & 0 & 0 & 0 & 0 & 0 & 0 & I & 0 & 0\,0 \\
0 & 0 & 0 & 0 & 0\,0\,0\,0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & I & 0\,0 \\
0 & 0 & 0 & 0 & 0\,0\,0\,0 & SL^3 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & I\,0 \\
0 & 0 & 0 & 0 & 0\,0\,0\,0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0\,I \\
I & SR^6 & 0 & 0 & 0\,0\,0\,0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0\,0
\end{pmatrix}$$

$$m(t) = (m_0(t), \cdots, m_{17}(t))^T \text{ and } m(t+1) = Tm(t)$$

8-bit words with carries: $m_0$, $m_3$, $m_7$, $m_8$, $m_{10}$, $m_{12}$, $m_{15}$, $m_{17}$. The diameter $d$ is equal to 44.

**Filter $FI$:**

$$\begin{aligned} FI(m(t)) = &(m_0(t) \oplus (m_3(t)_{>>>4})) \quad \oplus \\ &(m_7(t) \oplus (m_8(t)_{<<<2}))_{>>>5} \quad \oplus \\ &(m_{10}(t) \oplus (m_{12}(t)_{>>>3}))_{<<<1} \oplus \\ &(m_{15}(t) \oplus (m_{17}(t)_{>>>5}))_{<<<4} \end{aligned}$$

## B Matrix $T$ and $FI$ function for the 80-bit security level GLUON version

**Matrix $T$:** Parameters: $w = 12$ blocks of $r = 16$ bits. The value of $q$ is equal to:

$$-1196184691751347078913349747445350417991621293568616511077 3$$

$$T = \begin{pmatrix} 0 & I & 0 & 0 & 0 & 0\,0\,0 & 0 & 0 & SL^{12} & 0 \\ 0 & 0 & I & 0 & 0 & 0\,0\,0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & I & SR & 0\,0\,0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & I & 0\,0\,0 & SR^6 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & I\,0\,0 & 0 & 0 & 0 & 0 \\ 0 & I & 0 & 0 & 0 & 0\,I\,0 & 0 & 0 & 0 & 0 \\ SR^6 & 0 & 0 & 0 & 0 & 0\,0\,I & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0\,0\,0 & I & 0 & 0 & 0 \\ 0 & 0 & 0 & SL^{10} & 0 & 0\,0\,0 & 0 & I & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0\,0\,0 & 0 & 0 & I & 0 \\ 0 & 0 & 0 & 0 & 0 & 0\,0\,0 & 0 & 0 & 0 & I \\ I & 0 & SL^{11} & 0 & 0 & 0\,0\,0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

16-bit words with carries: $m_0$, $m_2$, $m_3$, $m_6$, $m_8$, $m_{11}$. The diameter $d$ is equal to 34.

**Filter $FI$:**

$$\begin{aligned} F(m) = &(m_0(t) \oplus (m_2(t)_{>>>3})) \quad \oplus \\ &(m_3(t) \oplus (m_6(t)_{<<<5}))_{>>>7} \quad \oplus \\ &(m_8(t) \oplus (m_{11}(t)_{>>>2}))_{>>>2} \end{aligned}$$

## C Matrix $T$ and $FI$ function for the 112-bit security level GLUON version

**Matrix $T$:** Parameters: $w = 9$ blocks of $r = 32$ bits. The value of $q$ is equal to:

$$-42288419239352779381136581495386884385959973$$
$$56223832672089412465139266477362728972043613$$

$$T = \begin{pmatrix} SR^{12} & I & 0 & 0 & 0\,0 & 0 & 0 & 0 \\ 0 & 0 & I & 0 & 0\,0 & 0 & 0 & 0 \\ 0 & 0 & 0 & I & 0\,0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & I\,0 & SR^9 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0\,I & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0\,0 & I & SR^7 & 0 \\ 0 & 0 & 0 & SL^7 & 0\,0 & 0 & I & 0 \\ 0 & 0 & 0 & 0 & 0\,0 & 0 & 0 & I \\ I & 0 & SL^6 & 0 & 0\,0 & 0 & 0 & 0 \end{pmatrix}$$

32-bit words with carries: $m_0$, $m_3$, $m_5$, $m_6$, $m_8$. The diameter $d$ is equal to 42.

**Filter *FI*:**

$$\begin{aligned} F(m) = (m_0(t) \oplus (m_3(t)_{>>>15})) \quad & \oplus \\ (m_5(t) \oplus (m_6(t)_{<<<8}))_{>>>3} \oplus m_8(t)_{>>>13} & \end{aligned}$$