International Journal of Science and Research (IJSR)
ISSN (Online): 2319-7064
Impact Factor (2012): 3.358

Efficient and Secure Data Sharing By Applying AES Algorithm with Anonymous Id Assignment

Barla Rakesh[1], Veladanda RamaKrishna[2]

[1]M.tech student, Department of CSE, Anurag Group of Institutions, Hyderabad, India

[2]Assistant Professor, Department of CSE, Anurag Group of Institutions, Hyderabad, India

Abstract: ***Security is a basic requirement of an organization in the world to keep their information secure from their competitors. Various techniques and algorithms were developed by research in order to achieve secure data sharing. We propose a technique for anonymous sharing of private data between N parties is developed. This technique is used to allocate these node ID numbers ranging from 1 to N and also apply encryption on private data. These assignments are anonymous in that the identities received are unknown to the other members of the group. Animosity between other members is verified in an information theoretic sense when private communication channels are used. This type of serial numbers assignment allows more complex data to be shared and has applications to other problems in privacy preserving data mining, animosity avoidance in communications and distributed database access. The prescribed computations are distributed without using a trusted third party central authority. Existing and new techniques for assigning anonymous IDs are examined with respect to trade-offs between communication and computational requirements. The proposed technique also finds distributed environment with minimal communication among parties and ensures higher degree of privacy with Advanced Encryption Standard (AES). It generates more secured item sets among multiple parties without affecting mining performance and optimal communication among parties with high privacy and zero percentage of data leakage. The new techniques are built on top of a secure sum data mining operation using Newton's identities and Sturm's theorem.***

Keywords: Anonymization and deanonymization, multiparty computation, privacy preserving data mining, secure sum algorithm, Advanced Encryption Standard (AES).

1. Introduction

Privacy is essential to trusted collaboration and interactions to protect against malicious users and falsified activities. Privacy is much needed to protect source of information, destination of information, route of information transmission and the information content itself. Security is a basic requirement of an organization in the world to keep their information secure from their competitors. A variety of techniques and algorithms were developed by research in order to achieve secure data sharing. Main aim is to protect the data from hacker during data sharing. Popularity of internet as a communication medium whether for personal or business use depends in part on its support for anonymous communication. Businesses also have lawful reasons to engage in anonymous communication and avoid the consequences of identity exposure. For example, to allow dissemination of summary data without revealing the identity of the entity the underlying data is associated with, to protect whistle-blower's right to be anonymous and free from political or economic retributions [1]. Cloud-based website management tools [2] provide capabilities for a server to anonymously capture the visitor's web actions. The major problem of sharing privately held data so that the individuals who are the subjects of the data cannot be identified has been researched extensively [3]. Researchers have also investigated the relevance of anonymity and/or privacy in various application domains: patient medical records, e-mail, social networking [8], etc.

A different form of anonymity, as used in secure multiparty computation, allows multiple parties on a network to jointly carry out a global computation that mainly depends on data from each party while the data held by each party remains anonymous to the other parties [3]. The secure computation function widely used in the literature is secure sum that allows parties to compute the sum of their individual inputs without disclosing the inputs to one another. This type of function's are popular in data mining applications and also helps characterize the complexities of the secure multiparty computation [3].

This work deals with efficient techniques for assigning identifiers (IDs) [10] to the nodes of a network in such a way that the IDs are anonymous using a distributed computation without using a trusted third party central authority. Given N nodes, this assignment is basically an incarnation of the integers {1……N} with each ID being known only to the node to which it is assigned. Proposed main technique is based on a method for anonymously sharing simple data and resulting methods for efficient sharing of complex data. We are having so many applications that require dynamic unique IDs for network nodes. Those IDs can be used as part of schemes for sharing/dividing data storage, communications bandwidth, and other resources anonymously and without inconsistency. These IDs are needed in sensor networks for security or for administrative tasks requiring consistency, such as configuration and monitoring of single and individual nodes, and download of binary code to these nodes. Differentiate anonymous ID assignment from anonymous communication; consider a situation where N parties wish to display their data in the form of cipher text collectively, but anonymously, in N slots on a trusted central authority third party site. These IDs can be used to assign the N slots to users [10], while anonymous communication can allow the parties to cover their identities from the third party.

2. Literature Survey

Bruce Schneier [11] proposed an algorithm based upon permutations and substitutions. That algorithm supports fixed key size approach to secure the data. It is Specification for Advanced encryption standard [12] provides the whole mechanism for encryption algorithm for which Rijmen provided the concept of AES. It mainly specifies the Rijndael algorithm a symmetric block cipher that can process data blocks of 128 bits, by using cipher keys. However, the lengths of Rijndael, which are required to handle additional key lengths and block sizes, are not adopted in this following standard.

This Algorithm provides flexibility to user to choose different key sizes. By using this algorithm throughput increases with increase in frequency level but the processing time is reduced at high frequency level. In this algorithm round time is the function of processing time so with frequency it also reduces. In this Rijndael provides a security equivalent to RSA of 3072 bit key and also overcome the drawbacks of DES and TDES. It is also concluded that if the throughput is increased then the same algorithm can be implemented on optical networks.

3. Secure Multiparty Computation

3.1 Motivation and Highlights

The Distributed computing considers has the scenario where a number of distinct, yet connected, computing parties wish to carry out a joint computation of some function. For example, these parties may be servers can who hold a distributed database system, and the function should to be computed may be a database update of some kind. The aim of *secure multiparty computation [3]* is to enable parties to carry that distributed computing tasks in a secure manner. at the same time as distributed computing classically deals with questions of computing under the threat of machine crashes and other accidental faults, for that secure multiparty computation is concerned with the possibility of cautiously malicious behavior by some adversarial entity. It is assumed that a protocol execution may come under "attack" by an external entity of the participating parties.

3.2 Security in Multiparty Computation

The parties under the control of the adversary are called corrupted, and follow the adversary's instructions. Here secure protocols should withstand any adversarial attack. In order to formally claim and prove that a protocol is secure, for that a precise definition of security for multiparty computation is required. A number of dissimilar definitions have been proposed and these definitions aim to ensure a number of important security properties that are general enough to capture most multiparty computation tasks. We now express the most central of these properties:

1) ***Privacy:*** None of any party should learn anything more than its prescribed output. In exacting, the only information that should be learned about other parties' inputs is what can be derived from the output itself.
2) ***Correctness:*** Each and every party is guaranteed that the output that it receives is correct.
3) ***Independence of Inputs:*** In this Corrupted parties must choose their inputs independently of the honest parties' inputs. Here we note that independence of inputs is *not* implied by privacy. For example, may be possible to generate a higher bid without knowing the value of the original one.
4) ***Guaranteed Output Delivery:*** Here corrupted parties should not be able to prevent honest parties from receiving their output. In other words, an adversary should not be able to disrupt the computation by carrying out a "denial of service" attack.
5) ***Fairness:*** In this corrupted parties should receive their outputs if and only if the honest parties also receive their outputs.

Adversarial Power:

Above informal definition of security omits one very important issue: the power of the adversary [2] can attack a protocol execution. Already we have mentioned, the adversary controls a subset of the participating parties in the protocol. On the other hand, we have not described the corruption strategy (i.e., How or when parties come under the "control" of the adversary), the allowed adversarial behavior (i.e., adversary does the passively gather information or can it instruct the corrupted parties to act maliciously), and the complexity of adversary is assumed to be (i.e., is it polynomial-time or computationally unbounded).Now we describe the main types of adversaries that have been considered:

1. ***Corruption strategy:*** Mainly corruption strategy [2] deals with the question of when and how parties are corrupted. We have two main models:

(a) **Static corruption model:** In this model, the adversary is given a fixed set of wanted parties .Here honest parties remain honest throughout, while corrupted parties remain corrupted.

(b) **Adaptive corruption model**: Quite than having a fixed set of corrupted parties, adaptive adversaries are given the capability of corrupting parties during the computation. If the choice of who to corrupt and when can be arbitrarily decided by the adversary and may depend on its view of the execution; for this reason, it is called adaptive. This type of strategy models the threat of an external "hacker" breaking into a machine during an execution. Here we note that in this model, one time a party is corrupted, it leftovers corrupted from that point on.

2. ***Allowed adversarial behavior:*** Another limitation that must be defined relates to the actions that corrupted parties are allowed to take. Again there are two main types of adversaries:

(a) **Semi-honest adversaries:** Semi-honest [2] adversarial model, even corrupted parties correctly follow the protocol specification. On the other hand, the adversary obtains the internal state of all the corrupted parties (including the transcript of all the messages received), and attempts to use this to learn information that should remain private. It is a rather weak adversarial model. Yet, there are some settings where it can realistically model threats to the system. The Semi-honest

adversaries are also called "honest-but-curious" and "passive".

(b) **Malicious adversaries:** In this model, the corrupted parties can *arbitrarily* deviate from the protocol specification, according to the adversary's directions. The common, providing security in the presence of malicious adversaries is ideal; it ensures that no adversarial attack can succeed. In this malicious adversaries are also called "active".

The above distinction regarding the complexity of the adversary yields two very different models for secure computation: the *information-theoretic* [9] model and the *computational* model. The information-theoretic setting, Here the adversary is not bound to any other complexity class (and in particular, isn't assumed to run in polynomial time). Thus, results in this model hold unconditionally and do not rely on any complexity or cryptographic hypothesis. The only hypothesis used is that parties are connected via ideally *private* channels (i.e., it is assumed that the adversary cannot eavesdrop or interfere with the communication between honest parties).

A. Shamir, [5] proposed a scheme to" How to share a secret" is called as Shamir's secret key sharing it is not secure against cheating. He do a small modification to his scheme retains the security and competence of the original, secure at the side of cheating, and preserves the property that its security does not depend on any unproven assumptions such as the intractability of computing number theoretic functions.

Here we concentrate on a particular example scheme that will be sufficient for our purposes in this chapter, namely Shamir's secret sharing scheme. This Shamir's scheme is based on polynomials [8] over a finite field F. The only necessary restriction on F is that |F| > n, however we will assume for concreteness and simplicity that F = Zp for some prime p > n. A value s ∈ F is shared by choosing a random polynomial fs(X) ∈ F[X] of degree at most t such that fs(0) = s. Then sending privately to player Pj the share sj = fs(j). The basic facts about this method are that any set of t or fewer shares contain no information on s, whereas it can easily be reconstructed from any t+1 or more shares

A Review of Secure Sum

Secure sum is frequently given as a simple example for secure multiparty computation. It here because of its applicability to data mining and because it demonstrates the difficulty and nicety involved in making and proving a protocol secure [10]. Distributed data mining algorithms often calculate the sum of values from individual sites. Imagining three or more parties and no collusion, the following scheme securely computes such a sum.

Assume that the value $v = \sum_{l=1}^{s} vl$ to be computed is known to lie in the range [0..n]. One site is chosen as the master site, numbered 1. The remaining all other sites are numbered 2..s. Site 1 generates a random number R, uniformly chosen from [0..n]. Sites 1 adds this to its local value v1 and immediately apply encryption on that value v1, and sends the sum R + v_1 mod n to site 2. Since the value R is chosen uniformly from [1...n], the number R + v_1 mod n is also distributed uniformly across this region, so site 2 has not learns anything about the actual value of v_1.

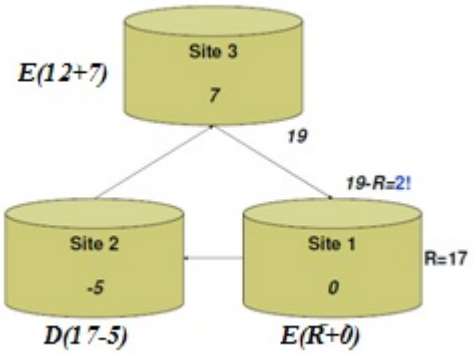

Figure 1: Secure computation of a sum

For the remaining sites *l*= 2..s — 1, the algorithm is as follows. Site *I* receives

$$V = R + \sum_{j=1}^{l-1} v_j \bmod n.$$

Since this value is uniformly distributed across [1...n], i learns nothing. Site i then compute and passes it to site *l* + 1.

$$R + \sum_{j=1}^{l} v_j \bmod n = (v_j + V) \bmod n$$

Site s performs the above step, and then sends the result to site 1. Site 1, knowing R, can subtract R and Apply decryption to get the actual result. Reminder that site 1 can also determine $\sum_{i=2}^{s} vl$ by subtracting vi. It is possible from the global result regardless of how it is computed, so site 1 has not learned or not gets anything from this computation. Figure 1 depicts how this scheme operates. This scheme faces an noticeable problem if sites collude. Sites *l* — 1 and *l* + 1 can evaluate the values they send/receive to determine the exact value for vi. This scheme can be extended to work for an honest majority. Each site divides vi into shares. The sum for each share is computed individually[10]. However, the path used is permuted for every share, such that there is no site has the same neighbor twice. To compute v_l, the neighbors of *l* from each iteration would have to collude. Varying the number of shares varies the number of dishonest (colluding) parties required to violate security.

4. Sharing Complex Data with AIDA

Now see the chance of more complex data is to be shared amongst the participating nodes. Each node n_i has a data item d_i of length b-bits which it wishes to make public unknowingly to the other participants.

As the number of bits per data item and the number of nodes becomes larger. To complete this sharing, we will utilize an indexing of the nodes. Techniques for finding such an indexing are developed in consequent sections. Suppose that each node n_i has a unique identification (ID) or serial number $s_i \in \{1, 2, ..., N\}$. Further, understand that no node has knowledge of the ID number s_i of any other node,

and that $s_1,\ldots,s_N$ are a random permutation of $1,\ldots,N$. This, yet again, is termed an Anonymous ID Assignment (AIDA)[7].

This AIDA may be used to allot slots with respect to time or space for communications or storage. It may be possible to simply have a database with central storage locations C_i such that each node simply stores its data there setting $C_{Si} := d_i$. This could occur if there was a trusted central authority, or if the storage operation $C_{Si} := d_i$ was untraceable [14].

We already given that there is no third party central authority (the situation for which secure sum was designed), secure sum can be used to finish the desired data sharing. Let 0 be a vector of b-bits. Each node creates a data item D_i of N b-bits. To numbering each of the N, b-bit components $1, 2,\ldots,N$ we have:

	1	2		s_i		N
$D_i =$	$\vec{0}$	$\vec{0}$	$\cdots$	d_i	$\cdots$	$\vec{0}$

The secure sum algorithm, given earlier in this paper, may now be used to collect the data items D_1 ,...., D_N. The group operation is bitwise exclusive-or, and each node n_i will choose $N-1$ random entries $r_{i,j}$, each composed of N . b randomly chosen bits while calculating one entry, e.g., $r_{i,i}$ to ensure sum (by XOR) equal to D_i.

5. Finding an AIDA

Here we present a simple algorithm for finding an AIDA which has several variants depending on the choice of the data sharing method at step (3) below. According to step one, random integers or "slots" between 1 and S are chosen by each node. Now node's position will be determined by its position among the chosen slots, but necessities must be made for collisions. The parameter 5 should be chosen so that $S \geq N$.

Algorithm (Find AIDA): Following nodes $n_1,\ldots,n_N$,use distributed computation (without any central authority) to find an anonymous indexing permutation s : $\{1,\ldots,N\} \rightarrow \{1,\ldots,N\}$.

1) First we allocate the number of assigned nodes as A = 0.
2) We make each unassigned node n_i chooses a random number r_i in the range 1 to S. Then a node assigned in a previous round chooses $r_i = 0$.
3) Now the generated random numbers are shared anonymously shared values by $q_1,\ldots,q_N$.
4) Here Let $q_1,\ldots, q_k$ denote a revised list of shared values with duplicated and zero values entirely removed where k is the number of distinctive or unique random values. Then nodes n_i which drew distinctive random numbers then determine their index S_i from the position of their random number in the revised list as it would appear after being sorted:
$$S_i = A + Card\{q_j : q_j <= n\}$$
5) Here update the number of nodes assigned: A = A + k.
6) If A < N then return to step (2).

Example (Execution of Algorithm to Find an AIDA): Assume that four nodes participate in searching for an AIDA. For example S = 10 and random number choices 6, 10, 6, 2 again in the first round. The choices of n_1 and n_3 are 5 and 6 respectively in the second round while n_2 and n_4 choose 0 as they will already have indexes assigned at that point. Now because of that a trace of critical steps in the procedure is shown in Table I. The final AIDA result is then $s_1 = 3$ for node m, $s_2 = 2$ for node n_2, $s_3 = 4$ for node n_3,and $s_4 = 1$ for node n_4.

Table 1: Trace of an AIDA Algorithm Execution

R	Step	A	r_1	r_2	r_3	r_4	q_1	q_2	q_3	q_4	s_1	s_2	s_3	s_4
1	2	0	6	10	6	2								
1	3	0	6	10	6	2	2	6	6	10				
1	4	0	6	10	6	2	2	10				2		1
1	5	2										2		1
2	2	2	5	0	6	0						2		1
2	3	2	5	0	6	0	0	0	5	6		2		1
2	4	2	5	0	6	0	5	6			3	2	4	1

The number of rounds this algorithm takes is modeled by a Markov chain [13]. Although there is no absolute upper bound is possible, we will see in later Section's that the performance is very good, as one might expect, when S is much larger than N. The various methods for sharing the random numbers at step (3).

The complicity resistance of AIDA depends upon the underlying secure sum algorithm used and the complicity resistance of that algorithm for a particular set of colluding nodes C. The strongest result possible can be obtained by using our simple, but inefficient, secure sum Algorithm 1:

Theorem (AIDA is N-Private): This Algorithm is resistant to the conspiracy of any subset of the participating nodes when the secure sum method of Algorithm is used.

Proof: Here we outline the essential step of the proof by viewing the AIDA algorithm at its final termination. Assume that there are M iterations of steps (2)-(6). Let $r^m{}_1,\ldots, r^m{}_N$ denote the random values chosen by nodes $n_1,\ldots, n_N$ at step (2) in iteration m. Denote by $i \rightarrow s(i)$ is the final permutation $s(i) = s_i$ produced by AIDA.

Here $t(i)$ denote any permutation of [N]. Suppose that the random choices by node n_i at iteration m during execution had been $\hat{r}_i^m = r^m_{t(i)}$ rather than $r^m{}_i$. The choice of random numbers would be similarly likely and would have resulted in the final assignment $n_i \rightarrow s(t(i))$. Here C denote the alliance of colluding nodes and $D = [N] \setminus C$ the remaining nodes. Given any desired permutation $p: D \rightarrow D$,define

$$p'(i) = p(i) \setminus i \in D;\ p'(i) = i \setminus i \in C.$$

The selection $t(i) = s^{-1}(p'(i))$ yields

$$n_i \rightarrow s(s^{-1}(p'(i))) = p'(i).$$

Thus all permutations of the values $\{s(i) | i \in D\}$ are similarly likely and this is independent of the number of iterations.

Corollary (AIDA Produces Random Permutations): From the above algorithm results in a permutation of the

participating nodes chosen from the uniform distribution (all assignments are similarly likely) when the secure sum method of Algorithm is used.

6. AES Algorithm

AES is similar to the Rijndael cipher proposed by two Belgian cryptographers, Vincent Rijmen and Joan Daemen by whom submitted a proposal to NIST during the AES selection process. Although Rijndael is a family of ciphers with different key and block sizes. For NIST,AES selected[11],[12] three members of the Rijndael family, has each every with a block size of 128 bits, but we have three different key lengths like 128 bits, 192 bits and 256 bits.

AES Algorithm mainly consists of two parts: The Computational flow of AES includes various steps to complete its operation. Those steps are briefly described given below and are shown in Figure. Four basic steps are involved in operations of AES computation flow: Add Round Key, Shift Rows and Mix Columns, Sub Bytes. Following step one Add Round Key means plaintext is updated with the result of XOR operation of each individual Byte of round key and element of plaintext. Then followed by Substitute Byte is a non-linear byte substitution, it uses substitution table (S-Box) which is built by composing of two transformations, first, the polynomial m(x) ('11B') is ported, followed by an affine transformation [11]. Let see in Shift Rows the rows of the plaintext block are cyclically shifted to generate another matrix. Finally the Mix Columns means transformation of four elements of each column of the plaintext by a polynomial multiplication. This type of multiplication is simple multiplication. This routine step is final step for encryption process. The Numbers of rounds are possible to encrypt the data. This varies from 10, 12, and 14 for different key sizes of 128,192,256 bits respectively.

Behind the scenes, the encryption routine takes the key array and uses it to generate a "key schedule" Expand this key stream by (4r+4) i.e. 32 bit words where r represents rounds {10, 12, 14}, Nk number of key segments {4, 6, 8}.It generates ith (32) bit word by XORing the (i - Nk)th word either with (i-1)th word or conditionally generated (i-1) word.

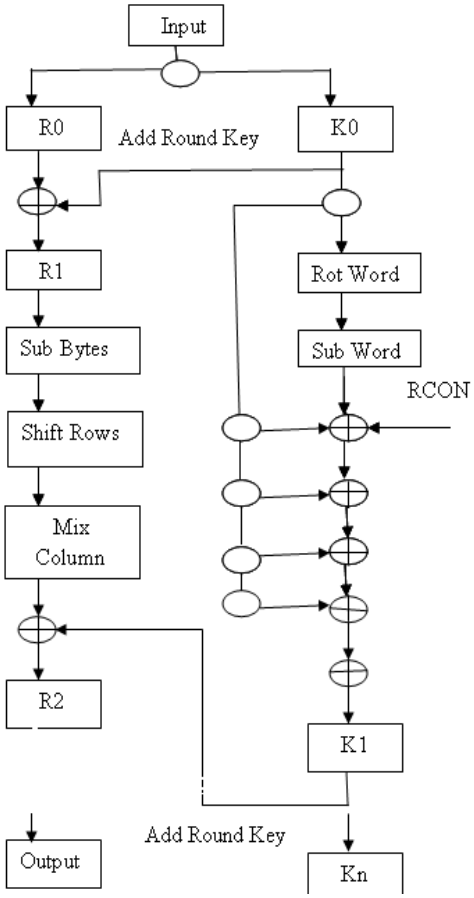

Figure 6.1: AES flow chart

$Nk \leq i \leq (4r+3)$
Rcon [i /Nk] = [02i/Nk, 00, 00,00]

Rcon represents the round contants. The round functions are generated from original key value (0x00 through 0x17). The variable Nk represents the size of the seed key in 32-bit words. The point is that there are now many keys to use instead of using once. These new keys are called the round keys to distinguish them from the original seed key. The main loop of the AES encryption algorithm performs four different operations on the state matrix, which are named as Sub Bytes, Shift Rows, Mix Columns, and Add Round Key in the specification [11],[12]. The Following Add Round Key operation is the same as the preliminary Add Round Key except that each time Add Round Key is called; the next four rows of the key schedule are used. The Sub Bytes routine is a substitution operation that takes each byte in the State matrix and substitutes a new byte determined by the S-box table. Shift Rows is a permutation operation that rotates bytes in the state matrix to the left. Row 0 of State is rotated 0 positions to the left, first row 1 is rotated 1 position left, then row 2 is rotated 2 positions left and similarly row 3 is rotated 3 positions left. The addition and multiplication are special mathematical field operations, are not the common addition and multiplication on integers.

Theory

The Processing time can be deliberate by fly technique and are as follows:

T (processing) = K+ V/F (1)
V= N × P+ offset (2)

K1: Constant part of the execution Time.
V: Variable part of the execution time.
F: System clock frequency.
N: Number of programmed bytes
P: Multiplication factor
Offset: Additive value
As per INFINEON [14] the values of factors P, K, offset are
K1= 4.48
P= 0.06
Offset = 0.27
Clock frequency = 16 MHz
Round Time = T (pr) × (Number of rounds+1) (3)

We observe the processing time of different key sizes and observed that processing time of 256 bit key is more as compared to 192 and 128 bit streams at each frequency level. Figure 3 shows the encryption time for different key lengths, and is observed that encryption through higher order key requires more time as compared to low order key. Encryption time of AES depends upon of key length. Different algorithms take different times to manage the different keys but in AES case encryption and decryption is independent of key length.

7. Conclusion

Ideas contributed by Dr. W. E. Clark play major role in this research. All non-cryptographic algorithms have been extensively simulated, and evaluated that the present work does offer a basis upon which implementations can be constructed. The communications requirements of the algorithms depend heavily on the underlying implementation of the chosen secure sum algorithm along with AES. On the other hand, merging the two layers could result in reduced overhead.

References

[1] **http://en.wikipedia.org/wiki/Wikipedia**
[2] Yehuda , Lindell and Benny Pinkas "*Secure Multiparty Computation for Privacy-Preserving Data Mining"* The Journal of Privacy and Confidentiality (2009) 1, Number 1, pp. 59 -98
[3] Zhu Han and Yan Lindsay Sun Electrical and Computer Engineering Department, University of Rhode Island *"Securing Cooperative Transmission in Wireless Communications"* .
[4] Salik Makda†, Ankur Choudhary_, Naveen Raman†, Thanasis Korakis†, Zhifeng Tao_, Shivendra Panwar "*Security Implications of Cooperative Communications in Wireless Networks"*.
[5] A. Shamir, "How to share a secret," *Commun. ACM,* vol. 22, no. 11, pp. 612-613, 1979.
[6] A. Yao, "Protocols for secure computations," in *Proc. 23rdAnn. IEEE Symp. Foundations ofComputer Science,* 1982, pp. 160-164, IEEE Computer Society
[7] S. S. Shepard, R. Dong, R. Kresman, and L. Dunning, "Anonymous id assignment and opt-out," in *Lecture Notes in Electrical Engineering*, S. Ao and L. Gleman, Eds. New York: Springer, 2010,
[8] D. Jana, A. Chaudhuri, and B. B. Bhaumik, "Privacy and anonymity protection in computational grid services," *Int. J. Comput. Sci. Ap-plicat.,* vol. 6, no. 1, pp. 98-107, Jan. 2009.
[9] J. Castella-Roca, V. Daza, J. Domingo-Ferrer, and F. Sebe, "Privacy homomorphisms for e-gambling and mental poker," in *Proc. IEEEInt. Conf. Granular Computing,* 2006, pp. 788-791.
[10] The Journal of Privacy and Congeniality (2009) 1, Number 1, pp.Secure Multiparty Computation for Privacy-Preserving Data Mining
[11] Federal Information Processing Standards, "Specification for advanced encryption standard" Publication 197, pp.1-47, 2001.
[12] Guido Bertoni; Aril Bircan; Luca Breveglieri; Pasqualina Fragneto; Marco Macchetti. Vittorio Zaccaria; "Performances of the Advanced Encryption Standard in embedded Systems with Cache Memory" IEEE transactions, 2003.
[13] C. M. Grinstead and J. L. Snell, "Chapter 11: Markov chains," in *Introduction to Probability,* 2nd ed. Providence, RI: Amer. Math. Society, 1997, pp. 510-510.
[14] D. M. Goldschlag,M. G. Reed, and P. F. Syverson, "Hiding routing information," in *Proc. Information Hiding*, 1996, pp. 137–150, Springer-Verlag.

Author Profile

Barla Rakesh received his B.Tech degree in computer science and Engineering from JNTUH in 2012 and pursuing M.Tech. Degree in Computer science and Engineering from JNTU Hyderabad. His subjects of interest include Computer Networking, Theory of Computer Science, cloud computing , Secure Computing, Design of Algorithms.

Veladanda RamaKrishna is working as an assistant professor in the Department of Computer Science Engineering, ANURAG GROUP OF INSTITUTIONS Venkatapur(V),Ghatkesar(M), Ranga Reddy District, Hyderabad-500088, Telangana State. His subjects of interest include Computer Networking, Theory of Computer Science, and Design of Algorithms.