This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TBDATA.2017.2702172, IEEE Transactions on Big Data

SEEN: A Selective Encryption Method to Ensure Confidentiality for Big Sensing Data Streams

Deepak Puthal, Xindong Wu, Surya Nepal, Rajiv Ranjan and Jinjun Chen

Abstract—Resource constrained sensing devices are being used widely to build and deploy self-organizing wireless sensor networks for a variety of critical applications such as smart cities, smart health, precision agriculture and industrial control systems. Many such devices sense the deployed environment and generate a variety of data and send them to the server for analysis as data streams. A Data Stream Manager (DSM) at the server collects the data streams (often called big data) to perform real time analysis and decision-making for these critical applications. A malicious adversary may access or tamper with the data in transit. One of the challenging tasks in such applications is to assure the trustworthiness of the collected data so that any decisions are made on the processing of correct data. Assuring high data trustworthiness requires that the system satisfies two key security properties: confidentiality and integrity. To ensure the confidentiality of collected data, we need to prevent sensitive information from reaching the wrong people by ensuring that the right people are getting it. Sensed data are always associated with different sensitivity levels based on the sensitivity of emerging applications or the sensed data types or the sensing devices. For example, a temperature in a precision agriculture application may not be as sensitive as monitored data in smart health. Providing multilevel data confidentiality along with data integrity for big sensing data streams in the context of near real time analytics is a challenging problem. In this paper, we propose a Selective Encryption (SEEN) method to secure big sensing data streams that satisfies the desired multiple levels of confidentiality and data integrity. Our method is based on two key concepts: common shared keys that are initialized and updated by DSM without requiring retransmission, and a seamless key refreshment process without interrupting the data stream encryption/decryption. Theoretical analyses and experimental results of our SEEN method show that it can significantly improve the efficiency and buffer usage at DSM without compromising the confidentiality and integrity of the data streams.

Index Terms—Big data stream, selective encryption, data confidentiality, data integrity, data security

1 Introduction

A large number of mission critical systems in areas such as disaster management, cyber physical infrastructure systems and SCADA (Supervisory control and data acquisition) are building the Internet of Things (IoT) applications by deploying a number of smart sensing devices in a heterogeneous environment. Data produced from a large variety of sources using sensing devices are streamed towards Data Stream Managers (DSM) for processing and decision making. This trend gives birth to an area, called big data stream [20][49]. The verity of applications and data sources makes the need for data dependability such that only trustworthy and dependable information is considered for decision making processes. Data security (i.e., more specifically ensuring data integrity and confidentiality) is an efficient and effective procedure to assure data trustworthiness/dependability, since DSM processes the data streams in near real time and performs the data analytics; the appropriate actions are performed based on the results from the analytics. It is thus important that data trustworthiness is assured during the lifecycle of big data stream processing. Recent research [2][3] highlighted the key contributions on lightweight security provenance in data both in transit and at rest by considering the example of SCADA systems for critical infrastructure.

The lifetime of a big data stream is very short because it is continuous in nature (i.e., the data can be accessed only once) [20][50]. Such data streams in critical applications have high volume and velocity, but the stream processing has to be done in near real time. It cannot follow the traditional store and process batch computing model [26]. To address this challenge, stream processing engines (such as Spark, Storm, S4, etc.) have emerged in the current era to provide the capability to commence big data processing in real time [21][49]. Stream processing engines (SPE) deal with two important advantages: (i) there is no need to store large volumes of data and (ii) it is capable of supporting real time computation needed by emerging applications. As the important decisions are made in critical applications by data streams analysis in near real time, it is important that such data are not accessed or tampered with by malicious adversaries. This brings one of the key and open research problems in big data streams; that is, how to ensure the end-to-end security for stream data processing. This includes guaranteeing data security properties (i.e. integrity, confidentiality, authenticity and freshness) [3][19][20].

There are different security requirements for different emerging critical applications. Let's consider some

- *D. Puthal is with University of Technology Sydney, Australia. E-mail: deepak.puthal@gmail.com.*
- *X. Wu is with University of Louisiana, Lafayette, USA, xwu@lousiana.edu*
- *S. Nepal is with CSIRO Data61, Australia. E-mail: Surya.Nepal@csiro.au.*
- *R. Ranjan is with Newcastle University, UK. E-mail: rranjans@gmail.com.*
- *J. Chen is with Swinburne Data Science Research Institute, Swinburne University of Technology, Australia. E-mail: jinjun.chen@gmail.com.*

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TBDATA.2017.2702172, IEEE Transactions on Big Data

applications such as disaster management, terrestrial monitoring, military monitoring, healthcare, cyber physical infrastructure systems, SCADA etc. that are the sources for big data streams [5][19][20][36]. Some applications, including terrestrial monitoring and disaster management, need data integrity so that the system has high confidence in the detected events from stream data processing; confidentiality is not that important in such applications [4][6][8]. Some applications such as military applications, healthcare, and SCADA need data confidentiality along with data integrity. The confidentiality of data depends not only on applications, but also on data types. For example, some applications need data confidentiality forever (i.e. strong confidentiality), whereas some applications need to maintain data confidentiality in real time (i.e. partial confidentiality). If we consider healthcare applications, personal health data need to be protected from outsiders and we need strong confidentiality for such applications [5], whereas in SCADA application data need to be protected in real time until a DSM detects the event [2]. There are still several applications including military monitoring that need different levels of data confidentiality [3] [4]. In such systems, there is no need for data confidentiality for normal sensed data, but it is needed for highly sensitive data such as movement in the battle field or detection of enemy activities. In this paper, we address the issue specified above by designing a novel security method for big sensing data streams.

The common approach to data security is to apply a cryptographic model. If the encryption keys are managed properly, data encryption applying a cryptographic method is the most widely recognized and secure way to transmit data. There are two basic sorts of cryptographic encryption strategies: asymmetric and symmetric. It is already proved that symmetric key cryptography is 1000 times faster than asymmetric key cryptography [22][23]. ECRYPT II has shown the 3,248-bit asymmetric key provides the same level of security as 128-bit symmetric key [23]. We thus focus on symmetric key cryptography to design a new security method for big data streams to ensure data confidentiality and integrity.

In order to address the aforementioned challenge, we have designed and developed a selective encryption method (SEEN) to secure and maintain confidentiality of big data streams according to sensitivity levels of the data. Our method is based on a typical shared key that is initialized and updated by a DSM without requiring retransmission. Furthermore, the proposed security method is able to recover keys by detecting lost keys and perform seamless key refreshment without interrupting ongoing data stream encryption/decryption. SEEN maintains different levels of data confidentiality along with data integrity. The main contributions of the paper can be summarized as follows:

- We have developed and designed a novel selective encryption method (SEEN) to secure and maintain confidentiality of big sensing data streams according to different data sensitivity levels. Our method is based on common shared keys and is initialized and updated by a DSM without requiring retransmission. Our method performs seamless refreshing of the shared key without disrupting ongoing data encryption or decryption.
- Our proposed model adopts different keys for the three levels of data confidentiality (i.e. no confidentiality, partial confidentiality and strong confidentiality) based on the data sensitivity levels. This model ensures the end-to-end security by protecting data from source device to cloud processing layer.
- We validate our proposed method by theoretical analyses and experimental results.
- We compare the SEEN method with a standard symmetric key solution (AES-128), DPBSV and DLSeF in order to evaluate the efficiency.

The rest of the paper is organized as follows. Section 2 gives a brief overview of the related works. Section 3 introduces our proposed system and the corresponding security method. Section 4 provides a detailed description of our security method, followed by its security analysis and performance evaluation in Sections 5 and 6, respectively. Finally, Section 7 concludes the paper by providing potential future directions for the work.

2 Related Works

In 2005, Stonebraker et al. [26] initially highlighted the eight requirements of real time stream processing which makes stream processing research more challenging and different to batch processing. In 2009, Nehme et al. [27] proposed a spotlight architecture to highlight the need for security in data streams and differentiate the security requirements of data (called *data security punctuations*) and query side security policies (called *query security punctuations*). There are a large number of security solutions proposed in the literature to protect data confidentiality and integrity by applying asymmetric and symmetric cryptography solutions [3][4][5][6][12][14]. In this section, we describe relevant work related to our research under the following three areas: stream processing, data stream security, and security solutions for data confidentiality and integrity.

2.1 Stream Processing

The Data Stream Management System, also known as STanford stREam data Manager (STREAM), was initially developed by Arsu et al. in 2003 [1]. STREAM is designed to deal with the high velocity data rates and the substantial numbers of continuous queries through thoughtful resource allocation. Most of the works carried out in the Data Stream Management System address different issues ranging from theoretical modelling and analysis to executing comprehensive models to deal with high speed data streams and response in real-time (near real-time). Research methodologies include: STREAM [32], Aurora [33], and Borealis [34].

In data stream management systems like STREAM [32], Aurora [33], and Borealis [34], queries issued by the same client in the meantime can share Seq-window

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TBDATA.2017.2702172, IEEE Transactions on Big Data

administrators. According to the STREAM framework, Seq-window administrators are reused by queries on indistinguishable streams. Rather than developing the sharing of parts between arrangements, Aurora research focuses on giving better execution over a vast numbers of queries. Aurora achieves this by clustering administrators as a basic performance entity. In Borealis, the data on input information criteria from query processing can be shared and changed by new approaching queries. StreamCloud is a large scalable reliable streaming system to handle large scale data streams on clouds [36]. StreamCloud utilizes a new parallelization strategy that separates input quarries into subqueries apportioned to free arrangements of hubs to reduce the circulation overhead. Even though numerous methodologies focus on scheduling and revising for QoS, distributing execution and computation by the same user at various times or by various user at the same time are not supported in stream processing engines. Other than common source Seq-windows as in DSMS, sharing intermediate computation results is a superior approach to improving performance.

The focus of these research was on the performance of query processing, but not much on the security issues in data stream. Nehme et al. [27] highlighted the security aspects of data stream; the following subsection describes details about security issues.

2.2 Stream Security

There have been several recent works on securing data streams [27][37][38][39][41][42][43] focusing on query security punctuations, i.e., access control over data streams. In spite of the fact that these frameworks support secure processing they are unable to avoid illegitimate data streams or data security. Punctuation based enforcement of access control on streaming data is proposed in [43]. Access control strategies are retransmitted each time, utilizing one or more security accentuations before the real data are transmitted. Both punctuation have prepared by streamshield (a unique filtration) for query plan. Secure query processing in a shared manner is proposed in [37]. From the streamshield concept, the authors show a three-phase system to enforce access control without presenting any unique operators, rewriting query, or influencing QoS. Supporting role-based access control through query rephrasing strategies is proposed in [39]. Query arrangements are reorganized and policies are mapped to an arrangement of guide and filter operations to authorize access control policies. The architecture in [41] utilizes a post-query channel to implement access control strategies on a stream level. The channel applies security arrangements before a client gets the outcomes from SPE, but after query preparing. Designing SPEs checking multilevel security imperatives has been tended to by authors in [38]. Xie et al. [42] adopt a Chinese Wall policy to protect and avoid sensitive data disclosure at DSMS. The focus of this research was on query security punctuation, however data security punctuation, i.e. end-to-end security between source and SPEs, is the mission. In our previous works [19][20][40][49], we have proposed data security over big sensing data streams to avoid integrity and authenticity. In this paper, we proposed an end-to-end security by protecting big data streams against confidentiality based on data sensitive level.

2.3 Data confidentiality and integrity

There are several existing works on data confidentiality, data integrity, and end-to-end security [3][4][5][6][12][14][35][44][45][46][47] while data are in transit. SPINS is a very popular and well accepted security protocol proposed for sensor networks at the very beginning in 2002 [47]. SPINS protocol has two blocks and those are (i) SNEP and (ii) μTESLA. SNEP provides data authentication, data confidentiality, and data freshness whereas μTESLA ensures authenticated broadcast. A lightweight security protocol named LiSP is designed by focusing on efficient rekeying without any interruptions [12]. The LiSP protocol requires just three hash functions and storage room for eight keys on average by reducing significant resource consumption. Lightweight Security protocol (named LSec) provides authentication and authorization of source sensing nodes along with a data confidentiality mechanism against intrusions and anomalies [4]. A novel lightweight security method is proposed for sensor data streams in [3]. This scheme relies on Bloom filters in data packets to encode source data and perform security verification over data streams at the base station. The location-aware end-to-end security framework for static sensor networks is proposed for node-to-node and node-to-sink authentications and data confidentiality in [44]. The protocol uses secret keys that are bounded to geographical area and nodes store a few keys based on its origin. Data aggregation in the intermediate node is a challenging task where the source is sensor networks. Pietro et al. [45] proposed a novel aggregation technique which provides both integrity and confidentiality over aggregated data and also detects false data injection efforts. A sensitive application oriented lightweight security solution is proposed in [14], which contains four components such as STKS (a secure triple-key scheme), SRAs (secure routing algorithms), SLT (secure localization technique) and a malicious node reveal method. A symmetric key based lightweight security scheme is proposed by considering energy consumption of hardware components in [5]. This scheme ensures integrity and confidentiality of the data collected from a WBAN (wireless body area networks), either data stored inside sensors or during data transmission towards a centralized controller for healthcare applications.

The focus of this research was on data confidentiality and/or integrity, however, none of the solutions talks about data confidentiality based on the data sensitivity levels. The selective encryption technique is popular and well accepted in the multimedia (image/video) computing domain [48]. However, in this domain researchers are not particularly aware of protecting data based on the confidentiality levels. We adopt the selective encryption concept to propose a new method for big sensing data streams.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TBDATA.2017.2702172, IEEE Transactions on Big Data

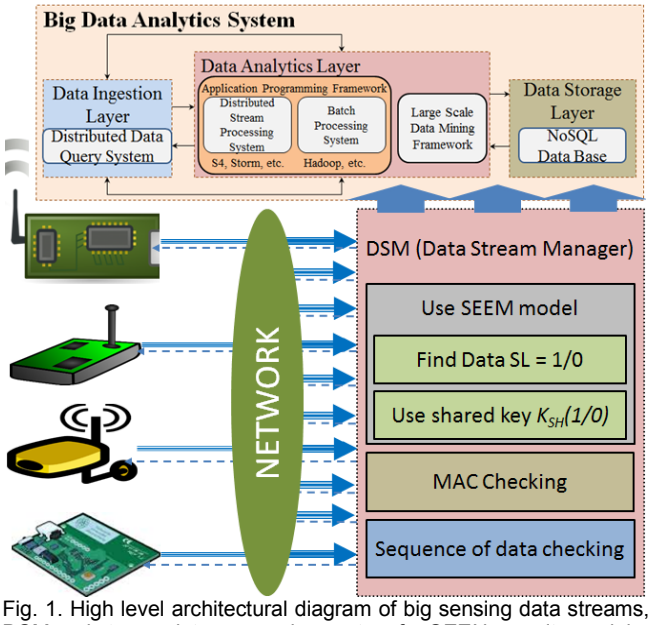

Fig. 1. High level architectural diagram of big sensing data streams, DSM and stream data processing system for SEEN security model.

3 Design Considerations

3.1 System Architecture

The overall architecture of a big sensing data streams including security and model is shown in Fig. 1. The architecture includes source sensing devices to transmit data to the DSM through wireless networks including our security model (SEEN). Big data stream processing is an emerging computing model which is particularly suitable for several application scenarios where huge volume and velocity of data (Big Data stream) must be processed in near real time (with a small delay). Several applications such as terrestrial monitoring, disaster monitoring, military monitoring and healthcare need data processing in near real time [5][18][19]. The needs of near real time processing include huge input data that discourages the use of instant storage, the obligation of generating rapid results, etc. We follow [19][20] to design a DSM, which is capable of handling high-volume and variety data streams from multiple sources. DSM dropped modified or data packets from malicious source to ensure only original sensed data available at SPE for analysis. SPE is shown at the top of the Fig. 1. In addition, the DSM is responsible for performing the security verification of the incoming data streams in near real time to synchronize with the processing speed of SPE (Spark Streaming, Apache Strom, Apache S4, etc.). For further information on stream data processing on datacenter, refer to [21].

Along with this, we consider that both source sensor and cloud data center deployed with Intrusion Detection Systems (IDS). Sensor based IDS monitor a sensor's behavior and generates alerts on potentially malicious activities onboard and network traffic [7]. IDS can be set inline, attached to a spanning port of a sensor. The idea here is to allow access to all packets we wish the IDS to monitor. LEoNIDS (low-latency and energy efficient network IDS) is a system that determines the energy expectancy trade off by giving both lower power utilization and lower recognition expectancy [8]. By highlighting the cloud based IDS, Lee et al. [9] proposed an intrusion detection system where the learning operators persistently process and give the redesigned methods to the discovery agents for efficient learning and real-time detection. It generally computes inter and intra audit record patterns; this can guide the data gathering process and simplify feature extraction from audit data. Xie et al. [10] proposed a novel technique to analyze the system (sensor) vulnerabilities and attack sources quickly and accurately.

In our architecture, the data streams are always in the encrypted format when they arrive at the DSM. Our idea is that while encrypting the data packets at the source, we attach sensitivity level of data to each individual data packet. In the SEEN method, we apply different keys to encrypt the data packets for different data sensitivity levels. The aim is to provide different confidentiality levels based on the applications as well as the sensitivity levels of the data. In a very generic representation, if we need n levels of data security then n-1 keys ($Shared\ Key(K_{SH})$) are required for encryption/decryption. In this paper, we are considering three levels of data confidentiality: strong confidentiality, partial confidentiality, and no confidentiality; and two keys (i.e. $k1$, $k2$) for encryption methods. The strong encryption method uses $k1$ and is used to provide strong confidentiality, and the weak encryption method uses $k2$ to support partial confidentiality. Note that we do not need to encrypt the data packets for no confidentiality.

$$Shared\ Key(K_{SH}) = \begin{cases} security\ level - n & key\ (1) \\ \vdots & \vdots \\ security\ level - 2 & key\ (n-2) \\ security\ level - 1 & key\ (n-1) \\ security\ level - 0 & no\ key \end{cases}$$

Data packets can be transmitted to DSM using two different ways of encrypting data: (i) encrypt the data stream and (ii) encrypt the data packets in the stream. In both these ways, we are going to apply encryption methods (strong/weak encryption) based on the data sensitivity or confidentiality level. The encrypted data stream applies to those sensors which are deployed with the sensitivity levels, whereas encrypted data packet applies to the sensors with different sensitivity levels for different types of data.

Here, we follow a three step process, data collection, security verification, and stream query processing at DSM as highlighted in Fig. 1. Our focus is to perform the security verification at DSM by providing an end-to-end security of big sensing data streams. It is also important to perform a security verification of a data stream before the stream query processing in order to maintain the originality of the data for SPE. The security verification needs to be done on-the-fly (i.e. near real-time) with a smaller buffer size. The queries including security verification can be defined as a directed acyclic graph; and each node is an operator and edges define the data flows between the nodes.

The above system architecture and security requirements

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TBDATA.2017.2702172, IEEE Transactions on Big Data

of big data streams [19][20] lead to the following two important features:

- Data packet needs to maintain confidentiality based on its sensitivity level.
- Need optimized buffer size at DSM prior to stream query processing.

Motivated by this problem, this paper aims to address the challenge of data integrity and multilevel confidentiality on real time massive data streams.

3.2 Adversary Model

We assume that a large number of sensor nodes are the sources of big sensing data streams that are fully connected and can communicate to the DSM through wireless networks. We assume that the DSM is aware of the network topology and initially deployed nodes. We assume that IDS is positioned at each source device and at the DSM so that source sensors and DSM are capable of detecting packet-loss attacks and data modifications [3]. The DSM is treated as fully secured and protected in our model as it resides at the cloud data center.

An attacker has several ways of attacking big sensing data streams:

- After the deployment, the nodes may be captured by the attacker who will then be able to access the data stored in these nodes, as well as reprogram them and control their actions. The attacker could therefore make nodes refuse to forward some of the packets (*Selective Forwarding* attack) or even all of them (*Blackhole* attack).
- The attacker may capture the data packets in the middle to get the information out of them and modify the content of a data packet. The attacker can therefore cause the loss of confidentiality (*confidentiality attack*) of sensitive information and data integrity (*integrity attack*).
- A replay attack (also well-known as playback attack) is a network based attack where a data stream is maliciously delayed or fraudulently repeated.

Compromising a node to drop packets and introducing interference in the network to access/tamper with the data are, from a high-level perspective, the two ways in which an attacker can disrupt data transmission through a packet-loss attack. For this reason, our adversarial model covers many different attacks that aim at causing packet losses. The other type of attack is to capture sensitive data packets and analyze to break the data confidentiality.

Each node whose IDS detects a packet loss attack, will investigate the loss; we assume the investigating source device to be trustworthy and not to report any false response. This assumption is particularly important for the Majority Voting algorithm adopted as part of our approach. However, we will also present a variant of this algorithm able to relax this constraint, and thus able to tolerate up to a confident number of colluding investigating source nodes.

3.3 Attack Models

There are mainly three threat approaches for attack models, i.e., attack centric, software centric and asset centric. An attack centric threat model always starts with an attacker, whereas a software centric threat model starts with system designing. An asset centric threat model follows the information collection and assets entrusted, so our proposed method is an asset centric threat model.

We assume that multiple simultaneous attacks can be carried out at the same time at various parts of the network. In fact, the strength of our approach is that multiple simultaneous investigations can be carried out.

The integrity of a big data stream ensures that a message sent from sources to the data center (DSM) is not modified by malicious intermediates. Authentication of big data streams ensures that the data are from legitimate sources to maintain the end-to-end security.

Data confidentiality (privacy) is a set of guidelines that restrict access or puts limitations on specific data streams. This guarantees that given data cannot be comprehended by anybody other than the desired receivers whether the data is in transit or at rest.

The effect on data confidentiality of a successful exploit of vulnerability on the target system as follows.

- *Strong confidentiality*: Only desired recipients can read the information.
- *Partial confidentiality*: There is considerable informational disclosure in some situations.
- *No confidentiality*: A total compromise of information as confidentiality is not a hard requirement.

TABLE 1
NOTATIONS.

Acronym	Description
S_i	ith source sensing device's ID
K_i	ith source sensing device's secret key
K_D	DSM secret key
k	Initial secret key
K_{SH}	Initial shared key generated by DSM
$K_{SH}(1)$	Shared key for strong encryption
$K_{SH}(0)$	Shared key for weak encryption
T	Time of packet generation
T'	Time of packet receive at DSM
R_1/R_2	Pseudorandom number
CAC	Centralize authentication code
SL	Data sensitivity level
MAC	Message authentication code
$E(\ ) / D(\ )$	Encryption/Decryption function
$H(\ )$	One-way hash function
$\oplus$	X-OR operation
$\|$	Concatenation operation

4 PROPOSED METHOD

In this paper, we propose a selective encryption method for big data stream (SEEN) which is furnished with key renewability and makes a tradeoff among security, performance and resource utilization. The SEEN security method's salient features are as follows:

- efficient key broadcasting without retransmission;
- ability to recover the lost keys with a proper detection;

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TBDATA.2017.2702172, IEEE Transactions on Big Data

- seamless key refreshment without interrupting the data streams; and
- maintain the data confidentiality based on the data sensitivity level.

We describe the proposed security method for big sensing data streams using four independent components: system setup, rekeying, new node authentication, and encryption/decryption. We refer readers to Table 1 for all notations used in describing our scheme. We made a number of sensible and practical assumptions to characterize the proposed security method. We describe those assumptions where necessary. We next describe independent components in detail.

4.1 Initial System Setup

We follow the symmetric key method for the initial system setup because of the limited resource availability at the source sensors [11]. In symmetric key encryption, hashing function need 5.9 µJ and encryption techniques 1.62 µJ whereas in an asymmetric key, RSA-1024 needs 304 mJ to sign and 11.9 mJ to verify and ECDSA-160 needs 22.82 mJ to sign and 45 mJ for verification [11]. So we choose to follow symmetric key methods for the initial setup. In the system setup process, DSM always starts the process to identify the authenticated source. After successful authentication, DSM shares the secret shared keys to the source sensors for encryption. The initial shared key setup phase is as follows:

DSM generates a pseudorandom number (R_1) and performs a hash function combined with its DSM secret key (K_D) to generate a unique secret shared key. Then DSM encrypts the generated shared key by using the pre deployed secret key (k) which is initialized during the network setup to generate Centralize Authentication Code (CAC). The DSM broadcasts the CAC to all the source sensors i.e. (1, …, n).

$K_{SH} = \{H(E(R_1, K_D))\}$

$CAC = E_k(K_{SH})$

$DSM \rightarrow S_{(1\cdots n)}\ \{CAC\}$

Once all the sensors receive the broadcast CAC from the DSM, sensors decrypt it by using a pre deployed secret key (k) (i.e. $K_{SH} = D_k(CAC)$). Here we show the operation for a single senor (i.e. i^{th} *sensor*). The following is the procedure to be performed at the sensor and sends an encrypted CAC to the DSM. The CAC contains source ID, random number as nonce, and a timestamp to avoid replay attack.

$K_{SH} = D_k(CAC)$

$CAC2 = E_{K_{SH}}(R_2 \parallel S_i \parallel T)$

$DSM \leftarrow S_i\ \{CAC2\}$

Once the CAC is received at the DSM, it decrypts and checks the source ID (S_i) for authentication and retrieves the corresponding sensor secret key from its data base ($K_i \leftarrow$ retrievekey (S_i)). It also checks the time stamp to avoid replay attacks. The complete procedure for authentication and replay attack avoidance is shown below.

$D_{K_{SH}}(R_2 \parallel S_i \parallel T)$

$K_i \leftarrow$ retrievekey (S_i) // for source authentication

$T' - T \leq \Delta T$ *(T-packet generated time; T'-Packet receive time)*

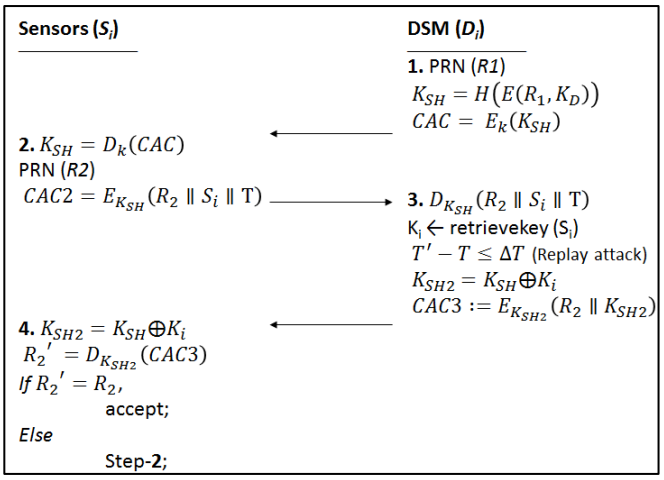

Fig. 2. Initial authentication methods with 4 step process.

It compares the received time frame (T) with its current time (T') to check the data freshness in order to avoid a replay attack (T - $T' \leq \Delta T$). If the time difference is less than ΔT , the DSM accepts the data packet otherwise the packets are discarded.

The DSM then generates a new key by performing X-OR on the existing shared key and sensor's secret key. The DSM uses this shared key to encrypt the nonce and sends back to the corresponding sensor for handshaking along with weak encryption shared key.

$K_{SH2} = K_{SH} \oplus K_i$

$CAC3 := E_{K_{SH2}}(R_2 \parallel K_{SH2(0)})$

$DSM \rightarrow S_i\ \{CAC3\}$

After a sensor (S_i) receives the data packet, it performs the same operation as DSM did to find the new shared keys to encrypt the data packets. It compares the decrypted nonce (R_2') with the nonce it has (R_2); if both are the same, then it accepts otherwise it rejects and starts a new authentication process. Received $K_{SH2(0)}$uses 64-bit key for weak encryption and K_{SH2} uses 128-bit key for strong encryption.

$K_{SH2} = K_{SH} \oplus K_i$

$R_2' := D_{K_{SH2}}(CAC3)$

If $R_2' = R_2$, then the sensors accept otherwise the process starts from the beginning. The complete authentication process is shown in Fig. 2, where we show the stepwise process with information flow.

4.2 Re-keying

After this initial key setup phase, the DSM shares the shared secret key with sensors for encryption. For the rekeying process, we follow a LiSP protocol [12] and modify it to make SEEN more data centric instead of communication centric. SEEN uses a key server (KS) at the DSM, that manages the security keys for both strong and weak encryption. We use 128-bit symmetric shared key for strong encryption and 64-bit symmetric key for weak encryption. Shared keys from KS are always chosen to perform the rekeying operation. Along with the shared key, individual sensors are able to perform the hash function.

In-order to make the system more secure, the shared key

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TBDATA.2017.2702172, IEEE Transactions on Big Data

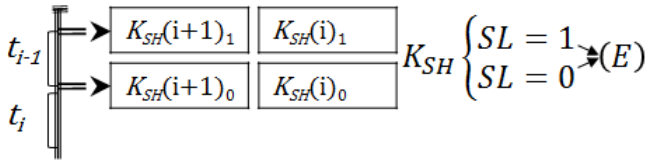

Fig. 3. Key selection

distribution for rekeying must be secure and fault tolerant; where "secure" means to maintain the confidentiality and authenticity and "fault tolerant" implies the capacity to restore the lost shared key (K_{SH}). In our SEEN method, we always use two kinds of control packets i.e. *UpdateKey* and *RequestKey*. *UpdateKey* is for periodically updating the shared key used by DSM, whereas *RequestKey* is used by sensors when they missed the shared key during the rekeying process.

We follow PRESENT [13] to generate the shared key at DSM and distribute the key before it is used for encryption at source sensors. The sensors have two buffer places for each key; that means four buffer places are required to save the keys as shown in Fig. 3. The front two shared key are always used for encryption and the back buffers contain the next shared key before the current shared key expires.

Algorithm 1. Rekeying process

t – time to rekeying
t′ – time to DSM starting the shared key distribution
δt – small time before t expires

1. At time t′: the DSM broadcasts (*UpdateKey*)
   UpdateKey ⇒ $E_{KSH}(K_{SH}(1) \parallel K_{SH}(0))$
2. Sensors use the current shared key (E_{KSH}) to get the next shared key
   $D_{KSH}(K_{SH}(1) \parallel K_{SH}(0))$
3. At time δt: If any sensor does not have the next shared key
   Sensors unicasts to DSM (*RequestKey*)
   RequestKey ⇒ $E_{KSH}(S_i \parallel t_i)$
4. After authentication, the DSM unicasts (*UpdateKey*)
   UpdateKey ⇒ $E_{KSH}(K_{SH}(1) \parallel K_{SH}(0))$

To ensure secure shared key distribution, the DSM initiates the shared key distributions by encrypting the control packet (*UpdateKey*) using the current shared key ($K_{SH}(i-1)$) to distribute the next shared key ($K_{SH}(i)$). The *UpdateKey* is always in the format of $E_{KSH^{(i-1)}}(K_{SH}^{i}(1) \parallel K_{SH}^{i}(0))$, *where* K_{SH} is the current shared key and all authenticated sensors have this key to perform the encryption. Let us assume the time to change the shared key is t; this means the DSM needs to initialize the shared key before the time t′. If the sensor did not get the shared key at time δt (t-t′= δt), then it initiates the *RequestKey*. The *RequestKey* always in the format of $E_{KSH}(S_i \parallel t_i)$, the source ID ($S_i$) along with time slot ($t_i$) encrypted with the current shared key (K_{SH}). Then DSM can decrypt the *RequestKey* control packet using current shared key (K_{SH}) and authenticate using the source ID. In such situations, the DSM sends an *UpdateKey* message to the corresponding sensors. Algorithm 1 shows the procedure for rekeying.

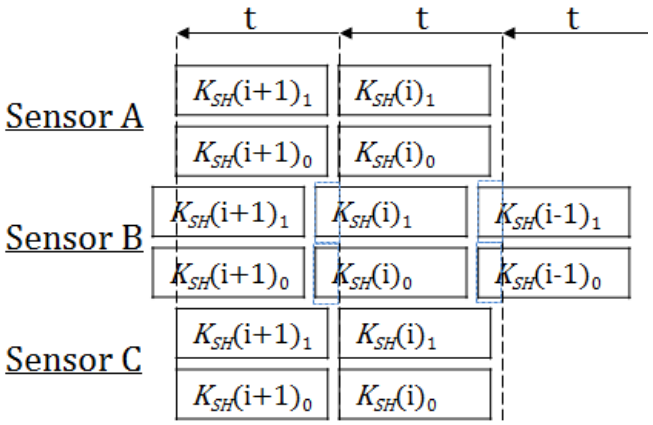

Fig. 4. Shared key management for Robustness to Clock Skews

4.3 New Node Authentication

Joining new nodes to the network is a common property of sensor networks. We assume that the source node is initialized by the DSM during the initial deployment [14]. In such cases, source sensors always start the process by authenticating with the DSM to get the current shared key. Sensors use a control packet (i.e. *InitKey*) to start the process. *InitKey* contains the source ID encrypted with the initially deployed secret key i.e. $E_k(S_j)$. Once the DSM receives the control packet, it checks its authenticity. If the DSM succeeds in the authentication process, then it follows the Initial key setup (from Fig. 1) phase to share the current shared key. The DSM uses the current shared key (K_{SH}) instead of generating a new key i.e. $K_{SH} = \{H(E(R_1, K_D))\}$. At the final stage of sharing the shared key, the DSM shares the keys along with a time stamp (t_i) to source sensors ($E_{K_{SH_2}}(R_2 \parallel K_{SH2(0)} \parallel t_i)$). For the robust clock skew and shared information details, the source sensor can get the information from its neighbours [12].

4.4 Reconfiguration

The DSM will configure the shared key at the time of the next rekeying process, if (1) any of the source sensors have been compromised; (2) any of the shared keys have been revealed; (3) a source node has overtly requested the shared key; or (4) a source has joined to participate in the data stream. The first condition forces all source devices to be reconfigured, whereas the final two issues focus on requesting that the source to be configured. The actions required for the issues highlighted above are summarized as follows:

(I) DSM withdraws the compromised nodes as the authenticated source, and if the $K_{SH}(i)$ has been disclosed previously. This may expose all earlier shared keys.

(II) DSM computes new shared keys for both strong and weak encryption and unicasts with control packets.

(III) DSM replies to the requesting source with current configuration.

(IV) DSM follows the authentication process, and if successful, DSM responds to the source by initializing an

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TBDATA.2017.2702172, IEEE Transactions on Big Data

InitKey control packet.

4.5 Encryption/Decryption

The above defined process makes both shared keys ($K_{SH}(1) \parallel K_{SH}(0)$) available at sensors. Note that $K_{SH}(1)$ *is* always used for strong encryption, whereas $K_{SH}(0)$ *is* always used for weak encryption. Each data block generated at sensors is a combination of two different parts. The first part is for integrity checking and maintaining the confidentiality level, whereas the other part is for the source authentication (i.e., $A_D = E_{K_{SH}(1)}(S_i \parallel {}^{1}/_{0} \parallel \mathrm{T})$. The authentication part is always encrypted using the strong encryption key; it contains the source ID for authentication, time stamp (T) to avoid replay attack, and a flag value 1/0;where 1 is for strong encryption of body part (highly sensitive data) or 0 for weak encryption of body part (low sensitivity data). In order to encrypt the data part of the packet, every sensor performs the XOR operation i.e. current shared keys $K_{SH}(1/0)$ with its own secret key (k_i), i.e. $K_{SH}(1)' = K_{SH}(1) \oplus K_i$ and $K_{SH}(0)' = K_{SH}(0) \oplus K_i'$ then it uses the newly generated key to encrypt the data packets. Shared key $E_{KSH(1)'}$ is always used for strong encryption, whereas shared key $E_{KSH(0)'}$ is used for weak encryption (DATA‖MAC). The above specified data block encryption is always based on the data sensitivity level and for data integrity and confidentiality.

The Capture layer of an IoT system (i.e. physical layer of sensor networks) is responsible for obtaining the context of data from the deployed environment using source sensors. This layer is also accompanied by classification methods, which mostly follow the unsupervised neural network methods, such as KSOM (Kohonen Self-Organizing Map) used to categorize the real time sensed data [15]. The word "sensor" not only signifies a sensing device but also applies to each data source that may deliver functional context information [16]. Ganesan et al. [17] proposed a similar kind of system, named DIMENSIONS, where authors extended sensors for computation, storage and system performance. We follow the KSOM technique to classify the sensed data at sensors to define the sensitivity level. KSOM uses data mining techniques and classification to extract the data sensitivity level. Few sensors are also pre-deployed with high sensitivity level, where all generated data packets are sent to the DSM with a high sensitivity level. The steps to select the encryption method and shared key are shown in Fig. 5. The strong encryption method always uses a shared key $E_{KSH(1)}$ for highly sensitive data, whereas the weak encryption method always uses the shared key $E_{KSH(0)}$ for low sensitivity data.

After data are received at the DSM, it always checks the authentication block and applies the strong encryption shared key to get the authentication information i.e. $D_{K_{SH}(1)}(S_i \parallel {}^{1}/_{0} \parallel \mathrm{T})$. Once it gets the source sensor ID, it checks its own database to find the match and confirms that data packets are from authenticated sources. After

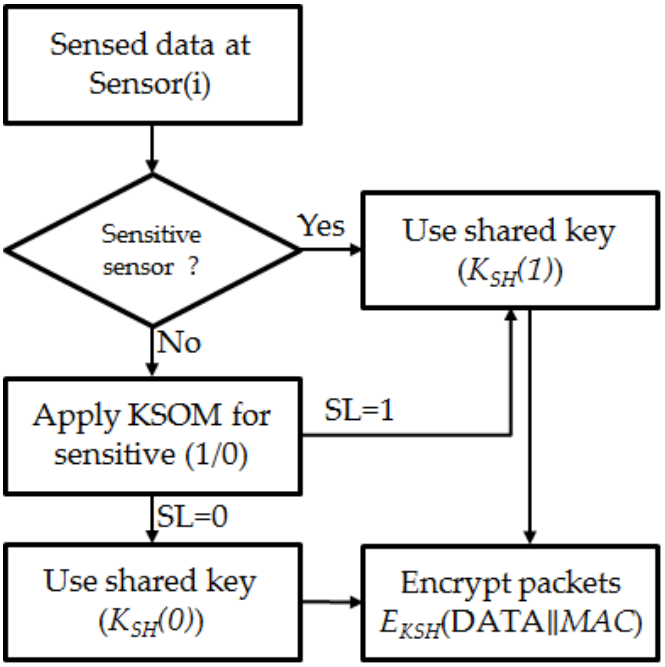

Fig. 5. Method to select encryption method based on the data sensitive level

successful authentication, the DSM compares the received time frame (T) with its current time (T') to check the data freshness in order to avoid a replay attack ($T - T' \leq \Delta T$). After successfully checking for a replay attack, the DSM retrieves the corresponding secret key of the sensors i.e. $K_i \leftarrow$ retrievekey (S_i) and checks the data sensitivity level to find out the shared key used for encryption. If the data sensitivity level is 1, then it performs the XOR operation i.e. $K_{SH}(1)' = K_{SH}(1) \oplus K_i$, else $K_{SH}(0)' = K_{SH}(0) \oplus K_i$. The computed new key is used for data decryption i.e. $D_{KSH(1)'}$ (DATA‖MAC) for high sensitivity data and $D_{KSH(0)'}$ (DATA‖MAC) for low sensitivity data. After data decryption, the DSM compares the MAC as an integrity check. The DSM always keeps the last shared key $K_{SH}(i\text{-}1)$ during the use of $K_{SH}(i)$. There is always the possibility of late arrival of data packets at a DSM because of the untrusted wireless communication medium.

4.6 Tradeoffs

The communication overhead of the proposed security method depends on the source network size. Each group of keys will have more chances of being compromised as the network size increases [12]; this in turn increases the chances of reconfigurations. The SEEN method accomplishes significant improvement over a traditional rekeying approach by broadcasting shared keys without retransmissions. Less frequent reconfigurations mean increased performance. In summary, a larger network improves the performance by more proficient broadcasting. Therefore, we can tradeoff between energy consumption and security to maximize the overall performance.

The proposed shared key management is made robust by synchronizing clocks among neighbors. As an example, Fig. 4 describes the time domain and key-slots of three

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TBDATA.2017.2702172, IEEE Transactions on Big Data

sensors i.e. A, B and C, where there is a clock skew among three nodes. Every *UpdateKey* control packet contains the time stamp to switch the shared key (t_i).

4.7 Required Resources for SEEN

4.7.1 Resources at Sensors

We follow [4] to define the communication overhead and power consumption theoretically. Following are the equations to define the communication overhead and power consumption

$$CO(\%) = \left(\frac{N_c \times 74.125}{\sum_{i-1}^{n} N_i \times 30}\right) \times 100 \quad (1)$$

$$PC_1 = 3(CSE + CSD) \quad (2)$$

$$PC_2 = (TNSP + CSE) + (TNRP + CSD) \quad (3)$$

$$PC_3 = (CSE + 2 \times CSD) \quad (4)$$

CO – Communication overhead

PC_1 – power consumption during node authentication (initial phase)

PC_2 – power consumption by a node during data transmission

PC_3 – power consumption during the rekeying

N_c – total number of connection

N_i – number of packets transferred by node S_i

CSE – computational power required by symmetric key encryption

CSD – computational power required by symmetric key decryption

$TNSP$ – total number of sent packets

$TNRP$ – total number of received packets

The communication overhead is always computed as a percentage and considers the total number of communications and the number of packets transferred by each sensor (*see equation 1*). The number of packets/total size is 74.125 bytes, whereas the data packet size is 30 bytes. Power consumption for initial node authentication is three times that required by both encryption and decryption processes (*see equation 2*). Each node needs a certain amount of power to participate in data transmission and data packet encryption. The normalized form of power consumption shows in *equation 3*. Sensors need power to decrypt the *UpdateKey* and also to initiate *RequestKey* during the rekeying process. Sensors always need more power to perform the encryption and decryption process. *Equation 4* shows the formulations for power consumption during rekeying.

4.7.2 Resources at DSM

The buffer utilizations needs to be optimized at the DSM, as the security mechanism of a big data stream needs to be performed in near real time because of the big data stream features [19][20]. Here we present a procedure to compute the halting time of a data block in a buffer before the stream data analysis is done. Let there be n number of sensors and each sends m number of data packets. We assume that the probability of attempt to success security verification at DSM is $\left(1/(n \times m)\right)$, or delays with probability $1 - \left(1/(n \times m)\right)$. We can compute Acquisition Probability as $A = \left(1 - \left(\frac{1}{(n \times m)}\right)^{((n \times m)-1)}\right)$ [25]. Based on the value of A, we can measure the halting time of the each individual data block; the halting time, represented as w, is $A \times (1 - A)$, where the value of w is inversely proportional to the value of A and security verification time of DSM.

5 Security Analysis

We follow [19][20][49] to define the following attack definitions and their properties. Based on these attack definitions, we have proved the following theorems and theoretical analysis.

Definition 1 (attack on authentication): An attacker *Ma* can attack on authenticity if it is an adversary capable of monitoring, intercepting, and introduce him/herself as an authenticated node to participate in the data stream.

Definition 2 (attack on integrity): An attacker *Mi* can attack on reliability if it is capable of monitoring the data stream and trying to access and/or modify the data block before DSM.

Definition 3 (attack on confidentiality): A malicious attacker M_c is an unauthorized party which has the ability to access or view the unauthorized big data streams.

Definition 4 (replay attack): A malicious attacker M_r is an unauthorized party which has the ability to intercept data packets and forward them later. This may be cause the loss of event detection during stream data analysis.

Theorem 1. *Strong encryption (128-bit) is always safer and takes a more computational power and time than weak encryption (64-bit) in SEEN security model.*

Proof. ECRYPT II proved that the key length of a 128-bit symmetric key provides the same strength of protection as a 3,248-bit asymmetric key [22][23]. Symmetric key cryptography becomes a natural choice for this purpose. It is mentioned with a proof that symmetric key cryptography is approximately 1000 times faster than strong public key ciphers [23]. From [19][23], it is comparatively easy for an attacker to read/modify packets, which are encrypted with a smaller key length. Crypto++ Benchmarks [24] also confirm that a smaller key length always takes less time to break or find the shared key. From the above, we conclude that the key length is directly proportional to the key domain size and also directly proportional to the time required to find all the possible keys (see Table 2). This means an attacker needs more computational time and resources to break 128-bit compared to 64-bit.

Theorem 2. *DSM can easily identify delayed data packets which have been intercepted by a replay attacker (M_r) using the SEEN security method.*

Proof. A replay attack is also broadly known as playback attack, where an attacker (M_r) intercepts the data packet(s) of data streams and forward later. Attacker also repeatedly send the data packets to block the DSM. This is carried out either by the source sensor or man-in-the-middle attack.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TBDATA.2017.2702172, IEEE Transactions on Big Data

In the SEEN security method, during encryption the source sensor always adds a time stamp i.e. T (sending time/packet generation time) at the header part of the data packets. This is in the format as $E_{K_{SH}(1)}(S_i \parallel 1/0 \parallel \mathrm{T})$ and is always used for authentication and data freshness. For every data packet, the DSM compares the received time frame (T) with its current time (T') to check the data freshness and to avoid a replay attack ($T - T' \leq \Delta T$). If the time difference is less then ΔT then the data packet is accepted otherwise it is discarded. Where ΔT tends to maximum time takes to transmit data from source to DSM. After successfully checking for replay attack, DSM follows the data decryption process.

Theorem 3. *In the SEEN security method, an attacker M_c cannot access or view the unauthorized high sensitive data stream, whereas M_c cannot read low sensitivity data stream in real time.*

Proof. By following Fig. 5, it is clear that every data stream or data packet within a stream is transmitted with a sensitivity level. Here in the SEEN method, we consider two sensitivity levels i.e. '1' for high and '0' for low. For highly sensitive data, sensors use 128-bit shared key i.e. $K_{SH}(1)$. The computational hardness of the shared key is shown in Table 2. It shows that the most advanced processor (Intel i7) takes decades to find all possible shared keys to perform the decryption operation. Attacker M_c cannot read the high sensitivity data (strong confidentiality).

For lower sensitivity data in the SEEN model, sensors use 64-bit shared key i.e. $K_{SH}(0)$. Attacker M_c also needs years to get all the possible keys to decrypt the data packets (see Table 2). The maximum time to update the shared key (i.e. t) is always less than the time required for 64-bit in Table 2. So attacker M_c can read the low sensitivity data but it is not possible in real time.

By following the above, we can confirm that the SEEN model maintains confidentiality for sensitive data and partial confidentiality for low sensitivity data.

Theorem 4. *An attacker M_a cannot forge the source to introduce itself as an authenticated source and attacker M_i cannot get the shared key K_{SH} to break data integrity in the proposed security method SEEN.*

Proof. According to IDS properties of a sensor [7][8], in SEEN security method the sensor always reports to the DSM if it is captured by an attacker M_a. In such situation, the DSM does not consider the data packets from that sensor for data analytics. Along with IDS report, DSM also checks the source authentication for each individual data packet, where data packets arrive in a format $A_D = E_{K_{SH}(1)}(S_i \parallel 1/0 \parallel \mathrm{T})$. DSM always applies the strong shared key (i.e. $K_{SH}(1)$) to check the source authentication. After decryption A_D of a data packet, DSM compares the S_i with its own database for source authenticate.

After a source device is authenticated, the DSM retrieves the corresponding secret key of the sensors i.e. $K_i \leftarrow$ retrieveKey(S_i) and checks the data sensitivity level to select the shared key used for encryption.

Based on the data sensitivity level, the DSM performs XOR operation i.e. $K_{SH}(1/0)' = K_{SH}(1/0) \oplus K_i$. The newly computed shared key will be used for data decryption i.e. $D_{KSH(1/0)'}$ (DATA ‖ MAC). After data decryption, DSM compares the MAC as an integrity check. Through the MAC check, the DSM confirms that the integrity of the data is intact.

TABLE 2
TIME TAKEN BY SYMMETRIC KEY (AES) ALGORITHM TO GET ALL POSSIBLE KEYS USING THE MOST ADVANCED INTEL I7 PROCESSOR [20].

Key Length	64	128
Key domain size	1.845e +19	3.4028e+38
Time (in days)	11415	1.9E+19

The major drawback (this is also applicable to all other security models) is that the confidentiality and integrity checks can be broken with a brute force attack.

Theorem 5. *The proposed SEEN requires a comparatively smaller buffer size compared to standard symmetric key solutions (i.e. AES-128) at DSM before stream query processing.*

Proof. Following *Theorem 3*, it is clear that the proposed SEEN security method provides a high level of data confidentiality for sensitive data, where it provides the partial confidentiality for low sensitivity data. We decrypt the header part for authentication (*see Theorem 4*) and data freshness (*see Theorem 2*); after successful authentication, we decrypt the data block for integrity checks. Another important mechanism is the different keys with the key length used for encryption/decryption. From *Theorem 1*, it is clear that the key length is directly proportional to the security verification and the security verification speed is inversely proportional to the buffer required for security verification. By combining the above, we conclude that the proposed SEEN security method needs a comparatively smaller buffer size. The evaluation is presented in the following section.

Forward secrecy

By following a standard symmetric key cryptography procedure, shared keys used for encrypting data packets are only used once and until they are expired (i.e. for time period t). Thus, previously used shared keys are worthless to an intruder even when a previously-used shared keys is known to the attackers. However, if an intruder continuously monitors the data stream for a long period of time, he/she can break the confidentiality of the low sensitivity data but not the high sensitivity data (see Table 2). At the same time, data integrity is always maintained.

6 EXPERIMENT AND EVALUATION

In order to evaluate the security strength and efficiency of the SEEN security method under the above specified adverse situations, we experimented in multiple simulation environments. The experiment was conducted using the in-house simulators on an Intel (R) Core (TM) i5-6300 CPU @ 2.40 GHz 2.50 GHz CPU and 8 GB RAM running on Microsoft Windows 7 Enterprise. We first

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TBDATA.2017.2702172, IEEE Transactions on Big Data

Scyther results : verify

Claim			Status	Comments
SEEN	S	SEEN,S2	Alive	Ok	No attacks within bounds.
		SEEN,S3	Nisynch	Ok	No attacks within bounds.
	D	SEEN,D2	Niagree	Ok	No attacks within bounds.
		SEEN,D3	Nisynch	Ok	No attacks within bounds.

Done.

Fig. 6. Scyther simulation results page for security verification.

verified the proposed security approach using Scyther [28]; second, we measured the performance of the approach using JCE (Java Cryptographic Environment) [29]; third, we computed the required buffer size to process our proposed approach using MatLab [30] to measure the efficiency of our method; finally, we used COOJA simulator in Contiki OS [31] to get the network performance of SEEN.

6.1 Security Verification

The SEEN security protocol is simulated in the Scyther simulation environment by using the underlying Security Protocol Description Language (.spdl). Scyther is an automatic security protocol verification tool that can be used to check the correctness of the security protocols. As per the Scyther model, we defined the roles of S and D, where S is a sensing device and D is the receiver (i.e. DSM). In this scenario, S and D have all information for encryption/decryption that is initialized in the system setup and rekeying phase. In this simulation environment, S sends the encrypted data packets to D for security verification. We introduced three types of attacks. First, an attacker changes the data packet while it is in the network. In the second, an adversary steals the property of source (S) and forwards the data packets to D pretending to be S. In the third, an adversary gets the data block to analyze and tries to read the data and replay the data packets. We experimented with 100 runs with 10 run intervals for individual claims with results as shown in Fig. 6. Here, we model the security method by following the previous section and used different key sizes (i.e. 64 bits; 128 bits) in random data packets. Here we follow the SEEN method to update the different keys (see Table 2).

Results: This experiment ranges from 0 to 100 instances in 10 intervals using different numbers of data blocks. We checked the data integrity and confidentiality after data packet authentication. As the key generation and distribution process is handled by DSM, so we assumed that none of the intruders have the shared secret key. We are using two different keys for the encryption process i.e. (K(0)) for weak encryption and ($K(1)$) for strong encryption. This also confuses the intruder in attempting to guess the key. During this experiment, we did not come across any potential attacks at the DSM to compromise the shared key, so it is secured in terms of confidentiality and integrity. Fig. 6 shows the result of the security verification experimented with in the Scyther simulation environment. Finally, we conclude that our model is secured against confidentiality and integrity attacks.

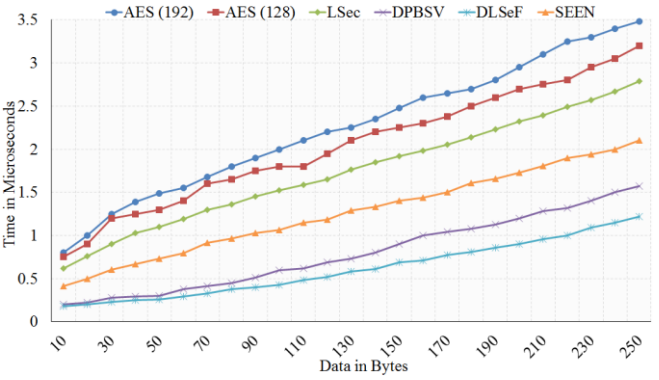

Fig. 7. Performance comparison SEEN method with AES-128, AES-192, LSec, DPBSV, and DLSeF.

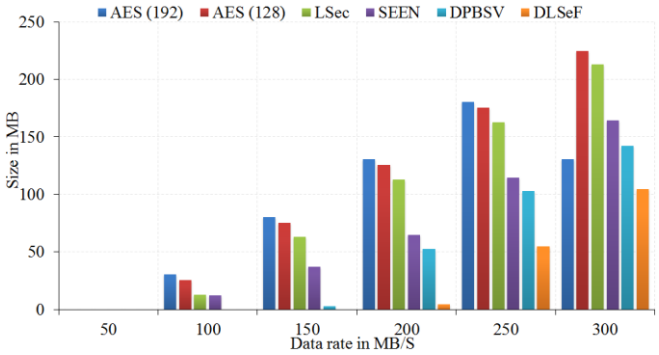

Fig. 8. Efficiency comparison by comparing required buffer size at DSM for security processing.

6.2 Performance Comparison

We used JCE (Java Cryptographic Environment) to experiment on and evaluate the performance of the SEEN method. JCE is the standard extension to the Java platform that provides an implementation context for cryptographic methods. The experiment is based on the features of the JCE in 64 bit Java virtual machine version 1.6. The security verification time of the experiment is computed in the DSM. The experiment outcomes for security verification are shown in Fig. 7. We performed experiments and compared security verification time with different sizes of data packets. We compare the performance of SEEN security with advanced encryption standard (AES-128, AES-192), LSec and our previously proposed model for big sensing data streams i.e. DPBSV and DLSeF [19][20][49].

Results: The experiment results of the SEEN security method are better than AES-128, AES-192 and LSec algorithms with different data packets as shown in Fig. 7. SEEN does not use the trusted part of sensor (i.e. TPM) and avoids confidentiality attacks in comparison to DPBSV and DLSeF (see Table 3). So even though the performance of SEEN is not as good as DPBSV and DLSeF, it is acceptable in any circumstance of sensor network applications. The performance of SEEN shows that it is more efficient and faster than AES-128, AES-192 protocols while providing the same level of security and removing some of the unrealistic assumptions of DPBSV and DLSeF.

6.3 Required Buffer Size

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TBDATA.2017.2702172, IEEE Transactions on Big Data

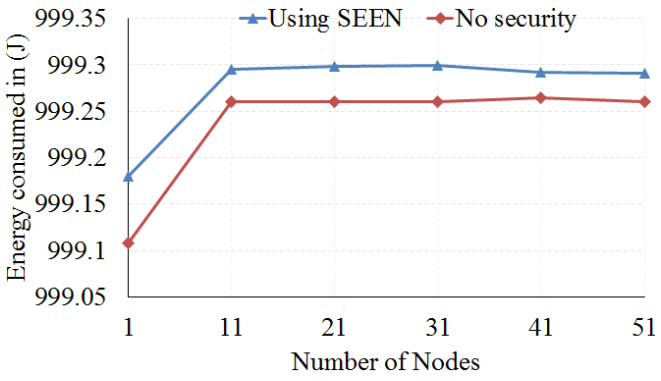

Fig. 9. Energy consumption.

The experiment for the required buffer size at DSM was carried out using a MATLAB Simulation tool. The buffer size is based on the security verification time at DSM (from Fig. 7) with respect to different velocity of big data streams. The performance is based on the verification time calculated as shown in Fig. 8. Here we compared our SEEN security method with standard AES-128, AES-192, LSec, DPBSV and DLSeF (see Fig. 8). The velocity of big data streams starts from 50 to 300 MB/S with a 50 MB/S interval. The required buffer size for SEEN is always smaller than the AES-128 algorithm with different rates of incoming data. Fig. 8 shows the minimum buffer size required at the DSM for the SEEN method in comparison with AES-128, AES-192, LSec, DPBSV and DSeF. The performance comparison proves that the SEEN method requires less buffer and is efficient in performing security verification without compromising security properties.

6.4 Network Performance

We tested our SEEN protocol using a COOJA simulator in Contiki OS to get the network performance (i.e. communication overhead and power consumption) [31]. We took the two most common types of sensor (i.e. Z1 and TmoteSky sensors) for network simulation. In this experiment, we checked the performance while computing and distributing the shared key.

For network simulation, we took a random area to deploy 51 nodes (i.e. 50 sensors and 1 DSM) in COOJA simulation environment. We took initial battery power of an individual sensor node $1x10^6$ J, power consumption for transmission is 1.6W and power consumption for reception is 1.2 W. Apart from these, we follow the default properties of sensors. We assume that the size of each data packet is 30 bytes, nonce 23 bits, secret key of 64/128 bits and token 4 bytes for the simulation [4].

In order to compute the performance of the communication overhead, our simulation used data packets of size 30 bytes of interval. We follow equation 1 to get the communication overhead. The performance of the communication overhead is computed as a percentage (%) with respect to the number of data packets as shown in Table 4. According to the network properties, the communication overheard is inversely proportional to the number of packets in the network as shown in Table 4.

For every connection, SEEN exchanges control packets for source/DSM authentication and shared key distributions based on the above specified packet sizes. This is an acceptable tradeoff between energy and security for the sensor node. The simulation results of energy consumption are shown in Fig. 9. The SEEN protocol required extra energy for the network authentication but its difference is very low. The energy consumption by using the SEEN protocol remains the same even by increasing the network size. We simulated the scenario using 50 nodes in 10 nodes interval as shows in Fig. 9.

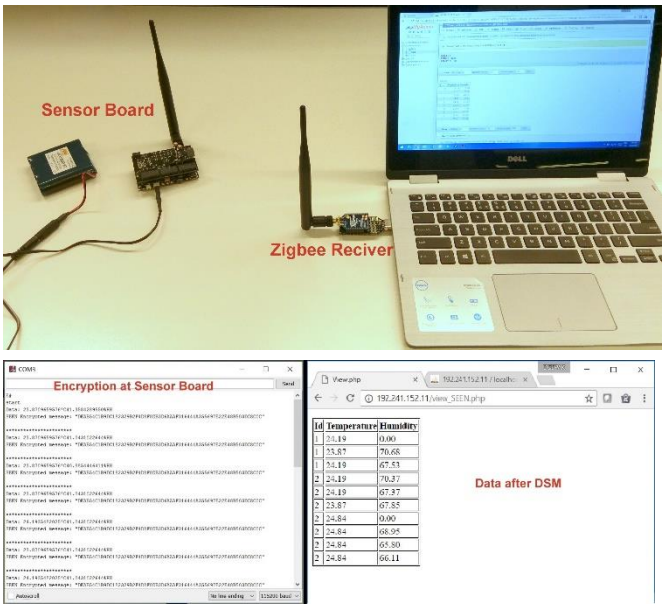

Fig. 10. Testbed implementation for end-to-end security using SEEN.

From all the above security analysis and experiments, we conclude that the proposed security method (i.e. SEEN) is secured (from multi-level confidentiality and integrity attacks), and efficient in terms of security verification speed and required buffer size at DSM (compared to AES-128, AES-192 and LSec). Table 3 shows the comparisons of SEEN security properties with AES-128 and existing DPBSV and DLSeF. It clearly shows that the proposed method provides the same level of security as AES-128 while reducing computational overhead.

6.5 Testbed Implementation

This section gives a testbed implementation scenario of our proposed method to prove that proposed SEEN security method work efficiently in real time streaming environment. We have taken waspmote smart cities board as the source of our data streams and a public cloud with specification as 512 MB Memory / 20 GB Disk / NYC1 - Ubuntu 14.04.5 x64 to deploy our DSM. We have deployed MySQL server database to store data streams after decrypted at DSM. Here in this testbed deployment, we plugged two sensors i.e. temperature and humidity sensors on smart cities board to get data. We use standard sensor network radio i.e. Zigbee network (IEEE 802.15.4) to stream data packets to cloud as shown in Fig. 10.

We used SEEN method to protect data streams in end-to-end basis. Sensed data encrypted using SEEN method at sensor board and followed by decrypted at DSM before store at MySQL server. The different level of encryption is used at sensor board in different instances without notifying DSM, and finally we found DSM can decrypt

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TBDATA.2017.2702172, IEEE Transactions on Big Data

TABLE 3
PERFORMANCE AND PROPERTIES OF SECURITY SOLUTIONS

	AES	DPBSV	DLSeF	SEEN
Authenticity	✓	✓	✓	✓
Integrity	✓	✓	✓	✓
Confidentiality	✓	P	P	✓
Trust on Sensor Node (TPM)	×	✓	✓	×
Computation	HIGH	LOW	LOW	LOW

P- Partially; TPM- Trusted Platform Module.

TABLE 4
COMMUNICATION OVERHEAD OF SEEN PROTOCOL

NP	10	20	30	40	50	60	70	80
CO (%)	25	23	12.8	11	8	6.8	6	6

NP- Number of packets; CO- Communication Overheard

the data packets and stored at MySQL server. We consider previous simulation results for big data streams performance, whereas this testbed implementation shows the evidence of SEEN method's compatibility in real time testbed environment.

7 CONCLUSION

In this paper, we proposed a Selective Encryption (SEEN) method to maintain confidentiality levels of big sensing data streams with data integrity. In SEEN, a DSM independently maintains an intrusion detection and shared key management as two major components. Our method has been designed based on a symmetric key block cipher and multiple shared keys use for encryption. By employing the cryptographic function with selective encryption, the DSM efficiently rekeys without retransmissions. The rekeying process never disrupts ongoing data streams and encryption/decryption. SEEN supports the source node authentication and shared key recovery without incurring additional overhead. We evaluated the performance of SEEN by security analyses and experimental evaluations. We found that our SEEN provides a significant improvement in processing time, buffer requirement and prevents data confidentiality and integrity from malicious attackers.

We will further investigate our proposed method to improve the efficiency of symmetric-key encryption. We are also planning to introduce the access control model over big sensing data streams, which will give access to the end user or query processor based on the data levels.

8 ACKNOWLEDGEMENT

Research in this paper is supported by AISRF-08140.

REFERENCES

[1] A. Arasu, et al. "STREAM: the stanford stream data manager (demonstration description)." In *ACM SIGMOD international conference on Management of data*, pp. 665-665, ACM, 2003.

[2] H-S. Lim, Y-S. Moon and E. Bertino, "Provenance-based trustworthiness assessment in sensor networks." In *Seventh International Workshop on Data Management for Sensor Networks*, pp. 2-7. ACM, 2010.

[3] S. Sultana, G. Ghinita, E. Bertino and M. Shehab, "A lightweight secure provenance scheme for wireless sensor networks." In *18th International Conference on Parallel and Distributed Systems (ICPADS)*, pp. 101-108, 2012.

[4] R. A. Shaikh, S. Lee, M. AU Khan and Y. J. Song, "LSec: lightweight security protocol for distributed wireless sensor network." In *IFIP International Conference on Personal Wireless Communications*, pp. 367-377. Springer Berlin Heidelberg, 2006.

[5] G. Selimis et al., "A lightweight security scheme for wireless body area networks: design, energy evaluation and proposed microprocessor design." *Journal of medical systems*, vol. 35, no. 5, pp. 1289-1298, 2011.

[6] G. Selimis, at al. "Evaluation of 90 nm 6 T-SRAM as Physical Unclonable Function for Secure Key Generation in Wireless Sensor Nodes", *in IEEE ISCAS Brazil*, pp. 567-570, 2011.

[7] M. Roesch,"Snort: Lightweight Intrusion Detection for Networks." *LISA*, vol. 99, no. 1, pp. 229-238. 1999.

[8] N. Tsikoudis, A. Papadogiannakis and E. P. Markatos, "LEoNIDS: a Low-latency and Energy-efficient Network-level Intrusion Detection System." *IEEE Transactions on Emerging Topics in Computing*, vol. 4, no. 1, pp.142-155, 2016.

[9] W. Lee and S. J. Stolfo, "Data Mining Approaches for Intrusion Detection." In *Usenix security*. 1998.

[10] Y. Xie, D. Feng, Z. Tan and J. Zhou, "Unifying intrusion detection and forensic analysis via provenance awareness." *Future Generation Computer Systems*, vol. 61, pp.26-36, 2016.

[11] A. S. Wander, N. Gura, H. Eberle, V. Gupta and S. C. Shantz, "Energy analysis of public-key cryptography for wireless sensor networks." In *Third IEEE international conference on pervasive computing and communications*, pp. 324-328. IEEE, 2005.

[12] T. Park and K. G. Shin, "LiSP: A lightweight security protocol for wireless sensor networks." *ACM Transactions on Embedded Computing Systems (TECS)*, vol. 3, no. 3, pp. 634-660, 2004.

[13] A. Bogdanov, et al. "PRESENT: An ultra-lightweight block cipher." In *International Workshop on Cryptographic Hardware and Embedded Systems*, pp. 450-466, 2007.

[14] T. A. Zia and A. Y. Zomaya, "A Lightweight Security Framework for Wireless Sensor Networks." *JoWUA*, vol. 2, no. 3, pp. 53-73, 2011.

[15] K. Van Laerhoven, "Combining the self-organizing map and k-means clustering for on-line classification of sensor data." In *International Conference on Artificial Neural Networks*, pp. 464-469. Springer Berlin Heidelberg, 2001.

[16] P. Ferreira and P. Alves, *Distributed context-aware systems*. Springer, 2014. DOI 10.1007/978-3-319-04882-6

[17] D. Ganesan, D. Estrin and J. Heidemann, "DIMENSIONS: Why do we need a new data handling architecture for sensor networks?." *ACM SIGCOMM Computer Communication Review, vol.* 33, no. 1. pp. 143-148, 2003.

[18] D. Puthal, S. Nepal, R. Ranjan and J. Chen, "A Secure Big Data Stream Analytics Framework for Disaster Management on the Cloud." In *18th International Conference on High Performance Computing and Communications*, pp. 1218-1225. 2016.

[19] D. Puthal, S. Nepal, R. Ranjan and J. Chen, "A dynamic prime number based efficient security mechanism for big sensing data streams." *Journal of Computer and System Sciences* vol. 83, no. 1, pp. 22-24, 2017.

[20] D. Puthal, S. Nepal, R. Ranjan and J. Chen, "DLSeF: A Dynamic Key Length based Efficient Real-Time Security Verification Model for Big Data Stream." *ACM Transactions on Embedded Computing Systems*, vol. 16, no. 2, pp. 51, 2017.

[21] R. Ranjan, "Streaming big data processing in datacenter clouds." *IEEE Cloud Computing*, vol. 1, no. 1, pp. 78-83, 2014.

[22] www.cloudflare.com (accessed on: 04.08.2014)

[23] J. Burke, J. McDonald and T. Austin, "Architectural support for fast symmetric-key cryptography." *ACM SIGARCH Computer Architecture News, vol.* 28, no. 5, pp. 178-189, 2000.

[24] Crypto++ Benchmarks. Available: http://www.cryptopp.com/benchmarks.html (accessed on: 30.07.2016)

[25] R. M. Metcalfe and D. R. Boggs, "Ethernet: distributed packet switching for local computer networks." *Communications of the ACM*, vol. 19, no. 7, pp. 395-404, 1976.

[26] M. Stonebraker, U. Çetintemel and S. Zdonik, "The 8 requirements of real-time stream processing." *ACM SIGMOD Record*, vol. 34, no. 4, pp. 42-47, 2005.

[27] R. V. Nehme, H-S. Lim, E. Bertino and E. A. Rundensteiner, "StreamShield: a stream-centric approach towards security and privacy in data stream environments." In *ACM SIGMOD International Conference on Management of data*, pp. 1027-1030. ACM, 2009.

[28] Scyther, http://www.cs.ox.ac.uk/people/cas.cremers/scyther/ (accessed on: 30.07.2016)

[29] N. Nagaratnam, L. Koved and Anthony Nadalin, *Enterprise Java Security: Building Secure J2EE Applications*. Addison-Wesley Professional, 2004.

[30] Matlab, http://au.mathworks.com/products/matlab/ (accessed on: 30.07.2016)

[31] Contiki operating system, http://www.contiki-os.org/ (accessed on: 30.07.2016)

[32] A. Arasu, et al. "Stream: The stanford data stream management system." *Technical Report 2004-20, Stanford InfoLab*, 2004.

[33] H. Balakrishnan et al., "Retrospective on aurora." *The VLDB Journal*, vol. 13, no. 4, pp. 370-383, 2004.

[34] D. J. Abadi et al., "The Design of the Borealis Stream Processing Engine." *CIDR*, vol. 5, pp. 277-289, 2005.

[35] F. Wu et al. "An efficient authentication and key agreement scheme for multi-gateway wireless sensor networks in IoT deployment." *Journal of Network and Computer Applications*, 2016.

[36] V. Gulisano, et al. "Streamcloud: An elastic and scalable data streaming system." *IEEE Transactions on Parallel and Distributed Systems, vol.* 23, no. 12, pp. 2351-2365, 2012.

[37] R. Adaikkalavan and T. Perez, "Secure shared continuous query processing." In *ACM Symposium on Applied Computing*, pp. 1000-1005. ACM, 2011.

[38] R. Adaikkalavan, X. Xie and I. Ray, "Multilevel secure data stream processing: Architecture and implementation." *Journal of Computer Security*, vol. 20, no. 5, pp. 547-581, 2012.

[39] J. Cao, B. Carminati, E. Ferrari and K.-L. Tan, "Acstream: Enforcing access control over data streams." In *IEEE 25th International Conference on Data Engineering*, pp. 1495-1498, 2009.

[40] D. Puthal, S. Nepal, R. Ranjan, and J. Chen, "Threats to Networking Cloud and Edge Datacenters in the Internet of Thing." *IEEE Cloud Computing*, vol. 3, no. 3, pp. 64-71, 2016.

[41] W. Lindner and J. Meier, "Securing the borealis data stream engine." In *10th International Database Engineering and Applications Symposium (IDEAS'06)*, pp. 137-147. IEEE, 2006.

[42] X. Xie, I. Ray, R. Adaikkalavan and R. Gamble, "Information flow control for stream processing in clouds." In *18th ACM symposium on Access control models and technologies*, pp. 89-100. ACM, 2013.

[43] R. V. Nehme, E. A. Rundensteiner and E. Bertino, "A security punctuation framework for enforcing access control on streaming data." In *IEEE 24th ICDE*, pp. 406-415, 2008.

[44] K. Ren, W. Lou and Y. Zhang, "LEDS: Providing location-aware end-to-end data security in wireless sensor networks." *IEEE Transactions on Mobile Computing*, vol. 7, no. 5, pp. 585-598, 2008.

[45] R. Di Pietro, P. Michiardi and R. Molva, "Confidentiality and integrity for data aggregation in WSN using peer monitoring." *Security and Communication Networks*, vol. 2, no. 2, pp. 181-194, 2009.

[46] E. Bertino, K-K R. Choo, D. Georgakopolous and S. Nepal. "Internet of Things (IoT): Smart and secure service delivery." *ACM Transactions on Internet Technology (TOIT)*, vol. 16, no. 4, pp. 22, 2016.

[47] A. Perrig, et al. "SPINS: Security protocols for sensor networks." *Wireless networks*, vol. 8, no. 5, pp. 521-534, 2002.

[48] F. Peng, X-q. Gong, M. Long and X-m. Sun, "A selective encryption scheme for protecting H. 264/AVC video in multimedia social network." *Multimedia Tools and Applications, pp.* 1-19, 2016.

[49] D. Puthal, S. Nepal, R. Ranjan and J. Chen, "A Synchronized Shared Key Generation Method for Maintaining End-to-End Security of Big Data Streams." In *50th Hawaii International Conference on System Sciences*, pp. 6011-6020, 2017.

[50] M. Dayarathna and T. Suzumura, "Automatic optimization of stream programs via source program operator graph transformations." *Distributed and Parallel Databases*, vol. 31, no. 4, pp. 543-599, 2013.

Deepak Puthal is a PhD student in the School of Computing and Communications at the University of Technology Sydney. He is also working as a graduate researcher at CSIRO Data 61, Australia. His research interests include big data analytics, Internet of Things, and information security.

Xindong Wu is a Professor in the School of Computing and Informatics at the University of Louisiana at Lafayette (USA) and a Yangtze River Scholar in the School of Computer Science and Information Engineering at Hefei University of Technology (China). He received the Bachelor's and Master's degrees in computer science from Hefei University of Technology, China, and PhD degree in artificial intelligence from the University of Edinburgh, United Kingdom. His research interests include data mining, knowledge-based systems, and Web information exploration. He is the steering committee chair of ICDM (IEEE International Conference on Data Mining). He is a Fellow of the IEEE and the AAAS.

Surya Nepal is a principal research scientist at CSIRO Data-61, Australia. His research interests include cloud computing, Big Data, and cybersecurity. He has a PhD in computer science from RMIT University, Australia.

Rajiv Ranjan is an associate professor (reader) in the School of Computing Science at Newcastle University, UK. His research interests include cloud computing, content delivery networks, and big data analytics for Internet of Things (IoT). He has a PhD in computer science from University of Melbourne.

Jinjun Chen is Deputy Director of Swinburne Data Science Research Institute and Director of Swinburne Big Data Lab at Swinburne University of Technology, Australia. His research interests include cloud computing, big data, data science, privacy and security. He has a PhD in computer science from Swinburne University of Technology, Australia