D-DOG: Securing Sensitive Data in Distributed Storage Space by Data Division and Out-of-order keystream Generation

[†]Jun Feng, [†]Yu Chen*, [‡]Wei-Shinn Ku, [§]Zhou Su
[†]Dept. of Electrical & Computer Engineering, SUNY - Binghamton, Binghamton, NY 13902
[‡]Dept. of Computer Science & Software Engineering, Auburn University, Auburn, AL 36849
[§]Dept. of Computer Science, Waseda University, Ohkubo 3-4-1, Shinjyuku, Tokyo 169-8555, Japan
{jfeng3, ychen }@binghamton.edu, weishinn@auburn.edu, zhousu@asagi.waseda.jp

***Abstract* - Migrating from server-attached storage to distributed storage brings new vulnerabilities in creating a secure data storage and access facility. Particularly it is a challenge on top of insecure networks or unreliable storage service providers. For example, in applications such as cloud computing where data storage is transparent to the owner. It is even harder to protect the data stored in unreliable hosts. More robust security scheme is desired to prevent adversaries from obtaining sensitive information when the data is in their hands. Meanwhile, the performance gap between the execution speed of security software and the amount of data to be processed is ever widening. A common solution to close the performance gap is through hardware implementation. This paper proposes D-DOG (*Data Division and Out-of-order keystream Generation*), a novel encryption method to protect data in the distributed storage environments. Aside from verifying the correctness and effectiveness of the D-DOG scheme through theoretical analysis and experimental study, we also preliminarily evaluated its hardware implementation.**

Keywords: Data Security, Distributed Storage, Stream Cipher, Encryption/Decryption.

I. INTRODUCTION

Data storage has been recognized as one of the main dimensions of information technology. The prosperity of network based applications leads to the moving from server-attached storage to distributed storage. Along with variant advantages, the distributed storage also poses new challenges in creating a secure and reliable data storage and access facility over insecure or unreliable service providers. Aware of that data security is the kernel of information security, a plethora of efforts has been made in the area of distributed storage security [7], [15], [19].

During past decades, most designs of distributed storage chose the form of either Storage Area Networks (SANs) or Network-Attached Storage (NAS) on the LAN level, such as a network of an enterprise, a campus, or an organization. Either in SANs or NAS, the distributed storage nodes are managed by the same authority. The system administrator has the access and control over each node, and essentially the security level of data is under control. The reliability of such systems is often achieved by redundancy, and the storage security is highly depending on the security of the system against the attacks/intrusion from outsiders. The confidentiality and integrity of data are mostly achieved using robust cryptograph schemes.

However, such a security system is not robust enough to protect the data in distributed storage applications at the level of wide area networks. The recent progress of network technology enables global-scale collaboration over heterogeneous networks under different authorities. For instance, in the environment of peer-to-peer (P2P) file sharing or the distributed storage in cloud computing environment, it enables the concrete data storage to be even transparent to the user [19]. There is no approach to guarantee the data host nodes are under robust security protection. In addition, the activity of the medium owner is not controllable to the data owner. Theoretically speaking, an attacker can do whatever he/she wants to the data stored in a storage node once the node is compromised. Therefore, the confidentiality and the integrity would be violated when an adversary controlled a node or the node administrator becomes malicious.

In the recent years, more and more scientific or enterprise applications have been developed based on the distributed data storage or distributed data computing techniques [9], [14], [15], [19], [20], [21]. Availability and performance are two of the most important metrics in these systems [24]. Data can be stored using encoding schemes such as short secret sharing, or encryption-with-replication. No matter which scheme is chosen, the cipher algorithm is either block cipher based or stream cipher based [8].

The general block cipher AES is designed mainly for the software application and is not effective for the hardware acceleration. Meanwhile, the general stream cipher schemes developed recently in eSTEAM project [5] follow two different directions. One is for the software application that emphasized the executing speed of software implementation. The other is hardware oriented, which focuses on the implementation on passive RFID tags or low-cost devices. For instance, the hardware security level for the profile 2 cipher was 80 bits [5], [11]. Although it may be adequate for the lower-security applications where low-cost devices might be used, it is not enough for the distributed storage network security application.

In this paper, we propose D-DOG (*Data Division and Out-of-order keystream Generation*), a high performance hardware implementation oriented stream cipher for distributed storage network. The D-DOG creates cipher blocks by dividing the plaintext data into multiple blocks and encrypting them, where the keystream is generated by

* Manuscript submitted on Sept. 10, 2009 to *the 2010 IEEE International Conference on Communications (ICC 2010)*, May 23 – 27, 2010, Cape Town, South Africa. Corresponding author: Yu Chen, Dept. of Electrical & Computer Eng., SUNY–Binghamton, Binghamton, NY 13902. E-mail: ychen@binghamton.edu, Tel.: (607) 777-6133, Fax: (607) 777-4464.

abstracting bits from the data blocks in a pseudorandom out-of-order manner.

The D-DOG avoids one of the weaknesses existing in modern stream ciphers resulted from the fixed length initialization vector (IV). Treating the data block as a binary stream, D-DOG generates the keystream by extracting *n* bits from the plaintext in a pseudorandom manner. The length of the keystream *n* is flexible and can be set according to different specific security requirements. The variable length keystream makes brute force attacks much more difficult. And the pseudorandom bit abstracting makes decrypted data stream still unrecognizable unless the keystream bits are inserted back to the original position.

The rest of the paper is organized as follows. Section 2 gives a brief overview of related work. Section 3 presents the principle of our D-DOG scheme and the detailed design is discussed in section 4. Section 5 illustrates the robustness of our design against some known attacks. Section 6 shows the simulation and experimental results. Section 7 summarizes this paper.

II. RELATED WORK

Securing sensitive and/or private data in distributed storage has been an important topic in security research community [6], [16], [20]. This section briefly overviews recent work in the modern stream cipher design area.

Stream ciphers are widely used to protect sensitive data at fast speeds [2], [22]. Although block ciphers have been attracting more and more attention, stream ciphers still are very important, particularly for military applications and to the academic research community. Stream ciphers are more suitable in environments where tight resource constraints are applied, i.e. in wireless mobile devices [3], [22], or wireless sensor networks [6]. When there is a need to encrypt large amount of streaming data, a stream cipher is preferred [2].

In recent years, a lot of efforts have been reported in this area and many interesting new stream ciphers have been proposed and analyzed. A popular trend in stream cipher design is to turn to block-wise stream ciphers like RC4, SNOW 2.0, and SCREAM [13]. In order to improve the time-data-memory tradeoff for stream cipher, a concept of Hellman's time-memory tradeoff [3] has been applied and it achieved obvious improvements [10]. The Goldreich-Levin [9] one-way function hard-core bit construction has been enhanced into a more efficient pseudo-random number generator BMGL [12] with a proof of security.

Efficient hardware implementations of stream ciphers are important in both high-performance and low-power applications [13]. This is the main trend of the stream cipher development in the future. Researchers have pointed out that RFID (Radio Frequency Identification) could be one of the next killer applications for hardware-oriented stream ciphers [22]. The second phase of the eSTREAM project in particular focused stream ciphers suited toward hardware implementation and currently there are eight families of hardware-oriented stream ciphers [5].

Normally there are two input parameters to a stream cipher, the password and an initialization vector (IV). In contrast with the user password being kept secret, the IV is public. As a consequence, attacks against the IV setup of stream cipher have been very successful [25]. Due to the weakness with the IV setup, more than 25% of the stream ciphers submitted to the eSTREAM project in May 2005 have been broken [1]. Some robust academic designs were broken also due to problems with the IV setup [25].

In this paper, we will investigate an alternative design approach for the self-encryption stream cipher scheme to avoid the shortcomings incurred by using public IV. Also, the robustness of a fixed length keystream has been weakened as the computing power which an adversary possesses has been growing. Instead, a variant length keystream will make brute force attacks computationally infeasible. To reach this goal, this paper will also introduce a novel keystream generation scheme.

III. D-DOG SCHEME RATIONALE

This section first introduces the idea of divide-and-store for confidentiality and integrity of the sensitive data in a distributed storage environment. Then an overview of the encryption/decryption operation of the D-DOG scheme is presented. The design details of the scheme are discussed in the next section.

A. Divide-and-Store Principle

The major design goal of the D-DOG strategy is the confidentiality and integrity of the sensitive/privacy data that is stored in the Internet based distributed storage infrastructure such as Grid Storage or Cloud Computing, where the data owner can control neither the reliability/security of the medium, nor the violation of the medium provider or administrators. Either the medium providers or an adversary who has successfully compromised a storage node could do whatever he/she wants to the data in the machine. Therefore, our purpose is to make it computationally infeasible to reveal any meaningful information from each ciphertext pieces.

Figure 1 illustrates the basic divide-and-store principles of the D-DOG scheme. We don't expect much from the remote nodes beyond the role of a storage medium provider.

At the local user side, the major functions include the following. When a data file is stored:

1) Generating the IV using three elements: the PIN from the user, the nonce generated by the system and the bits abstracted from the plaintext;
2) Constructing the keystream for encryption;
3) Encrypting the remained plaintext using the keystream;
4) Dividing the cipher text into multiple data blocks with fixed-length, the last block will be stuffed if it consists of fewer bits;
5) Allocating storage nodes in the network and sending each block to one of them;
6) Storing the PIN & keystream, and publish nonce.

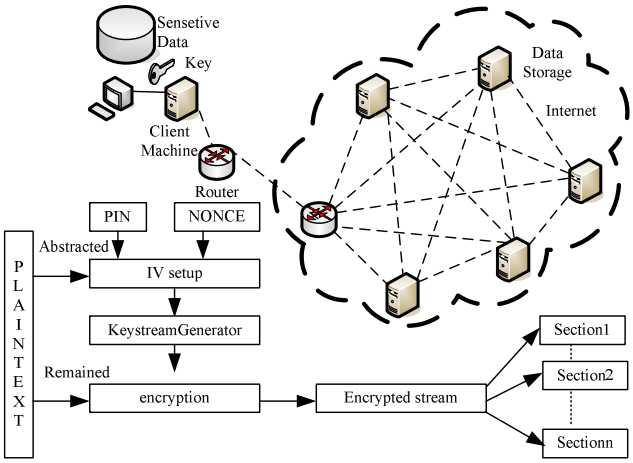

Figure 1. Illustration of the D-DOG scheme principle.

B. Basic Operations of the D-DOG Scheme

Similar to any other stream cipher, the D-DOG also encrypts the plaintext and decrypts the ciphertext by performing the bitwise adding calculation with a keystream:

$$Ciphertext = Plaintext \oplus Keystream \qquad (1)$$

As illustrated in Fig. 1, the D-DOG scheme has three inputs: the plaintext, the user key and a nonce. The plaintext is used as one of the inputs to generate the keystream. Therefore, extra efforts must be considered carefully for the requirements raised by software computation and hardware parallel processing.

Figure 2 and Figure 3 present the flowcharts of the encryption operation and decryption operation respectively. The plaintext is inputted into the Separation module, which takes the pseudo-random stream generated by RandomAddrGen1 as the address index and draws the corresponding bits from the plaintext. KeystreamGen makes use of the Key and the data from IV Initial as input and output keystream. The Separation module outputs two separate streams: stream1 and stream2. Then, they are encrypted by the exclusive or with the keystream

After the stream1 and stream2 are encrypted, they can be combined together, or they can be sent out directly. As marked with a shadowed background, both the Combining module and the RandomAddrGen2 are optional. If the combining module is used, the RandomAddrGen2 is used to produce the address for the bit insertion operation. For module Keystream Generator2, there are three encryption pseudo-random generator schemes according to the Figure 2, which are corresponding to different decryption methods, as shown in Figure 3.

Method (a): Usually, under the secure data storage architecture, we can adopt any so-far unbroken cipher algorithm into the module, including the block cipher. If the path (a) is removed from Figure 2, the D-DOG cipher scheme is no different from the normal encryption algorithm, except the separate module which introduces extra attack complexity tremendously.

Method (b): When path (a) is introduced into the Keystream Generator2, if the IV/nonce path (b) is disconnected, the decryption scheme should be different from what the scheme (a) is. When path (b) is removed, the plaintext decrypted from Keysteam3 is used as the IV/nonce, the plaintext from path (a) and Key from the user enters the Keystream Generator2 and generates the corresponding keystream to decrypt the Text1. The scheme provides more robust solution than (a).

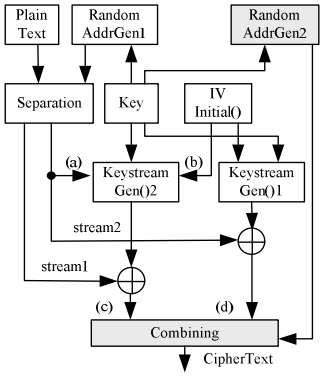

Figure 2. Flowchart of Encryption operation

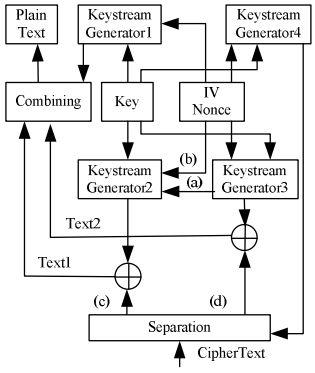

Figure 3. Flowchart of Decryption operation

Method (c): When both path (a) and path (b) in Figure 3 are considered, it can provide the same safe level of the keystream as the method (b), since the attacker cannot re-use the nonce for a replay attack. And method (c) can achieve more safety in the IV setup than method (b).

Summary: The main difference between methods (a) and (b) is that path (a) does not consider the operation performed by Keystream Generator3, and this leads to more flexibility. For methods (b) and (c), the designer has to consider the inter-communication between the Keystream Generator2 and Keystream Generator3 module. Especially when the algorithm is implemented in hardware, the time slot should be considered carefully.

IV. D-DOG DESIGN

This section first gives the preliminaries for the cipher construction, and then we present the detailed design of the novel D-DOG scheme.

A. Preliminaries

$u + v$: the addition operation, which means $u + v \bmod 2^{32}$;

$u \oplus v$: the bitwise exclusive-or of two words u and v;

{a, b}: the cascadence, for example, {4'he, 4'hf} = 8'hef;

$u <<< c$: c bit left rotation of a word u;

$u >>> c$: c bit right rotation of a word u, for example, 8'b10011110>>>2 is 8'b10100111;

We define four tables, P, Q, M, N to provide variables to the internal state, which are all 512 32bit words;

K: the 256 bit key;

IV: the 256 initialization vector of the cipher;

T: the 256 bit plaintext;

S: the keystream generated by the gen module;

$f(x) = (x>>>7) \oplus (x>>>13) \oplus (x>>>11) \oplus (x>>>18)$;

$g(x,y,z) = (x>>>7) \oplus (y>>>18) \oplus z(x \oplus y)$,

where *z* can be either of *P*, *Q*, *M*, *N*;

$h(x,X) = X(x0) + X(x1) + X(x2) + X(x3)$, where $x = \{x3, x2, x1, x0\}$;

random(i) is a pseudo-bitstream generator as a two-bits counter, which is determined by the number *i*.

B. Major Procedures Description

The encryption operation consists of three procedures: IVInitial(), KeystreamGen(), RandomAddrGen(). This subsection discusses the design and operation of four major procedures in detail.

1) IV Initial()

Initialization procedure is used to expand the key, the *IV* and the plaintext into the tables *P*, *Q*, *M*, *N*. The operations from a) to d) are shown below.

a. $K = \{K0, K1, \ldots, K7\}$, $IV = \{IV0, IV1, \ldots, IV7\}$, $T = \{T0, T1, \ldots, T7\}$, where *Ki*, *IVi*, *Ti* means a 32bit number;

b. Generate *W* in the following ways:

Initialize:

$Wi = Ki$ when $0 <= i < 8$;
$Wi = IVi - 8$ when $8 <= i < 16$;
$Wi = Ti - 16$ when $16 <= i < 24$;
$Wi = 0$ when $24 <= i < 2048$;
$Wi = f(Wi-11) + f(Wi-13) + f(Wi-7) + f(i-20)$ when $24 <= i < 2048$.

Then, we define procedure1 as follows:

$Wi = f(Wi-11) + f(Wi-13) + f(Wi-7) + f(i-20)$ when $0 <= i < 2048$.

The minus operation is on the module *2048*, for example, *i-11* means *i-11* mod (*2048*).

Run the procedure1 *1024* steps.

c. Initialize *P*, *Q*, *M*, *N* by the following procedure.

$P(i) = W(i)$,
$Q(i) = W(i+512)$,
$M(i) = W(i+1024)$,
$N(i) = W(i+1536)$, when $0 <= i < 512$

d. Initialize the internal state by the following procedure.

Run the keystream generation *2048* rounds without the output. Or Run the procedure1 *2048* rounds.

2) KeystreamGen()

The user is allowed to pick up keys and IVs and put them into the keystream generator modules respectively. The keys can be the same, but the IVs should not be the same. It is capable of expanding the key and initialization vector into the internal state more randomly, and it achieves certain synchronization between the sender and the receiver. Different from other stream cipher design, the plaintext is inputted to the setup phase to generate inner state for the keystream generation if needed.

The *P*, *Q*, *M*, *N* tables are changed every step. And all the elements in the table will be renewed in 512 rounds. The parallel pseudo code is run as follows:

Do begin
  $j = random(P(0))$;
  For $i = 0$ *to* 511 *do begin*
    $P(i) = P(i) + P(i+128) + g(P(i+256),P(i+384)),P)$;
    $Q(i) = Q(i) + Q(i+128) + g(Q(i+256),Q(i+384),Q)$;
    $M(i) = M(i) + M(i+128) + g(M(i+256),M(i+384),M)$;
    $N(i) = N(i) + N(i+128) + g(N(i+256),N(i+384),N)$;
    $X = P(if\ j = 0)\ or\ Q(if\ j = 1)\ or\ M(if\ j = 2)\ or\ N(if\ j = 3)$;
    $S = h(X(i) + X(i+37),X)$;
  End
End while (*the keystream bits length is the same as the plaintext*)

3) RandomAddrGen()

The *RandomAddrGen* generates a random index, here we takes the user's PIN and a nonce as input and the output is an integer *seed*, which is used as the seed of the random number generator *G*. The output random number sequence $\{r_0, r_1, \ldots, r_{n-1}\}$ indicates which bits are selected and abstracted from the message (plaintext) to form the keystream. Therefore, we have:

$$seed = F(PIN, nonce) \quad (2)$$

$$\{r_0, r_1, \ldots, r_{n-1}\} = G(seed) \quad (3)$$

Where $\{r_0, r_1, \ldots r_{n-1}\}$ is a random number sequence generated continuously by *G*. Here we adopt a continuous add modulo method to avoid collision and out-of-bound problem, which is:

$$r'_k = (r_k + r_{k-1}) \bmod (m-k) \quad (4)$$

Another advantage of this simple algorithm is that it raises the bar of the brute-force attack and can be easily and quickly implemented by hardware.

V. ROBUSTNESS ANALYSIS

This section briefly discusses the robustness of the D-DOG scheme against some of the well known attacks.

Period Attack: For D-DOG cipher, the 65536 internal states ensure that the period of the keystream is extremely large. Because of the fact that the internal state evolves in a nonlinear way, its period is hard to determine. But we can get the average period of the keystream is estimated to be about 2^{65535}, if we assume that the invertible next-state function of D-DOG cipher is random.

Linear Relations Attack: The large secret table of D-DOG cipher is updated during the keystream generation process, so it is extremely difficult to develop linear relations linking the input and output bits of the table.

Brute-Force Attack: Brute force attacks are observed very often. The internal state of the D-DOG cipher is about 65536 bit, and the average period is about 2^{65535}, which is enough to resist any brute-force attack so far. In addition, since the D-DOG cipher use the highly-nonlinear feedback in the keystream generation, the period of the keystream is

variable, which make any attempt that is to attack the stream generated by the separate module is unavailable.

Time-Memory-Data Tradeoff Attack: The cost of time/memory/data tradeoff attacks on stream ciphers is $O(2^{n/2})$, where n is the number of inner states of the stream cipher. Due to the choice of the length of the inner state, the time-memory-data tradeoffs attacks costs is $O(2^{32767})$, which means it is impracticable to execute such method.

Algebraic Attack: The idea of an algebraic attack is to come up with a "small" set of equations satisfied by input states, output states, and unknown intermediate states, and then solve the equations, or, for a distinguisher, see whether the equations have a solution. However, it is very challenging to apply algebraic attacks to recover the secret key because the output and feedback functions of D-DOG cipher are highly non-linear.

Correlation Attacks: In order to find a relevant correlation in the cipher, the following questions can be addressed: Is there a linear relation at bit level between some input and output bits? Is there a particular relation between some input bit vector and some output bit vector? However, because the output and feedback functions of our D-DOG cipher are highly non-linear, it is very hard to apply the correlation attacks to recover the secret key.

Differential Analysis Attacks: The idea of a differential attack is that some "small" differences in input states have a perceptible chance of producing "small" differences after the first step of the computation, the second step of the computation, etc. However, the D-DOG cipher uses the 32-to-32-bit mapping similar to that being used in Blowfish and rotation method to diffuse the small difference into the whole table, which leads to the large difference in the output. Therefore, it is difficult to guess the key by the differential attacks.

VI. EXPERIMENT AND SIMULATION

The D-DOG cipher operation would be tedious and time-consuming to the data owner. It would be preferred if the whole operation can be done automatically with very low time overhead. In this paper, therefore, from the users' perspective, we proposed to implement the whole D-DOG operation in an embedded accelerator using reconfigurable hardware devices such as FPGAs (Field Programmable Gate Array). This accelerator pushes the job down to the lower layer of the data communication protocol set and makes it transparent to applications.

For the convenience of hardware implementation, we divided the data file into fixed-length blocks. In fact, storing fixed-length block at each node makes it more difficult for adversaries to get useful information to reassemble and/or decrypt the ciphertext blocks.

In order to evaluate the performance and the correctness of this design, we implemented the D-DOG cipher algorithm by Modelsim and Synplify on Altera CycloneII FPGA. Since it is merely a prototype, we use the same keystream generator module to generate the random stream sequence, and use a fixed key and nonce as the input. The FPGA selected is Altera CycloneII EP2C20F484C8. Modelsim version is 6.2g, and Synplify version is 8.5, Quartus version is 7.2.

Table 1 Comparison between D-DOG and other ciphers

Cipher	Key Size (bit)	Frequency (MHz)	SLICE	Memory (bit)	Data Width (bit)	Throughput (Mbps)
D-DOG	**256**	**178**	**151**	**49152**	**8**	**1424**
AES128	**128**	**130**	**595**	**32768**	**16**	**208**
Trivium*	**80**	**207**	**41**	**-**	**1**	**207**
Grain128*	**128**	**181**	**48**	**-**	**1**	**181**
MICKEY 128*	**128**	**200**	**190**	**-**	**1**	**200**

* Data comes from reference paper [4], however, the keysize of Grain and Mickey becomes 80 bit in the final eStream portfolio due to the hardware environment constraint, such as RFID Tag.

For comparison, we also implemented the AES with 128bit level into the FPGA chip with Synplify, the following table is the Quartus result comparison. We select AES cipher and the three stream cipher in the final portfolio. We compare our D-DOG scheme with AES for two reasons. First, AES is the one of the most popular ciphers used today, and many hardware storage adopt AES as their crypto method, such as Seagate Inc. The second reason is that there is no standard stream cipher to compare and eStream project has been using AES as the reference to evaluate newly developed stream ciphers [5].

As shown in the table, the estimated executing frequency of the D-DOG on FPGA device is 178 MHz and the throughput is 1424Mbps. Compared with AES, it is a light-weighted design since much less hardware resources are consumed. Although D-DOG consumes more resources than Grain and Trivium, its application environment focuses on the throughput and Key Size instead of resources. The result in Throughput column indicates that the D-DOG outperformed the others, including the AES.

(a) Original image **(b) Output of the encryption**

Figure 4. Illustration of the SE encryption effects

To verify the effectiveness of the D-DOG encryption, Figure 4(a) is chosen as the original example data need to be protected and stored in the distributed storage space. Figure 4(b) presents the output of the cipher process. Obviously, the D-DOG cipher effectively scrambled the original image

to a random looking un-recognizable image. Then, the output has been used as input of the decryption operation. The original image has been recovered successfully.

VII. CONCLUSIONS

In this paper, we propose D-DOG, a novel steam cipher encryption for data security in distributed storage. We verified the correctness and effectiveness of the D-DOG encryption scheme through simulation and synthesis on top of reconfigurable hardware devices (FPGAs). The result is encouraging since we are considering implementing the D-DOG operations using FPGAs due to the overhead and computing power requirement. We are going to push this task to lower layer of data processing. Hence, the operations including encryption, decryption, data division and reassembly are transparent to the higher layer application programs and users.

In fact, what reported in this paper is not complete enough to be implemented in practice. More works have been scheduled. First of all, the D-DOG cipher will be tested on the NIST Test Suite [18], the DIEHARD battery of tests [17], the ENT test [23] and the eSTREAM tests [5]. Tests will be performed on the internal state as well as on the extracted output. Furthermore, more various statistical tests on the key setup function will be conducted.

Furthermore, we will study the performance in the context of real applications. A testbed consisting of 16 nodes is under construction at Binghamton University supported by the DURIP award. Each node is equipped with a NetFPGA board that provides four network interfaces with a gigabyte data rate. A reconfigurable prototype of the D-DOG scheme will be implemented on top of the new testbed. We will explore the tradeoffs between the performance and resource utility by the D-DOG system.

REFERENCES

[1] D. J. Bernstein, "Which eSTREAM ciphers have been broken?" http://www.ecrypt.eu.org/ stream/, submitted 2008.

[2] A. Biryukov, "Block Ciphers and Stream Ciphers: The State of the Art," Lecture Notes in Computer Science, in Proceedings of the COSIC Summer course, 2003.

[3] A. Biryukov and A. Shamir, "Cryptanalytic time /memory /data tradeoffs for stream ciphers," in Proceedings of Asiacrypt'00, no. 1976 in Lecture Notes in Computer Science, pp. 1–13, Springer-Verlag, 2000.

[4] Philippe Bulens, Kassem Kalach, Fran¸cois-Xavier Standaert and Jean-Jacques Quisquater, "FPGA Implementations of eSTREAM Phase-2 Focus Candidates with Hardware Profile", www.ecrypt.eu.Org/stream/papersdir /2007/024.pdf

[5] eSTREAM Project, http://www.ecrypt.eu.org/stream.

[6] N. Fournel, M. Minier, and S. Ubeda, "Survey and Benchmark of Stream Ciphers for Wireless Sensor Networks," the *Workshop in Information Security Theory and Practices* (WISTP'07), Crete, Greece, May 8-11, 2007.

[7] J. A. Garay, R. Gennaro, C. Jutla, and T. Rabin, "Secure Distributed Storage and Retrieval," in Theoretical Computer Science, 1997.

[8] G. A. Gibson and R. V. Meter, "Network Attached Storage Architecture," Communications of the ACM, 43(11): 37 – 45, 2000.

[9] O. Goldreich and L. A. Levin, "A hard core predicate for any one way function," in Proceedings of Symposium on Theory of Computing – STOC'89 , pp. 25–32, ACM Press, 1989.

[10] J. D. Golic, "Cryptanalysis of alleged A5 stream cipher," in *Advances in Cryptology – EUROCRYPT'97*, vol. 1233 of *Lecture Notes in Computer Science*, pp. 239–255, edited by W. Fumy, Springer-Verlag, 1997.

[11] T. Good and M. Benaissa, "Hardware performance of eSTREAM phase-III stream cipher candidates," the State of the Art of Stream Ciphers Workshop (SASC'08), Lausanne, Switzerland, Feb. 13-14, 2008.

[12] J. Hastad and M. Naslund, "Improved analysis of the BMGL keystream generator," in *Proceedings of the Second NESSIE Workshop*, 2001.

[13] D. Hwang, M. Chaney, S. Karanam, N. Ton, and K. Gaj, "Comparison of FPGA-Targeted Hardware Implementations of eSTREAM Stream Cipher Candidates," the State of the Art of Stream Ciphers Workshop (SASC'08), Lausanne, Switzerland, Feb. 13-14, 2008.

[14] G. S. Jedhe, A. Ramamoorthy, and K. Varghese , "A Scalable High Throughput Firewall in FPGA", In 16th International Symposium on Field-Programmable Custom Computing Machines , pp43-52, 2008.

[15] V. Kher and Y. Kim, "Securing Distributed Storage: Challenges, Techniques, and Systems," StorageSS'05, Fairfax, Virginia, USA, Nov. 11, 2005.

[16] P. Kocher, J. Jaffe and B. Jun, "Differential power analysis", *Advances in Cryptology* (Crypto '99), Lecture Notes in Computer Science, 1666 (1999), Springer-Verlag, 388-397.

[17] G. Masaglia. "Die Hard: A battery of tests for random number generators," http://www.stat.fsu.edu/pub/diehard/.

[18] National Institute of Standards and Technology, "A statistical test suite for the validation of random number generators and pseudo random number generators for cryptographic applications," NIST Special Publication 800-22, http://csrc.nist.gov/rng, 2001.

[19] R. Ranjan, A. Harwood, and R. Buyya, "Peer-to-Peer-based Resource Discovery in Global Grids: A Tutorial," IEEE Communications Surveys & Tutorials, Vol. 10, No. 2, 2nd Quarter, 2008.

[20] D. Saha, A. Mukherjee, "Pervasive Computing: A Paradigm for the 21st Century," IEEE Computer, IEEE Computer Society Press, pp. 25-31, March 2003.

[21] P. E. Sevinc, M. Strasser, and D. Basin, "Securing the Distribution and Storage of Secrets with Trusted Platform Modules," Workshop in Information Security Theory and Practices (WISTP'07), Crete, Greece, May 8-11, 2007.

[22] A. Shamir, "Stream Ciphers: Dead or Alive?" invited talk, ASIACRYPT 2004, Jeju Island, Korea, Dec. 5-9, 2004.

[23] J. Walker. "A pseudorandom number sequence test program," http://www.fourmilab.ch/random.

[24] Y. Ye, I. Yen, L. Xiao, and B. Thuraisingham, "Secure, Highly Available, and High Performance Peer-to-Peer Storage Systems", in *11th IEEE High Assurance Systems Engineering Symposium*, pp383-391, 2008.

[25] E. Zenner, "Why IV Setup for Stream Ciphers is Difficult," in *Proceedings of Dagstuhl Seminar on Symmetric Cryptography*, Jan. 2007.