Hardware Implementations of Multi-output Welch-Gong Ciphers

C.H. Lam, M. Aagaard, and G. Gong

Department of Electrical and Computer Engineering
University of Waterloo
Waterloo, Ontario N2L 3G1, CANADA

(March 2009)

Abstract

The WG family of ciphers provides keystreams with mathematically proven randomness properties such as ideal two-level autocorrelation, balance, long period, ideal tuple distribution, and high and exact linear complexity. In this paper, we extend the mathematical analysis of WG family of ciphers to multi-output WG (MOWG) ciphers and demonstrate that MOWG ciphers provide a large design space of hardware architectures and implementation options with tradeoffs in security, area, and performance, including options that are competitive with the eSTREAM Phase-3 Profile-2 finalists.

1 Introduction

Many hardware-oriented stream ciphers in the eSTREAM stream cipher project [17] are designed to defeat algebraic attacks by using nonlinear feedback shift registers (NFSRs) or, more generally, updating the state of the stream cipher nonlinearly. Because very few theoretical results exist for the NFSRs, the security of such stream ciphers relies on the difficulty of analyzing the design itself.

The WG family of ciphers is designed to be analyzed mathematically. The WG ciphers have been proven to guarantee a variety of desirable randomness properties, such as ideal two-level autocorrelation, balance, long period, ideal tuple distribution, and high and exact linear complexity. These properties fulfill a number of security requirements, especially for encryption and authentication in radio frequency identification (RFID) systems. The low-autocorrelation property and the ideal 2-level autocorrelation property provide tamper resistance for RFID tags and prevent side-channel attacks, since the power spectral density of the key-stream sequence is flat. The good correlation property could be used to perform tag anti-collision and reader anti-collision in RFID systems at the signal-transmission level for free.

To defeat the algebraic attacks on LFSR based stream ciphers, WG-ciphers rely on nonlinear Boolean functions with a large number of inputs, a high degree and complex algebraic normal forms (ANFs). To overcome the implementation complexity of the Boolean function, the WG ciphers are designed in polynomial form instead of ANF. Implementation of such functions require multiplications over small finite-fields, which can be done efficiently using lookup tables or optimal normal-basis multipliers.

In this paper, we extend the theory of WG ciphers to *multi-output* WG (MOWG) ciphers, describe a variety of hardware design options for MOWG ciphers, and demonstrate that MOWG ciphers offer a large design space with tradeoffs in security, area, and performance, including options that are competitive with the eSTREAM Phase-3 Profile-2 finalists.

A MOWG stream cipher consists of a MOWG keystream generator that produces a long pseudo-random keystream. Conventional WG ciphers produce one output bit; MOWG ciphers over $\mathbb{F}_{2^m}$ output up to approximately $m/2$ bits, allowing for significant performance improvements with no additional hardware cost. The keystream is XORed with the plaintext to produce the ciphertext. The MOWG keystream generators use

WG permutations as the filtering functions. The WG permutations have very large ANFs and can be implemented in polynomial form using finite field arithmetic. The WG transform is fed by an LFSR. The two principal design parameters for a MOWG cipher are the size of the finite field $\mathbb{F}_{2^m}$ and the primitive polynomial $p(x)$ for the LFSR. The internal state of the cipher is the $m \times l$ set of flip-flops in the LFSR, where l is the degree of $p(x)$. The constraint on the number of output bits is $d \leq m - K/l$, where K is the size of the secret key in bits (Section 2).

We have implemented MOWG ciphers over $\mathbb{F}_{2^7}$, $\mathbb{F}_{2^{11}}$, and $\mathbb{F}_{2^{29}}$ with LFSRs ranging from 5 to 23 stages. Our hardware design options for the WG transform include random logic, ROM tables, and GF multipliers that were pipelined, superpipelined, or superpipelined with reuse. These design options offer a variety of tradeoffs in area and performance. We outline the best options for different choices of MOWG parameters and different implementation technologies (e.g., ASIC vs FPGA). The area and speed characteristics of ASICs are very different from FPGAs, and the design options must be chosen to take maximum advantage of the features of the implementation technology.

We have benchmarked MOWG ciphers over $\mathbb{F}_{2^7}$ with a 23-stage LFSR against Trivium-64 [2] and Grain-16 [3], the two leading contenders for hardware ciphers in eSTREAM. On an ST Microelectronics 90 nm ASIC cell library, a MOWG cipher was significantly smaller than either Trivium-64 or Grain-16 and competitive in optimality (ratio of throughput to area) with Grain-16. On the ASIC, using Trivium-64 as a baseline of 1.00, Grain-16 has an optimality of 0.18 and MOWG has an optimality of 0.13. Based on several comparisons of the hardware-oriented ciphers in eSTREAM, all of the other eSTREAM ciphers have relative optimalities below 0.05 [1, 4, 6, 5].

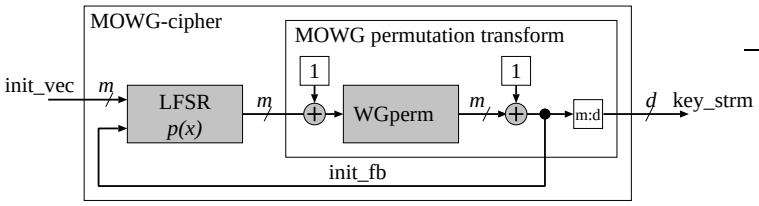
This paper is organized as follows: Section 2 gives an overview of the mathematics and security analysis of MOWG ciphers, Section 3 covers the hardware design options and tradeoffs, Section 4 describes our implementations of MOWG ciphers and compares our results to Trivium-64 and Grain-16.

2 Mathematics and Security Analysis

We use the following notation: $\mathbb{F}_2 = GF(2)$, finite field with 2 elements: 0 and 1; $\mathbb{F}_{2^n} = GF(2^n)$, i.e., extension field of $GF(2)$ with 2^n elements. The autocorrelation of a binary sequence with period v is defined as the difference between their agreements and disagreements when the symbol 0 maps to 1 and 1 maps to -1. If all the out-of-phase autocorrelation is equal to -1, then the sequence is said to have *ideal two-level autocorrelation*. The linear complexity, LC, of a binary sequence is defined as the size in bits of the shortest LFSR required to generate that sequence.

A MOWG cipher can be regarded as a nonlinear filter. A MOWG cipher (Figure 1) consists of a linear feedback shift register, followed by a MOWG permutation transform. The MOWG permutation transform $\mathrm{MOWG}(x)$ is defined in terms the permutation WGperm. The LFSR is based on an l degree primitive polynomial $p(x)$ over $\mathbb{F}_{2^m}$. The LFSR generates a maximal-length sequence over $\mathbb{F}_{2^m}$. This simple design generates a keystream whose period is $2^n - 1$, where $n = lm$, and that is easy to analyze for various cryptographic properties. The input signal `init_vec` and feedback signal `init_fb` are used only in the initialization phase of operation. When the cipher is running, the only feedback is within the LFSR. The absence of feedback paths allows for efficient pipelining of the MOWG permutation transform. The output of the cipher is a vector of d keystream bits, where $d \leq m - K/l$, where K is the size of the secrete key. We denote a MOWG generator with an LFSR of l stages over $\mathbb{F}_{2^m}$ that outputs d bits as an (m, l, d)MOWG generator. Note that an $(m, l, 1)$MOWG generator is not necessarily an (m, l)WG generator, because WG generators have a trace function from $\mathbb{F}_{2^m}$ to $\mathbb{F}_2$ after the permutation (see details in the appendix).

Figure 1 contains the equations for the MOWG-cipher and WG-permutation. The WG permutation exists only if $m(\mathrm{mod}3) \neq 0$. We generally prefer values of m where $m(\mathrm{mod}3) = 2$, because they result in smaller values of k, which in turn reduces the number of multiplications to calculate $2^k - 1$ (Section 3.2). For



Mathematical parameters

m	Bit-width of cipher
$g(x)$	Generating polynomial for $\mathbb{F}_{2^m}$
$p(x)$	Primitive polynomial for LFSR
l	Degree of $p(x)$.

Find k such that $3k \equiv 1 \; mod \; m$.

$$\begin{aligned} \mathrm{MOWG}(x) &= \mathrm{WGperm}(x+1)+1 \\ \mathrm{WGperm}(x) &= x + x^{r_1} + x^{r_2} + x^{r_3} + x^{r_4} \\ \text{where } x &\in \mathbb{F}_{2^m} \end{aligned}$$

$$\begin{aligned} r_1 &= 2^k + 1 \\ r_2 &= 2^{2k} + 2^k + 1 \\ r_3 &= 2^{2k} - 2^k + 1 \\ r_4 &= 2^{2k} + 2^k - 1 \end{aligned}$$

Figure 1: Architecture and definitions for MOWG ciphers

example, as we increment m from 11 to 13 $k = 4, 7, 5, 9$. For designs that implement the WG permutation using multipliers, we prefer values of m that allow us to use an optimal normal basis multiplier [32].

We consider a value of m to be a reasonable choice for a MOWG cipher if $m \bmod 3 \neq 0$ and either of the following hold: m is small enough to allow a table-based implementation of the permutation ($m \leq 11$), or $m \bmod 3 = 2$ and m has an optimal normal basis. The reasonable values of m within the range of $7 \ldots 29$ are: 7,8,10,11,23,29.

2.1 Selection of Parameters for Hardware Implementations

Once the bit width m has been chosen, the next step is to pick a basis and a defining polynomial for $\mathbb{F}_{2^m}$. The WG cipher can be implemented using either a normal or polynomial basis. The choice of basis affects:

- To guarantee the 1-order resiliency of the MOWG cipher, the basis must be at least 1-order resilient.
- Implementation of WGperm. In normal bases x^{2^i} is implemented simply as a hardwired rotation, $+1$ is bitwise negation, and optimal normal basis multipliers are smaller than other multipliers.
- Primitive polynomial for the LFSR. To guarantee the periodic two-level autocorrelation security property of the MOWG cipher, the LFSR and WG transform must use the same basis for $\mathbb{F}_{2^m}$. The primitive polynomial is of the form $a_n x^n + a_{n-1} x^{n-1} + \cdots + a_0 x^0 + \gamma$. If a polynomial basis is used, γ is of the form α^k where α is a root of $g(x)$ and values of k can be found such that multiplication by α^k can be implemented with wired shifts and a few XOR gates. If a normal basis is used, then multiplication by γ generally requires a multiplication by a constant that is significantly more expensive in area.

If the WG permutation will be implemented using Galois-field multipliers, then an optimal normal basis is preferred. The small size of the multipliers more than makes up for the area and delay cost of the multiplication by a constant in the LFSR. If the WG permutation will be implemented with random logic or a ROM, then a polynomial basis is preferred, because this allows the multiplication by γ in the LFSR (Figure 2) to be implemented with wired shifts and a few XOR gates. These area-driven reasons for the choice of basis correlates with the clock period, in that table-based implementations of WGperm are used with smaller finite fields, which in turn result in higher clock speeds. Using a multiplier in the LFSR feedback path would be the dominant critical path and limit the clock speed of the cipher.

After the generating polynomial has been chosen, the primitive polynomial $p(x)$ for the LFSR is chosen. The degree l of $p(x)$ must be sufficiently large so that the size of the internal state $n = lm$ is twice the size of the secret key. The degree should also be sufficiently large to give the LFSR a long period. Within those

constraints, we also want a minimal number of taps and want the multiplication by γ to be implemented as efficiently as possible. A less important effect is that depending on implementation details, WG keystream generators can be re-initialized in as few as $2l$ clock cycles.

When selecting design parameters, some general restrictions that apply to all keystream generators must also be met. For example, the algebraic degree of the filtering function and its algebraic immunity must be high to prevent algebraic attacks, *etc*. Table 1 lists some options for the parameter value and the resulting amount of security for the cipher. The table lists only a few of the options. Many other options exist, even within the range of values shown. The linear complexities of MOWG keystream sequences can be easily computed from the formula summarized in [26].

Table 1: Example MOWG ciphers and security levels

m	l	n	LC_{WGK}	C_{AA}	ONB	$T_d > 2^K$
7	5	35	$2^{12.6}$	$2^{40.82}$		$2 \ldots 3$
7	11	77	$2^{16.9}$	$2^{53.98}$		$2 \ldots 3$
7	13	91	$2^{17.8}$	$2^{56.74}$		$2 \ldots 3$
7	23	161	$2^{21.02}$	$2^{66.11}$		$2 \ldots 3$
11	15	165	$2^{23.36}$	$2^{80.57}$	$\surd$	$2 \ldots 5$
11	16	176	$2^{24.7}$	$2^{81.7}$	$\surd$	$2 \ldots 6$
11	24	264	$2^{27.6}$	2^{90}	$\surd$	$2 \ldots 6$
11	32	352	$2^{29.6}$	$2^{95.8}$	$\surd$	$2 \ldots 6$
29	11	319	2^{45}	2^{182}	$\surd$	$2 \ldots 17$

n	Size of the internal state (or LFSR) in bits
LC_{WGK}	Linear complexity of the cipher
C_{AA}	Complexity of an algebraic attack on the cipher
ONB	$\surd$ denotes that optimal normal bases exist

2.2 Multi-output WG Key Stream Generators

Dobbertin showed that WGperm(x) is a permutation of $\mathbb{F}_{2^m}$ [8]. Here we refer to WGperm(x) as a *WG permutation* to match the hardware implementation of the core part of the WG keystream generator. We refer to MOWG(x) as a *WG permutation transform*. (The exponents in the monomial trace terms of the WG permutation are represented in the unified form for the both cases $m = 3k - 1$ and $m = 3k - 2$ in [26].)

In Figure 1, we output d bits from the m-bits of the WG permutation transform MOWG(x). MOWG key-stream sequences inherit many desirable randomness properties of WG keystream sequences, as stated in Proposition 1 and proved in the appendix.

Proposition 1 *Any output sequence of an (m, l, d)MOWG keystream generator satisfies the following randomness properties. 1) The period is $d(2^n - 1)$. 2) The linear complexity is equal to $(d \cdot LC_{WGK})$ where LC_{WGK} is the linear complexity of the WG keystream sequence. 3) The autocorrelation takes value $-d$ for all shifts τ except for $\tau = kd + j, 0 \leq j < d, 1 \leq k \leq 2^m - 2$.*

From Proposition 1, there are at most $d(2^m - 2)$ autocorrelation values of a MOWG keystream sequence that are not equal to $-d$ ($d < m$). For the MOWG keystream generators with at least an 80-bit secret key, the probability that the autocorrelation is not equal to $-d$ is negligible. In the following, we consider the nonlinearity of the MOWG permutation transform and the range for d for the security of the MOWG generator.

Nonlinearity of MOWG. An (m, d) multi-output Boolean or vectorial Boolean function is defined as a function maps from an m bit vector space to a d bit vector space. The nonlinearity of a (m, d)-function is defined as the minimum nonlinearity of all the linear combination of d component functions, which is introduced by Nyberg [28] and studied by Chabaud and Vaudenay [9]. The nonlinearity of a (m, d)MOWG

permutation transform is not known theoretically. However, our experimental results showed that the nonlinearity is high. This result allows us to output multiple bits of the WG permutation transform for increased performance. Also, the (m, d)MOWG permutation transform is balanced, i.e., each d-bit pattern occurs exactly 2^{m-d} times.

Output size of MOWG. Note that MOWG(x) is a permutation. From l consecutive elements of $\{a_i\}$, we can recover one state of the m-sequence $\{b_i\}$. Thus the initial state of the LFSR can be recovered. Therefore, for the MOWG generator, $d < m$. A proper d to output d bits from the WG permutation transform can be determined by the exhaustive search complexity for keys. In detail, assume that r consecutive elements of the output vectors are known. Then in order to recover the corresponding b_i, we need 2^{m-d} guesses for each b_i. For l consecutive b_i's, we need to verify $T_d = 2^{(m-d)l}$ guesses. Thus, to prevent this guessing attack, we require that $T_d > 2^K$ where K is the size of the secret key in bits. In the last column of Table 1, we list the ranges of d for (m, l, d)MOWG, with $K = 80$ for $m = 7$ and $m = 11$, and with $K = 128$ for $m = 29$.

2.3 Security Analysis

A. Impact of Good Correlation Property: The use of the WG or MOWG steam cipher fulfills multiple security requirements, especially when it is applied for encryption and authentication in radio frequency identification (RFID) systems. Usually, there is no tamper resistant module in low cost RFID systems. However, the ideal 2-level autocorrelation of WG cipher and the low autocorrelation property possessed by MOWG cipher provide tamper resistance for RFID tags. The WG cipher also can prevent side-channel attacks since the power spectral density of a WG keystream sequence is flat. Another benefit of god correlation of WG or MOWG keystream sequences is for reader anti collision in RFID systems [16] at signal transmission level which is obtained for free, i.e., the reader collision is prevented since different readers sends different ciphertext for which the keystream sequences having low correlation, an analogue to code division multiple access (CDMA) technologies [31]).

B. Resistance against Various Attacks: As discussed in [26], to provide protection against time/memory/data tradeoff attacks, the size of the internal state (or length of LFSR in bits) must be at least $2K$, i.e., $n \geq 2K$, and to provide protection against algebraic attacks [10], the complexity of the algebraic attack that recovers the secret key K of the WG generator is approximately $7/64 \cdot \binom{n}{\lceil m/3 \rceil + 1}^{log_2 7}$. The same result is also applied to the MOWG generator. Another type of attack is proposed in [30] [29], which needs LC_{WGK} consecutive bits. This attack can be prevented by choosing l and m such that the WG or MOWG sequences have large linear complexity. Various correlation attacks can be prevented since the nonlinearity of WG transform or MOWG permutation transform is very high. Algebraic attacks can be defeated completely by changing the design of the MOWG cipher to incorporate `init_fb` as an additional input to the WG transform. The cost of this design change is that the feedback will be needed during run mode, thereby eliminating the ability to pipeline the transform. This option is potentially attractive only for small MOWG ciphers whose datapaths are not pipelined.

For chosen-IV attacks, by running the initialization phase for $2l$ clock cycles, the differentials between two chosen IVs involve most of the internal state, and so make such attacks infeasible. For fault attacks, the MOWG transform has three-valued addition correlation, i.e., $\sum_{x \in \mathbb{F}_{2^m}} (-1)^{WG(x)+WG(x+a)}$ takes only three values $0, \pm 2^{(m+1)/2}$ for all $0 \neq a \in \mathbb{F}_{2^m}$. This prevents the fault attack in [22].

3 WG in Hardware

After the mathematical parameters of the cipher have been chosen, the first implementation decision is how to implement the WG transform. There are three options: ROM, random logic, or Galois-field multipliers.

In ASICs, the primary decision factor is area. We found that the best option was random logic for small ciphers (e.g., $\mathbb{F}_{2^7}$) and multipliers for larger sizes. A ROM was always larger and slower than the other options, although, at 11-bits, all three options are relatively close and so the choice will depend on the specific characteristics of the implementation technology.

In FPGAs, because dedicated block memories are a separate resource from regular FPGA cells, there is not a direct tradeoff in area between ROM-based designs and the other choices. For FPGAs, the primary decision factor is clock speed. For $\mathbb{F}_{2^7}$, random logic was must faster than a ROM. For $\mathbb{F}_{2^{11}}$ and $\mathbb{F}_{2^{29}}$ we pipelined the Galois field multipliers into two stages. This gave us a critical path to just two LUTs,[1] which is much faster than the access time for a ROM.

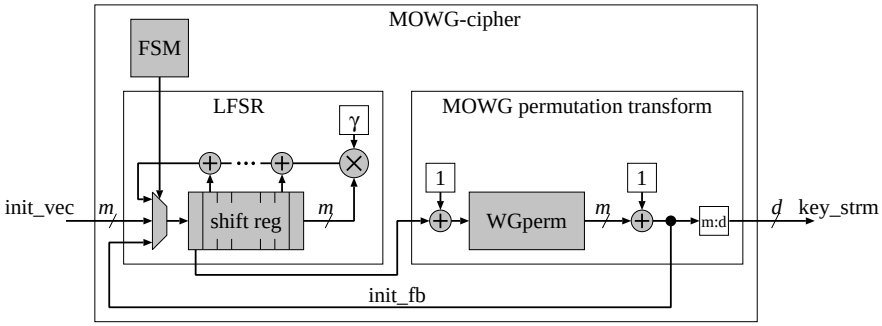


Figure 2: Detailed architecture

Figure 2 is a more detailed version of the high-level architecture from Figure 1. The state machine (FSM) controls which phase the cipher is in: load key and IV, initialize, or run. The duration of each phase is determined by the number of stages in the LFSR l and the latency through WGperm L. The FSM is implemented using an l-bit one-hot counter and a 2-bit binary counter for the phase. The key and IV loading phase lasts l clock cycles, during which the input to the LFSR is `init_vec`. The initialization phase last $2lL$ clock cycles, during which the input to the LFSR is the XOR of the LFSR feedback and `init_fb` from the WG-permutation. During the run phase, `init_fb` is not used, which enables the WG-permutation to be deeply pipelined.

During initialization, the FSM's l-bit counter increments and the LFSR shifts once every $L + 1$ clock cycles. (During the run phase, the LFSR shift on every clock cycle and the FSM's counters are irrelevant.) If the implementation of WGperm is not combinational, then $L \neq 0$ and we must implement the LFSR and FSM using registers with chip-enables. For an FPGA, the only cost of the chip enables is a slight increase in the setup time. But, for an ASIC, using chip enables is expensive in both time and space. In ASICs, flip-flops with a chip enable have a significantly longer setup time than flip-flops without chip enables (e.g. 0.42 ns vs 0.06 ns for the ST 90nm library that we used). At clock speeds in the neighborhood of 1GHz, this setup time consumes almost half of the clock cycle and limits the amount of computation that can be done in a clock cycle.

When we need a chip enable, we propagate a valid bit through WGperm and use the valid bit as the chip enable for the FSM and LFSR. In designs that reuse the multipliers (Section 3.3), the valid bits in WGperm also serve as the select line to multiplexers on the inputs to the multipliers.

[1] "LUT" refers to a 4:1 lookup table, which is the basic building block of combinational logic in FPGAs

Mathematically, either Galois or Fibonacci style feedback may be used. Galois-style feedback has a shorter delay, but we were able to find primitive polynomials of just a few terms, which made the difference between the two styles insignificant. We chose Fibonacci-style feedback to be consistent with the mathematical presentations of the WG-cipher.

The multiplier at the end of the shift register multiplies a constant γ by the output of the shift register. The value of γ is determined by the primitive polynomial $p(x)$ implemented by the shift register. In a polynomial basis, the constant γ is the root of the generating polynomial raised to some power (e.g., $\gamma = \alpha^3$). When α^k has a low Hamming weight, multiplication by γ simplifies to wired-shifts and a few XOR gates. When choosing $p(x)$, it is important to find a γ that allows this optimization. Because the multiplication in the LFSR is on the feedback path, it cannot be pipelined. A full multiplier would almost always make this the critical path in the design.

The rest of this section contains: table-based designs for WGperm (Section 3.1), mathematical optimizations for multiplier based implementations of WGperm (Section 3.2), and hardware implementation optimizations and options relating to pipelining and reuse (Section 3.3).

3.1 Table-Based Design

For a table based design, we compute all 2^m values of $\mathrm{WGperm}(x+1)+1$. This table can then be implemented in hardware either as random logic or a ROM. The exponents of x in $\mathrm{WGperm}(x+1)$ are given in [27].

We have the option to put registers on the output of the permutation transform to cut the critical path through WGperm and the multiplexers on the LFSR input. The increase in clock speed comes at the expense of increased area, because we need to add chip-enables to the flip-flops in the LFSR and FSM. In our (7,23,3)MOWG cipher, it was slightly advantageous to put registers on the output of the permutation transform. For MOWG ciphers over larger fields, the registers are definitely advantageous.

3.2 Mathematical Optimizations

A direct implementation of WGperm requires seven multiplications and an inversion. The equations for the r_i exponents in the permutation can be optimized to reduce the number of multiplications from seven to four and to replace the inversion with a small number of multiplications. In Figure 3 we rewrite r_2 in terms of r_1 and introduce s to capture a common subexpression between r_3 and r_4.

Simplifications:

$$2^{2k}+2^k = (2^k+1)\times 2^k$$
$$2^{2k}-2^k = (2^k-1)\times 2^k$$

WGperm exponents:

$$s = 2^k-1$$
$$r_1 = 2^k+1$$
$$r_2 = 2^{2k}+r_1$$
$$r_3 = s\times 2^k+1$$
$$r_4 = 2^{2k}+s$$

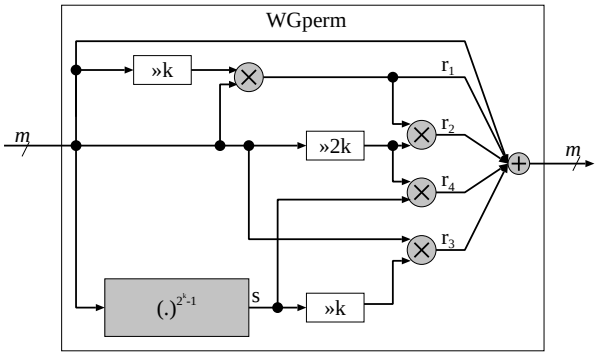


Figure 3: Multiplier-based implementation of WGperm

We have reduced the implementation from seven multiplications to four, plus an inversion in r. Inversions are expensive (e.g., six multiplications for $\mathbb{F}_{2^{29}}$). We remove the inversion by rewriting s as a

summation using the equation: $2^k - 1 = \sum_{i=0}^{k-1} 2^i$. We then optimize the summation using a recursive divide-and-conquer approach that generates common subexpressions, resulting in a logarithmic number of multiplications based on the value of k (Equation 1). This approach is similar to a common method for performing inversions. The disadvantage of doing the inversion in our situation is that the number of multiplications is logarithmic in m, while for $2^k - 1$, the number of multiplications is logarithmic in k and $k = \lceil m/3 \rceil$.

As an example, Figure 4 shows how $2^{10}-1$ evaluates mathematically using Equation 1 and is then implemented in hardware. The math can be matched against the hardware by reading the expression from inside to outside starting with $(1 + 2^1)$. The implementation is usually pipelined at the level of one multiplication per stage, or in the case of superpipelining, the multiplier itself is a two-stage pipeline (Section 3.3).

$$\sum_{i=0}^{j} = \begin{cases} 2^0 & : \quad j = 0 \\ \left(1 + 2^{\lfloor \frac{j-1}{2} \rfloor + 1}\right) \sum_{i=0}^{\lfloor \frac{j-1}{2} \rfloor} 2^i & : \quad \text{odd } j \\ \left(\left(1 + 2^{\lfloor \frac{j-1}{2} \rfloor + 1}\right) \sum_{i=0}^{\lfloor \frac{j-1}{2} \rfloor} 2^i\right) + 2^j & : \quad \text{even } j \end{cases} \tag{1}$$

$$\begin{aligned} & 2^{10} - 1 \\ &= \sum_{i=0}^{9} 2^i \\ &= \left(1 + 2^5\right)\left(\left(1 + 2^2\right)\left(\left(1 + 2^1\right)2^0\right) + 2^4\right) \\ &= \left(1 + 2^5\right)\left(\left(1 + 2^2\right)\left(1 + 2^1\right) + 2^4\right) \end{aligned}$$

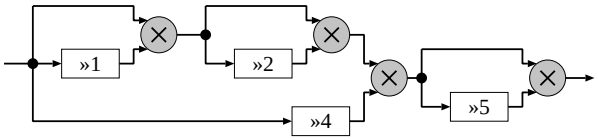


Figure 4: Mathematics and implementation of $2^{10} - 1$

3.3 Pipelining and Reuse

For bit-widths at which GF-multipliers are used (e.g. 11 or more bits), the delay through WGperm is much larger than that through the state-machine or LFSR. Fortunately, WGperm is easily pipelined. A good starting point is to match the clock-cycle to the delay of a single GF-multiplier. If the resulting clock speed is still insufficient and the multiplier is the critical path, the GF-multiplier itself may be pipelined (which we characterize as super-pipelining). Broadly speaking, there are four possible critical paths: the LFSR feedback, the multiplier, and the feedback from WGperm to the LFSR. The relative delays of these paths is dependent on m, $p(x)$, and the implementation technology. In this section, we mention a variety of techniques we used to balance out the delays of these paths.

When pipelining or super-pipelining, we include a stage register after the $+1$ negation at the end of the permutation. These registers prevent the path through the XOR, negation, and multiplexed inputs to the LFSR from becoming the critical path. Although this path is relatively short in terms of the logic delay, the long setup time constraint on the LFSR flip flops with chip enables can easily cause this to be the critical path.

Super-pipelining can be used to either double the throughput with a small increase in area, or halve the the area with a small decrease in throughput (Figure 5c). We illustrate the design options with reuse and

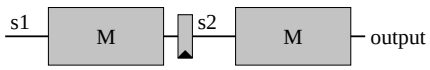


Figure 5a: Pipeline

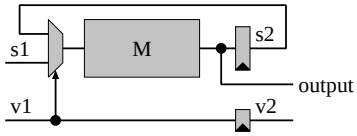


Figure 5b: Pipeline with reuse

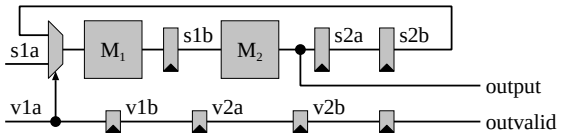


Figure 5c: Superpipelining with reuse

	No-reuse	Reuse
Pipelining	A, τ, T	$\frac{1}{2}A$, τ, $\frac{1}{2}T$
Superpipelining	A, $\frac{1}{2}\tau$, $2T$	$\frac{1}{2}A$, $\frac{1}{2}\tau$, T

Figure 5d: Design options with ideal effect on area A, clock period τ, and throughput T

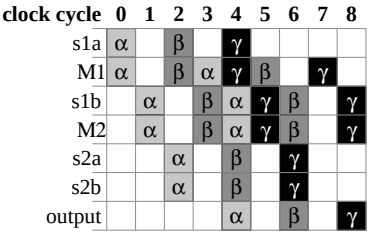


Figure 5e: Superpipelining with reuse

Figure 5: Superpipelining and reuse options

superpipelining using a circuit M that can be superpipelined into two subcircuits $\mathtt{M}_1$ and $\mathtt{M}_2$. In the baseline pipeline (Figure 5a) M blocks are used in both the first and the second pipeline stages. With reuse (Figure 5b), we reduce both the area and throughput by 50%. Superpipelining by itself (not shown) decomposes each of the two M blocks into an $\mathtt{M}_1$ block and an $\mathtt{M}_2$ block, doubling the throughput of the baseline with only a small increase in area for additional pipeline-stage registers. Combining superpipelining and reuse (Figure 5c) ideally uses half the area of the baseline design and provides the same throughput. The actual area is slightly more than ideal due to the multiplexers and extra registers. The actual throughput is slightly less than ideal due to the extra delay through the multiplexers. The behaviour of Figure 5c is shown in Figure 5e, where data parcels α, β, and γ proceed through the pipeline. In steady state, the $\mathtt{M}_1$ and $\mathtt{M}_2$ blocks are always busy (clock cycles 2–5 for $\mathtt{M}_1$ and 3–6 for $\mathtt{M}_2$). New data arrives every other clock cycle, because each data parcel uses the $\mathtt{M}_1$ block and an $\mathtt{M}_2$ blocks twice on its traversal through the pipeline. Figure 6 shows the implementation of WGperm over $\mathbb{F}_{2^{29}}$ with superpipelining and reuse.

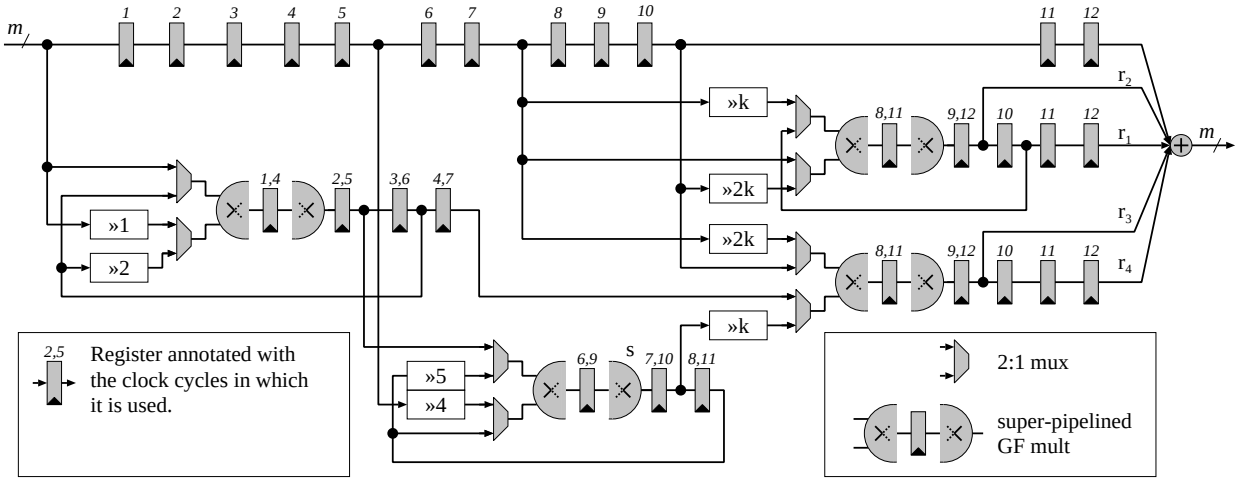


Figure 6: WG-permutation with superpipelining and reuse for $\mathbb{F}_{2^{29}}$

Table 2: A General Method to Load the Key and IV into the LFSR

$n = ml > 2K, e = \lceil \frac{2K}{3l} \rceil$	
$3e < m$	$S_{2j} = k_{3ej}, \cdots, k_{3ej+2e-1} \| IV_{3ej}, \cdots, IV_{3ej+e-1} \| 0 \cdots 0$ $S_{2j+1} = k_{3ej+2e}, \cdots, k_{3ej+3e-1} \| IV_{3ej+e}, \cdots, IV_{3ej+3e-1} \| 0 \cdots 0$ $j = 0, \cdots, \lfloor l/2 \rfloor$
$2e < m < 3e$	$S_{2j} = k_{mj}, \cdots, k_{mj+2e-1} \| IV_{mj}, \cdots, IV_{mj+m-2e-1}$ $S_{2j+1} = k_{mj+2e}, \cdots, k_{mj+m-1} \| IV_{mj+m-2e}, \cdots, IV_{mj+m-1}$ $j = 0, \cdots, \lfloor l/2 \rfloor$

Reuse allows the number of multipliers to be cut in half. However in an ASIC, the overall area of the WG-cipher is reduced by significantly less, because of the relatively large size of the LFSR. In contrast, reuse can be very helpful in an FPGA, because the total area is almost always constrained by the number of LUTs, not the number of flip-flops.

In an ASIC, reuse can be helpful when area is much more important than performance. For the WG over $\mathbb{F}_{2^{11}}$ with superpipelining, reuse reduced the area from 32.5 ksqum to 25.8 ksqum, a savings of 22%.[2] For the largest cipher we implemented ($\mathbb{F}_{2^{29}}$), reuse with superpipelining reduced the area from 280 ksqum to 180 ksqum, a savings of 35%.

3.4 Key Initialization and Re-synchronization

We generalize the key initialization scheme of a $(29, 11)$WG generator to a general case (m, l). Note that the key initialization for WG and MOWG are the same. We denote the state of the LFSR as $S_0, \cdots, S_{l-1}$ where $S_i \in \mathbb{F}_{2^m}$, a m-bit vector, secret key bits, $(k_0, \cdots, k_{K-1})$, and initial vector, $IV = (IV_0, \cdots, IV_{K-1})$ where $k_i, IV_i \in \mathbb{F}_2$ and K is the size of the secret key. The key and IV loading for the specific examples used in this paper are in the appendix. Once the LFSR has been loaded with the key and IV, the keystream generator is run for $2l+1+L$ clock cycles, after which the outputs of WGperm may be used as the keystream.

4 Results

This section summarizes implementations of the MOWG-cipher for $\mathbb{F}_{2^7}$, $\mathbb{F}_{2^{11}}$ and $\mathbb{F}_{2^{29}}$. The range of ciphers illustrate the different implementation options of random logic, ROM, pipelining, superpipelining, and reuse. We compare MOWG over $\mathbb{F}_{2^7}$ to the eSTREAM ciphers Grain and Trivium for an ASIC on an ST Microelectronics 90nm process and an Altera Stratix-II FPGA.

4.1 Summary of MOWG Cipher Implementations

The smallest MOWG cipher we implemented is for $\mathbb{F}_{2^7}$. We explored a variety of LFSRs with between 5 and 23 stages. This range offers a wide spectrum of possible levels of security, with corresponding increases in area and decreases in optimality as the security increases. For MOWG over $\mathbb{F}_{2^7}$, we implemented the design using random logic synthesized from the table of MOWG-permutation-transform values. For this small size field, random logic was smaller and faster than a ROM. With a 5-stage LFSR, the small amount of unique internal state (just the 35 variables in the LFSR) makes it relatively vulnerable to attack. The purpose of this

[2] "ksqum" is thousands of square microns.

example is show how the security of a MOWG cipher can be easily controlled and that increasing the size of the LFSR does not affect the performance of the circuit.

We do not have access to a memory compiler for the ST 90 nm process. The memory compiler we have is from ARM and supports a family of IBM processes between 250 and 130nm. MOWG over $\mathbb{F}_{2^{11}}$ is an interesting design point for an ASIC, because random logic, a ROM table, and GF multipliers are all reasonable options. With a 500MHz clock, the ROM-based design was the same area as the multiplier-based design. However, 500MHz was the fastest clock that the ROM could support, while the superpipelined multiplier design went up to 1.67GHz, where it achieved a much higher overall optimality.

For the ASIC implementation of $\mathbb{F}_{2^{29}}$, the critical path is in the LFSR from through the multiplier and XOR gates. For the LFSR, we explored both Galois and Fibonacci style feedback. Galois-style feedback was slightly faster and larger than Fibonacci, making the two roughly equivalent in terms of their affect on the performance/area ratio of the cipher.

The (29,11)WG stream cipher was carefully analyzed in the eSTREAM project. The revised version appeared in [26], which does not suffer the chosen IV attack in [33]. The linear space attack introduced in [30] and [33] needs $2^{45.0415}$ consecutive bits. Since the amount of keystream that can be generated with a single key/IV pair is strictly less than the required amount, this attack is not a threat to WG cipher designed according to given specifications. The $(29, 11)$ WG cipher has very strong security and can be employed in applications in scenarios which physical layer protection is needed. From the results in this paper, the (29,11,17)MOWG cipher preserves the essential security properties of the WG cipher.

4.2 Area and Performance Comparison Results

The stream ciphers Grain, MICKEY-128, and Trivium are the eSTREAM phase-3 finalists for hardware. In a variety of comparisons on both ASICs and FPGAs, Trivium-64 dominates in optimality as measured by the ratio of performance to area. Grain-16 consistently comes in second, with a relative optimality of approximately 0.20 compared to Trivium-64. The other ciphers are generally below 0.05 in relative optimality [1, 4, 5].

Our ASIC area and performance results are for an ST Microelectronics 90 nm CMOS cell library (CORX90GPHVT) with nominal delay values and using Synopsys Design Compiler for synthesis. The results are based on Design Compiler's estimate of area and clock speed prior to place-and-route. When comparing with ROM-based designs, we used an IBM 130nm process for both the multiplier based design and the table based designs. The ROM was created using Arm's ROM compiler. Our FPGA area and performance results are for Altera Stratix II series FPGA device EP2S15F484C using Mentor Graphics PrecisionRTL for logic synthesis and Altera Quartus for physical synthesis. All results are for post place-and-route. The source code of Trivium and Grain that we compare against were provided generously by Gabriel Southern, Son Nguyen, and Kris Gaj [6].

5 Conclusions

WG ciphers were designed to have many desirable randomness properties that can be verified mathematically. In this paper, we extend the previous analytical work on WG ciphers to multi-output WG ciphers. For an (m, l, d)MOWG-cipher with a K-bit secret key, we show that up to $d = m - l/K$ bits may be produced securely by the MOWG cipher without any cost in hardware or performance. We have implemented a variety of MOWG ciphers of different sizes and have explored the design space of ROM, random-logic, and GF-multiplier based implementations. For multiplier-based implementations, we show how pipelining, superpipelining, and reuse provide tradeoffs in area and performance. For $\mathbb{F}_{2^7}$, random logic is the smallest and fastest option. For $\mathbb{F}_{2^{11}}$, all three implementation styles achieve similar area and performance, but

Table 3: Area and performance results

	ST 90nm					Stratix II				
	Area	Speed	Perf	Opt	ROpt	Area	Speed	Perf	Opt	ROpt
Trivium-64	21.5	1.25	80.00	4.02	1.00	451	140	8960	19.87	1.00
Grain-16	21.8	1.00	16.00	0.73	0.18	490	192	3040	6.20	0.31
MOWG(7,5,3)	4.1	1.43	4.29	1.05	0.26	82	260	780	9.51	0.48
MOWG(7,11,3)	5.9	1.43	4.29	0.73	0.18	126	260	780	6.19	0.31
MOWG(7,23,3)	8.3	1.43	4.29	0.52	0.13	222	260	780	3.51	0.18
MOWG(11,15,6)	39.0	1.67	8.33	0.21	0.05	888	350	2100	2.36	0.12
MOWG(29,11,17)	187	1.0	8.50	0.05	0.01	4184	218	1853	0.44	0.02

- Area is measured as ASIC: kilo square microns, FPGA: cells.
- Clock speed is measured as ASIC: GHz, FPGA: MHz.
- Performance is speed $\times$ d bits/clock-cycle
- Opt (optimality) is Perf/Area, ROpt is optimality relative to Trivium-64

Table 4: Parameter values for MOWG implementations

m	l	Basis	L	Polynomials
7	5	poly	1	$g(x) = x^7 + x + 1$ $p(x) = x^5 + x^2 + \alpha$
7	11	poly	1	$g(x) = x^7 + x + 1$ $p(x) = x^{11} + x^2 + \gamma$, where$\gamma = 1 + \alpha^4 + \alpha^5$
7	23	poly	1	$g(x) = x^7 + x + 1$ $p(x) = x^{23} + x^2 + \alpha^5$, where $\alpha^5 = 0000010$
11	15	poly	10	$g(x) = x^{11} + x^{10} + x^8 + x^6 + x^2 + x + 1$ $p(x) = x^{15} + x^4 + \alpha^7$, where $\alpha^7 = 00000001000$
11	15	norm	10	$g(x) = x^{11} + x^8 + x^6 + x^2 + x + 1$ $p(x) = x^{15} + x^2 + x + \alpha^{352}$
29	11	norm	12	$g(x) = x^{29} + x^{28} + x^{24} + x^{21} + x^{20} + x^{19} + x^{18} + x^{17} + x^{14} + x^{12} + x^{11} + x^{10} + x^7 + x^6 + x^4 + x + 1$ $p(x) = x^{11} + x^{10} + x^9 + x^6 + x^3 + x + \gamma$ where $\gamma = \alpha^{464730077}$

superpipelined multipliers edged out the other options because of the higher clock speed that they provide. As the size of the finite fields increase, reuse becomes more effective, but we found that even with $\mathbb{F}_{2^{29}}$ superpipelining was slightly more optimal than superpipelining with reuse. MOWG ciphers have the highest optimality for small size finite fields. The additional security of the larger fields comes at the cost of rapid growth in area.

We compared our implementation of a MOWG cipher over $\mathbb{F}_{2^7}$ with a 23-stage LFSR and 3 bits of output to Trivium-64 and Grain-16. On both an ST 90nm cell library and a Stratix II FPGA, the MOWG cipher is about half the size of the other ciphers and allows a higher clock speed. Taking into account throughput in bits/clock-cycle, the MOWG cipher comes in third behind Trivium-64 and Grain-16, but well ahead of the other eSTREAM contenders. With their proven security properties and wide range of design

options for optimizing area, performance, and security, MOWG ciphers are an attractive option.

References

[1] M. Feldhofer. Comparison of low-power implementations of Trivium and Grain. *SASC 2007*

[2] Christophe De Cannière. Trivium: A stream cipher construction inspired by block cipher design principles. In *ISC*, pages 171–186, 2006.

[3] M Hell, T Johansson, W Meier. Grain: a stream cipher for constrained environments. *Int'l J. Wireless and Mobile Computing* (2)1. 2007

[4] P Bulens, K Kalach, FX Standaert, JJ Quisquater. FPGA implementations of eSTREAM phase-2 focus candidates with hardware profile. *SASC 2007*.

[5] D Hwang, M Chaney, S Karanam, N Ton, K Gaj. Comparison of fpga-targeted hardware implementations of estream stream cipher candidates. *SASC 2008*

[6] K Gaj, G Southern, R Bachimanchi. Comparison of hardware performance of selected Phase II eSTREAM candidates. *SASC 2007*

[7] L. Batina, J. Lano, N. Mentens, S. B. Ors, B. Preneel and I. Verbauwhede, Energy, Performance, area versus security trade-offs for stream ciphers, *SASC 2004*

[8] H. Dobbertin, Kasami power functions, permuation polynomials and cyclic difference sets, *Difference sets, sequences and their correlation properties (1998)*, pp. 133-158, NATO Adv. Sci. Inst. Ser. C Math. Phys. Sci., 542.

[9] F. Chabaud and S. Vaudenay. Links between Differential and Lin- ear Cryptanalysis, *EUROCRYPT94, Advances in Cryptology*, Lecture Notes in Computer Science, vol. 950, Springer Verlag, pp. 356-365, 1995.

[10] N. Courtois and W. Meier, Algebraic Attacks on Stream Ciphers with Linear Feedback, *Advances in Cryptology - Eurocrypt 2003*, LNCS 2656, pp. 346-359, Springer-Verlag, 2003.

[11] N. Courtois, Fast Algebraic Attacks on Stream Ciphers with Linear Feedback, *Advances in Cryptology-CRYPTO 2003*, LNCS 2729, pp. 176-194, Springer-Verlag, 2003.

[12] N. Courtois, Algebraic Attacks on Combiners with Memory and Several Outputs, *Cryptology ePrint Archive, Report 2003/125*, http://eprint.iacr.org/, 2003.

[13] N.Courtois and Pieprzyk J., Cryptanalysis of Block Ciphers with Overdefined Systems of Equations, *Advances in Cryptology-ASIACRYPT 2002*, volume LNCS 2501. Springer-Verlag, 2002.

[14] J. Dillon, and H. Dobbertin, New Cyclic Difference Sets with Singer Parameters, *Finite Fields and Their Application*, vol.10, No.3, pp. 342-389, July 2004.

[15] J. Dillon, Multiplicative Difference Sets via Additive Characters, *Designs, Codes and Cryptography*, Vol. 17, pp. 225-236, Sep. 1999,

[16] *EPCglobal Standards*, http://www.epcglobalinc.org/standards/.

[17] eSTREAM - The ECRYPT Stream Cipher Project, *http://www.ecrypt.eu.org/stream/*

[18] S. W. Golomb, and G. Gong, *Signal Design for Good Correlation: For Wireless Communication, Cryptography, and Radar*, Cambridge University Press, 2005.

[19] G. Gong, S. W. Golomb, and H.Y. Song, A note on low correlation zone signal sets, *IEEE Trans. on Information Theory*, Vol. 53, No. 7, July 2007, pp. 2575 - 2581.

[20] G. Gong, Theory and applications of q-ary interleaved sequences, *IEEE Trans. on Inform. Theory*, vol. 41, No. 2, March 1995, pp. 400-411.

[21] G. Gong, and A. Youssef, Cryptographic Properties of the Welch-Gong Transformation Sequence Generators, *IEEE Transactions on Information Theory*, vol. 48, No. 11, pp. 2837-2846, Nov. 2002.

[22] J.J. Hoch and A. Shamir, Fault Analysis of Stream Ciphers, *Cryptographic Hardware and Embedded Systems - CHES 2004*, Lecture Notes in Computer Science, vol. 3156, Springer, pp. 240-253, 2004.

[23] W. Meier, E. Pasalic, and C. Carlet, Algebraic Attacks and Decomposition of Boolean Functions, *Advances in Cryptology EUROCRYPT-2004*, LNCS 3027, pp.474-491, Springer-Verlag, 2004.

[24] R. Mullin, I. Onyszchuk, and S. Vanstone, Optimal Normal Bases in $\mathrm{GF}(\mathrm{p}^n)$, *Discrete Applied Mathematics*, vol. 22, pp. 149-161, 1989.

[25] Y. Nawaz and G. Gong, The WG Stream Cipher, *eSTREAM: The ECRYPT Stream Cipher Project*, Report 2005/033, Available at http://www.ecrypt.eu.org/stream/wg.html

[26] Y. Nawaz and G. Gong, WG: A family of stream ciphers with designed randomness properties, *Information Sciences*, ELSEVIER, Vol. 178, No. 7, April 1, 2008, pp. 1903-1916.

[27] J. S. No, S. W. Golomb, G. Gong, H. K. Lee, and P. Gaal, Binary Pseudorandom Sequences of period $2^n - 1$ with Ideal Correlation Properties, *IEEE Transactions on Information Theory*, Vol. 44, No. 2, pp. 814-817, March 1998.

[28] K. Nyberg. On the construction of highly nonlinear permutations, *EUROCRYPT 92, Advances in Cryptology*, Springer Verlag, Lecture Notes in Computer Science 658, pp. 92-98, 1993.

[29] S. Ronjom and T. Helleseth, Attacking the Filter Generator over $GF(2^m)$, *eSTREAM: The ECRYPT Stream Cipher Project*, Report 2007/011, Available at http://www.ecrypt.eu.org/stream/papersdir/2007/011.pdf

[30] S. Rønjom, G. Gong and T. Helleseth, On attacks on filtering generators using linear subspace structures, *Sequences, Subsequences, and Consequences, Lecture Notes in Computer Science*, S.W. Golomb *et al.* (Eds.), Vol. 4893, pp. 204-217, Springer-Verlag, 2007.

[31] M.K. Simon, J. K. Omura (Author), R.A. Scholtz, B.K. Levitt, *Spread Spectrum Communications Handbook*, Mcgraw-Hill (Tx), 1094.

[32] B. Sunar, and C. Koc, An Efficient Optimal Normal Basis Type II Multiplier, *IEEE Transactions on Computers*, vol. 50, No. 1, pp. 83-88, Jan. 2001.

[33] H. Wu, and B. Preneel, Resynchronization Attacks on WG and LEX, *Fast Software Encryption 2006*, LNCS 4047, pp. 422-432, Springer-Verlag 2006.

Appendix

A. Proof of Proposition 1

Let $\{\alpha_0, \cdots, \alpha_{m-1}\}$ be a basis of $\mathbb{F}_{2^m}$ over $\mathbb{F}_2$ and $\{\beta_0, \cdots, \beta_{m-1}\}$ be its dual basis, i.e.,

$$Tr(\alpha_i \beta_j) = \begin{cases} 1 & \text{if } i = j \\ 0 & \text{otherwise.} \end{cases}$$

Since $F(x)$ is an element of $\mathbb{F}_{2^m}$, then $F(x)$ can be represented by

$$F(x) = \sum_{i=0}^{m-1} f_i(x)\alpha_i, f_i(x) \in \mathbb{F}_2 \leftrightarrow F(x) = (f_0(x), \cdots, f_{m-1}(x)) \tag{2}$$

where the component functions are given by

$$f_i(x) = Tr(\beta_i F(x)), 0 \leq i < m. \tag{3}$$

Thus $F(x)$ is associated with a binary vector $(f_0(x), \cdots, f_{m-1}(x))$, and each component is a polynomial function mapping from $\mathbb{F}_{2^m}$ to $\mathbb{F}_2$, which has a natural correspondence with a boolean function in m variables. If there is i with $0 \leq i < m$ such that $\beta_i = 1$, then $f_i(x) = WG(x)$. Note that $\beta_i \neq 1$ for all i,

$i = 0, \cdots, m-1$ when $\{\alpha_0, \cdots, \alpha_{m-1}\}$ is a normal basis. Thus, in the case of the normal bases, none of the component function of $F(x)$ is equal to the WG transformation.

Let $\mathbf{b} = \{b_i\}$ be a sequence generated by the LFSR, and let

$$U(x) = F \circ Tr_m^n(x)) \leftrightarrow U(x) = (u_0(x), \cdots, u_{m-1}(x))$$

where $\circ$ means the composition of two functions and $u_j(x)$ is a function from $\mathbb{F}_{2^n}$ to $\mathbb{F}_2$ given by

$$u_j(x) = f_j(Tr_m^n(x)), j = 0, \cdots, m-1 \tag{4}$$

where f_j is given by (3). Let $\mathbf{a} = \{a_i\}$ whose elements are given by $a_i = F(b_i), i = 0, 1, \cdots$. Thus we can write $a_i = (a_{i,0}, a_{i,1}, \cdots, a_{i,m-1}), i = 0, 1, \cdots$. Let $\mathbf{a}_j = \{a_{i,j}\}$. Then $\mathbf{a}_j$'s are component sequences generated by the WG permutation transform. From (2) and (4), we have

$$a_{i,j} \quad = \quad f_j(b_i), j = 0, \cdots, m-1, i = 0, 1, \cdots$$

and $u_j(x)$ is the trace representation of the component sequence. For each subset of d component sequences $\mathbf{a}_{j_1}, \mathbf{a}_{j_2}, \cdots, \mathbf{a}_{j_d}$ with $0 \le j_1 < j_2 < \cdots < j_d \le m-1$ and $d < m$, we concatenate each d bits from those d component sequences, and the resulting sequence is the output of (m, l, d) MOWG generator, denoted by $\mathbf{u} = \{u_i\}$.

Note that U is referred to as an *interleaved sequence* in [20], and the component sequences $\mathbf{a}_j$'s are sequences with subfield factorization in [18]. Thus, the linear complexities and autocorrelation functions of the component sequences can be determined from the theory of subfield factorization sequences in [18].

Lemma 1 *With $\mathbf{a}_j, j = 0, \cdots, m-1$ defined above, then*

1. *The linear complexity of all component sequences $\mathbf{a}_j, j = 0, \cdots, m-1$ are equal, which is equal to the linear complexity of the WG key stream sequence $\mathbf{s}$, i.e., $LC_{\mathbf{a}_j} = LC_{\mathbf{s}}, j = 0, \cdots, m-1$.*
2. *The cross correlation of any two component sequences or the out-of-phase autocorrelation of each component sequence takes value -1 at all shifts except for those $\tau = kd$ where $d = \frac{2^n-1}{2^m-1}$ and $k = 1, \cdots, 2^m - 2$.*

Proof. Note that β_j is constant. Thus, the exponents of x in $Tr(\beta_j F(x))$ are the same as those in $Tr(F(x))$. According to Theorem 3.17 in [18], the first assertion is true. For each j, $Tr(\beta_j F(x))$ is balanced (i.e., $Tr(\beta_j F(x)) = c, c \in \mathbb{F}_2$ has 2^{m-1} solutions in $\mathbb{F}_{2^m}$). According to Theorem 1 in [19], the second assertion follows. □

Using a similar technique for interleaved sequences used in [20] and Lemma 1, the assertions of Proposition 1 is established.

B. Key Initializations for (7, 23) and (11,15) WG Generators

Table 5: $(m, l) = (7, 23), K = 80, e = 3$

$S_0 = k_0, \cdots, k_5 \| IV_0,$	$S_1 = k_6 \| IV_1, \cdots, IV_6$
$S_2 = k_7, \cdots, k_{12} \| IV_7,$	$S_3 = k_{13} \| IV_8, \cdots, IV_{13}$
$\vdots$	
$S_{20} = k_{70}, \cdots, k_{75} \| IV_{70},$	$S_{21} = k_{76} \| IV_{71}, \cdots, IV_{76}$
$S_{22} = (k_{77}, k_{78}, k_{79}, IV_{77}, IV_{78}, IV_{79}, 0)$	

Table 6: $(m, l) = (11, 15), K = 80, e = 4$

$S_0 = k_0, \cdots, k_7 \|\| IV_0, IV_1, IV_2,$	$S_1 = k_8, K_9, k_{10} \|\| IV_3, \cdots, IV_{10}$
$S_2 = k_{11}, \cdots, k_{18} \|\| IV_{11}, IV_{12}, IV_{13},$	$S_3 = k_{19}, k_{20}, k_{21} \|\| IV_{14}, \cdots, IV_{21}$
$S_4 = k_{22}, \cdots, k_{29} \|\| IV_{22}, IV_{23}, IV_{24},$	$S_5 = k_{30}, k_{31}, k_{32} \|\| IV_{25}, \cdots, IV_{32}$
$S_6 = k_{33}, \cdots, k_{40} \|\| IV_{33}, IV_{34}, IV_{35},$	$S_7 = k_{41}, k_{42}, k_{43} \|\| IV_{36}, \cdots, IV_{43}$
$S_8 = k_{44}, \cdots, k_{51} \|\| IV_{44}, IV_{45}, IV_{46},$	$S_9 = k_{52}, k_{53}, k_{54} \|\| IV_{47}, \cdots, IV_{54}$
$S_{10} = k_{55}, \cdots, k_{62} \|\| IV_{55}, IV_{56}, IV_{57},$	$S_{11} = k_{63}, k_{64}, k_{65} \|\| IV_{58}, \cdots, IV_{65}$
$S_{12} = k_{66}, \cdots, k_{73} \|\| IV_{66}, IV_{67}, IV_{68},$	$S_{13} = k_{74}, k_{75}, k_{76} \|\| IV_{69}, \cdots, IV_{76}$
$S_{14} = k_{77}, k_{78}, k_{79} \|\| IV_{77}, IV_{78}, IV_{79} \|\| 00000$	