A Dynamic Key Length based Approach for Real-Time Security Verification of Big Sensing Data Stream

Deepak Puthal[*], Surya Nepal[†], Rajiv Ranjan[†], and Jinjun Chen[*]

[*]Faculty of Engineering and Information Technology
University of Technology, Sydney, Australia
[†]Digital Productivity Flagship
Commonwealth Scientific and Industrial Research Organization (CSIRO), Australia

{deepak.puthal,jinjun.chen}@gmail.com,
{Surya.Nepal, Raj.Ranjan}@csiro.au

Abstract. The near real-time processing of continuous data flows in large scale sensor networks has many applications in risk-critical domains ranging from emergency management to industrial control systems. The problem is how to ensure end-to-end security (e.g., confidentiality, integrity, and authenticity) of such data stream for risk-critical applications. We refer this as an *online security verification* problem. Existing security techniques cannot deal with this problem because they were not designed to deal with high volume, high velocity data in real-time. Furthermore, they are inefficient as they introduce a significant buffering delay during security verification, resulting in a requirement of large buffer size for the stream processing server. To address this problem, we propose a Dynamic Key Length Based Security Framework (DLSeF) based on the shared key derived from synchronized prime numbers; the key is dynamically updated in short intervals to thwart Man in the Middle and other Network attacks. Theoretical analyses and experimental results of DLSeF framework show that it can significantly improve the efficiency of processing stream data by reducing the security verification time without compromising the security.

Keywords: Security; Sensor Networks; Big Data Stream; Key Exchange.

1 Introduction

A variety of applications, such as emergency management, SCADA, remote health monitoring, telecommunication fraud detection, and large scale sensor networks, require real-time processing of data stream, where the traditional store-and-process method falls short of addressing the challenge [1]. These applications have been characterized to produce 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 while ensuring end-to-end security [6]. More formally, the characteristics of big data are defined by “4Vs” [10, 11] such as Volume, Velocity, Variety, and Veracity. The streaming data from sensing source meets these characteristics. Our focus in this paper is thus on secure processing of high volume, high velocity data stream.

Big data stream is continuous in nature and it is critical to perform real-time analysis as: (i) the life time of the data is often very short (i.e., the data can be accessed only once) [2, 3] and (ii) the data is utilized for detecting events (e.g., flooding of highways, collapse of railway bridge, etc.) in real-time in many risk-critical applications (e.g., emergency management). Since big data stream has high volume and velocity, it is not economically viable to store and then process (as done in the batch computing model). Hence, stream processing engines (e.g. Spark, Storm, S4) have evolved in the recent past that have capability to undertake real-time big data processing. Stream processing engines offer two significant advantages: firstly, they circumvent the need to store large volume 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 the big data stream computation as the stream processing engines can now be easily scaled [2, 4, 5] in response to changing volume and velocity.

Though, the stream data processing has been studied in the past several years within the database research community, the focus has been on query processing [13], distribution [14] and data integration. Data security related issues, however, have been largely ignored. Since 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 sensors. Data from these sensors 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 decision making 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 stream in near real-time processing. We refer 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 Attack. In addition, these sensors have limited processing power, storage, and energy; hence, there is a requirement to develop light-weight security verification schemes. Furthermore, data streams need to be processed on-the-fly in a correct sequence. In this paper, we address these issues by designing an efficient approach for online security verification of big data streams.

The most common approach for ensuring data security is to apply the cryptographic methods. In the literature, there are two most common types of cryptographic encryption methods: asymmetric and symmetric key encryption. Asymmetric key encryption methods (e.g., RSA, ElGamal, DSS, YAK, Rabin, etc.) perform a number of exponential operations over a large finite field. Therefore, they are 1000 times slower than the symmetric key cryptography [15, 16]. Hence, efficiency become an issue if asymmetric key such as the Public Key Infrastructure PKI [18] is applied to end-to-end security of big data streams. Thus, the symmetric key encryption is the most efficient cryptographic solution for such applications. However, existing symmetric key methods (e.g., DES, AES, IDEA, RC_4, etc.) fail to meet the requirements of real time security verification. Hence, there is a need to develop an efficient and scalable approach for performing the online security verification of big data stream. 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 approach is based on a common shared key that is generated by exploiting synchronize prime number. 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 the reduced communication overhead, our approach is able to do security verification on-the-fly (with minimum delay) with minimal computational overhead.
- Our proposed approach adopts dynamic key length from the set of 128-bit, 64-bit, and 32-bit. This enables faster security verification at DSM without compromising the security. Hence, our approach is suitable to process high volume of data without any delay.
- We compare our proposed approach with the standard symmetric key solution (AES) in order to evaluate the relative computational efficiency. The results show that our approach 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. Section 3 describes our proposed solution, DLSeF. Section 4 presents the formal security analysis of our approach. Section 5 evaluates the performance and efficiency of the approach through extensive experiments. Section 6 concludes our work and points out to potential future directions.

2 Background and The Problem Definition

Data stream management system has been studied in the past several years with the main focus on query processing, data distribution, and data integrity. The security aspects have been largely overlooked. Nehme et al. [17] initially highlighted the need of security framework in streaming data. They divided the security problem into two: data security problem (also known as data security punctuation) and query security problem (also known as query security punctuation). Data security punctuation deals with the data security, whereas query security punctuation deals with the security and access control during the query processing. They extensively work on access control by focusing on both data security and query security punctuation in their papers [7, 17]. For example, FENCE, a continuous access control framework in dynamic data stream environments, deals with both data and query security restrictions [7]. It gives low overhead which is suitable for data stream environments. Similarly, ASSIST, an application system based effective and efficient access control framework, is proposed to protect streaming data from unauthorized access [8]. They have implemented ASSIST on top of StreamInsight, a commercial stream processing engine. In this paper, we are focusing on the data security punctuation, where our approach is to protect the data efficiently from potential attacks from/on untrusted intermediaries before the data reaches to the DSM.

Figure 1 shows an overall architecture for big data stream process from source sensing devices to the data processing center including our proposed security framework. We refer to [29] for further information on stream data processing in datacenter cloud. In sensor networks, data packets from the source are transmitted to the sink (data collector) through multiple intermediaries hops (e.g. routers and gateways).Collected data at sink node is forward to the DSM as data stream which may also pass through many untrusted intermediaries. The number of hops and intermediaries depend 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, the traditional communication security techniques [9, 12] are not sufficient to provide end-to-end security. In our framework both the queries and data security related techniques are handled by DSM in coordination with the on-field deployed sensors. 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 is stored in the big data storage system supported by cloud infrastructure. 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.

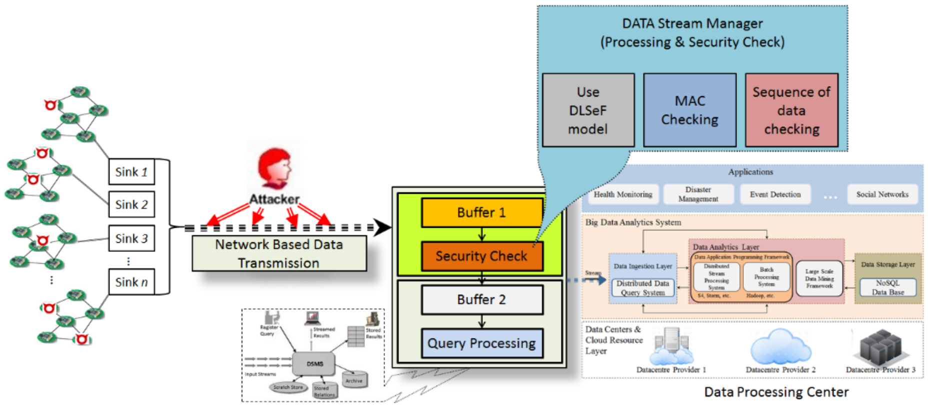


Fig. 1. High level of Architecture from Source Sensing Device to Big Data Processing Center.

The discussion in the above architecture clearly identifies the following most important features for security verification for big data stream processing. In summary, they include: (a) security verification needs to be performed in real time (on-the-fly),

(b) framework has to deal with high volume of data at high velocity, (c) data items should be read once in the prescribed sequence, and (d) original data is not available for comparisons which is widely available in store-and-process batch processing paradigm. These features need to be enabled by the big data stream processing framework in addition to meeting the end-to-end data security requirements.

Based on the above features of big data stream processing, we categorize existing data security methods into two classes: communication security [9, 12] and server side data security [26, 27]. Communication security deals with the data security between two nodes when it is in motion, and does not deals with the end-to-end security, server side data security approaches focus on ensuring the security of data when it is at rest in repository. They are not suitable to use in the big data stream. Furthermore, symmetric cryptographic based security solutions are either static shared key or centralized dynamic key. In static shared key, we need to have a long key to defend from potential attackers. It is well known that length of the key is always proportional to the security verification time and hence longer keys leading elevated computation time are not suitable for applications that need to do real-time processing over high volume, high velocity data. For the dynamic key, centralize processor rekey and distribute keys to all the sources; this is a time consuming process. Moreover, big data stream is always continuous in nature and impossible to put data in halt for rekeying process. To address this problem, we propose a distributed and scalable approach for big data stream security verification.

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

3 Proposed Approach

Our approach is motivated by the concept of moving target defense. The basic idea is that 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 approach, Dynamic Key Length Based Security Framework (DLSeF), provides the robust security by changing both key and key length dynamically. In our approach, if an intruder/attacker eventually hacks the key, 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 approach dynamically changes the both across the sessions. Similar to any secret key based symmetric key cryptography, our DLSeF approach consists of 4 independent components and related processes: system setup, handshaking, rekeying, and security verification. As stream processing is expected to be performed in near real time processing, we assume that data packets should not take more than few hours to reach DSM, as the end-to-end delay is an important QoS parameter to measure the performance of sensor networks [28]. Table 1 provides the notations used in modelling our approach. We next describe the approach.

Table 1. Notations used in our approach

Acronym	Description
S_i	i^{th} Sensor's ID.
K_i	i^{th} sensor's secret key.
K_{si}	i^{th} sensor'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 sensor 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 sensors.
t	Interval time to generate the prime number.
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.
$\oplus$	Bitwise X-OR operation.
$\|$	Concatenation operation.
$DATA$	Fresh data at sensor before encryption.

3.1 DLSeF System setup

We have made a number of realistic and practical assumptions while designing and modelling our approach. First, we assume that DSM has all deployed sensor's identities (IDs) and secret keys because the network is untrusted. Sensors 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 steps. The first three steps are for the sensors and DSM where they authenticate with each other and the next two steps are for the generating shared key. The shared key is utilized during the handshaking process.

Step 1: A sensor (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 DSM: $\boldsymbol{S_i} \rightarrow$ **DSM:** $P1$

Step 2: Upon receiving the message, the DSM decrypts $P1$ (i. e. $D_{K1}(P_1)$) and retrieves the corresponding source ID ($S_i \leftarrow retrieveKey(K_i)$). If the source sensor's ID matches with its own database, then 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 sensor for DSM authentication as follows: $\boldsymbol{S_i} \leftarrow$ **DSM:** P_2

Step 3: Corresponding sensor 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 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 sensor sends an acknowledgement (ACK) to DSM. The ACK is encrypted with the new key, which is computed using hash of the current key ($K3 \leftarrow H(K2)$). The encrypted ACK is denoted as $P_3 \leftarrow E_{K3}(ACK)$, and sends to DSM as follows: $\boldsymbol{S_i} \rightarrow$ **DSM:** P_3

Step 4: The DSM decrypts the ACK ($ACK \leftarrow D_{K3}(P_3)$) to confirm that the sensor is now ready to establish the session. The current secret key is updated using the hash of 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 $\boldsymbol{S_i} \rightarrow$ **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 existing shared key ($K4 \leftarrow H(K3)$). We update the shared key in every transaction to ensure the strength of security for handshaking.

3.2 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 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 the security while avoiding large prime number size. We achieve this by dynamically changing the key size as described next.

The dynamic prime number generation function is defined in Algorithm II. 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 of 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 I. After a certain time interval, the next shared key is generated by applying Algorithm II where the size is determined by Algorithm I as follows:

$Prime(P_i)$ periodically computes the relative prime number at both the sensor and DSM ends after a time interval t, which are updated based on function

$KeyLength(\,)$. The shared secret key (K_{SH}) generation process needs K_d, and P_i. In the handshaking process, DSM transmits all properties required to generate shared key to sensors ($K_d, t,\ P_i,\ Prime\ (\,), KeyLength(\,),\ K_{SH},\ KeyGen$) as follows: $\boldsymbol{S_i} \leftarrow$ **DSM:** $\{K_d, t, P_i,\ Prime\ (\,), KeyLength(\,), K_{SH},\ KeyGen\}$

All of these above transferred information are stored in the trusted part of source for future rekeying process (e.g., TPM) [25]).

3.3 DLSeF Rekeying

Our proposed approach 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 following the steps described in Algorithm I. We change the key periodically in 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 need 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. In some cases, data packet can arrive at DSM after shared key is updated. Such data packets are encrypted using previous shared key. We add a time stamp field to individual data packet to identify the encrypted shared key. If the data is encrypted using previous key then the DSM uses $K_{SH}-$ key for the security verification; otherwise, it follows the normal process.

The above defined *DLSeF Handshaking* process makes sensors aware about the *Prime* (P_i), *KeyLength*, and *KeyGen*. We now describe the complete secure data transmission and verification process using those functions and keys. As mentioned above, our approach uses the synchronized dynamic prime number generation Prime (P_i) on both sides, i.e., sensors and DSM as shown in Fig. 1. 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 II. The prime number generation process (Algorithm II) always calls Algorithm I to fetch shared key length information and associated time interval. Sensors generate the shared key $K_{SH}=(E(P_i,K_d))$ using the prime number P_i, and DSM's secret key $E(P_i,K_d)$. We use the secret key of DSM to improve the robustness of the security verification process. Each data block is associated with the authentication and integration tag and contains two different parts. One is encrypted DATA based on shared key K_{SH} for integrity checking (i.e., $I_D=DATA\bigoplus K_{SH}$), and the other part is for the authenticity checking (i.e., $A_D=S_i\bigoplus K_{SH}$). The resulting data block $((DATA\bigoplus K_{SH})\ \|\ (S_i\bigoplus K_{SH}))$ is sent to DSM as follows: $\boldsymbol{S_i} \rightarrow$ **DSM:** $\{(I_D\|A_D)\}$.

3.4 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. 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 the data integrity.

4 Security Analysis of DLSeF

In this section, we provide a theoretical analysis on our approach. We made the following assumptions: (a) any participant in our scheme cannot decrypt the data that was encrypted by 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) 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 the intruders.

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

Definition 1 (attack on authentication). A malicious attacker M_a can attack on the authenticity if it is capable to monitor, intercept, and introduce 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 on the integrity of the data if it is an adversary capable to monitor the data stream regularly and try to access and modify the data blocks before it reaches to DSM.

Theorem 1: The data security of data streams 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 source and DSM. The dynamic shared key length is either 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 [16]. Even smaller symmetric key provides more security as it is never shared publicly. 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 [16]. 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 128 bit symmetric key takes 3136e +19 nanoseconds (more than a month), 64 bit symmetric key takes 3136e +19 nanoseconds (more than a week), and so on. We fixed the time interval (t) to generate prime number 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 I). Dynamic shared key is computed based on the calculated prime number and associated properties initialized accordingly (See

Algorithm II). Based on these calculation, we conclude that an attacker cannot intercept within the interval time *t*. The key has been already changed four times before an attacker knows the key and this knowledge is not known to the attackers. Data blocks arrived after 20 hours are discarded as they might be compromised.

ALGORITHM I. 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
18: Return ($x_n$) // use to initialize $x_{n-1}$for next iteration.
```

Table 2. 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 is calculated in Algorithm II 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 [24]. In our approach, 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 intianized by DSM at DLSeF Handshaking process and the interval time is *t* (see Algorithm I).

By applying the *Algorithm II*, 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 genertion is based on the value of *m, wherem* 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 seniors and DSM. In Algorithm II, 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.

ALGORITHM II. 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 $\mathrm{S}(m) = 1$ then
8:        GO TO: 13
9:  Set $m := 6k + 5$
10: If $\mathrm{S}(m) = 1$ then
11:       GO TO: 13
12: $k := \lfloor k^3 + \sqrt{k} \rfloor \, mod \; 17 + k$
13: GO TO: 3
14: $P_i = m$
15: Return ($P_i$) // calculated new prime number
```

$$\mathrm{S}_1(x) = \frac{(-1)}{\left\lfloor \frac{\lfloor\sqrt{x}\rfloor}{6} \right\rfloor + 1} \sum_{k=1}^{\left\lfloor \frac{\lfloor\sqrt{x}\rfloor}{6} \right\rfloor + 1} \left\lfloor \left\lfloor \frac{x}{6k+1} \right\rfloor - \frac{x}{6k+1} \right\rfloor, \ \mathrm{S}_2(x) = \frac{(-1)}{\left\lfloor \frac{\lfloor\sqrt{x}\rfloor}{6} \right\rfloor + 1} \sum_{k=1}^{\left\lfloor \frac{\lfloor\sqrt{x}\rfloor}{6} \right\rfloor + 1} \left\lfloor \left\lfloor \frac{x}{6k-1} \right\rfloor - \frac{x}{6k-1} \right\rfloor$$

$$S(x) = \frac{\mathrm{S}_1(\mathrm{x}) + \mathrm{S}_2(\mathrm{x})}{2}$$

If $S(x) = 1$ then *x* is prime, otherwise *x* is not a prime.

$x \not\equiv 0 \; mod \; i \; \forall \; 1 \leq \mathrm{i} \leq \mathrm{x} - 1,$ if *x* is prime.

Put the value of *x* as a prime number, then derivations as follows:

$$\Rightarrow \left\lfloor \left\lfloor \frac{\mathrm{x}}{6\mathrm{k}+1} \right\rfloor - \frac{\mathrm{x}}{6\mathrm{k}+1} \right\rfloor = -1$$

Same as $\left\lfloor \left\lfloor \frac{\mathrm{x}}{6\mathrm{k}-1} \right\rfloor - \frac{\mathrm{x}}{6\mathrm{k}-1} \right\rfloor = -1$

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

$$\mathrm{S}_1(x) = \frac{(-1)}{\left\lfloor \frac{\lfloor\sqrt{x}\rfloor}{6} \right\rfloor + 1} \sum_{k=1}^{\left\lfloor \frac{\lfloor\sqrt{x}\rfloor}{6} \right\rfloor + 1} (-1) = 1$$

Same $\mathrm{S}_2(x)$ is also 1 and then $S(x) = \frac{\mathrm{S}_1(\mathrm{x}) + \mathrm{S}_2(\mathrm{x})}{2} = 1$

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 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 sensor, but M_a can introduce him/ herself as an authenticated node to send its information. In our approach, a sensor (S_i) sends$\big((I_D) \parallel (A_D)\big)$, where the second part of the data block $(S_i \oplus K_{SH})$ is used for authentication check. DSM decrypts this part of the data block for authentication check. 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 on big data stream.

Theorem 4: An *attacker* M_i cannot read the shared key K_{SH} within the time interval t in 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 approach calculates P_i randomly after time t and then uses the value P_i to generate K_{SH} as described in *Theorem 1 and 2.*

5 Experimental Evaluation

The proposed DLSeF security approach though deployed in 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 the adverse conditions, we experimented with two different approaches in two different simulation environments. We first verify the security approach using Scyther [22], and then measure the efficiency of the approach using JCE (Java Cryptographic Environment) [23].

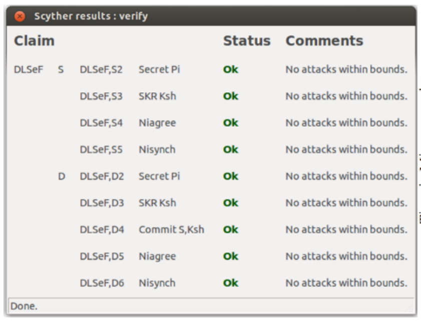
5.1 Security Verification

The protocols in our proposed approach is written in Scyther simulation environment using Security Protocol Description Language (.spdl). According to the features of Scyther, we define the role 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 are 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 two types of attacks by adversaries. In the first type of attacks, a malicious attacker change the data while it is being transmitted from S to D through intermediaries (integrity attack). In the second type of attacks (authentication attack), an adversary acquires the property of S and sends the data packets to D pretending that it is from S. We experimented with 100 runs for each claim and found out no attacks at D as shown in Fig. 2(a).

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 approach 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 protocols verification too,l to verify our proposed model.

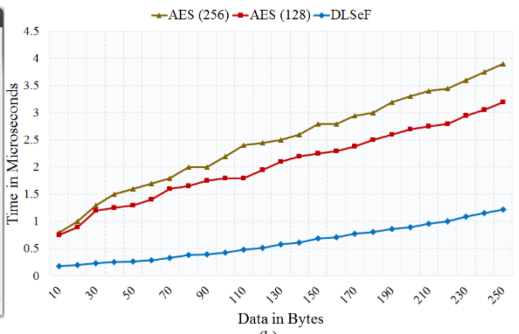
Results: We did our simulation using a different number of data blocks in each run. Our experiment ranged from 10 to 100 instances with 10 intervals. We checked authentication for each data block, whereas the integrity check is performed on the selected data blocks. As the key generation process is saved in the trusted part of the sensors, no one can get access to those information except the corresponding sensor. Hence, we did not find any authentication attacks. For integrity attacks, it is hard to get shared key (K_{SH}), as we are frequently changing 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. 2(a) shows the result of security verification experiments in Scyther environment. This shows that our approach is secured from integrity and authentication attacks.


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.

(a)



(b)

Fig. 2. (a) Scyther simulation result of successful security verification at DSM. (b) Performance of our approach compared in efficiency to 128 bit AES and 256 bit AES.

5.2 Performance Comparison

Experiment model: It is clear that the actual efficiency improvement brought by our approach highly depends on the size of the key and rekeying without further communication between sensor and DSM. We have performed experiments with different size of data blocks. The results of our experiments are given below.

We compare the performance of our proposed model DLSeF with advanced encryption standard (AES), the standard symmetric key encryption algorithm [20, 21]. Our approach is efficient compared with two standard symmetric key algorithm such as 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 size. 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 method. We experimented with many-to-one communication. All sensors node communicate to the single node (DSM). All sensors have the similar properties whereas the destination node is more powerful to initialize the process (DSM). The rekey process is executed at all the nodes without any intercommunication. Processing time of data verification is measured at DSM node. Our experimental results are shown in Fig. 2(b).

Results: The performance of our approach is better than the standard AES algorithm when different sizes of the data blocks are considered. Fig. 2(b) shows the processing time of the DLSeF approach in comparison with base 128-bit AES, and 256-bit AES for different size of the data blocks. The performance comparison shows that our proposed approach is efficient and faster than the baseline AES protocols.

From the above two experiments, we conclude that our proposed DLSeF approach is secured (from both authenticity and integrity attacks), and efficient (compare to standard symmetric algorithms such as 128-bit AES and 256-bit AES).

6 Conclusion and Future Works

In this paper, we have proposed a novel authenticated key exchange approach, namely Dynamic Key Length Based Security Framework (DLSeF), which aims to provide real-time security verification approach for big sensing data stream. Our approach has been designed based on the symmetric key cryptography and dynamic key length. By theoretical analyses and experimental evaluations, we showed that our DLSeF approach has provided significant improvement in the security processing time, and prevented malicious attacks on authenticity and integrity. In our approach, we decrease the communication and computation overhead by performing dynamic key initialization at both sensor and DSM, which in effect eliminates the need of re-keying and decreases the communication overhead. We plan to pursue a number of research avenues in future. The foremost is to perform a comparative study of our work with other techniques like RC_4, RC_6. We will further investigate the technique to develop a moving target defense strategy for the Internet of Things.

References

1. Stonebraker, M., Çetintemel, U., Zdonik, S.: The 8 requirements of real-time stream processing. ACM SIGMOD Record, 34(4), 42-47 (2005).
2. Bifet, A.: Mining Big Data in Real Time. Informatica (Slovenia), 37(1), 15-20 (2013).
3. Dayarathna, M., Suzumura, T.: Automatic optimization of stream programs via source program operator graph transformations. Distributed and Parallel Databases, 31(4), 543-599 (2013).
4. Demirkan, H., Delen, D.: Leveraging the capabilities of service-oriented decision support systems: Putting analytics and big data in cloud. Decision Support System 55(1), 412-421 (2013).
5. Tien, J. M.: Big data: Unleashing information. Journal of Systems Science and Systems Engineering, 22(2), 127-151 (2013).
6. Manyika, J., Chui, M., Brown, B., Bughin, J., Dobbs, R., Roxburgh, C., Byers, A.: Big data: The next frontier for innovation, competition, and productivity. (2011).
7. Nehme, R. V., Lim, H. S., Bertino, E.: FENCE: Continuous access control enforcement in dynamic data stream environments. In Proceedings of the third ACM conference on Data and application security and privacy, 243-254 (2013).

8. Cao, J., Kister, T., Xiang, S., Malhotra, B., Tan, W. J., Tan, K. L., Bressan, S.: Assist: Access controlled ship identification streams. InTransactions on Large-Scale Data-and Knowledge-Centered Systems XI, 1-25 Springer (2013).
9. Puthal, D.: Secure Data Collection and Critical Data Transmission Technique in Mobile Sink Wireless Sensor Networks. M.Tech Thesis, National Institute of Technology, Rourkela, 2012.
10. Bahrami, M., Singhal, M.: The Role of Cloud Computing Architecture in Big Data. In Information Granularity, Big Data, and Computational Intelligence, 275-295 (2015).
11. McAfee, A., Brynjolfsson, E.: Big data: the management revolution. Harvard business review, (90), 59-69 (2012).
12. Puthal, D., Sahoo, B.: Secure Data Collection & Critical Data Transmission in Mobile Sink WSN: Secure and Energy efficient data collection technique. LAP Lambert Academic Publishing: Germany, 2012.
13. Deshpande, A., Ives, Z., Raman, V.: Adaptive query processing. Foundations and Trends in Databases, 1(1), 1-140 (2007).
14. Sutherland, T. M., Liu, B., Jbantova, M., Rundensteiner, E. A.: D-cape: distributed and self-tuned continuous query processing. In Proceedings of the 14th ACM international conference on Information and knowledge management, 217-218 ACM (2005).
15. Burke, J., McDonald, J., Austin, T.: Architectural support for fast symmetric-key cryptography. ACM SIGOPS Operating Systems Review, 34(5), 178-189 (2000).
16. www.cloudflare.com (accessed on: 04.08.2014)
17. Nehme, R. V., Lim, H. S., Bertino, E., Rundensteiner, E. A.: StreamShield: a stream-centric approach towards security and privacy in data stream environments. In Proceedings of the ACM SIGMOD International Conf. on Management of data, 1027-1030 ACM (2009).
18. Park, K. W., Lim, S. S., Park, K. H.: Computationally efficient PKI-based single sign-on protocol, PKASSO for mobile devices. IEEE Trans. on Computers, 57(6), 821-834 (2008).
19. Stamp, J., Campbell, P., DePoy, J., Dillinger, J., Young, W.: Sustainable security for infrastructure SCADA. Sandia National Laboratories, Albuquerque, New Mexico (www. sandia. gov/scada/documents/SustainableSec urity. pdf) (2003).
20. Pub, N. F. 197: Advanced encryption standard (AES). Federal Information Processing Standards Publication, 197, 441-0311 (2001).
21. Heron, S.: Advanced Encryption Standard (AES). Network Security, 2009(12), 8-12 (2009).
22. Scyther, [Online] http://www.cs.ox.ac.uk/people/cas.cremers/scyther/
23. Pistoia, M., Nagaratnam, N., Koved, L., Nadalin, A.: Enterprise Java 2 Security: Building Secure and Robust J2EE Applications. Addison Wesley Longman Publishing, Inc. (2004).
24. Kaddoura, I., Abdul-Nabi, S.: On formula to compute primes and the nth prime. Applied Mathematical science, 6(76), 3751-3757 (2012).
25. Nepal, S., Zic, J., Liu, D., Jang, J. A mobile and portable trusted computing platform. EURASIP Jour. on Wireless Communication and Networking, 2011(1), 1-19 (2011).
26. Zissis, D., Lekkas, D.: Addressing cloud computing security issues. Future Generation computer systems, 28(3), 583-592 (2012).
27. Liu, C., Zhang, X., Yang, C., Chen, J.: CCBKE—Session key negotiation for fast and secure scheduling of scientific applications in cloud computing. Future Generation Computer Systems, 29(5), 1300-1308 (2013).
28. Akkaya, K., Younis, M.: An energy-aware QoS routing protocol for wireless sensor networks. In Distributed Computing Systems, 23rd International Conf. on, 710-715. (2003).
29. Ranjan, R.: Streaming Big Data Processing in Datacenter Clouds. IEEE Cloud Computing, 1(1), 78–83 (2014).