DPBSV- An Efficient and Secure Scheme for Big Sensing Data Stream

Deepak Puthal[*], Surya Nepal[†], Rajiv Ranjan[†], and Jinjun Chen[*]
[*]Faculty of Engineering and Information Technology, University of Technology, Sydney (UTS), Australia
[†]Digital Productivity Flagship, Commonwealth Scientific and Industrial Research Organization (CSIRO), Australia
E-mail: {deepak.puthal, jinjun.chen}@gmail.com, {Surya.Nepal, Rajiv.Ranjan}@csiro.au

Abstract— **Stream processing has become an important paradigm for the massive real-time processing of continuous data flows in large scale sensor networks. While dealing with big data stream in sensor networks, Stream Processing Engines (SPEs) must always verify the authenticity, and integrity of the data as the medium of communication is untrusted, where malicious attackers can access and modify the data. Existing technologies for data security verification are not suitable for data streaming applications, as the verification in real time introduces significant overheads. In this paper, we propose a Dynamic Prime Number Based Security Verification (DPBSV) scheme for big data stream processing. Our scheme is based on common shared key that is updated dynamically by generating synchronized pair of prime numbers. Theoretical analyses and experimental results of our DPBSV scheme show that it can significantly improve the efficiency as compared to existing approaches by reducing the security verification overhead. Our approach not only reduces the verification time, but also strengthens the security of the data by constantly updating the shared keys.**

Keywords—Security; Sensor Networks; Big Data Stream; Key Exchange; Cloud Computing.

I. Introduction

A number of application scenarios (e.g., telecommunications, network security, large-scale sensor networks, etc.) require real-time processing of data streams, where the applicability of the traditional method “store-and-process” is limited [20]. There are a wide range of applications that require cloud-based data stream processing (e.g., data from large scale sensors, information monitoring, web exploring, data from social networks such as Twitter and Facebook, surveillance data analysis, financial data analysis, etc.) [12]. These applications produce high speed, real time, and large volume data as input, and hence require a novel paradigm for data processing. As a result, a new computing paradigm based on Stream Processing Engines (SPEs) has emerged. SPEs deal with the specific types of challenges and are intended to process data streams with a minimal delay [16, 17].

Some applications such as network monitoring and fraud detection are producing data, which is beyond the capability of traditional data processing infrastructures. These applications require real-time processing of very high-volume data streams (also known as *big data stream*). The complexity of big data is defined through V^4’s: 1) *volume* –referring to terabytes, petabytes, or even exabytes (1000^6 bytes) of stored data, 2) *variety* – referring to the unstructured, semi-structured and structured data from various sources like social media (e.g. Twitter, Facebook etc.), sensors, surveillances, image or video, medical records etc., 3) *velocity* – referring to the high speed at which the data is in/out, and 4) *veracity* – referring to the quality of data. These features present significant opportunities and challenges for big data stream processing [12]. Big data stream is continuous in nature and it is important to perform the real-time analysis as the life time of the data is often very short (applications can access the data only once) [22, 25]. As the volume and velocity of the data is large, storing huge volume of data for later analysis is considered impractical; hence, the traditional store-and-process batch computing model is not suitable.

Though processing big data stream has emerged as one of the important topics of research, the secure data stream processing has received little attention from researchers. Some of these data streams arise from the mission critical applications (e.g., environmental monitoring, military application, etc.), where data streams need to be secured [4]. The problem is further exacerbated when thousands to millions of small sensors simultaneously produce data streams for real-time analytics. The hard question is can we efficiently undertake secure processing of thousands of data streams while meeting mission critical data processing constraints (e.g., minimizing data processing overheads). In addition, compared to the conventional store-and-process, these sensors have limited processing power, storage, bandwidth, and energy [5]. Furthermore, data streams need to be processed on-the-fly in a prescribed sequence.

One of the security threats is the man-in-the-middle attack, in which a malicious attacker can access or modify the data stream from sensors. This situation arises as it is not possible to monitor a large number of sensors deployed in the untrusted environment [5]. The common approach is to apply cryptographic model for securing the data streams. Keeping data encrypted is the most common and safe choice to secure data in transmission subject to safeguarding of encryption keys. There are two prominent cryptographic encryption algorithms available: asymmetric and symmetric. Asymmetric-key encryption algorithms (e.g., RSA, ElGamal, DSS, YAK, Rabin, etc.) perform a number of exponential operations over a large finite field. Therefore, they are approximately 1000 times slower than symmetric key cryptography [7, 8]. Efficiency can become a serious issue if asymmetric-key cryptology based infrastructure such as the Public-Key Infrastructure PKI [29] is applied to big data stream. Thus, symmetric-key encryption is the most efficient cryptographic solution for such applications. However, symmetric-key algorithms (e.g., DES, AES, IDEA, RC_4, etc.) do not scale when subjected to the real-time, on-the-fly processing of big data streams.

In this paper, we present the design and development of a Dynamic Prime-Number Based Security Verification (DPBSV) scheme. Our scheme is based on the notion of common shared key that is dynamically and periodically updated by generating synchronized prime numbers. The synchronized prime number enables to reduce the communication overhead without compromising the security. Our scheme is suitable for big data streams as it verifies the security (confidentiality and integrity) on-the-fly (with minimum delay), hence leading to reduced communication overhead. The scheme uses much smaller key length (64-bit) as against symmetric cryptographic algorithms. This enables faster security verification processing of streams at DSM (Data Stream Manager). The same level of security is maintained by updating the shared keys dynamically. Dynamic key generation is based on the random prime numbers, which is initialized and synchronized at sensors and DSM. We save on network overhead as our scheme do not require DSM and sensor node to communicate after the initial handshaking key step.

Our proposed scheme is efficient in comparison to AES, as it reduces the computational load and execution time significantly. The main contributions of the paper can be summarized as follows:

- We present a secure big data stream processing scheme.
- We design and develop an efficient Dynamic Prime-Number Based Security Verification (DPBSV) scheme for big data streams.
- We evaluate DPBSV scheme both theoretically and empirically. Our analysis show that it is efficient when applied to big data streams in comparison to standard AES.

The rest of this paper is organized as follows. Section II provides the background on big sensor data stream and corresponding security related work. Section III provides a motivating example in big data stream as well as detailed analysis to our research problem. Section IV describes DPBSV key exchange scheme. Section V presents the security analysis of scheme formally. Section VI evaluates the performance and efficiency of scheme through experimental results. Section VII concludes the paper and points out future work.

II. Proposed Secure Data Stream Architecture

A. *Big Data Stream*

Data stream processing is an emerging computing paradigm which is particularly suitable for application scenarios where huge amount of data (Big Data) must be processed in real-time (with small delay). Unlike traditional database systems where query processing is done over archived (i.e. the data needs to be stored based on a pre-defined schema prior to processing) data, SPE processes real-time time streaming data on-the-fly. The need for on-the-fly processing arises from the high-volume and high velocity input data that cannot be persisted for later analysis due practical reasons (e.g., data storage overhead). DSM handles streams of tuples similar to how a conventional database system handles relations. In addition, DSM undertakes the security verification of the data blocks on-the-fly.

Cloud computing has become platform of choice for processing big data due to its on-demand elasticity extremely low-latency and massively parallel processing architecture [18]. It supports the most efficient way to obtain actionable information from big data stream [21-24]. Figure 1 shows our cloud based architecture for big data stream processing systems consisting of data sources, the cloud data centers, and the DSM framework. We refer to [10] for further information on stream data processing in datacenter cloud. All the query and security related data processing tasks are handled by the DSM components. It is important to note that the security verification of streaming data has to be performed in real-time with a fixed buffer size before the actual stream query processing step. At last, the processed data is stored in the cloud storage. Queries registered in DSM are defined as “continuous” since they are continuously applied to the *streaming* data flows. Results are sent to the user each time the streaming data satisfies the query predicate. The queries (including security verification operation) are defined as a directed acyclic graph where each node is an operator and edges define data flow.

It is clear from the above description that security verification is one of the critical requirements for big data stream processing. We not that security verification step as proposed in our DSM framework adds to overall stream processing time. Hence, the major challenge for DSM is to reduce this additional security verification overhead. This is critical for big data stream due to the high volume and velocity. Hence, in our DSM approach security verification is done on-the-fly (with minimal overhead).

B. *Symmetric key cryptography based security verification methodology*

The Data Encryption Standard (DES) has been a standard symmetric key algorithm since 1977. However, it was cracked rather easily. In 2000, the Advanced Encryption Standard (AES) [1] replaced the DES to meet the ever-increasing requirements of data security. The Advanced Encryption Standard (AES), also known as the Rijndael algorithm, is a symmetric block cipher that can encrypt data blocks of 128 bits using symmetric keys of 128, 192 or 256 bits [1, 2]. AES was introduced to replace the Triple DES (3DES) algorithm. AES was acquainted with supplant the Triple DES (3DES) algorithm utilized for a decent measure of time all around. Hence, we have compared our proposed solution against AES.

Symmetric keys are smaller in size than asymmetric keys, so they have less computational burden. The 128-bit symmetric key provides the same strength of protection as a 3,248-bit asymmetric key [8]. Since the aim is to perform the security verification on-the-fly (real-time), the symmetric key cryptography becomes a natural choice due to its scalability. It is noted in the literature that symmetric key cryptography is approximately 1000 times faster than strong public key ciphers [7]. However, it is comparatively easy to read/modify the symmetric key cryptography as it has small key size [7]. To circumvent this problem, we periodically apply a synchronized dynamic prime number (P_i) generation algorithm at both source and DSM. This algorithm leads to confusion for the malicious attackers. The procedure *Prime*(P_i) is calculated and synchronized on both source and DSM ends. This intelligent modification makes the process overall security verification process faster and prevents potential attacks. We explain this algorithm in-detail later in this paper.

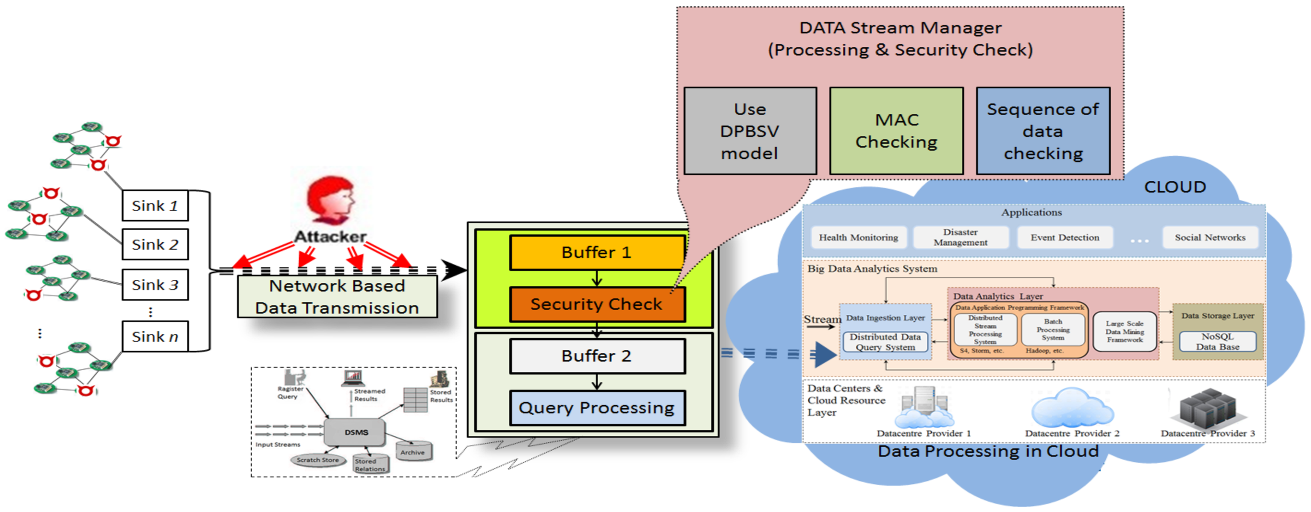


Figure 1. Overlay of our architecture from source sensing devices to cloud data processing center.

We assume that deployed source nodes operate in two modes: trusted mode, and untrusted mode. In the trusted mode, the nodes operate in a cryptographically secure space and adversaries cannot penetrate this space. Nodes can incorporate Trusted Platform Module (TPM) to design trusted mode of operation. The TPM is a dedicated security chip following the Trust Computing standard specification for cryptographic microcontroller system [11]. TPM provides a cost effective way of "hardening" of many of recently deployed applications, those are previously based on software encryption algorithms with keys kept on a host's disk [11]. It provides a hardware based trust, which contains cryptographic functionality such as key generation, store, and manage embedded in the chip. The detailed architecture can be found in [11]. We assume that the proposed *Prime (Pi)* and *secret key calculation on source nodes are conducted* in the trusted mode.

III. Motivation and Problem Analysis

The above discussion on the DSM framework architecture above clearly outlines the following most important requirements as regards to the secure processing of the big data stream:

1. Security verification needs to be performed in real time (on-the-fly).
2. Verification framework has to deal with high volume and high velocity data.
3. Data items can be read once in the prescribed sequence.
4. Unlike store-and-process paradigm, original data is not available for comparisons in context of stream processing paradigm.

Based on the above features of big data stream, we have categorized existing security methods into two classes: Communication Security [9, 19] and Server Side data security [6, 26, 27]. Communication security deals with handling security data in motion. On the other hand Server Side is concerned with securing the data security when it is at rest. The security threats and solutions proposed in the literature are not suitable for secure processing of the big data streams due to the following reasons.

The communication security techniques are mainly proposed for networks communication. The network communication related attacks are broadly divided into two types: external and internal. To avoid such attacks, security solutions have been proposed for each individual TCP/IP layers. For the physical layer, proposed solutions include spread spectrum communication, jamming reports, accurate and complete design of the node physical package, etc.; for the data link layer, proposed solutions include error correcting codes, collision detection and avoidance techniques, rate limiting etc.; for the network layer, proposed solutions include link layer encryption and authentication, multipath routing, identity verification and authenticated broadcast, etc.; and for the transport layer, proposed solutions include packet authentication [9, 19]. These solutions can avoid the communication threats but not suitable for dealing with new challenges posed by high volume and high velocity big data.

The server side data security is mainly proposed for physical data center, when data is at rest and accessed through applications. There are several potential attacks for such data such as data interruption, interception, privacy breach, impersonation, session hijacking, programming flaws, software modification, software interruption, defacement, disrupting communications, hardware interruption, and hardware modification, etc. Several solutions have been proposed to protect data and cloud servers from such attacks such as privacy in multitenant environment, data protection from disclosure, access control, software security, service availability, access control, application security, data security (e.g. data in transit, data at rest, reminisce etc.), cloud management control security, virtual cloud protection, hardware security, and hardware reliability etc. [6, 13, 26, 27]. However, these proposed solutions are tailored towards *store-and-process* paradigm, hence not feasible for on-the-fly big data stream processing.

Existing symmetric cryptographic based security solutions for data security are based on either static shared key or centralized dynamic key. In static shared key, we need to have a long key to defend from a potential attacker. Length of the key is always proportional to security verification time. Based on the requirement of big data stream processing (specified above), it is clear that security verification should be done in real-time. For the dynamic key management solution, centralized rekeying processing and distribution of keys to all the sources is a time consuming process. Big data stream is always continuous in nature and often huge in volume. This makes it impossible to pause the data movement while the rekeying, distribution, and synchronization processes finish. To address this problem, we are proposing a scheme for big data stream security verification without the need of rekeying. The benefits include reduction of the communication overhead and increase in the efficiency of security verification process at DSM.

Our proposed scheme is as follows: we use a common shared key for both sensors and DSM. The key is updated dynamically by generating synchronized prime numbers without the need of having communication between them. This reduces the communication overhead, as required by Rekeying process of existing methods, without compromising the security. Due to the reduced communication overhead, our scheme performs the security verification with minimum delay. The communication is required at the beginning for the initial key establishment and synchronization because DSM sends all the keys and key generation properties to the sources in this step. There is no further communication between the source sensor and DSM after handshaking, which increases the efficiency of the solution. Based on the shared key properties, individual source updates their dynamic key independently.

IV. Dynamic Prime-Number Based Security Verification– DPBSV

This section describes the DPBSV scheme. Similar to any secret key based symmetric key cryptography, DPBSV scheme consists of 4 independent components: system setup, handshaking, rekeying, and security verification. Table 1 provides the notations used in describing scheme. We next describe the scheme in details.

Table 1. Notations

Acronym	Description
S_i	ith Sensor's ID.
K_i	ith Sensor's Secret key.
K_{si}	ith Sensor's Session Key.
K_{enc}	Generated Key for the Authentication.
K_{SH}	Secret shared key of Sensors and DSM
K/K'	Encrypted with Sensor's Secret key for User Authentication.
$C/C'/C''$	Calculated hash Value.
r	Random Number Generated by 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.
$Prime(P_i)$	Prime number generation Function.
$KeyGen$	Key Generation procedure.
$\oplus$	Bitwise X-OR operation.
$\parallel$	Concatenation Operation.
$DATA$	Fresh Data at Sensor before Encryption.

A. *DPBSV System setup*

We have made a number of realistic and practical assumptions while defining our scheme. First, we assume that DSM has all deployed sensor's identities (IDs) and secret keys because the network is fully untrusted. We allow increased number of key exchanges between the sensors and DSM for the initial session key establishment process to achieve stronger security. Our aim is to make this session more secure because we transmit all the secret information of key generation to sensors. Second, we assume that each sensor node S_i knows the identity of its DSM.

Step 1: A sensor (S_i) generates a random number r and sends it to the DSM with its identity as $\{S_i, r\}$. There are n numbers of sensors deployed and those are $S_1, S_2, S_3, ..., S_n$ and S_i is the id of ith sensor. In our scheme, sensors do not communicate between each other to reduce the communication overhead. Proposed scheme also updates the dynamic shared key on both ends to prevent potential attacks from traffic behavior analysis.

1. $S_i \rightarrow$ DSM: $\{S_i, r\}$.

Step 2: Once the DSM receives the request from a sensor, it retrieves the corresponding sensor's secret key, i.e., $K_i \leftarrow retrieveKey(S_i)$ first and then DSM selects a random session key K_{si} i.e. $K_{si} \leftarrow radomdKey(\,)$. In order to share this session key with corresponding sensor (S_i), DSM generates a key based on a selected session key and corresponding sensor's private key $K_{enc} = K_{si} \oplus K_i$. Following the generated key (K_{enc}) DSM encrypts the generated key with the session key $K = E_k(K_{si}, K_{enc})$ and it performs the hash function $C = H(K_{enc} \parallel K \parallel r)$. Finally, DSM sends the value of C and K_{enc} to S_i.

$K_{enc} = K_{si} \oplus K_i$, from random selected session key K_{si}.

$K = E_k(K_{si}, K_{enc})$

$C = H(K_{enc} \parallel K \parallel r)$

2. $S_i \leftarrow$ DSM: $\{C, K_{enc}\}$

Step 3: Corresponding sensor gets its session key K_{enc} based on its own secret key $K_{si} = K_{enc} \oplus K_i$ and finds out the value of K' based on the value of K_{si} and K_{enc}, *i.e.* $K' = E_k(K_{si}, K_{enc})$. Next it computes the hash $H(K_{enc} \parallel K' \parallel r)$ and checks whether or not it is equal to C. If the hashes are equal and $K = K'$, S_i can authenticate DSM. However, if it is not equal, then S_i ends the protocol. Following the authentication, it transmits $C' = H(1 \parallel K_{enc} \parallel K' \parallel r)$ to DSM as follows.

$K_{si} = K_{enc} \oplus K_i$, to extract the session key for won.

$K' = E_k(K_{si}, K_{enc})$

$C' = H(1 \parallel K_{enc} \parallel K' \parallel r)$

3. $S_i \rightarrow$ DSM: $\{C'\}$.

Step 4: After receiving C', DSM compares it with $H(1 \parallel K_{enc} \parallel K \parallel r)$ to check whether or not they are equal. If they are equal, DSM authenticates S_i. Otherwise, the protocol is terminated. After authentication by DSM and sensor, DSM and S can share the session key K_{si} and $C'' = H(2 \parallel K_{enc} \parallel K \parallel r)$.

$C'' = H(2 \parallel K_{enc} \parallel K \parallel r)$

4. $S_i \leftarrow$ DSM: $\{C''\}$

B. *DPBSV Handshaking*

DSM sends its all properties to sensors $\{S_1, S_2, S_2, ..., S_n\}$ based on their individual session key. Generally, the larger the prime number of secret shares used in the pairwise key establishment process, the better security will the pairwise key achieve. However, using a larger prime number for the secret shares requires a greater computation time. In order to make the security verification lighter and faster, we reduce the prime number size. The dynamic prime number generation function is defined in Theorem 2 (described later in this paper). We calculate the prime number on both sensor and DSM sides to reduce communication overhead and minimize the chances of disclosing the shared key.

Step 5: $Prime(P_i)$ computes the relative prime number on both sides with a time interval *t.* In the handshaking process, it transmits all its procedures to generate the key and prime number such as $(K_d, t, P_i, Prime(P_i), K_{SH}, KeyGen)$.

5. $S_i \leftarrow$ DSM: $\{K_d, t, P_i, Prime(P_i), K_{SH}, KeyGen\}$

In this step, DSM sends all the parameters and properties of *KeyGen* to source sensors. The transferred information stored in trusted part of sensor (e.g., TPM).

C. *DPBSV Rekeying*

We propose a novel rekeying mechanism that calculates prime numbers dynamically on both source sensors and DSM independently. In the proposed scheme, small size of the key leads to faster security verification. However, a small key size can be relatively easy to crack. To counter this issue, the key pair is periodically updated. In the event of key compromise at sensors, DSM undertakes key resynchronization process with the sensor as described next. The source sensor executes the step 3 to reinitialize and resynchronize key pair with the DSM. We assume that the secret key information is managed by the senor in a trusted fashion such as by employing the TPM hardware.

In the following, we are presenting an alternative approach to rekeying and the corresponding analysis in terms of efficiency.

Step 6: The above defined *DPBSV Handshaking* process relays information related to the *Prime* (P_i) and *KeyGen* to the sensors. We next describe the secure data transmission and verification process based on above functions and keys. As mentioned above, the proposed scheme applies the synchronized dynamic prime number generation Prime (P_i) on both sides, i.e., sensors and DSM. 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 cycle of prime generation process is based on the value of the prime number and the specified time interval. Sensors generate the shared key $K_{SH} = H(E(P_i, K_d))$ using the prime number P_i and DSM secret key K_d. Each data block is associated with the authentication tag and contains two different parts. First is the encrypted DATA based on its secret key K_i and shared key K_{SH} for integrity checking (i.e., $I_D = DATA \oplus K_{SH} \oplus K_i$), and the second part is concerned with the authenticity checking (i.e., $A_D = S_i \oplus K_{SH}$). The resulting data block is: $((DATA \oplus K_{SH} \oplus K_i) \parallel (S_i \oplus K_{SH}))$.

$I_D = DATA \oplus K_{SH} \oplus K_i$

$A_D = S_i \oplus K_{SH}$

6. $S_i \rightarrow$ DSM: $\{ H(I_D \parallel A_D)\}$.

D. *DPBSV Security Verification*

According to the features of big data stream, security verification should be performed in real time (with minimal delay). The following step shows the verification model. Next step explains how the DSM verifies the authenticity and integrity of each or selected data block.

Step 7: The DSM verifies whether the data was modified while in transit and it was sent by an authenticated sensor node. The DSM first checks the authenticity and integrity of specific data block A_D. The approach selects next block to be checked for authenticity and integrity based on specified random interval such as I_D (configurable variable). This random variable is calculated based on the corresponding prime number *i.e.* $j = P_i \% 7$. The calculated values vary from *0* to *6*, i.e., the maximum interval of *6* blocks and if the value of *j* is *0*, then it will verify every data block. 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 for the integrity check according to the value of j. Given K_i, the DSM decrypts data and checks MAC for integrity check $DATA = I_D \oplus K_{SH} \oplus K_i$.

$S_i = A_D \oplus K_{SH}$

$DATA = I_D \oplus K_{SH} \oplus K_i$

V. Security Analysis of DPBSV

In this section, we provide theoretical analysis of the proposed scheme and prove that it can ensure both authenticity and integrity of streaming data.

A. *Security Proof*

Assumption 1**:** No one can decrypt data that was encrypted by a symmetric-key algorithm, unless it has the session/shared key which was used to encrypt the data by the sensor.

Assumption 2**:** DSM is deployed on a trusted server.

Assumption 3**:** Sensor's secret key, *Prime (Pi)* and *secret key calculation* procedures are deployed on the trusted hardware such as TPM, hence they are safe from intruders. Similar to most cryptological analysis of public-key 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 is an adversary who is capable of monitoring, intercepting, and introducing itself as an authenticated source node which can send data streams to the DSM.

Definition 2 (attack on integrity): A malicious attacker M_i is an adversary capable of monitoring the data stream and is able to modify the stream while it is in transit.

Theorem 1: The security is not compromised by reducing the size of shared secret key (K_{SH}).

Proof: We reduce the size of the prime number to make the key generation process faster and efficient. The ECRYPT II recommendations on key length say that 128-bit symmetric key provides the level of protection as a 3,248-bit asymmetric key. Smaller keys can also provide desired security levels as long as it is not shared publically. Advanced processor (*Intel i7 Processor*) takes about 1.7 nanoseconds to try out one key from one block. With this speed it would take about *1.3 × 10*^{12} *× the age of the universe* to check all the keys from the possible key set [8] of asymmetric scheme. By reducing the size of the prime number; we speed up the security verification process at DSM (see Table 2). As shown in Table 2, 64 bit symmetric key takes 3136e +19 nanoseconds (more than a month), so we safely concluded that updating the prime number every week (*i.e. t=168 hours*) will not compromise the security of the system. Dynamic shared key is computed based on the prime number. Hence we conclude that attacker cannot crack the shared key within the interval time *t*. Further, the shared key is updated without exchanging information between the sensors and DSM. This leads to confusion for the adversaries who may try to intercept the data flow. The key has been already changed four times before an attacker knows the key and this knowledge is not known to the attackers.

Table 2. Notations Symmetric key (AES) algorithm takes time to get all possible keys using 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 nanosecond)	1435	1e+05	7.301e +09	3136e +19	5.7848e +35

Theorem 2: Dynamically generated prime number P_i in Algorithm I 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 initially initialize the value of *k* based on this primary test formula. Our prime generation method is based on the nth prime number generation and from the extended idea of [3]. In our scheme, 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 DPBSV Handshaking process and the interval time is t.

ALGORITHM I. DYNAMIC PRIME NUMBER GENERATION

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

From the *Algorithm I,* we calculate the new prime number P_i based on the previous one P_{i-1}. The complete process of the prime number calculation is based on the value of *m* and *m* is initialized from the value *k*. The value of k is constant at source because it is calculated from current prime number. This process is initialized during *DPBSV Handshaking*. Since the value of k is the same on both sides, the procedure *Prime (Pi)* returns identical values. In Algorithm I, the value of *S(m)* computed as below [3].

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

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

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

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

$x \not\equiv 0 \bmod i \ \forall\ 1 \leq i \leq x-1$, if *x* is prime

Then put the value of *x* as a prime number, then

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

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

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

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

Same $S_2(x)$ is also 1 and then

$$S(x) = \frac{S_1(x)+S_2(x)}{2} = 1$$

Hence, the property of $S(x)$ is proved.

Theorem 3: An *attacker M_a* cannot read the secret information from sensor node (S_i) or introduce itself as an authenticated node in DPBSV.

Proof: Following *Definition 1*, we know that an attacker M_a can gain access to the shared key K_{SH} by monitoring the network thoroughly, but M_a cannot get secret information such as *Prime (Pi) and KeyGen*. 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 tap into the data stream, however M_a can introduce himself/ herself as the authenticated node and start sending false information to DSM. In our scheme, sensor (S_i) sends$\left((DATA \oplus K_{SH} \oplus K_i) \parallel (S_i \oplus K_{SH})\right)$, 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 the 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 marks it is not an authenticated sensor node. All required secured information for prime number and key generation procedure are stored at trusted part of the sensor node (i.e., TPM). According to the features of TPM, the attacker cannot get the information from TPM as discussed before. Hence we conclude that *attacker M_a* cannot attack or get access to the big data stream.

Theorem 4: An *attacker M_i* cannot read the shared key K_{SH} within the time interval t in DPBSV 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*, because our proposed scheme calculate P_i randomly after time *t* and then use the value P_i sensor to generate K_{SH}. For more details on computation analysis, readers can refer to *Theorem 1*.

VI. EXPERIMENT AND EVALUATION

In order to evaluate the efficiency and effectiveness of the proposed DPBSV scheme under the adverse conditions, we observe each individual data blocks for authentication check and selected data blocks for integrity attacks. The integrity attack verification interval is dynamic in nature and the data verification is done at the DSM only.

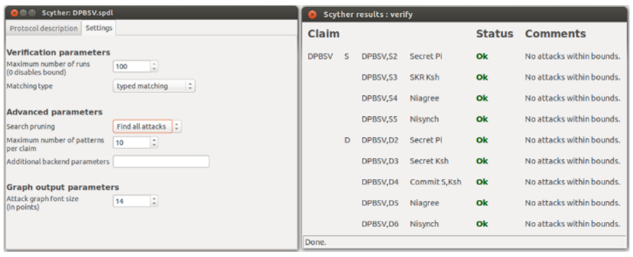
To validate our proposed scheme, we experimented with two different approaches by using different simulation environments. We first verify the security scheme using Scyther [14], and then measure the efficiency of the scheme using JCE (Java Cryptographic Environment) [15].

A. *Security Verification*

The scheme 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). Next, S and D have all the required information that are exchanged during the handshake process. This enables D and S to update their 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 the adversaries. The first type of attack is defined for the transmission between S and D (integrity) and the second attack is defined where an adversary acquires the property of S and sends the attack data packets to D (authentication). In our experiments, we evaluated all packets at D (DSM) for security verification. We experimented with 100 numbers of runs for each claim (also known as bounds) and found out the number of attacks at D as shown in Figure 2. Apart from these, we follow the default properties of Scyther.

Attack model: Many types of cryptographic attacks can be considered. In our case, we focus on integrity attack and authentication attack as discussed above. In integrity attack, an attacker can only observe encrypted data blocks/packets being transmitted over the network, that contain information about sensed data as shown in Figure 1. The attacker can perform a brute force attack on captured packets by systematically testing every possible keys, and we assumed that he/she is able to determine when the attack is successful. In authentication attack, an attacker can observe source node, and try to get the behavior of the source node. We assume that he/she is able to determine the source node's behavior. In such case, the attacker can introduce an authenticated node and act as the original source node. In our concept, we are using trusted module in sensor to store the secret information and procedure for key generation and encryption (such as TPM).

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 DPBSV scheme 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 fixed the key size 64 bits (see Table 2). We used Scyther an automatic security protocols verification tool to verify our proposed scheme.



Verification parameters
Maximum number of runs (0 disables bound): 100
Matching type: typed matching

Advanced parameters
Search pruning: Find all attacks
Maximum number of patterns per claim: 10
Additional backend parameters:

Graph output parameters
Attack graph font size (in points): 14

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

Done.

Figure 2. Scyther simulation environment with parameters and result page of success security verification at DSM.

Results: We did our simulation using the variable numbers of data block in each run. Our experiment ranges from 10 to 100 instances with 10 intervals. We check authentication for each data block, whereas the integrity check is performed on the selected data blocks. As our secure information such as K_d, t, P_i, $Prime(P_i)$, K_{SH}, $KeyGen$ are stored within the trusted module of the sensor, no one can get access to those information except the corresponding sensor. Without these information, attackers cannot authenticate encrypted data blocks. Hence, we did not find any attacks for authentication check. For integrity attacks, it is hard to get shared key (K_{SH}), as we are frequently changing the shared key (K_{SH}) based on the dynamic prime number P_i on both source sensor (S_i) and DSM. In the experiment, we did not encounter any attack in integrity check. Figure 2 shows the result of security verification experiments in Scyther environment. This shows that our scheme is secured from integrity and authentication attacks. From the observations above, we can conclude that our proposed scheme is secure.

B. *Performance Comparison*

Experiment model: It is clear that the actual efficiency improvement brought by our scheme highly depends on the size of 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 scheme DPBSV with advanced encryption standard (AES), the standard symmetric key encryption algorithm [1, 2]. Our scheme was compared with two standard symmetric key algorithms: 128-bit AES and 256-bit AES. This performance comparison experiment is 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 has the properties of DSM (more powerful to initialize the process). 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 Figure 3; the result validates the theoretical analysis presented in section IV.

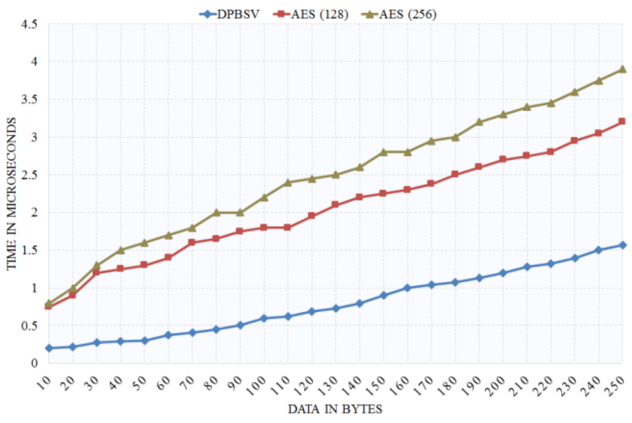


Figure 3. Performance of proposed scheme compared in efficiency to 128 bit AES and 256 bit AES.

Results: The performance of our scheme is better than the standard AES algorithm when different sizes of the data blocks are considered. Figure 3 shows the processing time of the proposed DPBSV scheme 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 scheme is more efficient and faster than the baseline AES protocols.

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

VII. Conclusions and Future Works

In this paper, we have proposed a novel authenticated key exchange scheme, namely Dynamic Prime-Number Based Security Verification (DPBSV), which aims to provide efficient and fast (on-the-fly) security verification scheme for big data stream. Our scheme has been designed based on the symmetric key cryptography and random prime number generation. By theoretical analyses and experimental evaluations, we showed that our DPBSV scheme has provided significant improvement in the processing time, and prevented malicious attacks on authenticity and integrity. In our scheme, we decrease the communication and computation overhead by dynamic key initialization at both sensor and DSM end, which in effect eliminates the need of rekeying 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 using the technique to develop a moving target defense strategy for the Internet of Things.

VIII. Acknowledgement

This research is funded by Australia India Strategic Research Grant titled "Innovative Solutions for Big Data and Disaster Management Applications on Clouds (AISRF-08140)" from the Department of Industry, Australia.

References

[1] PUB, NIST FIPS. "197: Advanced encryption standard (AES)." *Federal Information Processing Standards Publication* 197, pp. 441-0311, 2001.

[2] S. Heron, "Advanced Encryption Standard (AES)." *Network Security* 2009(12) pp. 8-12, 2009.

[3] I. Kaddoura and S. Abdul-Nabi, "On formula to compute primes and the nth prime." *Applied Mathematical science, 6*(76), pp.3751-3757, 2012.

[4] R. V. Nehme, et al., "StreamShield: a stream-centric approach towards security and privacy in data stream environments." In *ACM SIGMOD*, pp. 1027-1030, 2009.

[5] I. F. Akyildiz, et al., "Wireless sensor networks: a survey." *Computer networks, 38*(4), pp. 393-422, 2002.

[6] M. Benantar, R. H. High Jr, and M. K. Rathi, "Method and system for maintaining client server security associations in a distributed computing system." *U.S. Patent 6,141,758*, October 31, 2000.

[7] J. Burke, J. McDonald, and T. Austin, "Architectural support for fast symmetric-key cryptography." *ACM SIGOPS Operating Systems Review* 34(5), pp. 178-189, 2000.

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

[9] D. Puthal, "Secure Data Collection and Critical Data Transmission Technique in Mobile Sink Wireless Sensor Networks. " *M.Tech Thesis, National Institute of Technology, Rourkela*, 2012.

[10] R. Ranjan, "Streaming Big Data Processing in Datacenter Clouds." *IEEE Cloud Computing, 1*(1), pp. 78-83, 2014.

[11] S. Nepal, J. Zic, D. Liu, and J. Jang, "A mobile and portable trusted computing platform." *EURASIP Journal on Wireless Communications and Networking, 2011*(1), pp. 1-19, 2011.

[12] V. Gulisano, et al., "Streamcloud: An elastic and scalable data streaming system." *IEEE Transactions on* Parallel and Distributed Systems," *23*(12), pp. 2351-2365, 2012.

[13] B. R. Kandukuri, V. R. Paturi, and A. Rakshit, et al., "Cloud security issues." *IEEE International Conference on Services Computing, (SCC'09)*, pp. 517-520, 2009.

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

[15] M. Pistoia, et al., "Enterprise Java 2 Security: Building Secure and Robust J2EE Applications." *Addison Wesley Professional*, 2004.

[16] D. Carney, et al., "Monitoring streams: a new class of data management applications." In *Proceedings of the 28th international conference on Very Large Data Bases*, pp. 215-226, 2002.

[17] S. Chandrasekaran, et al., "TelegraphCQ: continuous dataflow processing." In *Proceedings of the ACM SIGMOD international conference on Management of data*, pp. 668-668, 2003

[18] D. Puthal, B. P. S. Sahoo, S. Mishra, and S. Swain., "Cloud Computing Features, Issues, and Challenges: A Big Picture." In *International Conference on Computational Intelligence and Networks (CINE)*, pp. 116-123, 2015.

[19] D. Puthal, and B. Sahoo. "Secure Data Collection & Critical Data Transmission in Mobile Sink WSN: Secure and Energy efficient data collection technique" *LAP Lambert Academic Pubilishing: Germany*, 2012. ISBN: 978-3-659-16846-8.

[20] M. Stonebraker, U. Çetintemel, and S. Zdonik, "The 8 Requirements of Real-Time Stream Procesing." *ACM SIGMOD Record, 34*(4), pp. 42-47, 2005.

[21] H. Demirkan and D. Delen, "Leveraging the capabilities of service-oriented decision support systems: Putting analytics and big data in cloud." *Decision Support Systems,55*(1), pp. 412-421, 2013.

[22] A. Bifet, "Mining big data in real time." *Informatica (Slovenia), 37*(1), pp. 15-20, 2013.

[23] J. Lu, and D. Li, "Bias correction in a small sample from big data." *IEEE Transactions on Knowledge and Data Engineering, 25*(11), pp. 2658-2663, 2013.

[24] J. M. Tien, "Big data: unleashing information." *Journal of Systems Science and Systems Engineering, 22*(2), pp. 127-151, 2013.

[25] M. Dayarathna and T. Suzumura, "Automatic optimization of stream programs via source program operator graph transformations." *Distributed and Parallel Databases, 31*(4), pp. 543-599, 2013.

[26] D. Zissis, and D. Lekkas, "Addressing cloud computing security issues." *Future Generation Computer Systems* 28(3) pp. 583-592, 2012.

[27] C. Liu, et al., "CCBKE-Session key negotiation for fast and secure scheduling of scientific applications in cloud computing." *Future Generation Computer Systems* 29(5) pp. 1300-1308, 2013.

[28] K. W. Park, S. S. Lim, and K. H. Park, "Computationally efficient PKI-based single sign-on protocol, PKASSO for mobile devices." *IEEE Transactions on Computers, 57*(6), pp. 821-834, 2008.