Rev Socionetwork Strat (2016) 10:105-123

An Approach to Security for Unstructured Big Data

Md. Ezazul ISLAM[1], Md. Rafiqul ISLAM[2] and A B M Shawkat ALI[3]

1) Department of Computer Science,
American International University-Bangladesh,
Dhaka, Bangladesh
2) Computer Science and Engineering Discipline, Khulna University,
Khulna, Bangladesh
3) School of Science and Technology, University of Fiji,
Lautoka, Fiji

ezaz.time@gmail.com
dmri1978@gmail.com
ShawkatA@unifiji.ac.fj

Received: 04 January 2016 / Accepted: 15 July 2016

Abstract. Security of Big Data is a huge concern. In a broad sense, Big Data contains two types of data: structured and unstructured. Providing security to unstructured data is more difficult than providing security to structured data. In this paper, we have developed an approach to provide adequate security to unstructured data by considering types of data and their sensitivity levels. We have reviewed the different analytics methods of Big Data to build nodes of different types of data. Each type of data has been classified to provide adequate security and enhance the overhead of the security system. To provide security to a data node, and a security suite has been designed by incorporating different security algorithms. Those security algorithms collectively form a security suite which has been interfaced with the data node. Information on data sensitivity has been collected through a survey. We have shown through several experiments on multiple computer systems with varied configurations that data classification with respect to sensitivity levels enhances the performance of the system. The experimental results show how and in what amount the designed security suite reduces overhead and increases security simultaneously.

Keywords: Big Data, Data security, Data analytics, Unstructured data, Security suite, Security standard

1 Introduction

The importance of the term Big Data is increasing day by day. However, recent research on Big Data security is at an initial stage [1]. Researchers and executives of IT-related organizations and blog writers have suggested different aspects of security by considering the volume, velocity, and variety of Big Data [2]. However, to our minds, there is no specific approach or framework to provide security to Big Data. For this reason we have developed an approach to providing security to Big Data by considering the different types of data. In the proposed framework the existing standards or algorithms for different services of security can be integrated to provide security to Big Data. Our research considers two dimensions of security: volume and variety. Since we can provide security by considering these two dimensions, velocity is handled by parallel programming. The volume of Big Data includes different varieties of data such as structured, semi-structured, and unstructured. In our research, the main goal is to provide security to unstructured data, which includes text, XML, images, video, and audio, for example.

The basic motivation of our research is to give an idea of a system that can secure data with a conventional encryption algorithm that uses minimum overhead. This research shows that it is better to have a varied group of encryption algorithms depending on the data types without having to use the strongest encryption. Securing data with the strongest encryption algorithm uses more overhead than choosing varied algorithms based on data types.

We have developed an approach to the security of unstructured Big Data by considering the existing standards or algorithms of security. This approach describes the analysis of unstructured data using data analytics technologies and by building a data node of databases, which contains different types of data such as text, XML, e-mail, images, video, and audio, for example. The authors see this as the next step in developing a security suite to provide security. The data analytics process can be conducted using different types of technologies, which will be discussed in Section 3. After analyzing the different types of data, further analysis classifies the data to get sensitivity levels according to which security standards or algorithms are selected. The ratios to classify data according to their sensitivity level are collected from personnel in various enterprises.

Finally, a scheduling algorithm inside the security suite works as an interface with the security suite and provides appropriate security to the proper type of data by considering the sensitivity level of the data. For multiple system platforms, the performance of the proposed security suite has been observed and analyzed to decide how the system interacts with the designed approach. To understand this, numerous experiments are conducted on several system environments, and the results are described in the later sections of this research.

Sections 2 and 3.2 present the literature about Big Data, and include existing security protocols to provide data security. Section 3 provides in depth knowledge of our proposed system. Section 3 also describes the sensitivity levels, the security suite, and the hypothetical model. Section 4 introduces the data sources and how they are retrieved, an analysis of the result, and a discussion on the analysis with respect to the hypothesis. Section 5 concludes this paper.

2 Background study

The increased possibility of theft and data damage is a big concern in unstructured data because of its huge scalability and less overhead. Therefore, encryption algorithms are necessary although mostly outdated. Until the 1970s, use of cryptography protocols (algorithms) were controlled and monitored by government bodies [3].

One of the most renowned and strongest block ciphers is DES. DES was developed in the early 1970s at IBM with a 64-bit key size and 64 bits of each block with another 16 rounds of 48 bit sub-keys for government purposes. 3DES, which applies DES three times for different keys, is applied in this research because it is more secure than DES and is still being used without any major flaws [4]. Data integrity refers to ensuring the accuracy and consistency throughout the life cycle of data [5]. To maintain data integrity several hash functions are used which ensure one-way conversion of a message for integrity. For our experiment, Snefu-256 and Tiger hash functions are used to maintain data integrity. However, almost all hash functions have some collision problems [6], [7]. In the case of a brute force attack, Snefu-256 can be broken with 288.5 operations [6] and Tiger can be broken by 262 or 244 operations [7]. Therefore, from this perspective, Snefu-256 is stronger than Tiger. CCM and HMAC-SHA-1 hash functions are both used to do two things: maintain integrity and provide authentication. However, CCM takes more time to execute than HMAC-SHA-1 for most of the cases in the experiment in this paper [8], so we use HMAC-SHA-1 instead of CCM.

To secure and authenticate XML data and content within XML files, XMLEnc (XML Encryption), XML DSig (XML Signature), SAML (Security Assertion Markup Language) and XACML (eXtensible Access Control Markup Language) are used. XML Encryption is used for XML document privacy. For XML Encryption key exchange, the key is encrypted with an asymmetric encryption algorithm [9]. XML

DSig has been used by several Web-based technologies in recent times for signing digital documents [10], and for signing a document when a certificate authority issues a certificate that is integrated with a key [11]. SAML is an open standard protocol for exchanging authentication and authorization data among several parties.

3 Framework of the proposed approach

Big Data is a collection of both structured and unstructured data [12]. In an analysis of Big Data, we can separate structured data using SQL queries and secure such data. Due to the several types of data in unstructured data, its security is a difficult task. Since unstructured data contains data types such as text, XML, e-mail, images, and video, for example, after the analysis of the data, a node of databases is built to store those data in that node, which holds different types of data. An algorithm interfaces that data node with a security suite, which contains several security algorithms to provide security to the data classified according to data type and sensitivity level. In this research, we studied several approaches to data analytics processes and proposed an approach to provide security to unstructured data using the designed security suite. The designed suite has all the essential services to maintain privacy and integrity with authentication and non-repudiation. Thus, our approach includes two processes: one applies data analytics and the other builds a security suite to provide security. The data analytics phase includes data filtering, clustering, and classification, which help to build a data node for the databases. The basic modules of the proposed design are shown in Figure 1.

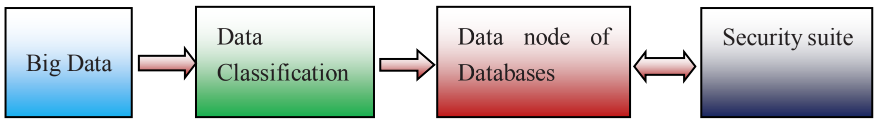

Figure 1. Pictorial view of the proposed approach

3.1 Classification according to sensitivity level

After classifying the data according to their types, different types of data are classified as sensitive, confidential, and public. Sensitive data are those which are protected by country or privacy regulations and data protected by sensitivity agreements. On the other hand, alteration or any unwanted destruction of data that can result in a moderate level of risk, are examples of confidential data. Data are denoted as public data when unauthorized disclosure, destruction, or alteration results occur with little or no risk to the environment where the data are accessed and stored. Sensitive data are the most valuable and require the highest level of security. To ensure security to this class of data, the strongest security standard/algorithms have to be implemented. Each confidential class of data has a middle level of sensitivity and requires a security algorithm with a good processing speed when executed and may be less strong than the protocol used for sensitive data. Public data will be open for all or they can be accessed with little authentication using only an ID and password. We classify the data according to sensitivity levels to provide an adequate level of security and enhance the performance of the system with respect to processing time. We have designed a code-based scenario for each class of data according to sensitivity as shown in Table 1. Since Big Data has a huge volume, providing different types of security services to all classes of data can degrade the processing performance of the targeted system [13]. However, if data can be further classified according to sensitivity levels, we can provide proper security services to each required class of data. The security varies according to classification based on sensitivity level. The major sources of Big Data are accessed by all (public class) or registered users only. There are several services for data security implementation, such as digital signatures or password verification schemes for authentication, cryptographic schemes for confidentiality or privacy, hash functions to provide data integrity, schemes with MAC (message authentication code) which provide user authentication as well as integrity of data, and an access control scheme for providing security according to the access rights of the users to the data. We put 50% of data in the public class, 30% to 40% data in the confidential class and the rest of the data in the sensitive class. The performance of the security aware system will be enhanced proportional to 50% since we do not provide security to 50% of data (See Table 2) or provide normal security with identification and authentication for registered users. The undefined data (UD) in Tables 2 and Table 3 are related to the public class of data. For further information about data classification and performance enhancement, users should refer to M. R. Islam et al. [13].

Table 1. Classes of sensitivity

Any type of Data	Sensitive	Confidential	Public
	01	02	03

Figure 2 represents all interactions among the modules of the whole system. After detecting data types and applying the proper sensitivity level according to the type, data are stored in the Data Node. An Interface Algorithm places requests to the security suite to apply the proper security protocol based on the Data Type and Sensitivity Level for the data available in the Data Node.

3.2 Security suite

Required and adequate security is provided by our security suite which is included in our newly designed approach. The basic goal of the security suite is to reduce execution time while providing adequate security to the structured data. The suite has four parts related to security issues: the first part is for identification and authentication and includes a digital signature scheme or password verification scheme; the second part is used for confidentiality, which contains encryption and decryption algorithms; the third part is for integrity and includes the hash functions, and the fourth and the last part is for integrity and authentication, which includes MAC schemes and access control schemes. Each of the sections is further divided into three sections which represent the three classes of data sensitivity. There is a scheduling algorithm which decides and activates appropriate security services from the selected section and provides adequate security according to the sensitivity level and data type. A detailed view of security suite is shown in Figure 2.

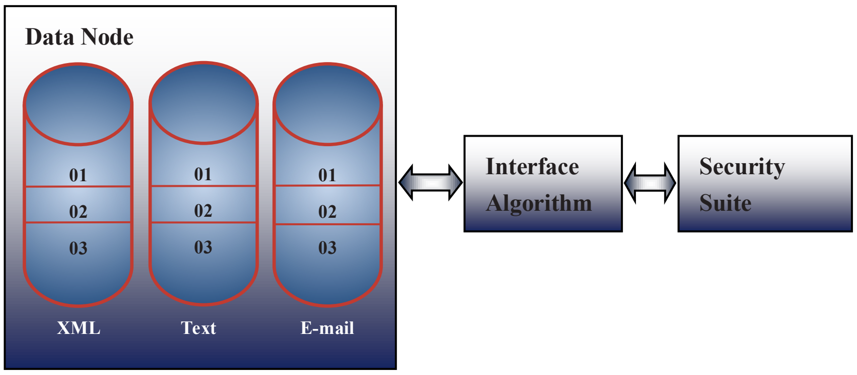

Figure 2. View of data node and security suite

In the design of the security suite, a mask/code is used for each service, such as CS, HF, and MC for privacy, integrity, and authenticity with integrity, respectively. To provide security to the data, the system accesses the mask/code and chooses an algorithm from the security suite based on the mask/code. For example, for the data with code TXCS01 (see Table 3), the algorithm chooses 3DES to provide cryptographic service to the data (See Table 3). We consider standards or algorithms of the different services according to the sensitivity levels of each type of data. For example, to maintain the privacy of text data in the sensitive class, we use 3DES. Table 2 shows the standards or algorithms for the main three services: privacy, integrity, and integrity with authentication. To secure text type data, the Diffie-Hellman key exchange scheme or digital certificates can be used to solve the key management problem. X.509, a standard, can be used as a digital certificate, for example.

Table 2. Security services and related algorithms

Service	Standard or algorithm for each class of data		
	01	02	03
Privacy	3DES	DES	UD
Integrity	Snefru-256	Tiger	UD
Authenticity and integrity	CCM	HMAC-SHA-1	UD

Table 3. Considered algorithms for text data

Type code	Service code	Sensitivity code	Algorithm
TX	CS	01	3DES
		02	DES
		03	UD
	HF	01	Snefru-256
		02	Tiger
		03	UD
	MC	01	CCM
		02	HMAC-SHA-1
		03	UD

E-mail is one of the most widely used internet services. Pretty Good Privacy (PGP) is the most commonly used standard for available cryptosystem algorithms. PGP includes authentication and confidentiality of the message both along with the key management. S/MIME (Secure/ Multipurpose Internet Mail Extensions) is a standard for security enhancement to MIME email. Cryptographic algorithms, digital signature and hash function provide integrity, authentication, and non-repudiation. PGP creates a random session key and is encrypted using the recipient's public key. The encrypted key is added to the encrypted message. In S/MIME Diffie-Hellman, a key agreement method is used for key exchange. Alternatively, digital certificates are used in both PGP and S/MIME for key management. However, differences exist in the key management models used by PGP and S/MIME to establish trust using digital certificates [14]. The security standards to provide security to e-mail messages are shown in Table 4.

XML Digital Signature (XML DSig) is used to provide integrity of the message, authentication, and non-repudiation. In our proposed system mask/code is used for services such as EC, DS, AC, and AP, in which each mask/code respectively refers to encryption, a digital signature, authentication, and access control. It may be used to sign XML resources and library resources such as JPEG files. XML Encryption (XML Enc.) is used to maintain confidentiality or privacy of the document. XML Enc. allows the encryption of selected parts of an XML document or the entire document. Both XML DSig and XML Enc define how to apply well established digital signatures and encryption algorithms such as DSS and 3DES [15]. SAML (Security Assertion Markup Language) is used to provide authentication, attributes, and authorization information. XACML (eXtensible Access Control Markup Language) is used to define access control polices in XML [16]. XML key management specification (XKMS) is a protocol proposed as a standard maintained by the W3C. It defines a way to register the public keys and distribution of keys used by the XML_SIG specification. XMLMS has two parts: the XML key registration specification (X_KRSS) and the XML key information service specification (X-KISS). To register public keys, X-KRSS is used and X-KISS is used to resolve the keys provided in an XML signature [17]. The security can be given to XML documents using the standards shown in Table 5. In Table 4, the service code is common and in Table 5, the sensitivity code is common and is shown as XX.

For multimedia content, a mask/code for the services of encryption and authentication, respectively, are short coded as EC and AT in the designed security suite. Vulnerability of copyright multimedia content, however, arises due to copying and content modification. Therefore, the protection and authentication of its content are significant [18]. Generally, digital water marking is a widely used technique to solve copyright protection problems of multimedia content in a network environment [19]. There are many applications available for watermarking. We use VHA (Video Hosting Authentication) for authentication purposes [20]. H.264 is the most commonly used video coding method and it has been extended to allow scalable video coding known as H.264/SVC [18], [21]. Encryption is used to maintain the privacy of the video content. Naïve encryption is an approach to encrypting the multimedia content. Naïve denotes AES in a cipher feedback mode [21]. T. Stutz and A Uhl showed that MPV (Message Privacy for Video) has the highest level of security. The MP-secure encryption scheme is an AES algorithm in a secure mode [19]. For these reasons, we have chosen VHA for authentication, Naïve encryption for confidential multimedia data, and MPV to encrypt sensitive multimedia content. Multimedia Internet Keying (MIKEY) [22] is designed to solve the key management problem for securing multimedia data. MIKEY uses a pre-shared key, public key encryption, the Diffie-Hellman (DH) key exchange, HMAC, authenticated DH, and reversed RSA to set up a common secret key for all communication scenarios. Detail about this key management scheme can be found in [18].

When designing the mask/code naming within the security suite, a two-character code is defined instead of one character for the sake of consistency and for similarity.

Table 4. Algorithms for text data

Data type	Service code	Sensitivity code	Standards
EM	XX	01	S/MIME
	XX	02	OpenPGP
	XX	03	UD

Table 5. Algorithms for XML

Data Type	Service code	Sensitivity code	Algorithm
XM	EC	XX	XML Enc
	DS	XX	XML DSig
	AC	XX	SAML
	AP	XX	XACML

Table 6. Algorithms for multimedia data

Data type	Service code	Sensitivity code	Standards
MD	EC	01	MPV
		02	Naïve
	AT	XX	VHA (auth)

3.3 Performance analysis – a hypothesis

Let P_k be the probability of data in k_{th} class where k = 1, 2, 3 (in our consideration). Therefore, P_k refers to the probability of data with k number of sensitivity levels. Let S denote the security suite for different types of security services, which include standards and algorithms related to each class of data. Let O be a function for overhead (processing time, memory used for the algorithms) of the security. If O (S) =1, the suite takes the full overhead needed. Let V_k be the value needed for the security of the data in k_{th} class. V_1 =1, for the data in sensitive class because we have to use almost all the services. V_2 = 0.6 to provide security to the data in the confidential class even though all the services may not be required. For the public data, V_3 = 0.1. O(S) is then computed as follows:

$$\begin{aligned} O(S) &= V_1P_1+V_2P_2+V_3P_3 \\ &= 1\times 0.3 + 0.6\times 0.23 + 0.1\times 0.47 \qquad (1) \\ &= 0.3 + 0.138 + 0.047 \\ O(S) &= 0.485 \approx 0.5 \end{aligned}$$

O(S) = 0.5. Therefore, 0.5 is needed for the overhead of the security suite. Since Big Data has big volume, this is a huge benefit in term of the performance of the system.

To study the sensitivity issue, we have sent a survey to several organizations in Bangladesh and found the following scenario.

Table 7. Data of several organizations

Type of organization	Sensitive (%)	Confidential (%)	Public (%)	Overhead	Saving (%)
Educational	20	25	55	0.405	59.5
Health care	65	20	15	0.785	21.5
Research	25	70	5	0.675	32.5
Real Estate	55	25	20	0.72	28
Software developer	40	45	15	0.685	31.5
Financial/Bank	55	25	20	0.725	27.5
Average	**43.34**	**35**	**21.66**	**0.66**	**44**

From Table 7 we can see that the health care organization has the highest percentage of sensitive data. If we provide security to the health care organization, we can save 21.5% overhead. On the other hand, the educational institution can save 59.5% system overhead when we provide security according to our security suite. Considering all six organizations, we can save 44% of system overhead by providing security according to the proposed security suite.

4 Data source, experimental results, and analysis

This section includes a description of how varied data are retrieved from various sources with some of the packages of the programming language used to conduct the data retrieval and storage. The next section contains the detailed results of all of the experiments and the following section is the analysis of those results.

4.1 Data source and retrieval

Among the various sources of online data, we have chosen Wikipedia data dump and Google search API for further analysis. They have a huge collection of image and text data used for non-profit usage. Both the Wikipedia and Google searches have separate data retrieval APIs that help to communicate with each of the servers' data service.

Data retrieval from the above mentioned sources has been conducted with the help of Java programming language. Java supports multiple packages for visiting various URLs for data, executing third party APIs, parsing HTML, reading images, reading text, deciding sensitivity levels and storing data into a MongoDB database. Data is dumped into a MongoDB database by the Java programming with some metadata that will be helpful during data type identification.

MongoDB is a NoSQL database management system that facilitates data storage and retrieval without any SQL query. Another merit of MongoDB is that it has no fixed table schema, which helps store data that have no fixed structure or schema. Unstructured data has no fixed schema.

Among numerous data sources, we have chosen Wikipedia and Google search APIs for experimental use. For example, the below mentioned URL is a Wikipedia API access function that helps to pass various parameters to the Wikipedia server to communicate and read desired types of data according to the requirements of the data requesting client.

For our experiment, we have used Java version j2se 1.7 to access various sources of data. Using the below mentioned links through the Java library we have downloaded text and images simultaneously from the Wikipedia server. Java communicates to the Wikipedia server, reads the response from the server, parses the response in XML format, and downloads the various types of data from the sources online. When conducting the experiments, the total size of the entire data is approximately 1024 megabytes.

http://en.wikipedia.org/w/api.php?action=query&list=allimages&aiprop=url&format=xml&ailimit=100&aifrom=Bangladesh

http://en.wikipedia.org/w/api.php?action=query&list=text&aiprop=url&format=xml&ailimit=100&aifrom=Bangladesh

http://ajax.googleapis.com/ajax/services/search/news?v=1.0&rsz=5&as_sitesearch=bdnews24.com&q=Bangladesh

http://ajax.googleapis.com/ajax/services/search/news?v=1.0&rsz=5&as_sitesearch=prothom-alo.com&q=Bangladesh

The above mentioned Google API link is used to access data using the Google Search API, which requires many parameters to signal the server about its desired data.

For the above links to download data using Java, we need several packages of the Java library. These packages are described in Figure 3:

javax.xml	**for parsing server response in XML format**
org.w3c.dom	**for traversing through XML nodes**
java.io	**for reading/writing text and images separately**
java.awt.image	**For rendering images**

Figure 3. Required Java packages and their purpose

To retrieve data from various sources we follow the following steps.

Search data > read data > analyze metadata > decide sensitivity > store into MongoDB

Through the help of the Java library and the Wikipedia or Google Search API, the data chunks are read on the Web along with other data (metadata), and then from those metadata, the data types are decided, such as whether the file is a text or an image, or XML. After deciding the data types from their extension or meta- tags, a random value is generated for the sensitivity level of each of the files whether it is a text, image or XML. Data type does not have to be decided at this point, but it is done by reading the extension of each of the files when downloading, which is easy. The data sensitivity level is assigned randomly. These random values are generated with the help of Java API. The sensitivity level is assigned randomly and is assigned by the user or manually set. Finally, after the data types and sensitivity levels are assigned in each of the executions of the Java application, each of the files is stored into a MongoDB data node.

MongoDB can store and manage unstructured data. NoSQL stands for Not Only SQL, so data can be accessed without even conventional SQL. Unstructured data has no fixed type of data. The key mechanism behind this is MongoDB's ability to store data with a key value pair, so that the data do not require any fixed table schema. MongoDB stores data in a JSON-like format. That format is known as BSON. World-renowned users of MongoDB are eBay, Foursquare, and The New York Times, for example. For our experiment, data are read and stored from/into MongoDB using Java API library classes. To do this, MongoDB provides a driver for MongoDB to Java connectivity.

4.2 Experimental results and analysis

For experimental use to prove our hypothesis, we have chosen two different systems with some different configurations. As our security suite is related to reducing overhead, we have chosen a variety of systems with varied configurations.

For System 1, the configuration is 1.81 with a GHz CPU with 2 GB memory. We used a Java library-based script that downloads data from the Web and stores them into

MongoDB. Another Java-based script reads data from MongoDB and applies the security suite. Then, one-by-one, the rest of the experimental cases are applied as described in Tables 9, 10, and 11. Three experiments were conducted for a system with configuration: 1.81; GHz CPU with 2 GB memory. For System 2, the configuration is: 2.66, GHz Dual Core CPU with 2 GB memory. A Java library-based script downloads data from Web and stores it into MongoDB. Another Java-based script reads the data from MongoDB and applies the security suite. Then, one-by-one, the rest of the experimental cases are applied as described in Tables 12, 13, and 14. Three experiments for a system with configuration: 2.66, GHz Dual Core CPU with 2 GB memory is also mentioned.

The table below illustrates how the experiments were performed. The experiments were done to measure the total execution of the security suite with respect to varied situations (see: security suite performance against column in below table).

Table 8. Test case to give an idea of all the experiments

	Tables	Same time		Security Suite Against :	
System 1	Table 9	Security Suite	**VS**	Apply 3DES on all Data	Experiment 1
	Table 10	Security Suite	**VS**	Sensitivity Level 1 on all Data	Experiment 2
	Table 11	Security Suite	**VS**	Sensitivity Level 2 on all Data	Experiment 3
System 2	Table 12	Security Suite	**VS**	Apply 3DES on all Data	Experiment 1
	Table 13	Security Suite	**VS**	Sensitivity Level 1 on all Data	Experiment 2
	Table 14	Security Suite	**VS**	Sensitivity Level 2 on all Data	Experiment 3

Table 9. Execution time for System 1 applying security suite and only 3DES

System 1 with 1.81 GHz, 2 GB RAM							
Applying Security Suite					Applying only 3DES		
Code	**Algorithm**	**Time 1**[a)]	**Time 2**[a)]	**Mean**	**Code**	**Algorithm**	**Mean**
TXCS01	3DES	407	421	414	TXCSXX	3DES	504
TXCS02	DES	46	36	41	TXCSXX	3DES	409
TXCS03	None	0	0	0	TXCSXX	3DES	525
TXHF01	Snefu-256	4093	4207	4150	TXHFXX	3DES	4750
TXHF02	Tiger	3779	4043	3911	TXHFXX	3DES	4401
TXHF03	None	0	0	0	TXHFXX	3DES	4314
TXMC01	CCM	2968	3031	2999.5	TXMCXX	3DES	4184
TXMC02	HMACSha1	221	126	173.5	TXMCXX	3DES	3359
TXMC03	None	0	0	0	TXMCXX	3DES	3078
XMEXXX	XML Enc	1297	1421	1359	XMEXXX	3DES	1632
XMDSXX	XML Dsig	1205	1235	1220	XMDSXX	3DES	1226
XMACXX	SAML	1066	1252	1159	XMACXX	3DES	1402
XMAPXX	XACML	922	1141	1031.5	XMAPXX	3DES	1395
Total		16004	16913	**16459**	Total		**31179**

a) Execution Time in Milliseconds

Percentage of improvement of **security suite** against **3DES**, S_1P_1:S_1P_1 = *(16458.5*100)/31179 = **52.78** %* (2)

Table 10. Execution time for System 1 applying security suite and sensitivity level 1

System 1 with 1.81 GHz, 2 GB RAM							
Applying Security Suite					Sensitivity Level 1		
Code	**Algorithm**	**Time 1**[a)]	**Time 2**[a)]	**Mean**	**Code**	**Algorithm**	**Mean**
TXCS01	3DES	407	421	414	TXCS01	3DES	402
TXCS02	DES	46	36	41	TXCS01	3DES	392
TXCS03	None	0	0	0	TXCS01	3DES	250
TXHF01	Snefu-256	4093	4207	4150	TXHF01	Snefu-256	5275
TXHF02	Tiger	3779	4043	3911	TXHF01	Snefu-256	4343
TXHF03	None	0	0	0	TXHF01	Snefu-256	3902
TXMC01	CCM	2968	3031	2999.5	TXMC01	CCM	3000
TXMC02	HMACSha1	221	126	173.5	TXMC01	CCM	3813
TXMC03	None	0	0	0	TXMC01	CCM	3297
XMEXXX	XML Enc	1297	1421	1359	XMEX01	XML Enc	1822
XMDSXX	XML Dsig	1205	1235	1220	XMDS01	XML Enc	1125
XMACXX	SAML	1066	1252	1159	XMAC01	XML Enc	1250
XMAPXX	XACML	922	1141	1031.5	XMAP01	XML Enc	1395
Total		16004	16913	**16458.5**	Total		**30266**

a) Execution Time in Milliseconds

Percentage of improvement of **security suite** against **sensitivity level 1**, S_1P_2:

*S_1P_2 = (16458.5*100)/30266 = **54.37** %* (3)

Table 11. Execution time for System 1 applying security suite and sensitivity level 2

System 1 with 1.81 GHz, 2 GB RAM							
Applying Security Suite					Sensitivity 2		
Code	**Algorithm**	**Time 1[a)]**	**Time 2[a)]**	**Mean**	**Code**	**Algorithm**	**Mean**
TXCS01	3DES	407	421	414	TXCS02	DES	47
TXCS02	DES	46	36	41	TXCS02	DES	62
TXCS03	None	0	0	0	TXCS02	DES	50
TXHF01	Snefu-256	4093	4207	4150	TXHF02	Tiger	4184
TXHF02	Tiger	3779	4043	3911	TXHF02	Tiger	4131
TXHF03	None	0	0	0	TXHF02	Tiger	4172
TXMC01	CCM	2968	3031	2999.5	TXMC02	HMAC-Sha1	187
TXMC02	HMACSha1	221	126	173.5	TXMC02	HMAC-Sha2	142
TXMC03	None	0	0	0	TXMC02	HMAC-Sha3	125
XMEXXX	XML Enc	1297	1421	1359	XMEX02	XML Dsig	1367
XMDSXX	XML Dsig	1205	1235	1220	XMDS02	XML Dsig	1172
XMACXX	SAML	1066	1252	1159	XMAC02	XML Dsig	1140
XMAPXX	XACML	922	1141	1031.5	XMAP02	XML Dsig	1257
Total		16004	16913	**16458.5**	Total		**18036**

a) Execution Time in Milliseconds

Percentage of improvement of **security suite** against **sensitivity level 2**, S_1P_3:

S_1P_3 = *(16458.5*100)/18036* = ***91.25** %* (4)

Table 12. Execution time for System 2 applying security suite and only 3DES

System with 2.66 GHz Dual Core CPU, 2 GB RAM							
Applying Security Suite					Applying only 3DES		
Code	**Algorithm**	**Time 1[a)]**	**Time 2[a)]**	**Mean**	**Code**	**Algorithm**	**Mean**
TXCS01	3DES	242	263	252.5	TXCSXX	3DES	234
TXCS02	DES	34	31	32.5	TXCSXX	3DES	239
TXCS03	None	0	0	0	TXCSXX	3DES	254
TXHF01	Snefu-256	3499	3809	3654	TXHFXX	3DES	3781
TXHF02	Tiger	3057	3027	3042	TXHFXX	3DES	3512
TXHF03	None	0	0	0	TXHFXX	3DES	4230
TXMC01	CCM	3287	3011	3149	TXMCXX	3DES	4289
TXMC02	HMACSha1	89	87	88	TXMCXX	3DES	3566
TXMC03	None	0	0	0	TXMCXX	3DES	3478
XMEXXX	XML Enc	1399	1208	1303.5	XMEXXX	3DES	1438
XMDSXX	XML Dsig	651	683	667	XMDSXX	3DES	1322
XMACXX	SAML	639	639	639	XMACXX	3DES	1305
XMAPXX	XACML	618	638	628	XMAPXX	3DES	1293
Total		13515	13396	**13455.5**	Total		**28941**

a) Execution Time in Milliseconds

Percentage of improvement of **security suite** against **3DES**, S_2P_1:

S_2P_1 = (13455.5*100)/28941 = **46.49** % (5)

Table 13. Execution time for System 2 applying security suite and sensitivity level 1

System with 2.66 GHz Dual Core CPU, 2 GB RAM							
Applying Security Suite					Sensitivity Level : 1		
Short Code	**Algorithm**	**Time 1[a)]**	**Time 2[a)]**	**Mean**	**Short Code**	**Algorithm**	**Mean**
TXCS01	3DES	242	263	252.5	TXCS01	3DES	290
TXCS02	DES	34	31	32.5	TXCS01	3DES	255
TXCS03	None	0	0	0	TXCS01	3DES	220
TXHF01	Snefu-256	3499	3809	3654	TXHF01	Snefu-256	3900
TXHF02	Tiger	3057	3027	3042	TXHF01	Snefu-256	3500
TXHF03	None	0	0	0	TXHF01	Snefu-256	3751
TXMC01	CCM	3287	3011	3149	TXMC01	CCM	3000
TXMC02	HMACSha1	89	87	88	TXMC01	CCM	3312
TXMC03	None	0	0	0	TXMC01	CCM	2985
XMEXXX	XML Enc	1399	1208	1303.5	XMEX01	XML Enc	1312
XMDSXX	XML Dsig	651	683	667	XMDS01	XML Enc	875
XMACXX	SAML	639	639	639	XMAC01	XML Enc	1110
XMAPXX	XACML	618	638	628	XMAP01	XML Enc	950
Total		13515	13396	**13455.5**	Total		**25460**

a) Execution Time in Milliseconds

Percentage of improvement of **security suite** against **sensitivity level 1**, S_2P_2:

S_2P_2 = (13455.5*100)/25460 = **52.84** % (6)

Table 14. Execution Time for System 2. Applying security suite and sensitivity level 2

System with 2.66 GHz Dual Core CPU, 2 GB RAM							
Applying Security Suite					Sensitivity Level : 2		
Code	**Algorithm**	**Time 1[a)]**	**Time 2[a)]**	**Mean**	**Code**	**Algorithm**	**Mean**
TXCS01	3DES	242	263	252.5	TXCS02	DES	41
TXCS02	DES	34	31	32.5	TXCS02	DES	36
TXCS03	None	0	0	0	TXCS02	DES	32
TXHF01	Snefu-256	3499	3809	3654	TXHF02	Tiger	4119
TXHF02	Tiger	3057	3027	3042	TXHF02	Tiger	3100
TXHF03	None	0	0	0	TXHF02	Tiger	3075
TXMC01	CCM	3287	3011	3149	TXMC02	HMACSha1	95
TXMC02	HMACSha1	89	87	88	TXMC02	HMACSha1	89
TXMC03	None	0	0	0	TXMC02	HMACSha1	91
XMEXXX	XML Enc	1399	1208	1303.5	XMEX02	XML Dsig	900
XMDSXX	XML Dsig	651	683	667	XMDS02	XML Dsig	855
XMACX X	SAML	639	639	639	XMAC02	XML Dsig	796
XMAPXX	XACML	618	638	628	XMAP02	XML Dsig	812
Total		13515	13396	**13455.5**	Total		**14041**

a) Execution Time in Milliseconds

Percentage of improvement of **security suite** against **sensitivity level 2**, S_2P_3:

S_2P_3 = (13455.5*100)/14041 = **95.83** % (7)

4.3 Discussion

Table 15. Time needed for security suite [from equation (2) to equation (7)] :

System	Vs 3DES Only(%)	Vs Sensitivity 1 (%)	Vs Sensitivity 2 (%)
1	52.78	54.37	91.25
2	46.49	52.84	95.83
Average	**49.635 %**	**53.605 %**	**93.54 %**

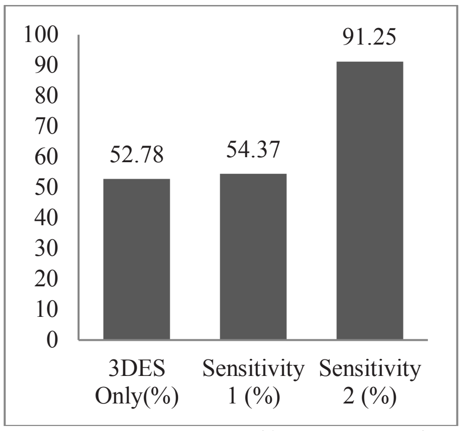

Figure 4. Percentage of improvement of security suite for System 1, prepared based on Table 15

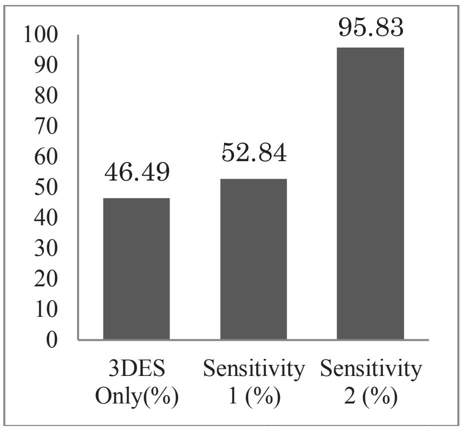

Figure 5. Percentage of improvement of security suite for System 2, prepared based on Table 15.

The data in Table 15, Figures 4 and Figure 5 show that in all aspects of the experiments performed, the security suite designed and proposed in our hypothesis proved to be the least overhead generating mechanism. In all situations and all platforms, application of our security took less execution time than applying one strong encryption algorithm on all data without categorizing them according to their sensitivity.

The execution time of a 1.81 GHz, 2 GB is 52.78% seconds when applying our designed security suite, against 3DES. On the other hand, the execution time of the same system 1 is 54.37% when applying sensitivity level 1 on the same data. Again, the execution time of the same system 1 is 91.25% when applying sensitivity level 2 on the same amount of data. Therefore, the security suite works the fastest because it takes the least time to execute for this system configuration.

The execution time of a 2.66 GHz, 2 GB is 46.49% seconds when applying our designed security suite, against 3DES. On the other execution time of the same system, 1 is 52.84% when applying sensitivity level 1 on the same data. Again, the execution time of the same system 1 is 95.83% when applying sensitivity level 2 on the same amount of data. Therefore, the security suite works the fastest because it takes the least time to execute for this system configuration.

In Figures 4 and 5, the time needed for the security suite is compared by applying 3DES, where sensitivity levels 1 and 2 increase for both systems. This indicates that if we use 3DES for all data to encrypt the system, it takes the maximum number of services that increase overhead. But instead of applying 3DES on the entire data, if we apply our security suite, then it only takes 49% of the time to execute. This proves that our security suite takes less time than applying the strongest encryption algorithm.

5 Conclusions

To provide adequate security to data and to decrease the overall processing overhead, our security system has been classified with respect to sensitivity levels. We have analyzed the system by considering the data of several organizations and have shown that classification according to sensitivity levels decreases the processing overhead of the system. In our work, we provided security to the unstructured data using existing security standards and algorithms according to the data types. A security suite has been built to provide security to the data. In section 3.3, the hypothesis that the security suite would take 50% of the total overhead means that we could reduce overhead by 50%.

To gain some accuracy and reliability, data about sensitivity ratios were collected through survey questionnaires. These survey questionnaires were used to collect data on sensitivity levels in various organizations and a summary of those sensitivity level ratios are given in Table 7, which shows that the percentage of sensitive data is 43.34%, confidential data is 35%, and the percentage of the rest of the public data is 21.66%. After conducting the experiments on two system platforms with varied configurations, the average overhead of those two systems are calculated and shown in Table 15 which shows that the security suite requires approximately 57% of the total overhead. This means that the amount of overhead used by the security suite is not much further from the assumed hypothesis stated in Equation (1). The security suite saves 43% of overhead.

In all aspects of the experiment, we have shown that application of our security suite works faster than not applying the security suite on the data. Therefore, for any type of data, our security suite can be used to reduce overhead.

References

1. Wu, Xindong., Zhu, Xingqua., Wu, Gong-Qin., Ding,Wei.: Data mining with big data. IEEE Transactions on Knowledge and Data Engineering. 26(1), 97-107, 2014
2. Grolinger, Katarina., Higashino, Wilson A., Tiwari, Abhinav., Capretz, Miriam AM.: Data management in cloud environments: NoSQL and NewSQL data stores. Journal of Cloud Computing.2 (22), 1-24, 2013
3. Rivest, Ronald Linn.: MIT Computer Science and Artificial Intelligence Laboratory Web Page, https://people.csail.mit.edu/rivest/pubs/Riv98e.pdf, 5 October 2015
4. Alanazi, Hamda., Zaidan, B.B., Zaidan, A.A., Jalab, Hamid.A., Shabbir, M., Al-Nabhani, Yahya.: New Comparative Study Between DES, 3DES and AES within Nine Factors. Journal of Computing. 2(3), 152-157, 2010
5. Wikipedia Homepage, https://en.wikipedia.org/wiki/Data_integrity, 2 October 2015
6. Merkle, Ralph C.: A fast software one-way hash function. Journal of Cryptology. 3(1), 43-58, 1990
7. Mendel, Floria., Rijmen, Vincent.: Cryptanalysis of the Tiger Hash Function. Advances in Cryptology - ASIACRYPT 2007. Springer Berlin Heidelberg, 2007
8. Dai, Wei.: Cryptopp.com Homepage, http://www.cryptopp.com/benchmarks.html, 2 October 2015
9. Oracle Homepage, http://docs.oracle.com/cd/E39820_01/doc.11121/gateway_docs/content/encryption_encrypt_settings.html, 2 October 2015
10. Wikipedia Homepage, https://en.wikipedia.org/wiki/XML_Signature, 2 October 2015
11. Eastlake, Donald E., Reagle, Joseph M., Solo, David.: World Wide Web Consortium Homepage, http://www.w3.org/TR/xmldsig-core, 2 October 2015
12. Demchenko, Yuri., Ngo, Canh., Membrey, Peter.: Architecture Framework and Components for the Big Data Ecosystem. System and Network Engineering, Graduate school of Sciences, University of Amsterdam, 2013
13. Islam, Md. Rafiqul., Habiba, Mansura.: Data Intensive Dynamic Scheduling Model and Algorithm for Cloud Computing Security, Journal of Computers. 9(8), 1796-1808, 2014
14. Tracy, Miles., Jansen, Wayne., Bisker, Scott.: Guidelines on Electronic Mail Security. NIST Special Publication 800-45, 2007
15. Nordbotten, Nils Agne.: XML and Web Services Security Standards. IEEE Communications Surveys & Tutorials. 11(3), 4-21, 2009
16. Islam, Mohd Rafiqul., Hasan, Mohd Toufiq., Ashaduzzaman, G. M.: An architecture and a dynamic scheduling algorithm of grid for providing security for real-time data-intensive applications. International Journal of Network Management. 21(5), 402-413, 2011
17. Doll, Shelley.: ZDNet Homepage, http://www.zdnet.com/article/xml-security-standards, 2 October 2015
18. Asghar, Mamoona Naveed., Ghanbari, Mohammad.: An Efficient Security System for CABAC Bin-Strings of H264/SVC. IEEE Transactions on Circuits and Systems for Video Technology. 23 (3), 425-437, 2013
19. Shi, F., Liu, S., Yao, H., Liu, Y., Zhang, S.: Scalable and credible video watermarking towards scalable video coding. Springer, 2010
20. Bhowmik, Deepayan.: White Horse eTheses Homepage, http://etheses.whiterose.ac.uk/1526/3/Bhowmik,_Deepayan.pdf, 16 September 2015
21. Stutz, Thomas., Uhl, Andreas.: A Survey of H264 AVC/SVC Encryption. IEEE Transactions on Circuits and Systems for Video Technology. 22(3), 325-339, 2012
22. Arkko, J., Carrara, E., Lindholm, F., Naslund, M., Norrman, K.: Internet Engineering Task Force Homepage, https://tools.ietf.org/html/rfc3830, 2 October 2015