DLSeF: A Dynamic Key Length Based Efficient Real-Time Security Verification Model for Big Data Stream

DEEPAK PUTHAL, University of Technology Sydney, Australia
SURYA NEPAL, CSIRO Data61, Australia
RAJIV RANJAN, Newcastle University, UK and CSIRO Data61, Australia
JINJUN CHEN, University of Technology Sydney, Australia

Applications in risk-critical domains such as emergency management and industrial control systems need near real-time stream data processing in large scale sensing networks. The key problem is how to ensure online end-to-end security (e.g. confidentiality, integrity, and authenticity) of data streams for such applications. We refer to this as an *online security verification* problem. Existing data security solutions cannot be applied in such applications as they cannot deal with data streams with high volume and velocity data in real-time. They introduce a significant buffering delay during security verification, resulting in a requirement for a large buffer size for the stream processing server. To address this problem, we propose a Dynamic Key Length Based Security Framework (DLSeF) based on a shared key derived from synchronized prime numbers; the key is dynamically updated at short intervals to thwart potential attacks to ensure end-to-end security. Theoretical analyses and experimental results of the DLSeF framework show that it can significantly improve the efficiency of processing stream data by reducing the security verification time and buffer usage without compromising security.

• Information systems→Data model extensions→Data Streams • Security and privacy→Security services→ Authentication→Multi-factor authentication.

General Terms: Algorithms, Performance, Security

Additional Key Words and Phrases: sensor networks, big data stream, key exchange, efficient, security, time synchronization

ACM Reference Format:

Deepak Puthal, Surya Nepal, Rajiv Ranjan, and Jinjun Chen. 2016. DLSeF: A Dynamic Key Length Based Efficient Real-Time Security Verification Model for Big Data Streams. *ACM Trans. Embedd. Comput. Syst.* 15, 4, Article 39 (Month YYYY), 25 pages.
DOI:http://dx.doi.org/10.1145/0000000.0000000

1. INTRODUCTION

A variety of applications, such as emergency management, SCADA (Supervisory Control and Data Acquisition), remote health monitoring, telecommunication fraud

This research is funded by the Australia India Strategic Research Grant titled "Innovative Solutions for Big Data and Disaster Management Applications on Clouds (AISRF - 08140)" from the Department of Industry, Australia. The research in this paper is also partially supported by ARC LP140100816.

The preliminary version of this paper has published in 16th International Conference on Web Information System Engineering (WISE 2015), Miami, Florida, USA, 2015.

Author's addresses: D. Puthal and J. Chen, Faculty of Engineering and Information Technology, University of Technology Sydney, Australia; S. Nepal, Commonwealth Scientific and Industrial Research Organization (CSIRO), Australia; R. Ranjan, Newcastle University, Newcastle upon Tyne, UK and Commonwealth Scientific and Industrial Research Organization (CSIRO), Australia; email: deepak.putahl@gmail.com.

Permission to make digital or hardcopies of part or all of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies show this notice on the first page or initial screen of a display along with the full citation. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credits permitted. To copy otherwise, to republish, to post on servers, to redistribute to lists, or to use any component of this work in other works requires prior specific permission and/or a fee. Permissions may be requested from Publications Dept., ACM, Inc., 2 Penn Plaza, Suite 701, New York, NY 10121-0701 USA, fax +1 (212) 869-0481, or permissions@acm.org.

© 2010 ACM 1539-9087/2010/03-ART39 $15.00

DOI:http://dx.doi.org/10.1145/0000000.0000000

detection and large scale sensing networks, require real-time processing of data streams, where the traditional store-and-process method falls short of the challenge [Stonebraker et al. 2005]. These applications have been characterized as producing high speed, real-time, sensitive and large volume data input, and therefore require a new paradigm of data processing. The data in these applications falls in the big data category, as its size is beyond the ability of typical database software tools and applications to capture, store, manage and analyze in real time [Manyika et al. 2011]. More formally, the characteristics of big data are defined by “4Vs” [Bahrami and Singhal 2015; McAfee et al. 2012]: Volume, Velocity, Variety, and Veracity; the streaming data from a sensing source meets these characteristics. Our focus in this paper is thus on providing end-to-end security for real-time high volume, high velocity data streams.

A big data stream is continuous in nature and it is critical to perform real-time analysis as: (i) the lifetime of the data is often very short (i.e. the data can be accessed only once) [Bifet 2013; Dayarathna and Suzumura 2013] and (ii) the data is utilized for detecting events (e.g. flooding of highways, collapse of railway bridge) in real-time in many risk-critical applications (e.g. emergency management). Since a big data stream in risk-critical applications has high volume and velocity and the processing has to be done in real-time, it is not economically viable and practically feasible to store and then process (as done in the traditional batch computing model). Hence, stream processing engines (e.g. Spark, Storm, S4) have emerged in the recent past that have the capability to undertake real-time big data processing. Stream processing engines offer two significant advantages. Firstly, they circumvent the need to store large volumes of data and secondly, they enable real-time computation over data as needed by emerging applications such as emergency management and industrial control systems. Further, integration of stream processing engines with elastic cloud computing resources has further revolutionized big data stream computation as stream processing engines can now be easily scaled [Bifet 2013; Demirkan and Delen 2013; Tien 2013] in response to changing volume and velocity.

Although stream data processing has been studied in recent years within the database research community, the focus has been on query processing [Deshpande et al. 2007], distribution [Sutherland et al. 2005] and data integration. Data security related issues, however, have been largely ignored. Many emerging risk-critical applications, as discussed above, need to process big streaming data while ensuring end-to-end security. For example, consider emergency management applications that collect soil, weather, and water data through field sensing devices. Data from these sensing devices are processed in real-time to detect emergency events such as sudden flooding, and landslides on railways and highways. In these applications, compromised data can lead to wrong decisions and in some cases even loss of lives and critical public infrastructure. Hence, the problem is how to ensure end-to-end security (i.e. confidentiality, integrity, and authenticity) of such data streams in near real-time processing. We refer to this as an *online security verification* problem.

The problem in processing big data becomes extremely challenging when millions of small sensors in self-organizing wireless networks are streaming data through intermediaries to the data stream manager. In these cases, intermediaries as well as the sensors are prone to different kinds of security attacks such as Man in the Middle Attacks. In addition, these sensors have limited processing power, storage, and energy; hence, there is a requirement to develop lightweight security verification schemes. Furthermore, data streams need to be processed on-the-fly in the correct sequence. In this paper, we address these issues by designing an efficient model for online security verification of big data streams.

The most common approach for ensuring data security is to apply cryptographic methods. In the literature, the two most common types of cryptographic encryption methods are asymmetric and symmetric key encryption. Asymmetric key encryption (e.g. RSA, ElGamal, DSS, YAK, Rabin) performs a number of exponential operations over a large finite field and is therefore 1000 times slower than symmetric key cryptography [Burke et al. 2000; Cloudflare 2014]. Hence, efficiency becomes an issue if an asymmetric key such as Public Key Infrastructure (PKI) [Park et al. 2008] is applied to securing big data streams. Thus, symmetric key encryption is the most efficient cryptographic solution for such applications. However, existing symmetric key methods (e.g. DES, AES, IDEA, RC_4) fail to meet the requirements of real-time security verification of big data streams because the volume and velocity of a big data stream is very high (refer to the performance evaluation section for the performance values). Hence, there is a need to develop an efficient and scalable model for performing security verification of big data streams. The main contributions of the paper can be summarized as follows:

— We have designed and developed a Dynamic Key Length Based Secure Framework (DLSeF) to provide end-to-end security for big data stream processing. Our model is based on a common shared key that is generated by exploiting synchronized prime numbers. The proposed method avoids excessive communication between data sources and Data Stream Manager (DSM) for the rekey process. Hence, this leads to reduction in the overall communication overhead. Due to this reduced communication overhead, our model is able to do security verification on-the-fly (with minimum delay) with minimal computational overhead.

— Our proposed model adopts a moving target approach, using a dynamic key length from the set 128-bit, 64-bit, and 32-bit. This enables faster security verification at DSM without compromising security. Hence, our model is suitable for processing high volumes of data without any delay.

— We compare our proposed model with the standard symmetric key solution (AES) in order to evaluate the relative computational efficiency. The results show that our model performs better than the standard AES method.

The rest of this paper is organized as follows. Section 2 gives the background and defines the problem space. Related works is discussed in Section 3. Section 4 describes our proposed solution, DLSeF. Section 5 presents the formal security analysis of our model. Section 6 evaluates the performance and efficiency of the model through extensive experiments. Section 7 concludes our work and points out potential future directions.

2. BACKGROUND AND THE PROBLEM DEFINITION

Fig. 1 shows the overall architecture for big data stream processing from source sensing devices to the data processing center including our proposed security framework. Refer to [Ranjan 2014] for further information on stream data processing in datacenter clouds. In sensor networks, data packets from the sources are transmitted to the sink (data collector) through multiple intermediary hops (e.g. routers and gateways). Collected data at sink nodes are then forwarded to the DSM as data streams may also pass through many untrusted intermediaries. The number of hops and intermediaries depends on the network architecture designed for a particular application. The intermediaries in the network may behave as a malicious attacker by modifying and/or dropping the data packets. Hence, traditional communication security techniques [Walters et al. 2007; Perrig et al. 2002; Chen et al. 2009] are not sufficient to provide end-to-end security. In our framework, both queries and data security related techniques are handled by DSM in coordination with the on-field deployed sensing devices. It is important to note that the security verification of streaming data has to be performed before the query processing phase and in near real-time (with minimal delay) with a fixed (small) buffer size. The processed data are stored in the big data storage system supported by cloud infrastructure [Puthal et al. 2015c]. Queries used in DSM are defined as "continuous" since they are continuously applied to the streaming data. Results (e.g. significant events) are pushed to the application/user each time the streaming data satisfies a predefined query predicate.

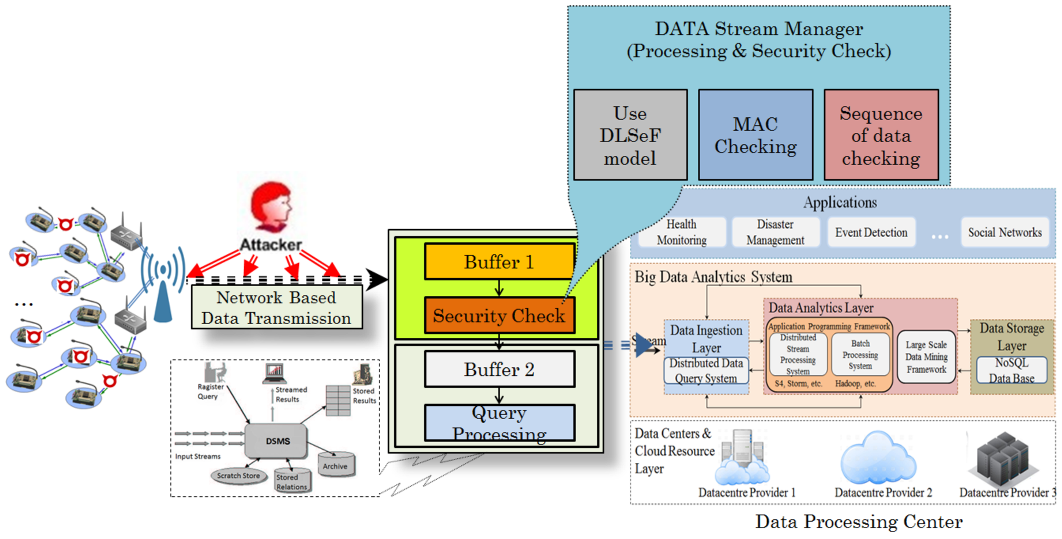


Fig. 1. High level of architecture from source sensing device to big data processing center.

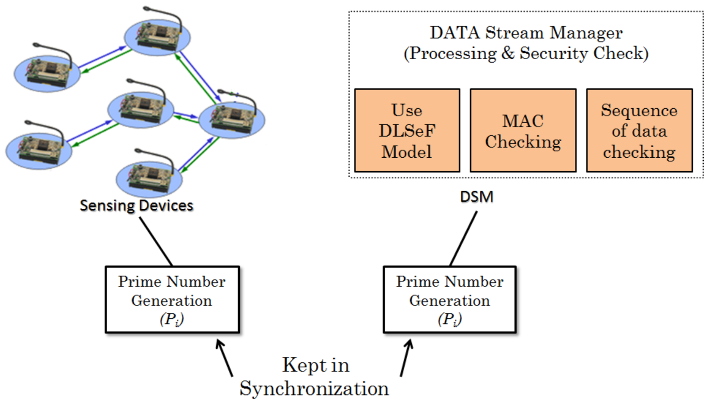

Sensing Devices
DATA Stream Manager
(Processing & Security Check)
Use DLSeF Model
MAC Checking
Sequence of data checking
DSM
Prime Number Generation (P_i)
Prime Number Generation (P_i)
Kept in Synchronization

Fig. 2. Pair of dynamic relative prime number generation, one at the DSM, and another in a distributed sensing device are maintained with a standard time interval based on key length.

The discussion of the architecture above clearly identifies the following most important requirements for security verification for big data stream processing. In summary, they include: (a) the security verification needs to be performed in real time (on-the-fly), (b) the framework has to deal with a high volume of data at high velocity, (c) the data items should be read once in the prescribed sequence, and (d) the original data is not available for comparisons which are widely available in a store-and-process batch processing paradigm. The above requirements need to be met by a big data stream processing framework in addition to end-to-end data security.

Based on the above requirements of big data stream processing, we categorize existing data security methods into two classes: communication security [Carman et al. 2000; Eschenauer et al. 2002] and server side data security [Zissis and Lekkas 2012; Liu et al. 2014]. Communication security deals with the data security between two nodes when it is in motion, and does not deal with the end-to-end security, whereas server side data security approaches focus on ensuring the security of data when it is at rest in a repository [Kandukuri et al. 2009]. The above listed security solutions are not suitable to use in the big data stream because of the four important features of big data stream stated before. Furthermore, symmetric cryptographic-based security solutions are either based on static shared key or centralized dynamic key [Daemen and Rijmen 2002; Heron 2009]. In static shared key, we need to have a long key to defend against potential attackers. It is well known that the length of the key is always proportional to security verification time (see Table 2); hence, longer keys are not suitable for applications that need to do real-time processing over high volume, high velocity data. For the dynamic key, centralized processors rekey and distribute keys to all the sources according to the standard symmetric key solution; this is a time consuming process. Moreover, a big data stream is always continuous in nature and it is impossible to halt data for a rekeying process. To address this problem, we propose a distributed and scalable model for big data stream security verification.

Our proposed model works as follows: we use a common shared key for both sensing devices and DSM. The key is updated dynamically by generating synchronized relative prime numbers without further communication between them after handshaking. This procedure reduces the communication overhead and increases the efficiency of the solution, without compromising security. Due to the reduced communication overhead, our model performs the security verification with minimum delay. Based on the shared key properties, individual source sensing devices update their dynamic key and key length independently.

The Data Encryption Standard (DES) has been a standard symmetric key algorithm since 1977. However, it can be cracked quickly and inexpensively. In 2000, the Advanced Encryption Standard (AES) [Pub, N. F. 2001] replaced the DES to meet the ever increasing requirements of data security. The Rijndael algorithm, i.e. Advanced Encryption Standard (AES), is a symmetric block cipher that encrypts data blocks of 128 bits using different sizes of symmetric keys such as 128, 192 or 256 bits [Pub, N. F. 2001; Simon 2009; Joan and Rijmen 2002]. AES was introduced to replace the Triple DES (3DES) algorithm used for a significant time universally. Hence, we have compared our proposed solution against AES.

3. RELATED WORK

Stonebraker et al. [Stonebraker et al. 2005] outlined eight necessities that a framework or system ought to meet to exceed expectations at a variety of real-time stream processing applications. We found stream data processing and security issues in data streams to be one recent research trend. We started working to address the security verification in data streams and in this paper we are presenting a novel solution towards this problem. In this section, we describe related works under the following two areas: stream processing and security solutions for security verification.

3.1 Stream Data Processing

Data streaming has turned into an important research topic for the stream processing of continuous data flows in several areas such as finance, telecommunications, and networking. The area has gained even more attention from researchers after the emergence of big data. Big data is a term connected to data sets whose size is past the capacity of accessible devices to attempt their securing, access, investigation or application in a sensible timeframe. Recently, it has been estimated that around four zettabytes of data are being generated per year from various sources [Tien 2013]. Many stream data processing technologies are available in the market. For example, StreamCloud is a large scalable elastic data streaming system for processing large data streams [Gulisano et al. 2012]. StreamCloud utilizes a novel parallelization strategy that splits queries into subqueries that are dispensed to independent sets of nodes in a way that minimizes the distribution overhead.

Arasu et al. initially proposed a Data Stream Management System (DSMS); it is called STanford stREam data Manager (STREAM) [Arasu et al. 2003]. It is intended to deal with high velocity data rates and substantial numbers of continuous queries through cautious resource allocation. Tatbul et al. [Tatbul et al. 2007] demonstrate the distributed load shedding issue as a linear optimization problem and propose two different solutions: a centralized approach and a distributed approach. The distributed approach works in light of metadata aggregation and propagation, whose unified execution is additionally accessible. Monitoring applications are those where floods of data, triggers, continuous prerequisites, and loose information are pervasive. In [Carney et al. 2002], a framework named Aurora is proposed towards observing data processing applications. This framework provides existing parts of database configuration and usage, additionally obliging creation of novel proactive information storage and processing concepts and methods. Chandrasekaran et al. [Chandrasekaran et al. 2003] proposed a dataflow system called TelegraphCQ for processing continuous queries over data streams. This is a novel architecture to support a dynamic query workload in unpredictable data stream situations. There are several existing solutions related to data processing but these do not deal with security issues.

3.2 Cryptographic Based Data Security

There are several cryptographic based security solutions, which broadly speaking are proposed to solve two different aspects of problems. These two different types of class are to protect two different modes of data i.e. communication security and server security. Communication security solutions are proposed to protect data when data is in motion whereas server security is to protect data when data is at rest. There are several security threats and security solutions exist in different communication layers [Walters et al. 2007; Chen et al. 2009; Perrig et al. 2002; Eschenauer and Gligor 2002]. Data Confidentiality, Data Integrity, Data Freshness, Availability, and Authentication are the major security threats and authors also give a clear list of security solutions for data communications. Authors also classified the layer-wise security threats and existing solutions (i.e. physical layer, data link layer, network layer, transport layer) for wireless communication when data is in motion.

There are numerous possible attacks when data is at rest such as data interruption, interception, impersonation, privacy breach, session hijacking, programming flaws, software interruption, software modification, defacement, disrupting communications, hardware interruption, and hardware modification, etc. Several existing solutions have been proposed to overcome these types of attacks as follows: data protection from disclosure, privacy in multitenant environments, application security, access control, software security, service availability, data security (data in transit, data at rest, reminisce), virtual cloud protection, cloud management control security, hardware security, and hardware reliability etc. [Zissis and Lekkas 2012; Liu et al. 2014; Kandukuri et al. 2009]. Zissis et al. [Zissis and Lekkas 2012] describe the complete architecture of cloud computing, features, services and security and trust related issues. They initially evaluate cloud security and present a feasible solution that disposes of these potential threats by identifying unique and novel security requirements. They propose security features in a cloud environment by introducing a third party and use PKI cryptography to certify the authentication, confidentiality and integrity of involved data and communications. Liu et al. [Liu et al. 2014] proposed a hierarchical KE scheme, i.e. HKE-BC, which gives more efficient secure scheduling in cloud computing environments and also cloud data auditing. It is designed with a two-phase layer iterative approach and proves the scheme by both theoretical analysis and experimental results. It reduces the overall time consumption in AKE without sacrificing the level of data security. There are several pieces of research available in this area but we are not going into detail. The server side data security (i.e. data is at rest) is mainly proposed for physical data center or cloud to access through applications.

There are also several existing security solutions in streaming environments and participatory sensing system [Nehme et al. 2013; Cao et al. 2013; Nehme et al. 2009; Wang et al. 2013]. Nehme et al. initially proposed an architecture to address the needs of data security and query security in streaming environments [Nehme et al. 2009]. They proposed a continuous access control architecture named StreamShield. They also proposed another solution named FENCE to solve the problem of continuous access control enforcement in data streams [Nehme et al. 2013]. They address security for both data and query processing in their solution. ASSIST is a system based access control framework to protect streaming data from unauthorized access [Cao et al. 2013]. Wang et al. [Wang et al. 2013] proposed a framework, named ARTSense, for participatory sensing networks to solve the problem of trust without identity. The above solutions are mainly designed for stream environments, and the security verification of data streams is not dealt with. Therefore, we focus on security verification of big data streams.

We would like to restate the four important requirements for security verification of big data stream processing: (a) near real time security verification, (b) deal with high volume and velocity of data, (c) the data items should only be accessed once, and (d) the original data is not available for comparisons. Existing solutions for communication security or server side security do not satisfy these requirements. Our work focuses on addressing all these requirements and we propose a novel light weight security model for big data streams. First, we proposed a Dynamic Prime Number Based Security Verification (DPBSV) scheme for big data stream processing, which is based on a common shared key that is updated dynamically by generating synchronized pairs of prime numbers [Puthal et al. 2015a; Puthal et al. 2016]. We proved our scheme is efficient by theoretical analyses and experimental results. The preliminary version of this paper contains the stream data processing architecture security requirements followed by proposal of a novel model to address the online security verification by ensuring end-to-end security (e.g. integrity, and authenticity) [Puthal et al. 2015b]. In this paper, we propose a new solution for real-time security verification (i.e. confidentiality, integrity and authenticity) on big data streams. Our model is efficient; we achieved efficiency by reducing the security computation time and buffer utilization.

4. PROPOSED MODEL

Our model is motivated by the concept of moving target defense. The basic idea is that the keys are the targets of attacks by adversaries. If we keep on moving the keys in spatial (dynamic key size) and temporal (same key size, but different key) dimensions, we can achieve the required efficiency without compromising the security. Our proposed model, Dynamic Key Length Based Security Framework (DLSeF), provides a robust security solution by changing both key and key length dynamically. In our model, if an intruder/attacker eventually hacks the key, the data and time period is selected in such a way that he/she cannot predict the key or its length for the next session. We argue that it is very difficult for an intruder to guess the appropriate key and its length as our model dynamically changes both across the sessions. Though the proposed model has weak confidentiality (eventually the intruder may able to detect the keys if he/she has sufficient processing and storage capabilities), it provides sufficient confidentiality for the duration of online real-time processing. Hence, such a weak confidentiality model is sufficient for a disaster management application scenario. It is important to note that no compromise is made on the authenticity and integrity of the data, which are important for making decisions from the data.

Table I. Notations used in our model

Acronym	Description
S_i	i^{th} source sensing device's ID
K_i	i^{th} source sensing device's secret key
K_{si}	i^{th} source sensing device's session key
kl	Key length
$K_1/K_2/K_3/K_4$	Initial keys for authentication
K_{SH}	Secret shared key calculated by the sensing device and DSM
K_{SH^-}	Previous secret shared key maintain at DSM
$P_1/P_2/P_3/P_4$	Communicated format during authentication
r	Random number generated by the sensing devices
t	Interval time to generate the prime number
j	Integrity checking interval
T	Timestamp added with data blocks
P_i	Random prime number
K_d	Secret key of the DSM
I_D	Encrypted data for integrity check
A_D	Secret key for authenticity check
$E(\,)$	Encryption function
$H(\,)$	One-way hash function
$Prime(P_i)$	Random prime number generation function
$KeyGen$	Key generation procedure
Key-Length ( )	Key length selection procedure
$\oplus$	X-OR operation
$\parallel$	Concatenation operation
$DATA$	Fresh data at sensing device before encryption

Similar to any secret key-based symmetric key cryptography, our DLSeF model consists of four independent components and related processes: system setup, handshaking, rekeying, and security verification. Stream processing is expected to be performed in near real-time. The end-to-end delay is an important QoS parameter to measure the performance of sensor networks [Akkaya and Younis 2003]. We are collecting data from sensor nodes to process for any emergency situation, data need to be collected at the DSM in real time. So we assume there should not be much delay on data arrival at the DSM for our model. Table 1 provides the notations used in our model. We next describe the model.

4.1 DLSeF System Setup

We have made a number of realistic and practical assumptions while designing and modelling our model. We assume that the DSM has all deployed sensing devices' identities (IDs) and respective secret keys because the network is untrusted. Sensing devices and DSM implement some common primitives such as hash function (H( )), and common key (*K1*), which are executed during the initial identification and system setup steps.

The proposed authentication process includes five different steps. The first three steps are for the sensing device and DSM authentication process and the final two steps are for the session key generation process as shown in Fig. 3. The shared key is utilized during the handshaking process.

Step 1:

A sensing device (S_i) generates a pseudorandom number *(r)* and encrypts it along with its own secret key K_i. The encryption process uses the common shared key (*K1*), which is initialized during the deployment. The output of encryption ($E_{K1}(r \parallel K_i)$) is denoted as *P1*. The output is then sent to the DSM: $S_i \rightarrow$ DSM: *P1*

Step 2:

Upon receiving the message, the DSM decrypts *P1* (i. e. $D_{K1}(P_1)$) and retrieves the corresponding source ID from secret key ($S_i \leftarrow retrieveKey(K_i)$). If the source sensor's ID is found in the database, it accepts; otherwise it rejects. The DSM computes the hash of the key to generate another key for encryption $K2 \leftarrow H(K1)$. The DSM then encrypts the pseudorandom number (*r*) with the newly generated key as $P_2 \leftarrow E_{K2}(r)$ and sends it to the source sensing device for DSM authentication: $S_i \leftarrow$ DSM: P_2

Step 3:

The corresponding sensing device receives the encrypted pseudorandom number and decrypts it to authenticate the DSM, *i.e.* $r' \leftarrow D_{K2}(P_2)$. It calculates the current secret shared key using the hash of the existing shared key i.e.$K2 \leftarrow H(K1)$. If the received random number is the same as the sensor had before (i.e. $r = r'$), the sensing device sends an acknowledgement (ACK) to the DSM. The ACK is encrypted with the new key, which is computed using the hash of the current key ($K3 \leftarrow H(K2)$). The encrypted ACK is denoted as $P_3 \leftarrow E_{K3}(ACK)$, and sent to the DSM: $S_i \rightarrow$ DSM: P_3

Step 4:

The DSM decrypts the ACK (*i.e.* $ACK \leftarrow D_{K3}(P_3)$) to confirm that the sensor is now ready to establish the session. The current secret key is updated using the hash of the existing secret key i.e. $K3 \leftarrow H(K2)$. After the confirmation of ACK, the DSM generates a random session key i.e. $K_{si} \leftarrow randomKey()$ for handshaking. The generated session key (K_{si}) is encrypted with the hash of the current key e.g. ($K4 \leftarrow H(K3)$) and then sent to individual sensors as $S_i \rightarrow$ DSM: $\{P_4\}$, where $P_4 \leftarrow E_{K4}(K_{si})$.

Step 5:

The sensor decrypts P_4 and extracts the session key for handshaking ($K_{si} \leftarrow D_{K4}(P4)$). It follows the same procedure as before, i.e. the current shared key is updated with the hash value of the existing shared key ($K4 \leftarrow H(K3)$). We update the shared key in every transaction to ensure the strength of security for handshaking. The complete authentication process works as shown in Fig. 3.

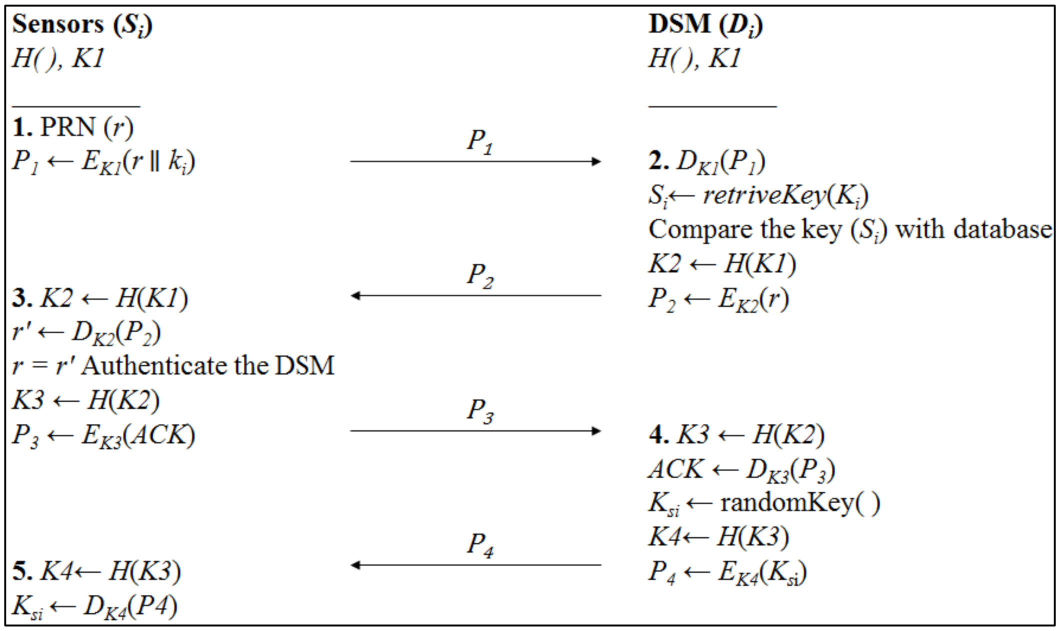

Sensors (S_i)
$H(\,), K1$
1. PRN (r)
$P_1 \leftarrow E_{K1}(r \parallel k_i)$
P_1
DSM (D_i)
$H(\,), K1$
2. $D_{K1}(P_1)$
$S_i \leftarrow retriveKey(K_i)$
Compare the key (S_i) with database
$K2 \leftarrow H(K1)$
$P_2 \leftarrow E_{K2}(r)$
P_2
3. $K2 \leftarrow H(K1)$
$r' \leftarrow D_{K2}(P_2)$
$r = r'$ Authenticate the DSM
$K3 \leftarrow H(K2)$
$P_3 \leftarrow E_{K3}(ACK)$
P_3
4. $K3 \leftarrow H(K2)$
$ACK \leftarrow D_{K3}(P_3)$
$K_{si} \leftarrow$ randomKey( )
$K4 \leftarrow H(K3)$
$P_4 \leftarrow E_{K4}(K_{si})$
P_4
5. $K4 \leftarrow H(K3)$
$K_{si} \leftarrow D_{K4}(P4)$

Fig. 3. Secure authentication of Sensor and DSM.

4.2 DLSeF Handshaking

In the handshaking process, the DSM sends the key generation and synchronization properties to sensors based on their individual session key (K_{si}) established earlier. Generally, a larger prime number is used to strengthen the security process. However, a larger prime number requires greater computation time. In order to make the rekeying process efficient (lighter and faster), we recommend reducing the prime number size. The challenge is how to maintain security while avoiding large prime number sizes. We achieve this by dynamically changing the key size as described next.

ALGORITHM 1. Dynamic Prime Number Generation

```
    Prime ($P_i$)
    1:  $P_{i-1} = P_i$
    2:  Set $k := \left\lceil \frac{P_{i-1}}{6} \right\rceil$
    3:  Set $m := 6k + 1$
    4:  If $m \geq 10^7$ then
    5:          $k := {}^{k}/_{10^5}$
    6:          GO TO: 3
    7:  If S($m$) = 1 then
    8:          GO TO: 14
    9:  Set $m := 6k + 5$
    10: If S($m$) = 1 then
    11:         GO TO: 14
    12: $k := \lfloor k^3 + \sqrt{k} \rfloor \bmod 17 + k$
    13: GO TO: 3
    14: $P_i = m$
Return ($P_i$) // calculated new prime number
```

ALGORITHM 2. Synchronization of Dynamic Key Length Generation

```
Key-Length ($x_{n-1}$)
1:  $x_{n-1} \leftarrow 64$ (for first iteration)
2:  $x_n \leftarrow x_{n-1} + x_{n-1} \cos x_{n-1}$
3:  $i \leftarrow x_n \% 3$
4:  If $i = 0$ then
5:    Set $kl \leftarrow 128$
6:        $t \leftarrow 720$ hours (1 month)
7:        $j \leftarrow$ no checking
8:  Else If $i = 1$ then
9:    Set $kl \leftarrow 64$
10:       $t \leftarrow 168$ hours (1 week)
11:       $j \leftarrow P_i \% 9$
12: Else
13:   Set $kl \leftarrow 32$
14:       $t \leftarrow 20$ hours (1 day)
15:       $j \leftarrow P_i \% 5$
16: End If
17: End If
Return ($x_n$) // use to initialize $x_{n-1}$for next iteration.
```

The dynamic prime number generation function is defined in Algorithm 1. This algorithm computes the relative prime number, which always depends on the previous prime number. This relation between the current and previous prime number process helps to synchronize the newly generated prime number. We have given the mathematical proofs of Algorithm 1, that the generated number will always be a prime number and will synchronize between source device and DSM (refer to Theorem 2). We calculate the prime number and shared key on both sensing sources and DSM ends to reduce communication overhead and minimize the chances of disclosing the shared key. The computed shared keys have multiple lengths (32 bit, 64 bit, and 128 bit) which are varied across the sessions. Initial key length is set to 64 bit and is dynamically updated as per the logic depicted in Algorithm 2. This algorithm selects the key length and the associated time interval to generate the shared key. The key and key length selection process follows based on the time taken to find all possible keys in the key domain by following Table II. In Table II, we compute the key domain size and time required to find all possible keys for different key lengths (i.e. 8, 16, 32, 64, and 128) by using the most advanced Intel i7 processor. So Algorithm 2 follows the properties from Table II to initialize the rekeying time interval according to the key length. After the time interval, the next shared key is generated by applying Algorithm 1 where the size is determined by Algorithm 2 as follows:

$Prime(P_i)$ periodically computes the relative prime number at both the sensor and DSM ends after a time interval t, which is updated based on function $KeyLength(\,)$. The shared secret key (K_{SH}) generation process needs K_d, and P_i. In the handshaking process, the DSM transmits all properties required to generate a shared key to sensors ($K_d, t, P_i, Prime\,(\,), KeyLength(\,), K_{SH}, KeyGen$) as follows: $S_i \leftarrow$ DSM: $\{E_{K_{si}}(K_d, t, P_i, Prime\,(\,), KeyLength(\,), K_{SH}, KeyGen)\}$

All of the transferred information outlined above is stored in the trusted part of the source for future rekeying processes (e.g. TPM) [Nepal et al. 2011].

4.3 DLSeF Rekeying

Our proposed model not only calculates the dynamic prime number to update the shared key without further communication after handshaking, but also proposes a novel way of dynamically changing key length at source and DSM according to steps described in Algorithm 2. We change the key periodically in the DLSeF Rekeying process to ensure that the protocol remains secured. If there are any types of key or data compromise at a source, the corresponding sensor is desynchronized with DSM instantly. Following that the source sensor needs to reinitialize and synchronize with DSM as described above. We assume that the secret information is stored in the trusted part of the sensor (e.g. TPM) and it is sent by the sensor to DSM for synchronization. According to the properties of the TPM, no one has access to the information stored inside the TPM. Only the sensor can access TPM properties. Even if the sensor is destroyed, an adversary cannot get the information from the trusted module of the sensor (i.e. TPM). In some cases, a data packet can arrive at DSM after the shared key is updated. Such data packets are encrypted using the previous shared key. We add a time stamp field to individual data packets to identify the encrypted shared key. If the data is encrypted using the previous key then DSM uses K_{SH}- key for the security verification; otherwise, it follows the normal process.

The above defined *DLSeF Handshaking* process makes sensors aware of the *Prime* (Pi), *KeyLength*, and *KeyGen*. We now describe the complete secure data transmission and verification process using those functions and keys. As mentioned above, our model uses the synchronized dynamic prime number generation Prime (Pi) on both sides, i.e. sensors and DSM, as shown in Fig. 2. At the end of the handshaking process, sensors have their own secret keys, initial prime number and initial shared key generated by the DSM. The next prime generation process is based on the current prime number and the time interval as described in Algorithm 1. The prime number generation process (Algorithm 1) always calls Algorithm 2 to fetch the shared key length information and associated time interval. Sensors generate the shared key KSH=($E(Pi,Kd)$) using the prime number Pi, and the DSM's secret key $E(Pi,Kd)$. We use the secret key of DSM to improve the robustness of the security verification process. We fixed the initial key length at 64 bits and 168 hours as the initial time interval for rekeying. Each data block is associated with the authentication and integration tag and contains two different parts. One is encrypted DATA based on shared key KSH for integrity checking (i.e. $ID=DATA\oplus KSH$), and the other is for authenticity checking (i.e. $AD=Si\oplus KSH$). The resulting data block (($DATA\oplus KSH$) ‖ ($Si\oplus KSH$)) is sent to DSM as follows: Si → DSM: {(ID‖(AD‖T))}. The time stamp which indicates the encrypted shared keys is always associated with the authentication part. We prefer to add the time stamp with the authentication part because the DSM can easily identify the data block if it is encrypted with the previous shared key. More details about the time stamp are described in the following subsection and the complete procedure of the key generation (rekeying) process is shown in Algorithm 3. This algorithm takes information from Algorithm 1 and Algorithm 2 to in order to perform the rekeying process. From Algorithm 1, Algorithm 3 takes the dynamic prime number (P_i) to compute a shared key K_{SH} and from Algorithm 2, it takes the key size and time interval for the rekeying process.

4.4 DLSeF Key Synchronization

Synchronization is one of the major issues during the rekeying process between sensors and the DSM. The shared key synchronization is based on the time interval set during the key length selection and key generation process. In our model, we define the time stamp as T, which is concatenated with encrypted data blocks that are sent to the DSM. The time initialization or interval to generate the shared key always maintains local time, i.e. t1, t2, t3 …, i.e. each site maintains its own time and increases it when the new key is generated. That means the time slot associated with key generation changes only when the key is changed.

The time slot appended with encrypted data blocks is used to identify the shared key by the DSM. When the DSM receives data blocks with the time stamp t_n, and it finds that the current time stamp is $t_{n+1,}$ then it decrypts the data blocks with the previous shared key i.e. K_{SH^-} as stated earlier. This time is updated on both ends after the interval of time, but DSM keeps the immediate previous time stamps with associate shared keys to decrypt data blocks which arrive after the shared keys are updated. The time duration to generate the key (rekeying) always depends on the key length selected by both sensors and DSM as in Algorithm 2. The generated shared keys are always synchronized because the key generation properties reside on both DSM and sensor. The prime number plays a vital role in synchronizing the shared key generation process, and the prime number will always be the same for both DSM and sensors (see Theorem 2).
In some adverse or natural disaster situations, source sensing devices may be lost or desynchronize with the shared key. In such situations, a source device starts the process from the beginning by sending its identification details to the DSM. If the DSM finds the received message is from an authenticated node, then it pass the current shared key along with its properties to the source device. Source devices can use the current key and time interval to encrypt the data blocks and perform the rekeying process. The key generation/rekeying process cannot be disclosed to anyone in any circumstances because we use the TPM at the source device to protect the rekey process. A TPM is a dedicated security chip following the trust computing standard specification for cryptographic microcontroller systems and provides hardware-based trust, which contains cryptographic functionality like key generation, store, and management in hardware.

ALGORITHM 3. Key Generation (Rekeying) Process at Sensor(S_i) and DSM(D)

```
1.  Session key ($K_{S_i}$) from Fig. 3
2.  Dynamic prime number ($P_i$) computed from Algorithm 1.
3.  Time interval (T) computed from algorithm 2.
        3.1  T= {t1, t2, t3, …}
             Here t1, t2, t3, … are the time intervals of key generation.
        3.2  Sensor ($S_i$) and DSM (D) update the key after the time interval from
             Algorithm 2.
4.  As stated before sensor and DSM have properties like $H( ), E$. The new key generation
    $K_{SH}= E_{K_{SH}}(\mathrm{H}(P_i, K_d))$.
5.  The encryption process at sensor happens in two steps
        5.1  $I_D=DATA \oplus K_{SH}$
        5.2  $A_D=S_i \oplus K_{SH}$
6.  $S_i \rightarrow$ DSM: $\{(I_D \| (A_D \| T))\}$
```

Algorithm 4. Security Framework for Big Sensor Data Stream

Description	Based on the prime number generation on both sensor and DSM ends, the proposed dynamic key length based security framework of big data stream works more efficiently than before without compromising security.
Input	the prime generation process $Prime(P_i)$, key length generation process *Key-Length* (x_{n-1}), key generation process $KeyGen$, and session key K_{enc}.
Output	Successful security verification without detecting any malicious attacks.
Step 1	DLSeF System setup
1.1	*Si → DSM: {$E_{K1}(r \parallel K_i)$}, ith sensor sends the random number with its identity which is encrypted with common shared key i.e. K1.*
1.2	*S_i ← DSM: {$E_{K2}(r)$}, DSM identifies the sensor and generates a new key which is the hash of current key for encryption K2 ← H(K1). Then DSM encrypts the random number and sends back to the ith sensor*
1.3	*S_i → DSM: {$E_{K3}(ACK)$}, ith sensor identifies the DSM by decrypting the packet. If sender is authenticated then it performs the hash of the current key (K3 ← H(K2)) to get a new key for encryption and sends back the acknowledgement.*
1.4	*S_i ← DSM: {$E_{K4}(K_{Si})$} DSM authenticates the last transaction and sends back to ith sensor with this format. DSM generates a session key K_{si} ← randomKey() and encrypts with the newly generated key (k4) with the hash function of current key (k3).*
1.5	*Sensor authenticates the packet and gets the session key for handshaking (K_{si} ← $D_{K4}(P4)$).*
Step 2	DLSeF Handshaking
	DSM sends its properties to individual sensors based on their individual session key. It includes the prime number generation and time interval to generation etc.
2.1	*DSM ← S_i: {$E_{K_{si}}(K_d, t, P_i,\ Prime(P_i), KeyLength\ (\,), K_{SH}, KeyGen)$}*
Step 3	DLSeF Rekeying
	Key updates on both source sensor and DSM and both are aware about the Prime (P_i) and KeyGen. Sensors generate the shared key $K_{SH} = H\big(E(P_i, K_d)\big)$ and each data block is associated with two different parts. One is encrypted i.e. $I_D = DATA \oplus K_{SH}$ and the other is for authenticity checking i.e., $A_D = S_i \oplus K_{SH}$.
3.1	*S_i → DSM: { $E_k(I_D \parallel (A_D \parallel T))$ }, these blocks for Authentication, integration, confidential check, and Time stamp for synchronization.*
Step 4	DLSeF Security Verification
	The DSM checks for authenticity in each data block A_D and checks for the integrity with random interval data blocks I_D and random value is calculated based on the corresponding prime number.
4.1	*DSM checks the timestamp (T) at every packet to get the key for decryption. If the timestamp is not the current one then it decrypts with K_{SH^-}.*
4.2	$S_i = A_D \oplus K_{SH}$ *For the authenticity check, the DSM gets the source ID. Once S_i is obtained, the DSM checks the source database and extracts the corresponding secret key K_i for the integrity check according to the value of j.*
4.3	$DATA = I_D \oplus K_{SH}$ *DSM calculates/decrypts data and checks MAC for integrity.*

4.5 DLSeF Security Verification

In this step, the DSM first checks the authenticity in each individual data block A_D and then the integrity with randomly selected data blocks I_D. The random value is calculated based on the corresponding prime number *i.e.* j=P_i% 5, when the key length is 32; j=P_i% 9 when the key length is 64; and there is no integrity verification when the key length is 128. We differ the integrity verification interval randomly for individual key lengths. We prefer to change the integrity verification interval more frequently when the key length is shorter because key length is inversely proportional to possibilities to read/modify the data. As the key length 128 is computationally hard and can last for a long time, we do not check the integrity verification. We update the shared key before there is a possibility of attack. The DSM also checks the time stamp of each individual data block to find the shared key used for encryption. For the authenticity check, the DSM decrypts A_D with shared key $S_i = A_D \oplus K_{SH}$. Once S_i is obtained, the DSM checks its source database and extracts the corresponding secret key K_i ($K_i \leftarrow retrieveKey(S_i)$). In the integrity check process, the DSM decrypts the selected data such as $DATA = I_D \oplus K_{SH}$ to get the original data and checks MAC for data integrity.

The complete mechanism beginning from source and DSM authentication to handshaking and security verification as mentioned in algorithmic format is shown in Algorithm 4. Algorithm 4 represents the description of the proposed mechanism as a stepwise process.

5. SECURITY ANALYSIS OF DLSEF

In this section, we provide a theoretical analysis of our model. We made the following assumptions: (a) any participant in our scheme cannot decrypt the data that was encrypted by the DLSeF algorithm unless it has the shared key which was used to encrypt the data; (b) as DSM is located at the big data processing system side, we assume that DSM is fully trusted and no one can attack it; and (c) a sensors' secret key, Prime (P_i) and secret key calculation procedures reside inside the trusted part of the sensor (such as the TPM) so that they are not accessible to intruders.

Similar to most security analyses of communication protocols, we now define the attack models for the purpose of verifying confidentiality, authenticity and integrity.

5.1 Security Proof

Definition 1 (attack on authentication). A malicious attacker M_a can attack the authenticity if it is capable of monitoring, intercepting, and introducing itself as an authenticated source node to send data in the data stream.

Definition 2 (attack on integrity). A malicious attacker M_i can attack the integrity if it is an adversary capable of monitoring the data stream regularly and trying to access and modify a data block before it reaches the DSM.

Definition 3 (attack on confidentiality): A malicious attacker Mc is an unauthorized party which has ability to access or view the unauthorized data stream before it reaches the DSM (within the time bound).

Theorem 1: The security is not compromised by changing the size of shared key (K_{SH}).
Proof: The dynamic prime number generation generates and updates the key on both sensor and DSM. The dynamic shared key length is 32 bit or 64 bit or 128 bit. The ECRYPT II recommendations on key length say that a 128-bit symmetric key provides the same strength of protection as a 3,248-bit asymmetric key [Cloudflare 2014]. An even smaller symmetric key provides more security as it is never shared publicly. An advanced processor (*Intel i7 Processor*) took about 1.7 nanoseconds to try out one key from one block. With this speed it would take about $1.3 \times 10^{12} \times$ *the age of the universe* to check all the keys from the possible key set [Cloudflare 2014]. By reducing the size

of the prime number, we vary the key length to confuse the adversary, but achieve faster security verification at DSM using the data reported in Table 2. Further, Table 2 shows that a 128 bit symmetric key takes 3136e +19 nanoseconds (more than a month), a 64 bit symmetric key takes 3136e +19 nanoseconds (more than a week), and so on. We fixed the time interval (*t*) to generate prime numbers and updated the shared key as follows: *t=720 hours* for 128-bit key length, *t=168 hours* for 64-bit length, and *t=20 hours* for 32-bit length key (see Algorithm 2). The dynamic shared key is computed based on the calculated prime number and associated properties initialized accordingly (see Algorithm 1). Based on these calculation, we conclude that an attacker cannot intercept within the interval time *t*. The key has already been changed four times before an attacker knows the key and this fact is not known to the attackers. Data blocks arriving after 20 hours are discarded as they might be compromised.

Table II. Time taken by symmetric key (AES) algorithm to get all possible keys using the most advanced Intel i7 Processor.

Key Length	8	16	32	64	128
Key domain size	256	65536	4.295e+09	1.845e +19	3.4028e+38
Time (in nanoseconds)	1435.2	1e+05	7.301e+09	3136e +19	5.7848e+35

Theorem 2: Relative prime number P_i as calculated in Algorithm 1 is always synchronized between the source sensors (S_i) and DSM.
Proof: The normal method to check the prime number is $6k+1$, $\forall k \in N^+$ (an integer). Here, we first initialize the value of k based on this primary test formula stated above. Our prime number generation method is based on the nth prime number generation concept and from the extended idea of [Kaddoura and Abdul-Nabi 2012]. In our model, the input P_i is the currently used prime number (initialized by DSM) and the return P_i is the calculated new prime number. Intially P_i is initialized by DSM at the DLSeF Handshaking process and the interval time is t (see Algorithm 2).

By applying *Algorithm 1*, we calculate the new prime number P_i based on the previous one P_{i-1} . The complete process of the prime number calculation and generation is based on the value of *m, where m* is initialized from *k*. The value of k is kept constant at source because it is calculated from the current prime number. This is initialized during *DLSeF Handshaking*. Since k is constant, the procedure *Prime* (P_i) returns identical values at both source sensors and DSM. In Algorithm 1, the value of *S(x)* is computed as follows, if the computed value is 1 then x is a prime; otherwise it is not a prime.

$$S_1(x) = \frac{(-1)}{\left\lfloor \frac{\lfloor\sqrt{x}\rfloor}{6} \right\rfloor + 1} \sum_{k=1}^{\left\lfloor \frac{\lfloor\sqrt{x}\rfloor}{6} \right\rfloor + 1} \left\lfloor \left\lfloor \frac{x}{6k+1} \right\rfloor - \frac{x}{6k+1} \right\rfloor ,$$

$$S_2(x) = \frac{(-1)}{\left\lfloor \frac{\lfloor\sqrt{x}\rfloor}{6} \right\rfloor + 1} \sum_{k=1}^{\left\lfloor \frac{\lfloor\sqrt{x}\rfloor}{6} \right\rfloor + 1} \left\lfloor \left\lfloor \frac{x}{6k-1} \right\rfloor - \frac{x}{6k-1} \right\rfloor$$

$$S(x) = \frac{S_1(x)+S_2(x)}{2}$$

If $S(x) = 1$ then x is prime, otherwise x is not a prime.
$x \not\equiv 0 \; mod \; i \; \forall \; 1 \leq i \leq x - 1,$ if x is prime.
Put the value of x as a prime number, then derivations as follows:
$\Rightarrow \left\lfloor \left\lfloor \frac{x}{6k+1} \right\rfloor - \frac{x}{6k+1} \right\rfloor = -1$
Same as $\left\lfloor \left\lfloor \frac{x}{6k-1} \right\rfloor - \frac{x}{6k-1} \right\rfloor = -1$

$\forall\, k$ within the specified range i.e 10^7, then

$$S_1(x) = \frac{(-1)}{\left\lfloor \frac{\lfloor\sqrt{x}\rfloor}{6} \right\rfloor + 1} \sum_{k=1}^{\left\lfloor \frac{\lfloor\sqrt{x}\rfloor}{6} \right\rfloor + 1} (-1) = 1$$

Same $S_2(x)$ is also 1 and then $S(x) = \frac{S_1(x)+S_2(x)}{2} = 1$
Hence, the property of $S(x)$ is proved.

Theorem 3: An *attacker* M_a cannot read the secret information from a sensor node (S_i) or introduce itself as an authenticated node in DLSeF.
Proof: Following *Definition 1* and considering the computational hardness of a secure module (such as TPM), we know that M_a cannot get the secret information for P_i generation, K_i and *KeyGen*. So there are no possibilities for the malicious node to trap the sensor, but M_a can introduce him/herself as an authenticated node to send its information. In our model, a sensor (S_i) sends $\left((I_D) \parallel (A_D)\right)$, where the second part of the data block $(S_i \oplus K_{SH})$ is used for an authentication check. The DSM decrypts this part of the data block for the authentication check. The DSM retrieves S_i after decryption and matches corresponding S_i within its database. If the calculated S_i matches with the DSM database, it accepts; otherwise it rejects the node as source and it is not an authenticated sensor node. Hence, we conclude that an *attacker* M_a cannot attack the big data stream.

Theorem 4: An *attacker* M_c cannot access or view the unauthorized data stream in our proposed DLSeF within the time bound.
Proof: Following *Algorithm 1*, it is clear that the prime numbers $Prime\,(P_i)$ are generated at sensors and DSM dynamically without any further communication. Shared secret key K_{SH} is calculated based on the generated prime number. Considering the computational hardness of secure modules (such as TPM), we know that M_c cannot get the secret information such as P_i generation, K_i and *KeyGen* within the time frame. Following *Definition 2*, we know that an attacker M_c can gain access to the shared key K_{SH} but no other information. In our scheme, source sensor (S_i) sends data blocks in the format $\left((DATA \oplus K_{SH} \oplus K_i) \parallel (S_i \oplus K_{SH})\right)$, where the first part of the data block $(DATA \oplus K_{SH} \oplus K_i)$ contains the original data. Getting the original data ($DATA$) is impossible from this because M_c does not have other information and at the same time the shared key K_{SH} is updated dynamically in the interval of time (t). If M_c has sufficient processing and storage capabilities, it may be able to get the shared key, but in the meantime our shared key must be changed. In such a case, M_c can read the message. This does not affect the application we are focusing on (e.g. disaster management) by stream data processing. So our model DLSeF provides weak confidentiality by not breaking the confidentiality in real time.

Theorem 5: An *attacker* M_i cannot read the shared key K_{SH} within the time interval t in the DLSeF model.
Proof: Following *Definition 2*, we know that an attacker M_i has full access to the network to read the shared key K_{SH}, but M_i cannot get correct secret information such as K_{SH}. Considering the method described in *Theorem 1*, we know that M_i cannot get the currently used K_{SH} within the time interval t (see Table 2), because our proposed model calculates P_i randomly after time t and then uses the value P_i to generate K_{SH} as described in *Theorems 1* and *2*.

Theorem 6: The proposed DLSeF requires a comparatively smaller buffer size than standard symmetric key solutions for security verification.
Proof: Following *Algorithm 3*, it is clear that the proposed DLSeF is a lightweight security model for security verification. We are decrypting the identity of sensing devices for authentication checks from every data block, whereas the selected data block decrypts for integrity checks. Another important mechanism is the key length used for encryption/decryption. As we are using the smaller key length to encrypt the data blocks, it also makes the security verification faster. These above two processes make the security verification much faster than other security mechanisms. As we all know, the speed of the security verification is directly proportional to the required buffer size. Finally, we conclude that the proposed DLSeF model for security verification needs a comparatively smaller buffer size. The evaluation proof is in the following section.

5.2 Forward Secrecy

As with other symmetric key procedures, shared keys used for encrypting communications are only used for a certain period of time (*t*) until the new prime number is generated. Thus, a previously used shared key or secret keying material is useless to a malicious attacker even if a secret key used in a previous session is known to the attackers. This is one of the major advantages of frequent changing of the shared key. In our model, we change the key with different key lengths. This is one of the reasons we did not choose static symmetric key cryptography or an asymmetric-key encryption algorithm.

6. PERFORMANCE EVALUATION

The proposed DLSeF security model, though deployed in a big sensor data stream in this paper, is a generic approach and can be used in other application domains. In order to evaluate the efficiency and effectiveness of the proposed architecture and protocol, even under adverse conditions, we experimented with different approaches in multiple simulation environments. We first measure the performance of sensor nodes by using a COOJA simulator in Contiki OS [Contiki 2015]; second, we verify the proposed security approach using Scyther [Scyther 2015]; third, we measure the performance of the approach using JCE (Java Cryptographic Environment) [Pistoia et al. 2004]; finally, we compute the minimum buffer size required to process our proposed approach by using MatLab [Matlab 2015] in-order to measure the efficiency of our method.

6.1 Sensor Node Performance

We tested the performance of sensors in a COOJA simulator in Contiki OS to measure the performance of sensors while running the proposed security verification model. We took the two most common types of sensor, i.e. Z1 and TmoteSky sensors, for our experiment and performance checking as shown in Fig. 4. In this experiment, we checked the performance of sensors while computing or updating the shared key and the highest possible number of shared key generation with specified energy level. Initially all sensor nodes have the same level of energy, 1.6 joule. [Kulik et al. 2002].

Fig. 4. The sensors used for experiment (a) Z1 low power sensor. (b) TmoteSky ultra low power sensor.

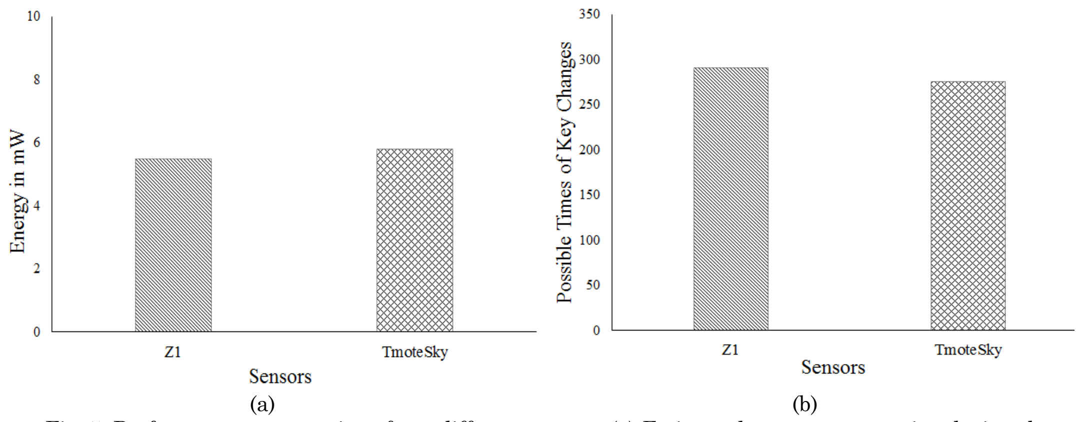


Fig. 5. Performance computation of two different sensors (a) Estimated power consumption during the key generation process. (b) Possible number of key generation with initial 1.6 J power of sensors.

Z1 sensor nodes are produced by Zolertia and are a low-power WSN module, designed as a universal purpose development platform for sensor network researchers. Most of the WSN communities prefer this because it supports most employed open source operating systems, like Contiki. COOJA is a network simulator for Contiki, which provides real time sensor node features for simulation.

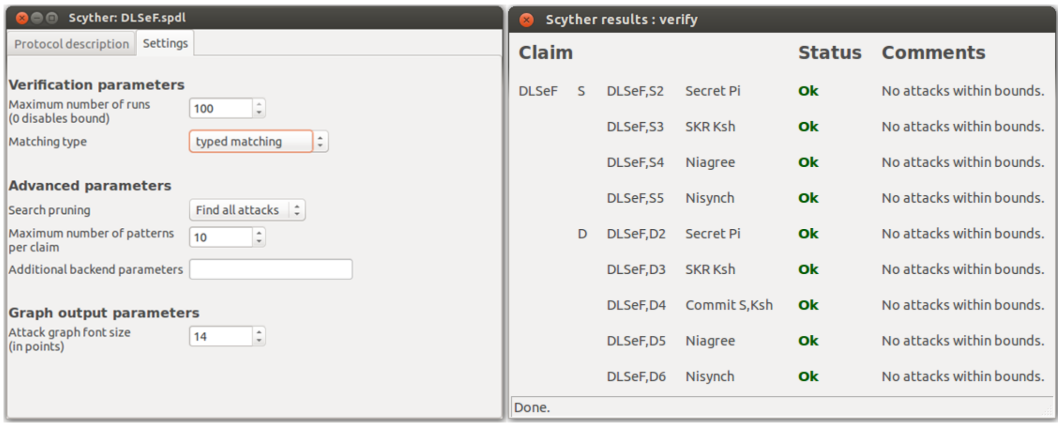
The Z1 sensor is equipped with the low power microcontroller MSP430F2617, which has features like a powerful 16-bit RISC CPU @16MHz clock speed, 8KB RAM, built-in clock factory calibration, and a 92KB Flash memory. Z1 hardware selection always guarantees robustness and maximum efficiency with low energy cost. Similarly, TmoteSky is an ultra-low power sensor. It is equipped with the low power microcontroller MSP430F1611, which has built-in clock factory calibration, 10KB RAM and a 48KB Flash memory.

From the features of the above two types of sensors, we successfully established in the COOJA Simulator that our key generation process works successfully in both types of sensors i.e. z1 sensor and TmoteSky sensor. These sensors can easily support our security approach. The energy consumption during the key generation process is shown in Fig. 5(a), and the maximum number of possible key generations in Fig. 5(b). On average, the above sensors can generate the shared key around 280 times which can support over a year to perform security mechanism. From this experiment, we conclude that our proposed security verification approach DLSeF is supported by most common types of sensors (tested with Z1 and TmoteSky sensors) and feasible for big sensing data streams to work for longer times.

6.2 Security Verification

The protocols in our proposed model are written in a Scyther simulation environment using Security Protocol Description Language (.spdl). According to the features of

Scyther, we define the roles of S and D, where S is the sender (i.e. sensor nodes) and D is the recipient (i.e. DSM). In our scenario, S and D have all the required information that is exchanged during the handshake process. This enables S and D to update their own shared key. S sends the data packets to D and D performs the security verification. In our simulation, we introduce three types of attack by adversaries. In the first type of attack, a malicious attacker changes the data while it is being transmitted from S to D through intermediaries (integrity attack). In the second type of attack (authentication attack), an adversary acquires the property of S and sends the data packets to D pretending that it is from S. In the third type of attack (attack on confidentiality), an adversary gets the data block to analyze and tries to read the data within the time bound. We experimented with 100 runs for each claim and found no attacks at D as shown in Fig. 6.



Scyther: DLSeF.spdl

Protocol description | Settings

Verification parameters

Maximum number of runs (0 disables bound): 100

Matching type: typed matching

Advanced parameters

Search pruning: Find all attacks

Maximum number of patterns per claim: 10

Additional backend parameters:

Graph output parameters

Attack graph font size (in points): 14

Scyther results : verify

Claim				Status	Comments
DLSeF	S	DLSeF,S2	Secret Pi	Ok	No attacks within bounds.
		DLSeF,S3	SKR Ksh	Ok	No attacks within bounds.
		DLSeF,S4	Niagree	Ok	No attacks within bounds.
		DLSeF,S5	Nisynch	Ok	No attacks within bounds.
	D	DLSeF,D2	Secret Pi	Ok	No attacks within bounds.
		DLSeF,D3	SKR Ksh	Ok	No attacks within bounds.
		DLSeF,D4	Commit S,Ksh	Ok	No attacks within bounds.
		DLSeF,D5	Niagree	Ok	No attacks within bounds.
		DLSeF,D6	Nisynch	Ok	No attacks within bounds.

Done.

Fig. 6. Scyther simulation environment with parameters and results page of successful security verification at DSM

Experiment model: In practice, attacks may be more sophisticated and efficient than brute force attacks. However, this does not affect the validity of the proposed DLSeF model as we are interested in efficient security verification without periodic key exchanges and successful attacks. Here, we model the process as described in the previous section and vary the key size between 32 bits, 64 bits, and 128 bits (see Table 2). We used Scyther, an automatic security protocol verification tool, to verify our proposed model.

Results: We did our simulation using a different number of data blocks in each run. Our experiment ranged from 10 to 100 instances with 10 intervals. We checked authentication for each data block, whereas the integrity check is performed on the selected data blocks. As the key generation process is saved in the trusted part of the sensors, no one can get access to that information except the corresponding sensor. Hence, we did not find any authentication attacks. For integrity attacks, it is hard to get the shared key (K_{SH}), as we frequently change the shared key (K_{SH}) and its length based on the dynamic prime number P_i on both source sensor (S_i) and DSM. In the experiment, we did not encounter any integrity attacks. Fig. 6 shows the result of security verification experiments in the Scyther environment. This shows that our model is secured from integrity and authentication attacks.

6.3 Performance Comparison

Experiment model: It is clear that the actual efficiency improvement brought by our model is highly dependent on the size of the key and rekeying without further communication between sensor and DSM. We have performed experiments with different sizes of data block. The results of our experiments are given below.
We compare the performance of our proposed model DLSeF with advanced encryption standard (AES), and our previously proposed model for big sensing data stream (DPBSV), the standard symmetric key encryption algorithm [Pub, N. F. 2001; Simon 2009]. Our model is efficient compared with DPBSV and two standard symmetric key algorithms, 128-bit AES and 256-bit AES. This performance comparison experiment was carried out in JCE (Java Cryptographic Environment). We compared the processing time with different data block sizes. This comparison is based on the features of JCE in Java virtual machine version 1.6 64 bit. JCE is the standard extension to the Java platform which provides a framework implementation for cryptographic methods. We experimented with many-to-one communication. All sensor nodes communicate to the single node (DSM). All sensors have similar properties whereas the destination node has more power to initialize the process (DSM). The rekey process is executed at all the nodes without any intercommunication. The processing time of data verification is measured at the DSM node. Our experimental results are shown in Fig. 7.

Results: The performance of our model is better than the standard AES algorithm when different sizes of data blocks are considered. Fig. 7 shows the processing time of the DLSeF model in comparison with base 128-bit AES, and 256-bit AES for different sizes of data blocks. The performance comparison shows that our proposed model is efficient and faster than the baseline AES protocols.
From the above two experiments, we conclude that our proposed DLSeF model is secured (from both authenticity and integrity attacks), and efficient (compared to standard symmetric algorithms such as 128-bit AES and 256-bit AES).

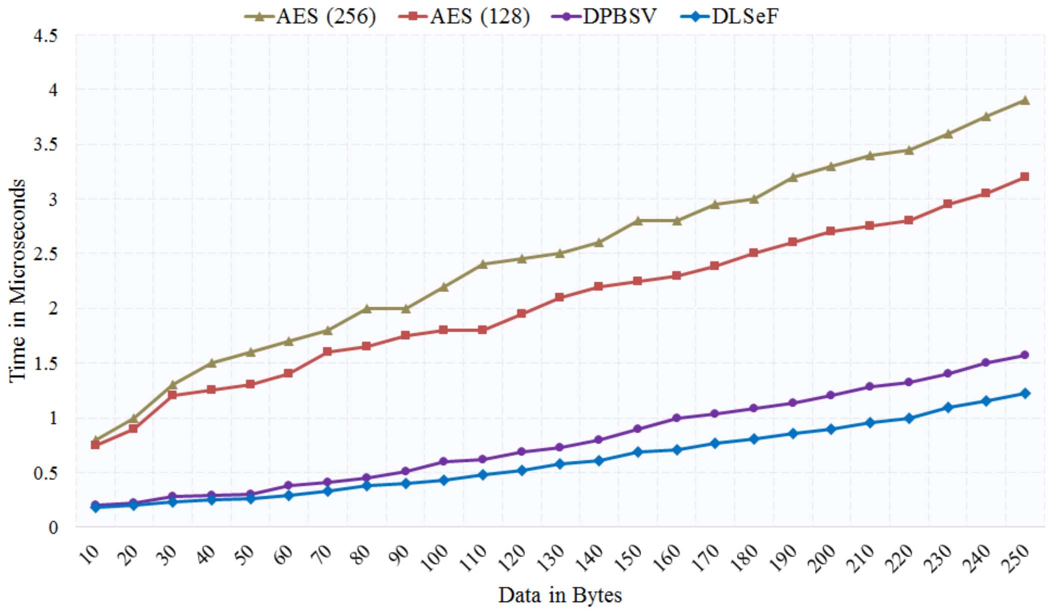


Fig. 7. Performance comparison of our scheme with DPBSV and standard AES algorithm i.e. 128 bit AES and 256 bit AES.

6.4 Buffer size Utilization

Experiment model: We experimented with the features of the DSM buffer by using MATLAB as the simulation tool [Matlab 2015]. This performance is based on the processing time performance calculated in Fig. 8. Here we compared our scheme with DPBSV and standard 128-bit AES and 256-bit AES, the same as the processing time performance comparison. The minimum size of buffer required to process security verification at DSM with various data rates starts from 50 to 250 MB/S with a 50 MB/S interval. Here we compare the efficiency of our proposed scheme (DLSeF).

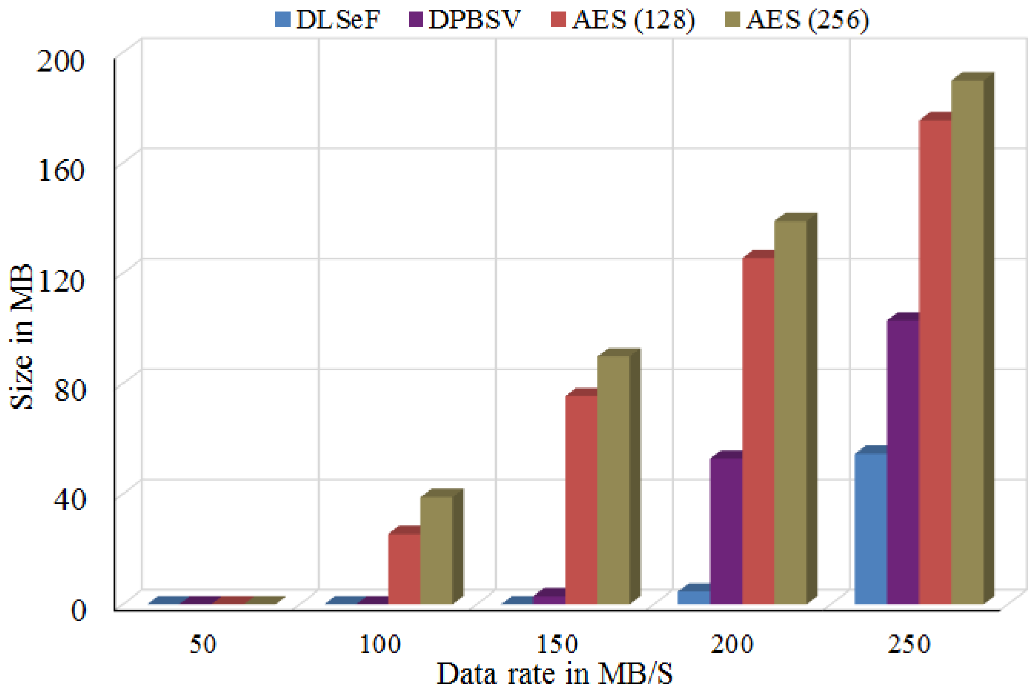


Fig. 8. Efficiency comparison of minimum buffer size required to process the security verification with various data rates to DSM.

Results: The performance of our scheme is better than the standard AES algorithm with different rates of data. Fig. 8 shows the minimum buffer size required to process security at the DSM and proposed DLSeF scheme performance compared with DPBSV and base symmetric key solutions such as 128-bit AES and 256-bit AES. The performance comparison shows that our proposed scheme is efficient and requires less buffer to process security than previous protocols.

From all the above experiments, we conclude that our proposed DLSeF model is secured (from authenticity, confidentiality, and integrity attacks), and efficient (compare to standard symmetric algorithms such as 128-bit AES and 256-bit AES and DPBSV). We also show that the proposed model needs less buffer during the security verification.

7. CONCLUSION AND FUTURE WORK

In this paper, we have proposed a novel authenticated key exchange protocol, namely Dynamic Key Length Based Security Framework (DLSeF), which aims to provide a real-time security verification model for big sensing data streams. Our model has been designed based on symmetric key cryptography and dynamic key length to provide more efficient security verification of big sensing data streams. Our proposed model is designed by two dimensional security i.e. not only the dynamic key but also the dynamic length of the key. By theoretical analyses and experimental evaluations, we showed that our DLSeF model has provided significant improvement in the security processing time, and prevented malicious attacks on authenticity, integrity and weak confidentiality. In our model, we decrease the communication and computation overhead by performing dynamic key initialization along with dynamic key size at both source sensing devices and DSM, which in effect eliminates the need for rekeying and decreases the communication overhead. The proposed security verification model is implemented before stream data processing (i.e. DSM) as shown in our architecture diagram. Several applications such as disaster management, event detection etc. need to filter the modified and corrupted data before stream data processing. These types of applications need only original and unmodified data for analysis to detect the event. The proposed DLSeF model performs security verification in near real time to synchronize with the performance speed of the stream processing engine. Our major concern is not to degrade the performance of stream processing by performing security verification near real time. Although the efficiency of big data stream security verification benefits greatly from an efficient AES and DPBSV scheme such as DLSeF, this is still not fast enough when verifying data blocks while maintaining as much data security and privacy as possible.

In the future, we plan to pursue a number of research avenues to improve the performance of the security verification on big data streams. In addition, we will perform a comparative study of our work with other symmetric key techniques like RC_5 and RC_6. We will further develop and investigate the technique for a moving target defence strategy for the Internet of Things.

ACKNOWLEDGMENTS

This research is funded by the Australia India Strategic Research Grant titled "Innovative Solutions for Big Data and Disaster Management Applications on Clouds (AISRF - 08140)" from the Department of Industry, Australia. The research in this paper is partially supported by Australian Research Council Linkage Project ARC LP140100816.

REFERENCES

Contiki operating system official website, http://www.contiki-os.org/

Matlab, [Online] http://au.mathworks.com/products/matlab/

Pub, N. F. 197. 2001. Advanced encryption standard (AES). Federal Information Processing Standards Publication, 197, 441-0311.

Scyther, [Online] http://www.cs.ox.ac.uk/people/cas.cremers/scyther/

www.cloudflare.com (accessed on: 04.08.2014)

Kemal Akkaya, and Mohamed Younis. 2003. An energy-aware QoS routing protocol for wireless sensor networks." In *Proceedings Distributed Computing Systems Workshops, 23rd International Conference on*, IEEE, 710-715.

Arvind Arasu, Brian Babcock, Shivnath Babu, Mayur Datar, Keith Ito, Itaru Nishizawa, Justin Rosenstein, and Jennifer Widom. 2003. STREAM: the stanford stream data manager (demonstration description). In *Proceedings of the ACM SIGMOD international conference on Management of data.* ACM, 665-665.

Mehdi Bahrami, and Mukesh Singhal. 2015. The Role of Cloud Computing Architecture in Big Data. In *Information Granularity, Big Data, and Computational Intelligence*, Springer International Publishing, 275-295.

Albert Bifet. 2013. Mining big data in real time. In *Informatica.* 37, 1. 15-20.

Jerome Burke, John McDonald, and Todd Austin. 2000. Architectural support for fast symmetric-key cryptography. In *ACM SIGOPS Operating Systems Review,* 34, 5,178-189.

Jianneng Cao, Thomas Kister, Shili Xiang, Baljeet Malhotra, Wee-Juan Tan, Kian-Lee Tan, and Stéphane Bressan. 2013. Assist: Access controlled ship identification streams. In *Transactions on Large-Scale Data-and Knowledge-Centered Systems XI*, Springer Berlin Heidelberg, 1-25.

David W. Carman, Peter S. Kruus, and Brian J. Matt. 2000. Constraints and approaches for distributed sensor network security. Technical Report 00-010, NAI Labs, Network Associates, Inc., Glenwood, MD.

Don Carney, Uğur Çetintemel, Mitch Cherniack, Christian Convey, Sangdon Lee, Greg Seidman, Michael Stonebraker, Nesime Tatbul, and Stan Zdonik. 2002. Monitoring streams: a new class of data management applications. In*Proceedings of the 28th international conference on Very Large Data Bases*, VLDB Endowment, 215-226.

Sirish Chandrasekaran, Owen Cooper, Amol Deshpande, Michael J. Franklin, Joseph M. Hellerstein, Wei Hong, Sailesh Krishnamurthy, Samuel R. Madden, Fred Reiss, and Mehul A. Shah. 2003. TelegraphCQ: continuous dataflow processing. In *Proceedings of the 2003 ACM SIGMOD international conference on Management of data*. ACM, 668-668.

Xiangqian Chen, Kia Makki, Kang Yen, and Niki Pissinou. 2009. Sensor network security: a survey. *IEEE Communications Surveys & Tutorials.* 11, 2, 52-73.

Joan Daemen, and Vincent Rijmen. 2002. AES the advanced encryption standard. In*The Design of Rijndael*, Springer.

Amol Deshpande, Zachary Ives, and Vijayshankar Raman. 2007. Adaptive query processing." *Foundations and Trends in Databases,* 1, 1, 1-140.

Miyuru Dayarathna and Toyotaro Suzumura. 2013. Automatic optimization of stream programs via source program operator graph transformations. *Distributed and Parallel Databases.* 31, 4, 543-599.

Haluk Demirkan, and Dursun Delen. 2013. Leveraging the capabilities of service-oriented decision support systems: Putting analytics and big data in cloud. In *Decision Support Systems.* 55, 1, 412-421.

Laurent Eschenauer, and Virgil D. Gligor. 2002. A key-management scheme for distributed sensor networks. In *Proceedings of the 9th ACM conference on Computer and communications security*, ACM, 41-47.

Vincenzo Gulisano, Ricardo Jimenez-Peris, Marta Patino-Martinez, Claudio Soriente, and Patrick Valduriez. 2012. Streamcloud: An elastic and scalable data streaming system." *Parallel and Distributed Systems, IEEE Transactions on.* 23, 12, 2351-2365.

Simon Heron. 2009. Advanced Encryption Standard (AES). In *Network Security*, no. 12. 8-12.

Issam Kaddoura, and Samih Abdul-Nabi. 2012. On formula to compute primes and the nth prime." Applied Mathematical science, 6, 76, 3751-3757.

Balachandra Reddy Kandukuri, V. Ramakrishna Paturi, and Atanu Rakshit. *2009.* Cloud security issues. In *Services Computing, SCC'09. IEEE International Conference*, IEEE, 517-520.

Joanna Kulik, Wendi Heinzelman, and Hari Balakrishnan. 2002. Negotiation-based protocols for disseminating information in wireless sensor networks. In *Wireless networks*. 8. 2/3, 169-185.

Chang Liu, Nick Beaugeard, Chi Yang, Xuyun Zhang, and Jinjun Chen. 2014. HKE-BC: hierarchical key exchange for secure scheduling and auditing of big data in cloud computing. In *Concurrency and Computation: Practice and Experience*, 28, 3, 646-660.

James Manyika, Michael Chui, Brad Brown, Jacques Bughin, Richard Dobbs, Charles Roxburgh, and Angela H. Byers. 2011. Big data: The next frontier for innovation, competition, and productivity.

Andrew McAfee, Erik Brynjolfsson, Thomas H. Davenport, D. J. Patil, and Dominic Barton. 2012. Big data. *The management revolution. Harvard Bus Rev,* 90, 10, 61-67.

Rimma V. Nehme, Hyo-Sang Lim, Elisa Bertino, and Elke A. Rundensteiner. 2009. StreamShield: a stream-centric approach towards security and privacy in data stream environments. In *Proceedings of the 2009 ACM SIGMOD International Conference on Management of data*, ACM, 1027-1030.

Rimma V. Nehme, Hyo-Sang Lim, and Elisa Bertino. 2013. FENCE: Continuous access control enforcement in dynamic data stream environments. In *Proceedings of the third ACM conference on Data and application security and privacy*, ACM, 243-254.

Surya Nepal, John Zic, Dongxi Liu, and Julian Jang. 2011. A mobile and portable trusted computing platform. In *EURASIP Journal on Wireless Communications and Networking.* 1, 1-19.

Ki-Woong Park, Sang Seok Lim, and Kyu Ho Park. 2008. Computationally efficient pki-based single sign-on protocol, PKASSO for mobile devices. In *Computers, IEEE Transactions on,* 57, 821-834.

Adrian Perrig, Robert Szewczyk, Justin Douglas Tygar, Victor Wen, and David E. Culler. 2002. SPINS: Security protocols for sensor networks." *Wireless networks,* 8. 5, 521-534.

Marco Pistoia, Nataraj Nagaratnam, Larry Koved, and Anthony Nadalin. 2004. *Enterprise Java security: building secure J2EE applications*. Addison Wesley Longman Publishing, Inc.

Deepak Puthal, Surya Nepal, Rajiv Ranjan, and Jinjun Chen. 2015a. DPBSV--An Efficient and Secure Scheme for Big Sensing Data Stream. In *Trustcom/BigDataSE/ISPA, IEEE*, vol. 1, 246-253.

Deepak Puthal, Surya Nepal, Rajiv Ranjan, and Jinjun Chen. 2015b. A Dynamic Key Length based Approach for Real-Time Security Verification of Big Sensing Data. In 16th International Conference on Web Information System Engineering (WISE), Springer International Publishing, 93-108.

Deepak Puthal, B. P. S. Sahoo, Sambit Mishra, and Satyabrata Swain. 2015c. Cloud computing features, issues, and challenges: a big picture. In *International Conference on Computational Intelligence and*

Networks (CINE), 116-123.

Deepak Puthal, Surya Nepal, Rajiv Ranjan, and Jinjun Chen. 2016. A dynamic prime number based efficient security mechanism for big sensing data streams. *Journal of Computer and System Sciences,* http://dx.doi.org/10.1016/j.jcss.2016.02.005

Rajiv Ranjan. 2014. Streaming Big Data Processing in Datacenter Clouds. In *IEEE Cloud Computing*. 1, 1, 78-83.

Michael Stonebraker, Uğur Çetintemel, and Stan Zdonik. 2005. The 8 requirements of real-time stream processing. In *ACM SIGMOD Record*. 34, 4, 42-47.

Timothy M. Sutherland, Bin Liu, Mariana Jbantova, and Elke A. Rundensteiner. 2005. D-cape: distributed and self-tuned continuous query processing. In *Proceedings of the 14th ACM international conference on Information and knowledge management*, ACM, 217-218.

Nesime Tatbul, Uğur Çetintemel, and Stan Zdonik. 2007. Staying fit: Efficient load shedding techniques for distributed stream processing. In *Proceedings of the 33rd international conference on Very large data bases*, VLDB Endowment, 159-170.

James M Tien. 2013. Big data: Unleashing information. *Journal of Systems Science and Systems Engineering,* 22, 2, 127-151.

John Paul Walters, Zhengqiang Liang, Weisong Shi, and Vipin Chaudhary. 2007. Wireless sensor network security: A survey." *Security in distributed, grid, mobile, and pervasive computing*. 2007. 1, 367.

Xinlei Wang, Wei Cheng, Prasant Mohapatra, and Tarek Abdelzaher. *2013.* Artsense: Anonymous reputation and trust in participatory sensing." In *INFOCOM, Proceedings IEEE*, 2517-2525.

Dimitrios Zissis, and Dimitrios Lekkas. 2012. Addressing cloud computing security issues." In *Future Generation computer systems.* 28, 3, 583-592.

Received xxxx; revised xxxx; accepted xxxx