DOI: 10.2298/CSIS121102040T

Efficient Implementation for QUAD Stream Cipher with GPUs

Satoshi Tanaka[1], Takashi Nishide[2], and Kouichi Sakurai[2]

[1] Graduate School of Information Science and Electrical Engineering,
744 Motooka, Nishi-ku, Fukuoka, Japan
tanasato@itslab.inf.kyushu-u.ac.jp
[2] Faculty of Information Science and Electrical Engineering,
744 Motooka, Nishi-ku, Fukuoka, Japan
{nishide@inf, sakurai@csce}.kyushu-u.ac.jp

Abstract. QUAD stream cipher uses multivariate polynomial systems. It has provable security based on the computational hardness assumption. More specifically, the security of QUAD depends on hardness of solving non-linear multivariate systems over a finite field, and it is known as an NP-complete problem. However, QUAD is slower than other stream ciphers, and an efficient implementation, which has a reduced computational cost, is required.
In this paper, we propose an efficient implementation of computing multivariate polynomial systems for multivariate cryptography on GPU and evaluate efficiency of the proposal. GPU is considered to be a commodity parallel arithmetic unit. Moreover, we give an evaluation of our proposal. Our proposal parallelizes an algorithm of multivariate cryptography, and makes it efficient by optimizing the algorithm with GPU.

Keywords: stream cipher, efficient implementation, Multivariate Cryptography, GPGPU.

1. Introduction

1.1. Background

Nowadays cryptography is a necessary technology for network communication. Multivariate cryptography uses multivariate polynominals system as a public key. The security of multivariate cryptography is based on the hardness of solving non-linear multivariate polynomial systems over a finite field [2]. Multivariate cryptography is considered to be a promising tool for fast digital signature, because it requires just computing multivariate polynomial system.

QUAD is a stream cipher, which uses a multivariate quadratic system [4]. Symmetric ciphers are used to authentication schemes [8] and signatures [10]. The security of QUAD depends on the multivariate quadratic (MQ) problem. Therefore QUAD has provable security like public key cryptography though it is a symmetric cipher. QUAD has high security, but it is very slow compared with other symmetric ciphers. When QUAD stream cipher is accelerated, we can realize high security communication with QUAD.

1.2. Related Works

Berbain et al. [3] provided efficient implementation techniques for multivariate cryptography including QUAD stream cipher on CPUs. They implemented 3 cases of QUAD instances, over $GF(2)$, $GF(2^4)$, and $GF(2^8)$. Arditti et al. [1] showed FPGA implementations of QUAD for 128, 160, 256 bits blocks over $GF(2)$. Chen et al. [6] presented throughputs of a GPU implementation of QUAD for 320 bits blocks over $GF(2)$. However the results show that GPU implementations are slower than ideal CPU implementaitons.

Most of these related works just implemented several QUAD instances. They did not evaluate computational costs of QUAD stream ciphers. Only Berbain et al. [3] showed the computational costs of QUAD with n unkonowns and m multivariate quadratics, which are $O(mn^2)$. We extended several implementation strategies for multivariate quadratic of Berbain et al. to GPU implementations and evaluated the computational cost of QUAD [12].

This is an extension work of our previous result [12]. We present extended GPU implementation results from $GF(2)$ case to $GF(2^p)$ cases, and comparisons with other works. Moreover, we refine the evaluations of computational costs of QUAD for general cases and optimized $GF(2)$ cases.

1.3. Motivation

Our goal is to implement efficient QUAD stream cipher. Since QUAD has a rigorous security proof as public key cryptography, we can use a fast and secure cipher when QUAD becomes fast like other stream ciphers.

1.4. Our Contribution

We provide two techniques to implemnt QUAD stream cipher. One is a parallel implementation for computing multivariate polynomials. The other is an optimization technique for implementing QUAD on GPUs.

In this paper, we discuss the computational time for generating keystreams of QUAD in more detail than [12]. Moreover, we report results of implementation of QUAD stream cipher over $GF(2)$, $GF(2^2)$, $GF(2^4)$, and $GF(2^8)$ on GPU.

2. CUDA Computing

2.1. GPGPU

Originally, Graphical Processing Units (GPUs) are process units for drawing the computer graphics. Recently, some online network games and simulators require very high level computer graphics. The GPU performance is growing to satisfy such requirements. Therefore, GPU has a large amount of power for computation.

GPGPU is a technique for any general process by using GPUs. In cryptography, it is used for some implementations. For example, Manavski proposed an implementation of AES on GPU, which is 15 times faster than an implementation on CPU, in 2007 [7]. Moreover, Osvik et al. presented a result of an over 30 Gbps GPU implementations of AES, in 2010 [9]. On the other hand, the GPGPU technique is also used for cryptanalysis. Bonenberger et al. used a GPU to generating polynomials of the General Number Field Sieve [5].

Because GPUs are designed based on SIMD, it is better to handle several simple tasks simultaneously. On the other hand, the performance of a GPU core is not higher than CPU. Therefore, if we use GPU for sequencial processing, it is not effective. In the GPGPU techniques, how to parallelize algorithms is an important issue.

2.2. CUDA API

CUDA is a development environment for GPU, based on C language and provided by NVIDIA. Pregnancy tools for using GPU have existed before CUDA is proposed. However, such tools as OpenGL and DirectX need to output computer graphics while processing work. Therefore, these tools are not efficient. CUDA is efficient, because CUDA uses computational core of GPU directly.

In CUDA, hosts correspond to computers, and devices correspond to graphic cards. CUDA works by making the host control the device. Kernel is a function the host uses to control the device. Because only one kernel can work at a time, a program requires parallelizing processes in a kernel. A kernel handles some blocks in parallel. A block also handles some threads in parallel. Therefore a kernel can handle many threads simultaneously.

NVIDIA GeForce GTX 480 In this paper, we use a GPU, which is named GeForce GTX 480 by NVIDIA. It is a high-end GPU of GeForce 400 series released in March 2010. GTX 480 is constructed by a Fermi architecture which is a new architecture. GTX 480 uses 15 streaming multiprocessors(SMs), which are constructed by 32 cuda cores instead of by 8 cuda cores.

3. Multivariate Cryptography

3.1. Cryptography

Cryptography is a technique to prevent data from being leaked by adversaries. Mainly, we use it on network communication. Cryptography is categorized into two types, one is symmetric key cryptography and the other is asymmetric key cryptography.

Symmetric Key Cryptography Symmetric key cryptography uses the same keys or functions in encryption and decryption. It has two types, block cipher and stream cipher. Block cipher encrypts message block by block size. Stream cipher uses pseudorandom number generators as keystream generators. A message is encrypted with keystream in sequence.

Asymmetric Key Cryptography Asymmetric key cryptography has two types of keys. One is a public key, which is used for encryption. The other is a private key for decryption.

3.2. Multivariate Polynomial Systems

Multivariate Polynomials We use a finite field $GF(q)$. Let $X = (x_1, \dots, x_n)$ be a n-tuple variable of $GF(q)$, we describe monomials as $\alpha^{(k)}_{s_1,\dots,s_k} \prod_{i=1}^{k} x_{s_i}$, where $k \geq 0$, $1 \leq s_1 \leq \cdots \leq s_k \leq n$. $\alpha^{(k)}_{s_1,\dots,s_k}$ is a coefficient of a k-dimensional monomial. Therefore, they consist of a coefficient and k variables. If a dimension of a monomial is 0, it is called a constant.

Multivariate polynomials contain a sum of monomials. Let $f^{(d)}(X)$ be a d-dimensional multivariate polynomial. It is denoted as Formula (1),

$$f^{(d)}(X) = \alpha^{(0)} + \sum_{k=1}^{d} \sum_{1 \leq s_1 \leq \cdots \leq s_k \leq n} \alpha^{k}_{s_1,\dots,s_k} \prod_{i=1}^{k} x_{s_i}. \tag{1}$$

Especially when $k = 2$, polynomials are called quadratics. Let $Q(X)$ be a multivariate quadratics, and Formula (2) presents $Q(X)$ with n unknowns,

$$Q(X) = \sum_{1 \leq i \leq j \leq n} \alpha_{i,j} x_i x_j + \sum_{1 \leq i \leq n} \beta_i x_i + \gamma, \tag{2}$$

where $\alpha_{i,j} = \alpha^{(2)}_{i,j}$, $\beta_i = \alpha^{(1)}_i$ and $\gamma = \alpha^{(0)}$.

Multivariate Polynomials Systems and MP Problem A multivariate polynomial $f(X)$ can be considered as a multivariate function, which computes results with some given variables. A multivariate polynomial system is a group of such functions. The multivariate polynomial system $MP(X)$ which is constructed with n unknowns and m d-dimensional polynomials is given in Formula (3).

$$MP(X) = \{f^{(d)}_1(X), \dots, f^{(d)}_m(X)\} \tag{3}$$

A multivariate quadratic system is a special case of the multivariate polynomial system, which uses quadratic functions $Q(X)$. The multivariate quadratic system $MQ(X)$ which is constructed with n unknowns and m quadratics is also given in Formula (4).

$$MQ(X) = \{Q_1(X), \dots, Q_m(X)\} \tag{4}$$

We assume that $MP(X)$ is constructed with m d-dimensional polynomials. MP problem is to find $X = (x_1, \dots, x_n)$ where $f^{(d)}_i(X) = 0$ for all $1 \leq j \leq m$. MP problem on a finite field is known as an NP-hard problem [11]. We can also define MQ problem for multivariate quadratic systems $MQ(X)$. It is also known as an NP-hard problem. The security of QUAD stream cipher depends on the MQ assumption.

3.3. QUAD Stream Cipher

QUAD is a stream cipher which is proposed by Berbain et al. [4]. However, it is a stream cipher, and the security of it is based on the MQ assumption.

Constructions QUAD uses a n-tuple internal state value $X = (x_1, \ldots, x_n)$ and a random multivariate quadratic system $S(x_1, \ldots, x_n)$ with m multivariate quadratic function $Q(X)$: $GF(q)^n \mapsto GF(q)$, such that

$$S(X) = \{Q_1(X), \cdots, Q_m(X)\}, \tag{5}$$

as a pseudorandom number generator. It is denoted by QUAD(q, n, r), where r is a number of output keystreams, and $r = m - n$. Usually, m is set to kn, where $k \geq 2$, and therefore $r = (k-1)n$.

Keystream Generation Let $m = kn$ and $S(X) = \{Q_1(X), \ldots, Q_{kn}(X)\}$ be divided two parts as $S_{it}(X) = \{Q_1(X), \ldots, Q_n(X)\}$ and $S_{out}(X) = \{Q_{n+1}(X), \ldots, Q_{kn}(X)\}$. The keystream generator of QUAD follows three steps, such that,

Computation Step
The generator computes values of system $S(X)$, where $X = (x_1, \ldots, x_n)$ is a current internal value.

Output Step
The generator outputs keystreams $S_{out}(X) = \{Q_{n+1}(X), \ldots, Q_{kn}(X)\}$ from values of $S(X)$.

Update Step
The current internal value $X = (x_1, \ldots, x_n)$ is updated to a next internal value with a n-tuple value $S_{it}(X) = \{Q_1(X), \ldots, Q_n(X)\}$ from $S(X)$.

The sketch illustrating the keystream generation algorithm is shown in Fig. 1. It indicates that the generator outputs keystreams by repeating the above three steps.

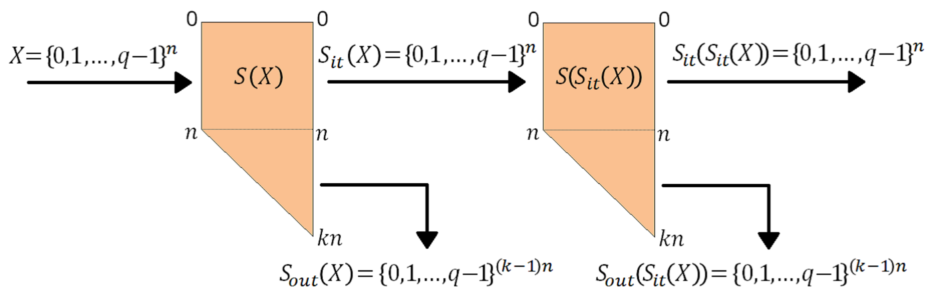
$X = \{0,1,\ldots,q-1\}^n$
0
$S(X)$
n
kn
$S_{it}(X) = \{0,1,\ldots,q-1\}^n$
$S(S_{it}(X))$
$S_{it}(S_{it}(X)) = \{0,1,\ldots,q-1\}^n$
$S_{out}(X) = \{0,1,\ldots,q-1\}^{(k-1)n}$
$S_{out}(S_{it}(X)) = \{0,1,\ldots,q-1\}^{(k-1)n}$

Fig. 1. Generating keystream of QUAD

The generated keystreams are considered to be a pseudorandom bit string and used to encrypt a plaintext with the bitwise XOR operationg.

Key and Initialization of Current State Berbain et al. [4] also provides a technique for initialization of the internal state $X = (x_1, \dots, x_n)$. For QUAD(q, n, r), we use the key $K \in GF(q)^n$, the initialization vector $IV = \{0, 1\}^{|IV|}$ and two carefully randomly chosen multivariate quadratic systems $S_0(X)$ and $S_1(X)$, mapping $GF(q)^n \mapsto GF(q)^n$ to initialize X.

The initialization of the internal state X follows two steps, such that,

Initially Set Step
We set the internal state value X to the key K.

Initially Update Step
We update X for $|IV|$ times. Let i be a number of iterating initially update and $IV_i = \{0, 1\}$ be a value of i-th element of IV, and we change the value of X to
- $S_0(X)$, where $IV_i = 0$, and
- $S_1(X)$, where $IV_i = 1$.

Computational Cost of QUAD The computational cost of multivariate quadratics depends on computing quadratic terms. The summation of quadratic terms requires $n(n+1)/2$ multiplications and additions. Therefore the computational costs of one multivariate quadratic is $O(n^2)$. QUAD(q, n, r) requires to compute m multivariate quadratics. Since $m = kn$, the computational cost of generating key stream is $O(n^3)$.

Security Level of QUAD The security level of QUAD is based on the MQ assumption. Berbain et al. [4] prove that solving QUAD needs solving MQ problem. However, according to the analysis of QUAD using the XL-Wiedemann algorithm which was proposed by Yang et al. [14], QUAD$(256, 20, 20)$ has 45-bit security, QUAD$(16, 40, 40)$ has 71-bit security, and QUAD$(2, 160, 160)$ has less than 140-bit security. Actually, secure QUAD requires larger constructions such as QUAD$(2, 256, 256)$, QUAD$(2, 320, 320)$.

4. Strategy

4.1. Existing Methods of Berbain et al.

Berbain et al. [3] provided efficent implementation techniques of computing multivariate polynomial systems for multivariate cryptography. In this paper, we use these strategies from [3].

- Variables are used as vectors. For example, C language defines int as a 32-bit integer variable. Therefore, we can use int as a 32-vector of boolean.
- We precompute each quadratic term. Because in multivariate quadratic systems, we must compute the same $x_i x_j$ for every polynomials, we can make efficient implementation by precomputing quadratic terms.
- We compute only non-zero terms in $GF(q)$. Because the probability of $x_i = 0$ is $1/2$ the probability of $x_i x_j = 0$ is $3/4$. Therefore, we can reduce computational cost to $1/4$.

4.2. Parallelizing on the GPU

In the GPGPU, the most important point is the parallelization of algorithms. Because the performance of GPU cores is worse than that of CPU, serial implementations with GPU are expected to be slower than CPU implementations.

Since all the polynomials of a multivariate quadratic system are independent of each other, parallelization of system is easy. We discuss how to parallelize a multivariate quadratics of system.

Summation of quadratic terms can be considered as summation of every element of a triangular matrix as the left side of Fig. 2. We assume that other elements of matrix are zero. Therefore we can compute summation of quadratic terms as summation of regular matrix as the right side of Fig. 2. Then, we can compute the summation as $\sum_{i=1}^{n}\sum_{j=1}^{n}\alpha_{i,j}x_ix_j$ in the following method.

1. We compute $S_k(x)=\sum_{i=1}^{n}\alpha_{k,i}x_kx_i$ for all k in parallel.
2. We compute $\sum_{k=1}^{n}S_k(x)$.

However computations increase trivial computations; we can make efficient implementations by parallelization with GPU.

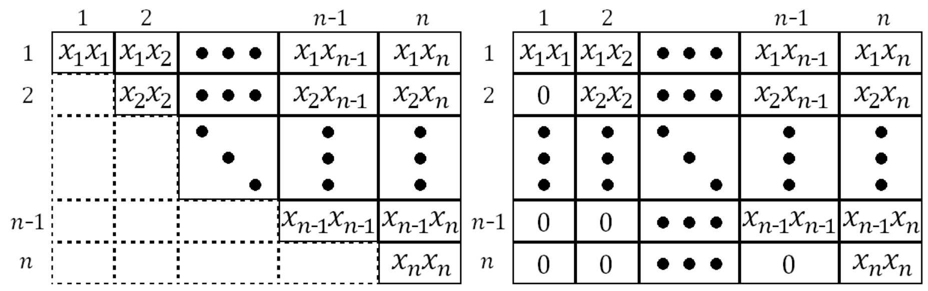

Fig. 2. Left: handling as a triangular matrix. Right: handling as a rectangle matrix with trivial 0 elements.

Next we reduce trivial computations of that way. We reshape a triangular matrix to a rectangular matrix as in Fig. 3, which presents the method of reshaping matrix. By this reshaping, we can reduce efficiently about 25% of the cost for computing an equation of a multivariate quadratic system.

4.3. Optimization on GPU architectures

On GPU implementations, we must consider characteristics of GPU. The strongest point of GPU is the computational power by processing cores, and a core is slower than a CPU. Therefore, non-active cores in a process affect results.

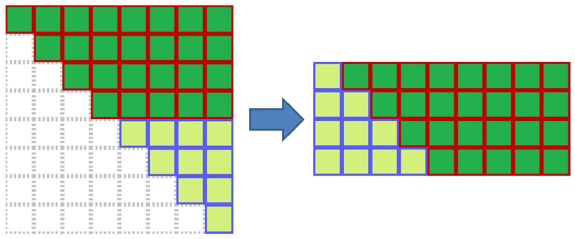

Fig. 3. Reshaping triangle to rectangle

Optimization of Matrix Calculation NVIDIA GeForce GTX 480 has 15 SMs, and every SM has 32 cuda cores. Since every SM handle 32 threads at a time, a process, which handles 32 threads, is not suitable for GPU implementations. Therefore, we should make sure that the number of threads in a process is divisible by 32. In the same way, we should make sure that an algorithm can be handled by 15 SMs in parallel. Finally, an algorithm should be paralleled as a multiple of 32×15.

An n-dimensional triangular matrix has $n(n+1)/2$ elements. Then a long side of a rectangle matrix, which is reshaped by an n-dimensional triangular matrix, has n or $n+1$ elements. However in $GF(2)$, a number of a long side's elements can be counted in a process, counting is a cost of computing a matrix. Therefore, we assume that $n = 30k$ where k is a natural number. In this way, a rectangle matrix is constructed by $15k \times (31k+1)$ in Fig. 4, and we can handle the rectangle matrix as $15k \times 32k$. Moreover, computing k-dimensional square submatrix from the matrix in parallel, we can reduce the $15k \times 32k$ matrix to a 15×32 matrix. Thus we can parallelize calculating of a matrix by 15 SMs and 32 cuda cores per SM.

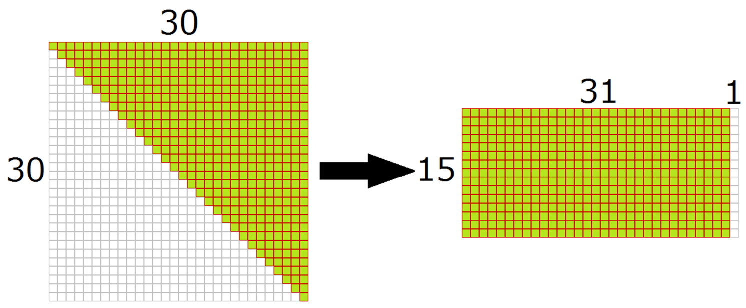

Fig. 4. Handling a $15k \times 32k$ matrix e.g. $k = 1$

Optimizing in Processes For realizing an efficient GPU implementation, we must design an algorithm as a chunk of similar small computings. Moreover, conditional branches are not suited to processing on GPU. Then we handle conditional branches as difference of kernels, and implement them by processing on CPU. In this case, we make kernels by difference of a number of non-zero terms. However, making all kernels every number of non-zero temrs is a heavy cost of implementaions. Therefore, we make kernels just every number of k. E.g. QUAD$(2, 320, 320)$, the maximum of k is 11, thus we have to make only 11 kernels.

4.4. Evaluation of GPU Implementation

Accelerating by strategies of Berbain et al. Originally, QUAD(q, n, n) requires $n(n+1)(n+2)$ additions and $2n^2(n+2)$ multiplications. Moreover, QUAD(q, n, n) requires $2n$ times computation of equations.

By the strategy of Berbain et al. [4], we can reduce multiplications of monomials. A monomial $x_i x_j$ is a common value for each polynomial. Then we need to calculate monomials only once. Since it takes $n(n+1)/2$ multiplications, we reduce multiplications from $2n^2(n+2)$ times to $n^2(n+3) + n(n+1)/2$ times.

Moreover, we can compute some polynomials at a time by vectorization of variables. In case of $q = 2^t$, i.e. using $GF(2^t)$, a 32-bit integer variable handles $32/t$ polynomials. Therefore QUAD(q, n, n) (=QUAD$(2^t, n, n)$) requires $\lceil t/16n \rceil (n+1)(n+2)$ additions and $\lceil t/16n \rceil n(n+3)/2 + n(n+1)/2$ multiplications.

Accelerating by parallelization In GPU implementations, we can parallelize some computational steps of evaluating multivariate polynomial systems.

By computing $x_i x_j$ in parallel for each i's, it takes n multiplications.

The computational cost of summations of row elements in a matrix requires $(n+1)(n+2)/2$ additions and multiplications for $64n/t$ polynomials. By parallelization using C cores in GPU, it takes $\lceil tn(n+2)/32C \rceil (n+1)$ additions and multiplications. Also, the computational cost of summations of column elements in a matrix can be reduced from $\lceil tn/16 \rceil (n+2)$ additions to $\lceil tn/16C \rceil (n+2)/2$ additions.

Actually, GTX 480 has 480 cores. Then it computes all the polynoimals at a time over $GF(2^t)$ field, where $tn/32 \leq 480$. Therefore, QUAD$(2^t, n, n)$ requires $\lceil (n+2)/P \rceil (n+1) + (n+2)/2$ additions and $\lceil (n+2)/P \rceil (n+1) + n$ multiplications for generating keys, where $P = 32C/tn$.

Acclerating for $GF(2)$ By the strategy of Berbain et al. [4], we can compute only non-zero terms in $GF(2)$. Because the probability of $x_i = 0$ is $1/2$ the probability of $x_i x_j = 0$ is $3/4$. Therefore, we can reduce computational cost to $1/4$.

Then we can compute an equation of QUAD$(2, n, n)$ by $\lceil (n+4)/4P \rceil (n+2)/2 + (n+4)/4$ additions and $\lceil (n+4)/4P \rceil (n+2)/2 + n/2$ multiplications, where $P = 32C/tn$.

Suppose the number of non-zero variables is $30k$. Then k-dimensional submatrix requires $k \times k$ additions. Using our strategy, we can compute the summations of QUAD$(2, n, n)$ by $(\lceil (15 \times 32nk)/16C \rceil + \lceil 15 \times 32n/16C \rceil)k + 15 + 32$ additions. Since the GTX 480 has 480 cores (i.e., $C = 480$), it requires $(\lceil nk/16 \rceil + \lceil n/16 \rceil)k + 47$ additions. For example, QUAD$(2, 320, 320)$ requires $20k^2 + 20k + 47$ additions.

5. Experiments

In this section, we present and discuss results of experiments. We used NVIDIA GeForce 480 GTX as a GPU, and also used Intel Core $i7$ $875K$ as a CPU. Moreover, the memory of implementation environment was 8GB.

5.1. Experimentations of Encryption

We implement QUAD$(2, n, n)$ on CPU and GPU; set $n = 32, 64, 128, 160, 256, 320, 512$, and measure the time of encrypting 5MB file. Also, we implement QUAD$(2^t, n, n)$ on GPU; set $t = 2, 4, 8$ and $n = 32, 64, 128, 160, 256, 320, 512$, and measure the time of encrypting messages 1000 times.

Moreover, we optimized GPU implementation of QUAD$(2, n, n)$ by our optimization strategies, and also measure the time of encrypting 5MB file.

5.2. Results

We present the results of QUAD$(2, n, n)$ implementations in Table 1, and QUAD$(2^t, n, n)$ in Table 2. Table 1 presents the time of encrypting 5MB files and throughputs of implementations of each QUADs. In Table 2 we compared our implementations of QUAD$(2^p, n, n)$, where $p = 1, 2, 4, 8$.

Table 1. Results of QUAD implementations.

QUAD(q,n,r)	Encryption time(s)		Throughputs(Mbps)	
	CPU	GPU	CPU	GPU
$(2, 32, 32)$	0.35	66.02	114.286	0.606
$(2, 64, 64)$	13.56	46.58	2.949	0.859
$(2, 128, 128)$	52.56	36.82	0.761	1.086
$(2, 160, 160)$	82.07	36.23	0.487	1.104
$(2, 256, 256)$	206.80	35.87	0.193	1.115
$(2, 320, 320)$	326.52	38.96	0.123	1.027
$(2, 512, 512)$	858.20	72.80	0.047	0.549

The image results of QUAD$(2, n, n)$ are shown in Fig. 5. Encryption time of CPU implementations follows square of n. However the computational cost of generating keystream is $O(n^3)$, number of generating keystream follows n.

On the other side, the results of GPU Implementations are not almost different. NVIDIA GeForce GTX 480 can use 15 blocks, and every block can use 32 threads, programs can handle 480 processes at the same time. QUAD$(2, n, n)$ requires $\lceil (n+4)/4P \rceil (n+2)/2 + (n+4)/4$ additions and $\lceil (n+4)/4P \rceil (n+2)/2 + n/2$ multiplications for generating keystreams, where $P = 32C/tn$. However when $(n+4)/4P \leq 1$ (i.e., $tn(n+4) = 128C$), we can generate keystreams of QUAD$(2, n, n)$ by only $(3n+8)/4$ additions and $n+1$ multiplications. On the GTX 480, we can set $C = 480$. Then we have that $n \leq 247$. Therefore the computational cost of QUAD$(2, n, n)$ is proportional to the number of unknowns n, and it generates keystreams in stable throughputs, when $n \leq 247$. Actually, we can see the decrease of the throughput of QUAD$(2, n, n)$ between $n = 256$ and 320.

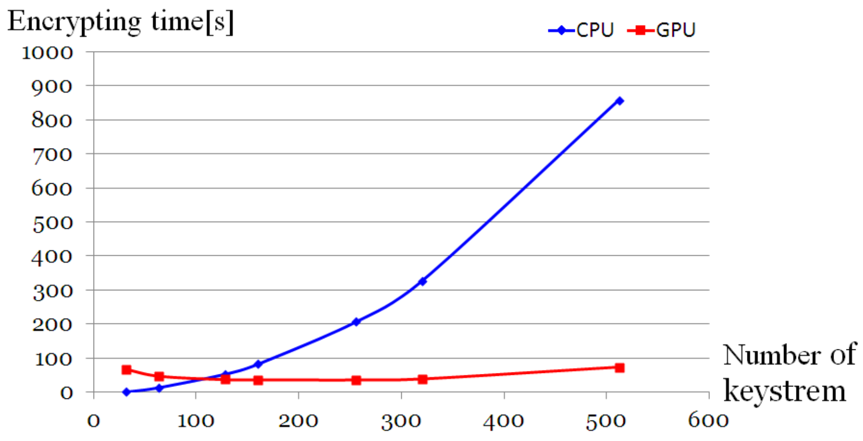

Fig. 5. Comparison of encryption time between CPU and GPU.

Table 2 and Fig. 6 also show the decrease of throughputs of QUAD$(2^t, n, n)$, where $t = 1, 2, 4, 8$ and $n = 32, 64, 128, 160, 256, 320, 512$. Especially, Fig. 6 shows that the higher the degree of $GF(2^t)$ is, the more drastic the decrease of the throughput of QUAD is. For example, when $n > 44$, the computational cost of QUAD over $GF(2^8)$ follows $O(n^3)$. Therefore the larger the number of unknowns n is, the slower the throughput of QUAD is.

Moveover, we provide the results of our optimized implementation with the results of a non-optimized implementation Table 3 shows that the throughputs of our optimized GPU implementations with the throughputs of our non-optimized GPU implementations, and ratio of GPU and CPU implementations. The results of GPU implementations show that the throughputs of optimized QUAD

Table 2. Implementation of QUAD$(2^t, n, n)$.

Unknowns	Througputs(Mbps)			
n	$GF(2)$	$GF(2^2)$	$GF(2^4)$	$GF(2^8)$
32	0.606	2.517	4.128	6.110
64	0.859	3.353	4.810	4.863
128	1.086	3.271	4.424	1.405
160	1.104	2.603	2.570	0.827
256	1.115	1.161	0.858	0.249
320	1.027	0.581	0.473	0.189
512	0.549	0.177	0.146	0.072

was improved compared with the non-optimized implementation. Our optimized implementations are 2.0 to 4.4 times faster than non-optimized implementation. We infer that the main cause of accelerated encryption time is due to the optimizations to handle $n = 30k$. Because computing n elements in serial is heavy to GPU, we can reduce serial computation by such handling.

Table 3. Throughputs of QUAD Implementations with Optimization and Non-Optimization.

QUAD(q,n,r)	Throughputs(Mbps)		Speed Up
	Non-optimized	Optimized	Rate(times)
$(2, 32, 32)$	0.606	1.234	x2.037
$(2, 64, 64)$	0.859	2.319	x2.699
$(2, 128, 128)$	1.086	4.132	x3.805
$(2, 160, 160)$	1.104	4.872	x4.413
$(2, 256, 256)$	1.115	4.115	x3.691
$(2, 320, 320)$	1.027	3.656	x3.560
$(2, 512, 512)$	0.549	1.494	x2.722

In Table 4, we compared related works Berbain et al. [3], Arditti et al. [1], and Chen et al. [6] for QUAD$(2, n, n)$, where $n = 128, 160, 256, 320, 512$, and compared throughputs of QUAD$(2^4, 40, 40)$ and QUAD$(2^{16}, 20, 20)$ with Berbain et al. [3]. According to Table 4 although our optimized implementation of QUAD$(2, 160, 160)$ is slower than Berbain et al. [3], our optimized implementation of QUAD$(2, 256, 256)$ and QUAD$(2, 320, 320)$ is faster than the GPU implementation of Arditti et al. [1] and Chen et al. [6]. [3] These results show that our optimized GPU implementation technique is suited to large QUAD constructions.

[3] Chen et al. also presented the CPU implmentations result of QUAD stream cipher. Although 6.10Mbps is the fastest known result, it was just a theoretic estimate [13] without a real implementation.

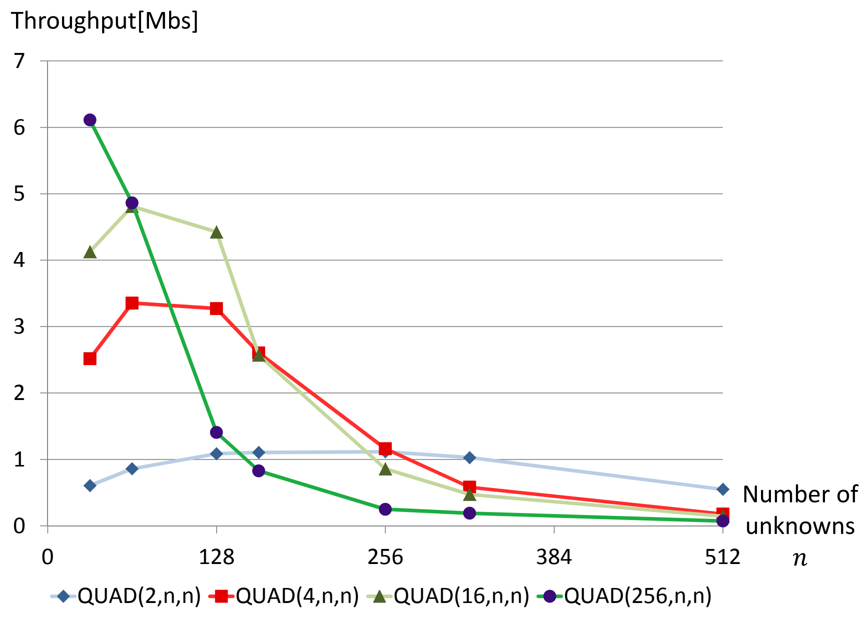

Fig. 6. Throughputs of QUAD implementations over $GF(2^t)$ $(t = 1, 2, 4, 8)$

On the other hand, Table 4 shows that both of our GPU implementation both for QUAD$(2^4, 40, 40)$ and QUAD$(2^8, 20, 20)$ are slower than Berbain et al. [3]. We infer that our implementation strategies are very specialized to $GF(2)$.

6. Conclusion

We presented and evaluated the GPU implementation techniques for QUAD stream cipher. Also we provided optimization techniques of QUAD to suit NVIDIA GeForce GTX 480.

Moreover, we carried out the experiments on the implementations of QUAD over $GF(2)$, $GF(2^2)$, $GF(2^4)$ and $GF(2^8)$. As a result, the larger the number of unknowns n is, the slower the throughput of QUAD is. However, when $tn(n + 2) \leq 32C$, it is stable. The condition for stable throughputs depends on the number of cores C. Although the GTX 480 has only 480 cores, the GTX 680, which is the latest high-performance GPU, has 1536 cores. Therefore, the throughput of QUAD$(2, n, n)$ is stable if $n \leq 439$. We expect that future GPUs allow efficient implementation of QUAD$(2, 512, 512)$ and more heavy constructions of QUAD.

Table 4. Comparison QUAD Implementations with Related Works.

	Throughputs(Mbps)			
	Our Works	Berbain et al. [3]	Arditti et al. [1]	Chen et al. [6]
	GPU(Optimized)	CPU	FPGA	GPU
QUAD$(2, 128, 128)$	4.132	N.A.	4.1	N.A.
QUAD$(2, 160, 160)$	4.872	8.45	3.3	N.A.
QUAD$(2, 256, 256)$	4.115	N.A.	2.0	N.A.
QUAD$(2, 320, 320)$	3.656	N.A.	N.A.	2.6
QUAD$(2^4, 40, 40)$	4.320	23.59	N.A.	N.A.
QUAD$(2^8, 20, 20)$	3.895	42.15	N.A.	N.A.

Acknowledgments. This work is partially supported by Japan Science and Technology agency (JST), Strategic Japanese-Indian Cooperative Programme on Multidisciplinary Research Field, which combines Information and Communications Technology with Other Fields, entitled "Analysis of Cryptographic Algorithms and Evaluation on Enhancing Network Security Based on Mathematical Science." The authors are grateful to three anonymous referees of ComSIS-2013 for improving this article.

References

1. Arditti, D., Berbain, C., Billet, O., Gilbert, H.: Compact fpga implementations of quad. In: Proceedings of the 2nd ACM symposium on Information, computer and communications security. pp. 347–349. ACM (2007)
2. Bard, G.: Algebraic cryptanalysis. Springer (2009)
3. Berbain, C., Billet, O., Gilbert, H.: Efficient implementations of multivariate quadratic systems. In: Selected Areas in Cryptography. pp. 174–187. Springer (2007)
4. Berbain, C., Gilbert, H., Patarin, J.: Quad: A practical stream cipher with provable security. Advances in Cryptology-EUROCRYPT 2006 pp. 109–128 (2006)
5. Bonenberger, D., Krone, M.: Factorization of rsa-170. Tech. rep., Tech. rep., Ostfalia University of Applied Sciences (2010)
6. Chen, M.S., Chen, T.R., Cheng, C.M., Hsiao, C.H., R., N., Yan, B.Y.: What price a provably secure stream cipher? (2010)
7. Manavski, S.: Cuda compatible gpu as an efficient hardware accelerator for aes cryptography. In: Signal Processing and Communications, 2007. ICSPC 2007. IEEE International Conference on. pp. 65–68. Ieee (2007)
8. Miyaji, A., Rahman, M.S., Soshi, M.: Efficient and low-cost rfid authentication schemes. Journal of Wireless Mobile Networks, Ubiquitous Computing, and Dependable Applications 2(3), 4–25 (2011)
9. Osvik, D., Bos, J., Stefan, D., Canright, D.: Fast software aes encryption. In: Fast Software Encryption. pp. 75–93. Springer (2010)
10. Pakniat, N., Eslami, Z.: A proxy e-raffle protocol based on proxy signatures. Journal of Wireless Mobile Networks, Ubiquitous Computing, and Dependable Applications 2(3), 74–84 (2011)
11. Patarin, J., Goubin, L.: Asymmetric cryptography with s-boxes is it easier than expected to design efficient asymmetric cryptosystems? Information and Communications Security pp. 369–380 (1997)

12. Tanaka, S., Nishide, T., Sakurai, K.: Efficient implementation of evaluating multivariate quadratic system with gpus. In: Innovative Mobile and Internet Services in Ubiquitous Computing (IMIS), 2012 Sixth International Conference on. pp. 660–664. IEEE (2012)
13. Yang, B.Y.: (private communicate) (2011)
14. Yang, B., Chen, O., Bernstein, D., Chen, J.: Analysis of quad. In: Fast Software Encryption. pp. 290–308. Springer (2007)

Satoshi Tanaka received a B.E. degree from National Institution for Academic Degrees and University Evaluation, Japan in 2010, and an M.E. degree from Kyushu University, Japan in 2012. Currently he is a candidate for the Ph.D. in Kyushu University. His primary research is in the areas of cryptography and information security.

Takashi Nishide received a B.S. degree from the University of Tokyo in 1997, an M.S. degree from the University of Southern California in 2003, and a Dr.E. degree from the University of Electro-Communications in 2008. From 1997 to 2009, he had worked at Hitachi Software Engineering Co., Ltd. developing security products. Since 2009, he has been an assistant professor in Kyushu University. His primary research is in the areas of cryptography and information security.

Kouichi Sakurai is Professor of Department of Computer Science and Communication Engineering, Kyushu University, Japan since 2002. He received B.E., M.E., and D.E. of Mathematics, Applied Mathematics, and Computer Science from Kyushu University in 1982, 1986, and 1993, respectively. He is interested in cryptography and information security. He is a member of IPSJ, IEEE and ACM.

Received: October 06, 2012; Accepted: January 31, 2013.