On Parallelizing the *CryptMT* Stream Cipher

Deian Stefan, David B. Nummey, Jared Harwayne-Gidansky and Ishaan L. Dalal
S*ProCom[2] // Dept. of Electrical Engineering,
The Cooper Union
New York, NY 10003
{stefan, nummey, harway, ishaan}@cooper.edu

***Abstract*—Fast stream ciphers are used extensively for encrypted data transmission in mobile networks and over multi-gigabit links. *CryptMT*, a recently proposed stream cipher, is one of the final candidates for standardization by the European Union's eSTREAM project. Cryptanalysis of *CryptMT* has discovered no feasible attacks thus far.**

We present a scalable technique for parallelizing *CryptMT* and present an area-efficient hardware implementation on a field-programmable gate array (FPGA). On the Xilinx Virtex-2 Pro FPGA, a 2× parallelization delivers throughputs of up to 16 Gbits/s while using minimal logic resources (1,782 slices). This is highly area-efficient compared to implementations of ciphers such as AES. Possibilities for higher degrees of parallelization are also discussed.

I. Introduction

Ciphers are cryptographic algorithms that encrypt *plaintext* data into *ciphertext*. There are two main classes of ciphers [1]: *stream ciphers*, such as RC4 and *CryptMT* [2], use a time-varying transformation to encrypt individual bits of plaintext; *block ciphers*, such as AES and DES, operate with a fixed transformation on large blocks of plaintext data. In general, stream ciphers have lower complexity and higher throughput than block ciphers. Due to these attractive properties, stream ciphers are widely used in embedded/mobile applications such GSM telephony (the A5 family of stream ciphers [3]), the 802.11 Wireless Encryption Protocol (WEP) [4] and Secure Sockets Layer (SSL) [5]–both use the RC4 stream cipher and the Bluetooth protocol (E0 stream cipher) [6].

In 2004, the European Union sponsored the *eSTREAM*[1] project to identify new stream ciphers suitable for widespread adoption. *eSTREAM* targets two different stream cipher profiles: 1) high performance (throughput) ciphers for software implementation and 2) hardware-oriented ciphers optimized for power and/or resource efficiency [7]. In April 2007, eSTREAM entered the third and final evaluation phase with *CryptMT* being one of the eight final software-profile candidates.

We begin with the notation and definitions necessary to understand *CryptMT*'s mathematical foundations. Then, we discuss *CryptMT*'s core modules: its pseudo-random number generator (PRNG) and the non-linear transformation. The architectural optimizations that allow efficient hardware implementations of this software-centric stream cipher are presented next, along with benchmarks of two field-programmable gate array (FPGA) based implementations. Finally, we elaborate upon a massive parallelization technique that allows CryptMT to scale to even higher throughputs.

[1] *http://www.ecrypt.eu.org/stream*

II. Structure of *CryptMT*

A stream cipher generates (encrypted) ciphertext by XOR'ing a *keystream* with the plaintext message. *CryptMT* is a *symmetric* stream cipher—both encryption and decryption utilize the same keystream generator; this generator must be initialized with a secret (mutually shared) *key* as well as an *initial vector* (IV) that need not be secret. The key and IV lengths can be chosen to vary from 128- to 2048-bits; larger key-lengths imply a higher resistance to brute-force (i.e., exhaustive key search) attacks. Like other stream ciphers, *CryptMT*'s keystream sequence is generated by a non-linear transformation of the output of a pseudo-random number generator (PRNG) [2].

A. Notation and word-wise operations

We follow the notation of [2], where the 128-bit word $\mathbf{x}$ is formed by concatenating four 32-bit words $\mathbf{x}[3]$, $\mathbf{x}[2]$, $\mathbf{x}[1]$ and $\mathbf{x}[0]$, i.e.,

$$\mathbf{x} = \mathbf{x}[3]\mathbf{x}[2]\mathbf{x}[1]\mathbf{x}[0]$$

- *Word-wise operations:* Operations with a '32' or '64' subscript operate on one 32-bit or 64-bit words at a time, respectively; after the operation is completed, these words are concatenated to form the 128-bit word. For example, addition is defined as

$$\mathbf{x} +_{32} \mathbf{y} = (\mathbf{x}[3]+\mathbf{y}[3])(\mathbf{x}[2]+\mathbf{y}[2])(\mathbf{x}[1]+\mathbf{y}[1])(\mathbf{x}[0]+\mathbf{y}[0]),$$

  where the 32-bit words are added *modulo* 2^{32} (no carries).
- *Word-wise shifts:* Similarly, word-wise shifts of $\mathbf{x}$ by S bits are defined as

$$\mathbf{x} \gg_{32} S = (\mathbf{x}[3] \gg S)(\mathbf{x}[2] \gg S)(\mathbf{x}[1] \gg S)(\mathbf{x}[0] \gg S)$$
$$\mathbf{x} \gg_{64} S = (\mathbf{x}[3]\mathbf{x}[2] \gg S)(\mathbf{x}[1]\mathbf{x}[0] \gg S)$$

- *Permutations:* We also define a *permutation* $\mathrm{perm}(\mathbf{x}, order)$ that re-orders the 32-bit words of $\mathbf{x}$ as *order*, e.g.,

$$\mathrm{perm}(\mathbf{x},\ 0132) = \mathbf{x}[0]\mathbf{x}[1]\mathbf{x}[3]\mathbf{x}[2] \quad (1)$$

TABLE I
WORD-WISE OPERATION BLOCKS IN *CryptMT*

Block	Operation	Example
`perm-1`	$\text{perm}(\mathbf{x}, 0321)$	$\mathbf{x}[0]\mathbf{x}[3]\mathbf{x}[2]\mathbf{x}[1]$
`pshift-1`	$\text{pshift}_{64}(\mathbf{x}, 2031, 3)$	$\mathbf{x}[2]\mathbf{x}[0]\mathbf{x}[3]\mathbf{x}[1] \oplus (\mathbf{x} \gg_{64} 3)$
`pshift-2`	$\text{pshift}_{32}(\mathbf{x}, 0321, 1)$	$\mathbf{x}[0]\mathbf{x}[3]\mathbf{x}[2]\mathbf{x}[1] \oplus (\mathbf{x} \gg_{32} 1)$
`pshift-3`	$\text{pshift}_{32}(\mathbf{x}, 3210, 16)$	$\mathbf{x}[3]\mathbf{x}[2]\mathbf{x}[1]\mathbf{x}[0] \oplus (\mathbf{x} \gg_{32} 16)$

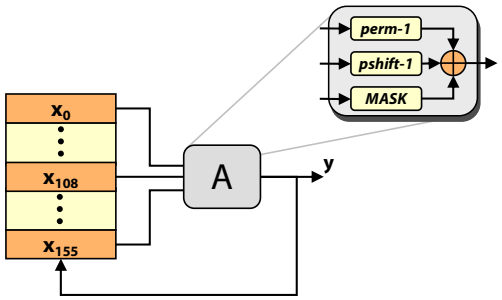

Fig. 1. Linear Pseudo-random Number Generator in *CryptMT*

- *Perm-shifts:* Permutations and w-bit word-wise shifts of S-bits are combined (by an XOR) to form the *perm-shift* operation $\text{pshift}_w(\mathbf{x}, order, S)$; pshifts are crucial to *CryptMT*'s structure. An example of a pshift is:

$$\text{pshift}_w(\mathbf{x},\ 0132,\ S) = \mathbf{x}[0]\mathbf{x}[1]\mathbf{x}[3]\mathbf{x}[2] \oplus (\mathbf{x} \gg_w S). \quad (2)$$

  CryptMT involves four kinds of pshifts and perms whose parameters are defined in Table I.

B. Linear Pseudo-random Number Generators

CryptMT's keystream generator combines a linear pseudo-random number generator (PRNG) with a non-linear multiplicative filter. The PRNG, called the *Simple Fast Mersenne Twister* (SFMT), is a type of generalized feedback shift-register (GFSR) from the *Mersenne Twister* (MT) long-period PRNG family [8]. SFMT has a huge period of $2^{19937} - 1$, operates on 128-bit words and has very good statistical characteristics.

SFMT consists of a 19,937-bit state vector (i.e., shift register) $\mathbf{x}$, stored as $N = 156$ 128-bit words, and the recurrence equation to generate a new word $\mathbf{x}_{N+j}$ is defined as:

$$\mathbf{x}_{N+j} = (\mathbf{x}_{N+j-1}\ \&\ \texttt{MASK}) \oplus \text{pshift}_{64}(\mathbf{x}_{M+j}, 2031, 3) \oplus \text{perm}(\mathbf{x}_j, 0321), \quad (3)$$

where '&' is the bitwise AND operator, $M = 108$ is the "far recurrence" offset, perm and pshift are as defined in (1) and (2), and the MASK[2] is 128 bits. Fig. 1 illustrates the state vector and the generation of one PRNG output word, $\mathbf{y}$; Table I lists the operations performed by the `perm-1` and `pshift-1` blocks.

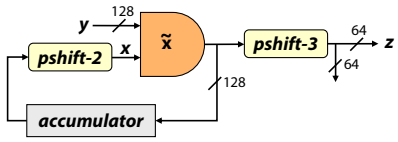

Fig. 2. Non-linear Multiplicative Filter

C. Non-linear Multiplicative Filter

If an attacker obtains access to a sequence of the linear PRNG's output that is at least twice the state size (i.e., $2 \times 19,937 \simeq 40$ kbits), they can use it to reveal the internal state of the linear feedback shift register using polynomial factorization algorithms such as Berlekamp-Massey [1], [9]. The cipher is then compromised since all future outputs can be predicted from that internal state. To prevent this, a non-linear transformation consisting of a "multiplicative filter" is applied to the PRNG output.

The multiplicative filter (Fig. 2) consists of an integer multiplier $\widetilde{\times}$ and a one-word accumulator; it takes two 128-bit word inputs: the PRNG output $\mathbf{y}$ and the perm-shifted feedback from the accumulator $\mathbf{x}$. The multiplication $\widetilde{\times}_{32}$ operates individually on the four 32-bit words $\mathbf{x}[i]$ and $\mathbf{y}[i]$ $(i = 0, \ldots, 3)$, as:

$$\mathbf{x}[i] \widetilde{\times}_{32} \mathbf{y}[i] = (2\mathbf{x}[i]\mathbf{y}[i] + \mathbf{x}[i] + \mathbf{y}[i]) \bmod 2^{32}, \quad (4)$$

which is essentially 33-bit (odd) integer multiplication [2]. The perm-shift (`pshift-2`) at the accumulator output further mixes the filter's internal memory as well as the upper bits with lower bits. Finally, the `pshift-3` block permutes the output of (4), after which the 16 most significant bits of each 32-bit words are dropped and the 16 least significant bits are used as the keystream output $\mathbf{z}$.

III. PARALLELIZING *CryptMT*

FPGAs consist of reprogrammable logic blocks (called *slices* or *logic elements*) and static block RAMs. An FPGA implementation of *CryptMT* can be optimized for throughput and/or area-efficiency by exploiting the algorithm as well as FPGA architecture.

A. Architectural Optimizations for Area-Efficiency

As shown in Fig. 1, the linear PRNG requires four 128-bit I/O operations (ops) per word output: 3 reads and 1 write (feedback). Contemporary FPGA block RAMs are dual ported, allowing two independent I/O ops per clock. One read is saved by buffering the PRNG output/feedback word and re-using it for the next output. Additionally, using the 'read-before-write' operational mode (if available) saves another I/O op by multiplexing a read/write on the same clock cycle; writing new data to an address on an input port simultaneously outputs (reads) the existing data at that address. This brings the total down to two I/O ops, allowing the PRNG to generate

[2] `MASK = ffdfafdf f5dabff ffdbffff ef7bffff`

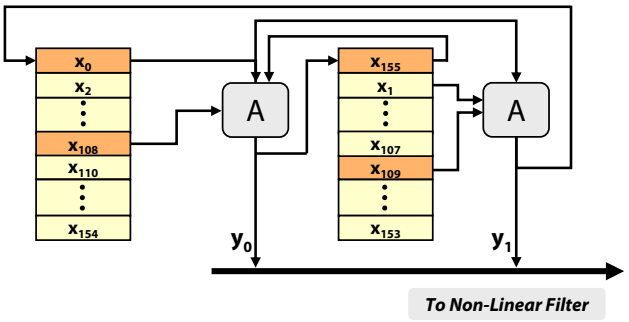

Fig. 3. SFMT 2-way Interleaved Parallelization

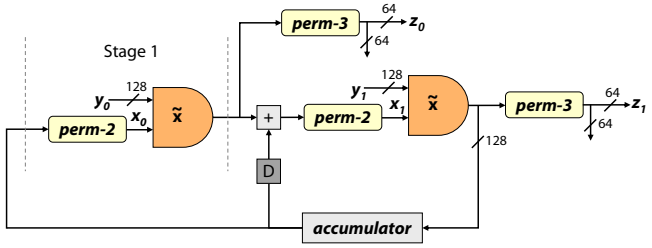

Fig. 4. Parallelized Non-linear Multiplicative Filter

one 128-bit output per clock while using only a single dual-ported block RAM. Finally, logic resources can be conserved by constructing the multiplicative filter from the embedded multipliers or "DSP blocks" available on newer FPGAs.

B. Parallelization: Scaling to Higher Throughputs

In parallelizing *CryptMT* for higher throughputs, the design must meet a very strict constraint: the parallelization methodology employed *must not* change any characteristics of the output stream, thus maintaining security (i.e., the algorithm is not modified) and backwards compatibility (e.g., a non-parallelized 'decrypter' can successfully work with the output of a parallelized 'encrypter', or vice versa). Meeting this constraint is challenging since *CryptMT* consists of *both* linear and non-linear components.

1) Interleave-Parallelizing the PRNG: Parallelization methodologies for the long-period class of PRNGs (including the Mersenne Twister) with various degrees of optimization exist [10], [11]. The *interleave-parallelization* (IP) approach of [10], in particular, already includes the area-efficient optimizations of Section III-A; here we extend it to SFMT.

Interleave-parallelization exploits the fact that SFMT's recurrences are at *constant offsets* (modulo N); thus, the N-word state-vector can be interleaved across multiple memory banks with static inter-bank connections. Multiple words can now be generated in one clock cycle by duplicating the circuitry that computes the PRNG's recurrence equation at the outputs of these banks.

Figure 3 illustrates 2-way interleave-parallelization (2-IP) for SFMT, where $N = 156$ and the recurrence offset $M = 108$. Outputs y_j depend on $\mathbf{x}[j-1]$, $\mathbf{x}[j]$ and $\mathbf{x}[j+M]$ (all modulo N, i.e., offsets wrap around); the outputs shown are computed in parallel as functions of:

$$y_0 \leftarrow f(\mathbf{x}[155],\ \mathbf{x}[0],\ \mathbf{x}[108])$$
$$y_1 \leftarrow f(\mathbf{x}[0] = y_0,\ \mathbf{x}[1],\ \mathbf{x}[109])$$

y_0 is buffered and need not be re-read to compute y_1.

2) Observations on Parallelizing the PRNG: The number of IP-banks n must be a factor of (i.e., evenly divide) N; equivalently, all n banks must have the same size. If this is *not* the case, the inter-bank connections are no longer static and IP becomes inefficient because of conditional routing overhead such as extra multiplexers, longer critical paths, etc. Thus, n-way parallelization of SFMT is possible where $\{n : N \bmod n = 0\}$.

We also note that that our original constraint of not modifying the output stream, and thereby maintaining backwards compatibility is met; an n_j-IP encrypter can be interfaced to an n_k-IP decrypter ($n_j \neq n_k$) by simply adding, depending on whether the intermediate channel is serial or parallel, the appropriate de-interleaver/re-interleaver at each end.

While other methods for parallelizing long-period PRNGs exist [10] that may be more efficient than IP, they invariably permute the words in the output stream. This is inconsequential if the output is being used for non-cryptographic applications such as simulations, but would violate our constraint and be non-backwards-compatible; we do not consider such methods.

3) Parallelizing the Non-linear Filter: As explained in Section II-C, the PRNG's output is transformed by the non-linear multiplicative filter (Fig. 2). The filter contains an accumulator (i.e., it has memory), and therefore cannot simply be duplicated like the recurrence circuitry if backwards-compatibility is to be maintained. Instead, an n-IP filter is implemented as a cascade of $n-1$ modified filter stages that contain delays and adders (*not* accumulators) with one standard filter stage. The delay-adder combinations act as effective accumulators—the memory is provided by the single accumulator in the first (standard) stage.

Figure 4 shows the structure of a 2-way parallelized multiplicative filter with two 128-bit inputs that produces two 64-bit ciphertext outputs per clock cycle. From a timing perspective, it is crucial to balance the delays used to pipeline the filter for higher speed with the delays required in the cascade to correctly compute the n-outputs.

IV. Hardware Implementation

We implemented non-parallelized and 2-way parallelized versions of CryptMT on the Xilinx *Virtex-II Pro* family of FPGAs. Here, we discuss the framework used to enable real-world deployment of CryptMT and address the performance of our implementations.

A. The CryptMT FPGA Framework

CryptMT must be initialized with a smaller PRNG called the "booter", which also generates the first $N \times 64$ ciphertext bits.

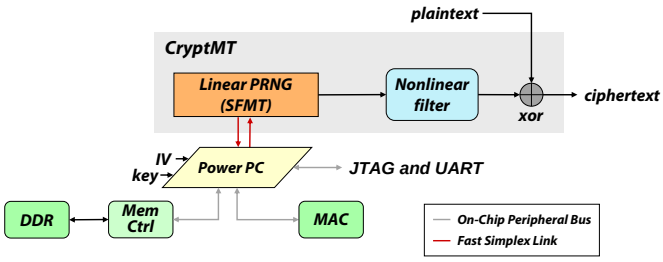

Fig. 5. Structure of the *CryptMT* Stream Cipher

Additionally, a CryptMT hardware peripheral on an FPGA must be able to communicate with the plaintext source and ciphertext destination (over TCP/IP, USB, etc.). To co-ordinate these tasks, we create a framework around a microprocessor that is embedded in the FPGA (a 'soft' processor constructed from FPGA logic can also be used). This processor, a *PowerPC 405* in our case, connects to the CryptMT peripheral via a point-to-point *Fast Simplex Link* (FSL) bus. To save logic, the one-time "booter" is executed in software on the PowerPC, which also transfers data (plaintext, ciphertext, initial state and control information) to/from the CryptMT peripheral.

Additionally, the PowerPC interfaces to the following on-chip peripherals via the Processor Local Bus (PLB) and On-chip Peripheral Bus (OPB):

1) A 256 MB Double Data Rate (DDR) SDRAM module (through a logic-based controller), providing additional memory for storing test run data.
2) An RS232 UART to communicate debug and control information to and from a PC.
3) A 10/100 Fast Ethernet MAC (and associated PHY link) for data transfer.
4) A USB-JTAG connection for debugging the PowerPC and transferring data.

B. Implementation

We implemented the standard *CryptMT* as well as a 2-way parallelized version in VHDL on the Xilinx *Virtex-II Pro* XC2VP30 FPGA that consists of 13,696 slices, 136 embedded multipliers and 136 18-kbit dual-ported block RAMs. Table II presents the resource usage and performance characteristics of both implementations. The standard version can be clocked up to 200.3 MHz (12.821 Gbits/s throughput) and the 2-parallel version up to 126.4 MHz (16.18 Gbits/s). As can be seen, the resource usage scales linearly when going to 2-way parallelization; the throughput, on the other hand, does not. We believe this discrepancy can be resolved and linear scaling for throughput attained by exploiting the denser packing and other advanced features on newer FPGA devices such as the Virtex-4 and 5.

V. Discussion

A. Comparison with other Stream Ciphers

CryptMT's area-efficiency (throughput/slice) is an order of magnitude better than recent fast FPGA implementations of AES [12], [13] and RC4 [14] that also used the Virtex-II family FPGAs; Table II includes a detailed comparison.

Table II also summarizes three recent eSTREAM Phase 3 hardware-oriented ciphers that were implemented on the Xilinx Spartan 3 FPGAs [15]. CryptMT started out as a software-oriented cipher and its implementations naturally have higher resource usage than some of the hardware-oriented eSTREAM candidates; however, as noted in Table II, CryptMT provides much larger keylengths and longer periods (i.e., much better resistance to attacks) than the hardware ciphers. For the hardware designer considering eSTREAM candidates, which cipher to use is a trade-off based on the amount of resources, speed and the security desired.

B. Cryptanalysis of CryptMT

The eSTREAM candidates have been subjected to external cryptanalysis; neither these nor the *CryptMT* authors' efforts have revealed any feasible attacks on the cipher thus far. Some cryptanalytic issues that were addressed include the long key length (128 to 2048-bits) which provides resilience against brute-force attacks, with the choice of keyspace size dependent on the user's needs; the extremely long internal state space which provides security against *time-memory-tradeoff (TMTO) attacks* [19]; and the 155-dimensional equidistribution of the PRNG and non-linear multiplicative filter which invalidate correlation attacks[20].

C. Massive Parallelization with Multiple CryptMT units

Even higher throughputs with similar area-efficiency can be achieved by interleaving the plaintext input across multiple *CryptMT* units in parallel ("massive parallelization"). This is different from the *internal* parallelization of CryptMT unit we have discussed so far; each unit in a massively parallel system can, of course, be internally parallelized too.

For such a massively parallel scheme to be secure, the outputs from the multiple PRNGs must be uncorrelated given just one common key/IV pair for initialization. A guaranteed way to achieve this is to have the PRNGs generate *non-overlapping sub-sequences* of the original long-period sequence.

Such multiple PRNGs must be initialized with successive states corresponding to the sub-sequence interval points, e.g. 2^{1000}, 2^{2000}, etc. Running a PRNG for 2^{1000} outputs to find these starting points is not feasible. However, a new technique for 'fast jump ahead' in long-period linear PRNGs (such as *CryptMT*'s) has recently been proposed [21]. It exploits polynomial arithmetic to calculate the jumps for the sub-sequences in 5-15 ms on current Intel/AMD CPUs. A proof-of-concept hardware demonstration of this technique exists [10], and could be ported to the CryptMT framework to enable massive parallelization.

VI. Conclusion

We have presented the first FPGA-based hardware implementation of the *CryptMT* stream cipher in both parallelized and non-parallelized forms; these implementations utilized several algorithmic and architectural optimizations and are

TABLE II
CRYPTMT RESOURCE USAGE, PERFORMANCE AND COMPARISON

Cipher	Slices (%)	Block RAMs (%)	Throughput (Gbits/s)	Mbps / Slice	Max Period (Bits)	Key Length (Bits)
CryptMT (standard)	878 (6.43%)	4 (2.94%)	12.821	14.60	$2^{19937}-1$	128-2048
CryptMT (2-IP)	1782 (13.01%)	8 (5.88%)	16.179	9.08	$2^{19937}-1$	128-2048
(eSTREAM HW ciphers)						
Trivium (×64) [16]	344 (9.60%)	—	13.504	39.26	$2^{64}-1$	80
Grain 128 [17]	50 (6.51%)	—	0.196	3.92	$2^{256}-1$	128
F-FCSR-16 [18]	471 (61.72%)	—	2.144	4.53	$2^{128}-1$	128
(other implementations)						
AES [12]	5177 (37.8%)	84 (61.7%)	21.54	4.16	—	128
AES [13]	422 (3.03%)	37 (27.2%)	1.30	3.15	—	128
RC4 [14]	138 (8.98%)	3 (2.20%)	0.12	0.87	N/A	8-128

substantially more area-efficient than previous FPGA implementations of ciphers such as AES. We also elaborated upon a methodology for parallelizing *CryptMT* that increases throughput while maintaining area-efficiency. Future work includes further optimization of *CryptMT* on FPGAs as well as a massively parallelized system of CryptMT units on a single FPGA that can encrypt/decrypt multiple Gigabit Ethernet links at line speeds.

ACKNOWLEDGMENTS

The authors would like to thank the anonymous reviewers for their comments and suggestions. They appreciate Prof. Om Agrawal's guidance on the mathematical aspects of this work, as well as the S*ProCom^2 research center and Prof. Fred L. Fontaine for providing the infrastructure for FPGA development.

REFERENCES

[1] A. Menezes, P. van Oorschot, and S. Vanstone, *Handbook of Applied Cryptography*. CRC Press, 1996.

[2] M. Matsumoto, M. Saito, T. Nishimura, and M. Hagita, "CryptMT stream cipher version 3," eSTREAM, ECRYPT Stream Cipher Project, Report 2005/001, 2007.

[3] T. Pionteck *et al.*, "Design of a reconfigurable AES encryption/decryption engine for mobile terminals," in *Proc. Int. Symp. Circuits and Systems ISCAS 2004*, vol. 2, May 2004, pp. 545–8.

[4] *Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications*, IEEE Std., 1999.

[5] *The SSL Protocol, Version 3*, Netscape Communications Corporation Std., 1996.

[6] *Wireless Medium Access Control (MAC) and Physical Layer (PHY) Specifications for Wireless Personal Area Networks (WPANs)*, IEEE Std., 2002.

[7] (2007, December) eSTREAM phase 3 technical background. ECRYPT - European Network of Excellence for Cryptology. [Online]. Available: http://www.ecrypt.eu.org/stream/technical.html

[8] M. Matsumoto and T. Nishimura, "Mersenne Twister: a 623-dimensionally equidistributed uniform pseudo-random number generator," *ACM Trans. Modeling and Comp. Simulation*, vol. 8, no. 1, pp. 3–30, Jan. 1998.

[9] H. C. van Tilborg, Ed., *Encyclopedia of Cryptography and Security*. Springer, 2005.

[10] I. L. Dalal and D. Stefan, "A hardware framework for the fast generation of multiple long-period random number streams," in *International Symposium on Field Programmable Gate Arrays*, February 2008.

[11] S. Konuma and S. Ichikawa, "Design and evaluation of hardware pseudo-random number generator mt19937," *IEICE Trans. on Info. and Systems*, vol. E88-12, no. 12, pp. 2876–2879, 2005.

[12] A. Hodjat and I. Verbauwhede, "A 21.54 Gbits/s fully pipelined AES processor on FPGA," in *12th Annual IEEE Symp. on Field-Programmable Custom Computing Machines FCCM '04*, Apr. 2004, pp. 308–309.

[13] D. Denning *et al.*, "An implementation of a gigabit ethernet AES encryption engine for application processing in SDR," in *IEEE 60th Vehicular Technology Conf. 2004-Fall*, vol. 3, Sep. 2004, pp. 1963–1967.

[14] P. Kitsos *et al.*, "Hardware implementation of the RC4 stream cipher," in *46th IEEE Intl. Midwest Symp. on Circ. and Sys. MWSCAS 2003*, vol. 3, Dec. 2003, pp. 1363–1366.

[15] D. Hwang *et al.*, "Comparison of fpga-targeted hardware implementations of estream stream cipher candidates," in *The State of the Art of Stream Ciphers*, 2008, pp. 151–162.

[16] C. D. Cannière and B. Preneel, "Trivium specifications," eSTREAM, the ECRYPT Stream Cipher Project, Tech. Rep., 2007. [Online]. Available: http://www.ecrypt.eu.org/stream/p3ciphers/trivium/trivium_p3.pdf

[17] T. J. Martin Hell and W. Meier, "Grain - a stream cipher for constrained environments," eSTREAM, the ECRYPT Stream Cipher Project, Tech. Rep., 2007. [Online]. Available: http://www.ecrypt.eu.org/stream/p3ciphers/grain/Grain_p3.pdf

[18] T. B. F. Arnault and C. Lauradoux, "Update on f-fcsr stream cipher," eSTREAM, the ECRYPT Stream Cipher Project, Tech. Rep., 2007. [Online]. Available: http://www.ecrypt.eu.org/stream/p3ciphers/ffcsr/ffcsr_p3.pdf

[19] J. Hong and P. Sarkar. (2005) Rediscovery of time memory tradeoffs. Cryptology ePrint Archive. [Online]. Available: http://eprint.iacr.org/2005/090.pdf

[20] M. Matsumoto, M. Saito, T. Nishimura, and M. Hagita, "Cryptanalysis of CryptMT: Effect of huge prime period and multiplicative filter," in *The State of the Art of Stream Ciphers*, 2006.

[21] H. Haramoto *et al.*, "Efficient jump ahead for $\mathbf{F}_2$-linear random number generators," 2006, gERAD Report G-2006-62, submitted for publication.