ISSN (Online) 2393-8021
ISSN (Print) 2394-1588

IARJSET

International Advanced Research Journal in Science, Engineering and Technology

ICRAESIT - 2015

B V Raju Institute of Technology, Narsapur

Vol. 2, Special Issue 2, December 2015

# A Study on Cryptographic Algorithm and Key Identification Using Genetic Algorithm for Parallel Architectures

**Sunar Mohammed Farooq[1], Nageswara Reddy Karukula[2], J.David Sukeerthi Kumar [3]**

Assistant Professor, SREC, Nandal, AP, India[1,2,3]

**Abstract:** Presently a day, the measure of exchange of information over huge system is expanding step by step. As there is increment in information exchange, at the same time there are security dangers that are emerging with it. In this paper we are proposing cryptographic frameworks that make utilization of hereditary calculation to safely exchange the information over expansive system. This methodology helps us to give high achievability to exchange information safely. The methods that we are utilizing for encryption & decoding are focused around hereditary calculation which will help us to effectively scramble the information which is resistible for any sort of outer assaults. At that point we will parallelize it utilizing Open Mp dialect which quickens the change of information to attain high security. At that point break down the execution for both code serial and parallel execution.

**Keywords:** The genetic algorithm, Linear generation  Equation, Genetic operator, Open MP.

## I. INTRODUCTION

Now a days computer networks are becoming more important for exchanging information. One of the most important requirement of these networks is to provide secure transmission of information from one place to another. Cryptography is one of the technique which provides most secure way to transfer the sensitive information from sender to intended receiver. Its main purpose is to make sensitive information unreadable to all other except the intended receiver. Any cryptography algorithm provides following features:
• Privacy/Confidentiality – only intended receiver can read the message.
• Integrity – Assurance that the data received by the receiver has not been altered in the way and it is same as original.
• Authentication – Process to prove the identity of communicating devices.

Wide varieties of cryptography algorithms are available for hiding the sensitive information. But most of the algorithms have many performance limitations such as memory requirement, execution time and computation power. So one of the solution to improve the performance of these algorithms is parallel computation. Parallel computation is a method in which several computations can be carried out simultaneously on two or more microprocessors. Parallel computation can be performed by using multicore and multiprocessor computers having multiple processing elements within a single machine.

## II. RELATED WORK

Cryptography is a process of protecting information by transforming it into unreadable form called cipher text. Cryptography not only protects data from theft or alteration, but can also be used for user authentication. In general, three are three types of cryptographic schemes typically used to accomplish these goals: secret key (or symmetric) cryptography, public-key (or asymmetric) cryptography, and hash functions. In all the cases, the initial unencrypted data is referred to as plaintext. Plain text is encrypted into cipher text, which will in turn (usually) be decrypted into usable plaintext.
Wide varieties of cryptography algorithms are available for hiding the sensitive information. Some of the cryptography algorithms are Advanced Encryption algorithm (AES), DES, HIGHT, International Data Encryption Algorithm (IDEA), Scalable Encryption algorithm (SEA), TEA, RC5, RC6, Skipjack etc. But most of these algorithms have many performance limitations such as memory requirement, execution time and computation power. Memory requirement for some of the algorithms are as follows (in bytes):

| **Cipher** | IDEA | TEA | SEA | AES | DES | HIGHT |
|---|---|---|---|---|---|---|
| Code Size | 1992 | 1050 | 732 | 6120 | 4314 | 888 |

Figure 1 Code size for different algorithms

All of the above cryptography algorithms performs arithmetic operations like byte substitution, rotation, XOR, finite field multiplication, modular multiplication, addition and subtraction. So, all of these algorithms requires more execution time. RC5, TEA, HIGHT, and SEA have moderate code size because of their simple structure. IDEA has a little larger code size as it needs complicated modular multiplicative inverse computation that also brings high burden for initialization. The code size of PRESENT mainly comes from bitwise permutation. AES's complicated algorithm structure results in its largest code size and Skipjack's overhead is due to its structure optimization for performance. TEA and SEA almost need no SRAM as there are no storage requirements for S-box and key schedule.

The SRAM overhead of SkipJack and RC4 mainly comes from S-box and that of RC5, IDEA, HIGHT and PRESENT mainly comes from key schedule. AES has more than two times of overhead compared with other candidates as it needs two S-boxes respectively for encryption and decryption. The SRAM overhead represents the worst case as performance and memory tradeoff can be considered. If performance degradation is allowed, the S-box can be stored in other place, such as program memory and EEPROM, and the key schedule can be generated only when it is to be used. Then, the SRAM can be shared with other function modules.

RC4 is the best in terms of encryption/decryption, because in this clock cycles overhead which is consistent. However, taken initialization into account, RC4 is no longer the best, because initialization is required before each encryption/decryption process of RC4, which corresponds to larger encryption/decryption overhead. SkipJack seems to be a better choice because for this encryption/decryption overhead is a little higher than RC4, but there is no initialization overhead. The main drawback of skipjack is relatively large code size because of it's special structure. The whole encryption/decryption process of SkipJack can be expressed as a complicated expression, so that cipher text can be directly expressed in terms of plaintext, not immediate results. For AES algorithm encryption/decryption overhead is little better compared to RC5 algorithm.

However, the complicated structure of AES algo. leads to the highest program memory overhead among the candidates. IDEA behaves well during encryption and decryption process, although it has the highest initialization overhead. The initialization overhead of IDEA mainly comes from computing modular multiplicative inverse. Compared with AES and RC5 algorithms, encryption/decryption overhead of IDEA is much better. Among the embedded computing oriented algorithms, HIGHT algorithm provides better performance than SEA and PRESENT. Because of simple structure, no. of clock cycles taken and memory overhead together, HIGHT may be the optimal candidate. It achieves better performance with less memory overhead. SEA and PRESENT does not have better performance. SEA algorithm requires a relatively large round number to provide adequate security, which is 2~7 times than other block ciphers. PRESENT algorithm has a basic operation of bitwise permutation. It has no performance overhead in hardware implementation as it only needs wiring, for software implementation it is very expensive because there is not direct assembly instruction support. So, Implementation of most of the cryptography algorithm has various overheads like memory requirement, execution time, power consumption etc.

One of the solution to reduce the execution time for these cryptography algorithms is by using parallel computation. Parallel computation is a method in which several computations can be carried out simultaneously on two or more microprocessors. Parallel computation can be performed by using multicore processors having multiple processing elements within a single chip. OpenMP is an application programming interface which provides various compiler directives and library routines for parallel implementation. So, by making the use of OpenMP API we can parallelize the execution of any specified cryptography algorithm between multiple cores in order to reduce the execution time of the algorithm.

We realize that, Cryptography is the investigation of encoding the data & delivering limitless information which is muddled by the third individual who is getting to information without former authorization of sender or collector. Presently a day different application like the informal communication, business applications, E-trade, in military or satellite correspondence and so forth need part of information transmission over substantial information system. Here is the fundamental sympathy toward protection comes into picture. As the above said territories are all that much helpless against assaults numerous scientists are chipping away at it. We as of now have numerous calculations for scrambling & unscrambling the information.

Encryption is a procedure of moving basic content into figure content and unscrambling is a methodology moving figure content into straightforward content which is meaningful And create a key which is an imparted between techniques of cryptography [9]. Encryption and decoding are inverse to one another for changing data [10].

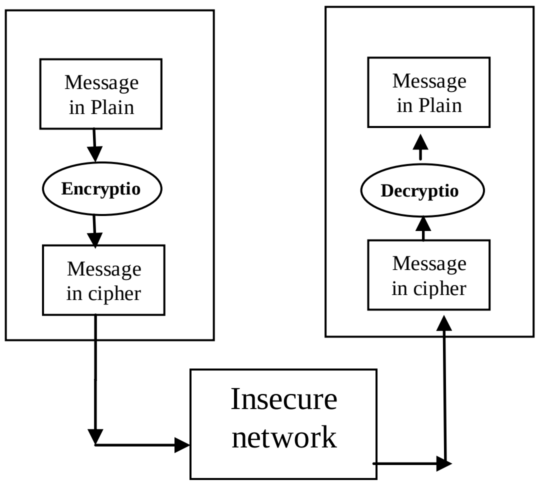

Fig 2 Encryption and Decryption

The hereditary calculation that we are utilizing takes mix of peculiarities of cryptography and hereditary administrator. The hereditary administrators that we are utilizing are hybrid & transformation. In hybrid we consolidate two strings to show signs of improvement string. The change is the methodology of changing any bit from the given set of bits. So here we can utilize the idea of Open Mp to parallelize the code. The era of arbitrary number can likewise be parallelized so we will dependably get distinctive randomized number from diverse distinctive strings. Here we are utilizing the idea of symmetric key.

In the cryptography there are two sorts by which we can encode and decode the content i.e. symmetric and asymmetric key. The cryptographic framework in which sender & receiver utilize the same key for encryption & decryption information is called symmetric key cryptography[1]. The Cryptographic framework in which sender and recipient utilize the distinctive key for encryption & unscrambling information is called unbalanced key cryptography. As we know the information to be exchange is extensive we require extremely proficient & quick calculation to scramble or decode it [9]. The parallelization of genetic algorithm yields better execution that will be extremely helpful for exchanging vast information safely utilizing this parallelized code [8]. Presently we will examine all the issues in subtle element.

### 2.1. Linear generator Equation:

Firstly here we a create a sequence of random number by using Congruential method then next generation number random number with the help of Crossover and Mutation which is a operator for genetic algorithm.

$$\mathbf{X_{n+1} = (s * X_n + T) \bmod m;} \qquad (1)$$

Xn= a randomly generated value initially it is assume.
s= multiplier which randomly generated
T= increment which randomly generated
m= modulus (0<m<10)

### 2.2. Generic Operators:

From the three operators of genetic algorithm i.e. CROSSOVER, SELECTION and MUTATION [6] here we will be using only two operators i.e. Crossover and Selection for our purpose.

### A. Crossover

It is a genetic operator which is used to join two strings to get an efficient string. In this process we take more than one parent process and generate a child solution for that process [2] this crossover genetic operator generate random number which perform in parallel.
There a three types of crossover[6]:

- Trade of uniform crossover
- one site Crossover
- Cut and splice

IARJSET
ISSN (Online) 2393-8021
ISSN (Print) 2394-1588
International Advanced Research Journal in Science, Engineering and Technology
ICRAESIT - 2015
B V Raju Institute of Technology, Narsapur
Vol. 2, Special Issue 2, December 2015

In this paper we will be concentrating on one site Crossover. This strategy is chosen as a result of its playing point over conventional systems that we can undoubtedly trade data through it viably according to our prerequisites. We realize that the information while moving is changed over into bits and afterward sent over the system so our next strategy i.e. MUTATION changes any bit randomly from the set of bit to achieve high security. As the bits are transformed its extremely troublesome for interloper to withdraw the data from it. As we are going to parallelise the code the diverse genetic algorithm bits.

## III. CRYPTOGRAPHY: GENETIC ALGORITHM APPROACH

### 3.1. Algorithm

#### 3.1.1 Generation Linear Equation of First order using Random Number [7].

In this we use Linear Equation for generating first order variables[5]. Initially we will consider four random variable s, T, m. $X_{n.}$ Now as per our formula of generating linear equation we will substitute this assumed 4 values to calculate next value. After that iteratively we will put the value of $X_{n+1}$ that is generated from the previous calculation. This iterative process will terminate only when it will complete the iterations equal to the length of the message. Here we can see that the initial parameters are unchanged throughout the process. A good randomization function must be used to get better initial values[6].

#### 3.1.2 Crossovers and Mutation:

After the above defined process we have some random numbers generated that are equal to the length of message. Now this numbers are converted into binary formats and length of each binary number is equated by padding some zeros at the beginning of that number[4]. Now here we will be using the crossover operation of genetic algorithm in which we will swap some bits of one string with the other string. At this stage two numbers are differently changed from their initial values to the new values. Let’s illustrate it with an example.

For, Example consider two numbers of 8 bit,

$187 \quad 10\underline{\mathbf{1111}}00 \quad \Longrightarrow \quad 10\underline{\mathbf{0001}}00$

$132 \quad 10\underline{\mathbf{0001}}00 \quad \Longrightarrow \quad 10\underline{\mathbf{1111}}00$

As of now we have just performed crossover now we will deal with mutation. Mutation is the process of changing the bit from the available set[3]. Here is the main function of mutation operator to change the bit any the selection of bit which is to be changed is done probabilistically so security increases. Finally after performing Crossover & mutation for several iterations we will get series of randomized number that we will again convert into decimal format for our further use.

Now the numbers which we will get finally will be in decimal format, so we will select the smallest number from it & the specific digit that is mutually decided between two parties will be subtracted from the ASCII value of our text message and the number which we will get after that is sent over the network along with the key.

### 3.2. For Example

#### 3.2.1 Encryption

In this method the simple text converted into cipher text, using above algorithm

• Let, message is ABCDEFGHIJ its length is 10.

Now applying linear Congruential equation, we will get the generated numbers that are similar in length of text message i.e.10. Let, numbers are: 106, 239, 538, 1211, 2725, 6132, 13798, 31046, 69854 and 157172. Now, Select two numbers from a group that starting (106,239), (538, 1211), (2725, 6132), (13798, 31046), (69854, 157172).

Then apply crossover and mutation iteratively:
106= 0000000001101010
239=0000000011101111

After performing crossover
0000000101101010 =362
0000000111101111=495

After applying mutation, numbers are (394, 271). Now we easily identified how group pair is changed from (106,239) to (394,271). However we find that both crossover and mutation increases security and after performing some iteration the whole data become secure.

Now after first iteration of crossover and mutation number will be obtained
(394, 271, 1018, 1371, 2885, 5652, 13318, 30886, 35215, 39069)
Now took a smallest number i.e. 271 and subtract second right digit from ASCII values of each character of message.

**Table 1 Encryption procedure**

The encrypted message is > @CA=@D@GJ
The Cipher text
{ 56,59,66,61,61,65,70,64,72,68 {key of {s, T, m, $X_n$ }

| *File Size* | *Serial Execution Time* | *Parallel Execution Time* | *Speed Up* |
|---|---|---|---|
| *1MB* | *0.33min* | *0.19min* | *1.67* |
| *10MB* | *1.5min* | *0.82min* | *1.81* |
| *50 MB* | *11.2min* | *5.6min* | *2* |
| *100MB* | *21.4min* | *9.5min* | *2.25* |
| *150MB* | *37.3min* | *13.4min* | *2.78* |

### 3.2.2 Decryption

At a user side we get a whole encrypted data along with key {s, T, m, $X_n$}. As we know decryption is an opposite process of encryption for that a receiver side inverse of encryption will be perform[8]. After that many iteration perform in decryption as they performed in encryption using cipher key.

**Table 2 decryption procedure**

| *Cipher text* | *Number to be added* | *ASCII Values* | *Input Data* |
|---|---|---|---|
| *56* | *9* | *65* | *A* |
| *59* | *7* | *66* | *B* |
| *66* | *1* | *67* | *C* |
| *61* | *7* | *68* | *D* |
| *61* | *8* | *69* | *E* |
| *65* | *5* | *70* | *F* |
| *70* | *1* | *71* | *G* |
| *64* | *8* | *72* | *H* |
| *72* | *1* | *73* | *I* |
| *68* | *6* | *74* | *J* |

## IV. RESULTS

As our primary point is to parallelize the project to get quick yield as per that we tried it for different number of words size. Here we display the table speaking to the throughput and the graphical representation.

### (A) For 2 core machine:

| *Input Data* | *Ascii Values* | *Generated number* | *Number to be substracted* | *Cipher text* |
|---|---|---|---|---|
| *A* | *65* | *394* | *9* | *56* |
| *B* | *66* | *271* | *7* | *59* |
| *C* | *67* | *1018* | *1* | *66* |
| *D* | *68* | *1371* | *7* | *61* |
| *E* | *69* | *2885* | *8* | *61* |
| *F* | *70* | *5652* | *5* | *65* |
| *G* | *71* | *13318* | *1* | *70* |
| *H* | *72* | *30886* | *8* | *64* |
| *I* | *73* | *35215* | *1* | *72* |
| *J* | *74* | *39069* | *6* | *68* |

**(B) For 4 core machine:**

| *File Size* | *Serial Execution Time* | *Parallel Execution Time* | *Speed Up* |
|---|---|---|---|
| *1MB* | *0.33min* | *0.11min* | *3* |
| *10MB* | *1.5min* | *0.48min* | *3.12* |
| *50 MB* | *11.2min* | *2.71min* | *4.12* |
| *100MB* | *21.4min* | *3.4min* | *6.3* |
| *150MB* | *37.3min* | *4.72min* | *7.89* |

**(C )For 8 core machine:**

| *File Size* | *Serial Execution Time* | *Parallel Execution Time* | *Speed Up* |
|---|---|---|---|
| *1MB* | *0.33min* | *0.07min* | *4.67* |
| *10MB* | *1.5min* | *0.28min* | *5.2* |
| *50 MB* | *11.2min* | *1.49min* | *7.51* |
| *100MB* | *21.4min* | *2.4min* | *8.9* |
| *150MB* | *37.3min* | *3.49min* | *10.67* |

**(D) Performance Analysis:**

| *File Size* | *2 Core* | *4 Core* | *8 Core* |
|---|---|---|---|
| *1MB* | *1.67* | *3* | *5.67* |
| *10MB* | *1.81* | *3.12* | *6.2* |
| *50 MB* | *2* | *4.12* | *8.51* |
| *100MB* | *2.25* | *6.3* | *9.9* |
| *150MB* | *2.78* | *7.89* | *11.67* |

We can see that parallelised project gives better yield as far as time taken by system to execute.

**X- axis**: File Size **Y-axis**: Type of machine

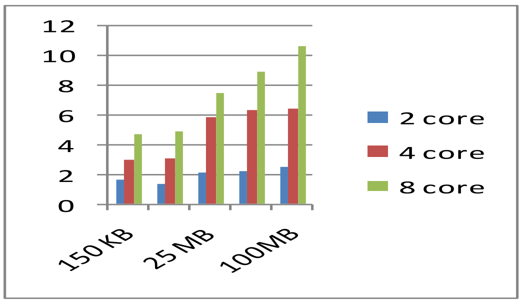

Fig 3 Graph showing the perfomance analysis

## V. FUTURE WORK

As we can see from the performance of our algorithm on different-different cores we can predict the future speed up for given large size of File.

**y axis** – File size

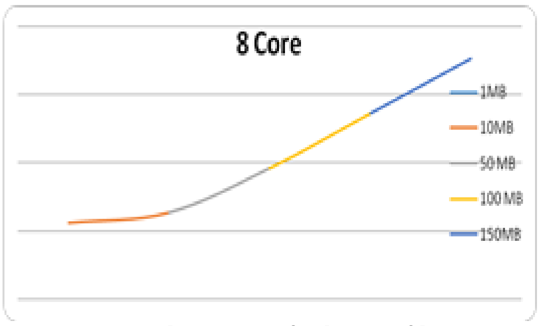

Fig 4 speed up Curve for known file size

In this curve, we can see that the speed up factor is seems to be increasing as there is increase in the file sizes in a constant growth rate. Now we are using logarithms for transfroming this equation into linear equation.

The linear equation is as follows,

$$Y = m + cX \qquad (2)$$

Where,
c = constant
m = speed up factor

**Projected values:**

**Table 3 projected values**

| *File Size* | *8 Core* | *Projected Values* |
|---|---|---|
| *1MB* | *5.67* | *5.67* |
| *10MB* | *6.4* | *6.4* |
| *50 MB* | *8.51* | *9.64* |
| *100 MB* | *9.9* | *13.69* |
| *150MB* | *11.67* | *17.7* |
| *200MB* | | *21.81* |
| *300MB* | | *29.92* |
| *400MB* | | *38.01* |
| *500MB* | | *46.13* |

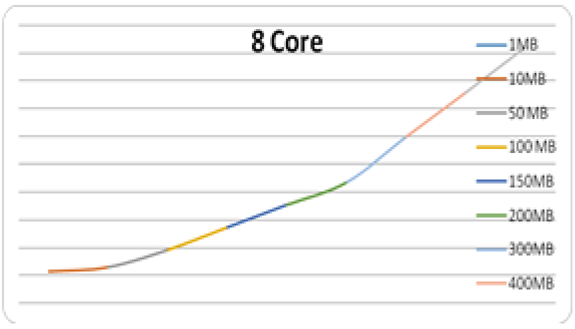

Fig 5 Speed up curve for projected values

## VI. PROOF

We can prove it by using mathematical formulas. Lets consider the speed up on 8 core machine for a given size file then it will follow the equation (2) i.e. $Y = c + mX$

**Solution:** We will use mathematical formulas for proving the above statement.

For 8 core machine,
Lets take initial values of X & Y
Y = 5.67 for file size of X = 1MB
Y= 6.4 for file size of X = 10MB
Putting this value in our equation,
We will get,
c = 5.58, m = 0.0811.
so now extending it to size of 100MB,150 MB it comes true.
For all values it will come true so we can say that this equation holds for all given size of file.

## VII. CONCLUSION

We can see that the genetic algorithm operator that we have used gives better performance as compared to other algorithm. On the off chance that we compose the same letter set in our message like ZZZZZZ still it will create diverse key and the worth for every letters in order. The quantity of times we iteratively rehash the methodology of hybrid & transformation we would be continue getting more secure encryption. Indeed fruitful parallelization yields better brings about terms of execution time. As we have not yet utilized all the operations of hereditary calculation in future we can utilized it show signs of improvement security.

## REFERENCES

[1] Benjamin Arazi, “Vehicular Implementations of Public Key Cryptographic Techniques”, IEEE TRANSACTIONS ON VEHICULAR TECHNOLOGY, VOL. 40, NO. 3,pp 646-653, AUGUST 1991.

[2] Murat Kantarcioglu, Wei Jiang, Ying Liu, and Bradley Malin, “A Cryptographic Approach to Securely Share and Query Genomic Sequences”, IEEE TRANSACTIONS ON INFORMATION TECHNOLOGY IN BIOMEDICINE, VOL. 12, NO. 5,pp 606-617, SEPTEMBER 2008.

[3] Jong-Bae Park, Young-Moon Park, Jong-Ryul Won, and Kwang Y. Lee, “An Improved Genetic Algorithm for Generation Expansion Planning”, IEEE TRANSACTIONS ON POWER SYSTEMS, VOL. 15, NO. 3, pp 916-922,AUGUST 2000.

[4] K. F. Man, K. S. Tang, and S. Kwong, “Genetic Algorithms: Concepts and Applications”, IEEE TRANSACTIONS ON INDUSTRIAL ELECTRONICS, VOL. 43, NO. 5,pp 519-534, OCTOBER 1996.

[5] R. H. Torres, G. A. Oliveira, J. A. M. Xexéo, W. A. R. Souza and R. Linden, “Identification of Keys and Cryptographic Algorithms Using Genetic Algorithm and Graph Theory”, IEEE LATIN AMERICA TRANSACTIONS, VOL. 9, NO. 2, APRIL 2011.

[6] Thang Nguyen Bui and Byung Ro Moon, “Genetic Algorithm and Graph Partitioning”, IEEE TRANSACTIONS ON COMPUTERS, VOL. 45, NO. 7,pp 841-855 JULY 1996.

[7] Yao-xue zhang, Kaoru Takahashi, Nor10 Shiratori, and Shoichi Noguchi, “An Interactive Protocol Synthesis Algorithm Using a Global State Transition Graph”, IEEE TRANSACTIONS ON SOFTWARE ENGINEERING, VOL. 14. NO. 3,pp 394-404, MARCH 1988.

[8] Sunar Mohammed Farooq K Nageswara Reddy “ Implementation of Intrusion Detection Systems For High Performance Computing Environment Applications. INTERNATION JOURNAL OF SCIENTIFIC ENGINEERING AND TECHNOLOGY RESEARCH ,Vol 4 issue 41, Oct 2015.

[9] Franciszek Seredynski and Albert Y. Zomaya, “Sequential and Parallel Cellular Automata-Based Scheduling Algorithms”, IEEE TRANSACTIONS ON PARALLEL AND DISTRIBUTED SYSTEMS, VOL. 13, NO. 10, pp 1009-1023,OCTOBER 2002.

[10] Yi-Ta Wu, and Frank Y. Shih, “Genetic Algorithm Based Methodology for Breaking the Steganalytic Systems”, IEEE TRANSACTIONS ON SYSTEMS, MAN, AND CYBERNETICS—PART B: CYBERNETICS, VOL. 36, NO. 1, pp 24-31,FEBRUARY 2006.

[11] Jun-Hong Chen, Ming-Der Shieh and Wen-Ching Lin, “A High-Performance Unified-Field Reconfigurable Cryptographic Processor”, IEEE TRANSACTIONS ON VERY LARGE SCALE INTEGRATION (VLSI) SYSTEMS, VOL. 18, NO. 8, pp 1145-1158,AUGUST 2010.