Journal of Computer and System Sciences ••• (••••) •••–•••

Contents lists available at ScienceDirect

Journal of Computer and System Sciences

www.elsevier.com/locate/jcss

A dynamic prime number based efficient security mechanism for big sensing data streams ☆

Deepak Puthal [a,*], Surya Nepal [b], Rajiv Ranjan [b,c], Jinjun Chen [a]

[a] Faculty of Engineering and Information Technology, University of Technology Sydney, Australia
[b] CSIRO Data61, Australia
[c] School of Computing Science, Newcastle University, UK

ARTICLE INFO

Article history:
Received 28 August 2015
Received in revised form 23 January 2016
Accepted 2 February 2016
Available online xxxx

Keywords:
Security
Sensor networks
Big data stream
Key exchange
Security verification

ABSTRACT

Big data streaming has become an important paradigm for real-time processing of massive continuous data flows in large scale sensing networks. While dealing with big sensing data streams, a Data Stream Manager (DSM) must always verify the security (i.e. authenticity, integrity, and confidentiality) to ensure end-to-end security and maintain data quality. Existing technologies are not suitable, because real time introduces delay in data stream. In this paper, we propose a Dynamic Prime Number Based Security Verification (DPBSV) scheme for big data streams. Our scheme is based on a common shared key that updated dynamically by generating synchronized prime numbers. The common shared key updates at both ends, i.e., source sensing devices and DSM, without further communication after handshaking. Theoretical analyses and experimental results of our DPBSV scheme show that it can significantly improve the efficiency of verification process by reducing the time and utilizing a smaller buffer size in DSM.

© 2016 Elsevier Inc. All rights reserved.

1. Introduction

1.1. Big data stream

A large number of applications, such as large scale sensors, information monitoring, web exploring, data from social networks like Twitter and Facebook, surveillance data analysis, and financial data analysis, deal with a large stream of data input, and consequently require an alternate ideal model of real-time data processing. As a result, a new computing paradigm based on Stream Processing Engines (SPEs) has appeared [15]. SPEs deal with the specific types of challenges and are intended to process data streams with a minimal delay [15–18]. In SPEs, data streams are processed in real time (i.e. on-the-fly) rather than batch processing after storing the data.

Several of these applications are approaching the bottleneck of current data streaming infrastructures and require real-time processing of very high-volume and high-velocity data streams (also known as *big data streams*). 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,

☆ The preliminary version of this paper is published in *14th IEEE International Conference on Trust, Security and Privacy in Computing and communications* (IEEE TrustCom-15) Helsinki, Finland, 2015 [47].

* Corresponding author.
E-mail addresses: deepak.puthal@gmail.com (D. Puthal), surya.nepal@csiro.au (S. Nepal), rranjans@gmail.com (R. Ranjan), jinjun.chen@gmail.com (J. Chen).

http://dx.doi.org/10.1016/j.jcss.2016.02.005
0022-0000/© 2016 Elsevier Inc. All rights reserved.

2) *variety* – referring to unstructured, semi-structured and structured data from different sources like social media (Twitter, Facebook etc.), sensors, surveillance, image or video, medical records etc., 3) *velocity* – referring to the high speed at which the data is handled in/out for stream processing, and 4) *veracity* – referring to the quality of data. These features introduce huge open doors and enormous difficulties for big data stream computing. A big data stream is continuous in nature and it is important to perform real-time analysis as the lifetime of the data is often very short (data is accessed only once) [22,25]. As the volume and velocity of the data is so high, there is not enough space to store and process; hence, the traditional batch computing model is not suitable.

1.2. Data stream security verification

Even though big data stream processing has become an important research topic in the current era, the data stream security has received little attention from researchers [43,44]. Some of these data streams are analyzed and used in very critical applications (e.g. surveillance data, military application, Supervisory Control and Data Acquisition (SCADA), etc.), where data streams need to be secured in order to detect malicious activities. The problem is exacerbated when thousands to millions of small sensors in self-organizing wireless networks become the sources of the data stream. How can we provide the security for big data streams? In addition, compared to conventional store-and-process, these sensors will have limited processing power, storage, bandwidth, and energy. Furthermore, data streams ought to be processed on-the-fly in a prescribed sequence. In this paper, we address these issues by designing an efficient architecture for real-time processing of big sensing data streams, and the corresponding security scheme.

The common approach to security is to apply a cryptographic model. Keeping data encrypted is the most common and safe choice to secure data in transmission, if the encryption keys are managed properly. There are two most common types of cryptographic encryption methods: 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 1000 times slower than symmetric key cryptography [7,8]. Efficiency becomes an issue if asymmetric-key based infrastructure such as the Public-Key Infrastructure PKI [28,29] is applied to big data streams. Thus, symmetric-key encryption is the most efficient cryptographic solution for such applications. However, symmetric-key algorithms (e.g. DES, AES, IDEA, RC_4) fail to meet the requirements of real-time big data streams security processing due to the properties of big data (i.e., 4 Vs). Hence, there is a need for an efficient scheme for securing big data streams.

1.3. Motivation

The discussion above led to four most important features of the big data stream from the point of view of security verification:

1. Security verification needs to be performed in near 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 the store-and-process paradigm, original data is not available for comparisons in the context of the stream processing paradigm.

In light of the above features and properties of big data streams, we classified existing security solutions into two classes: Communication Security [9,19,48,49] and Server side data security [26,27,30,36]. Communication security deals with data security when it is in motion and server side security deals with data security when it is at rest. The security threats and solutions proposed in the literature are further discussed in the related works section. Those proposed solutions are suitable for *store-and-process*, however are not plausible for big data streams.

Another major motivation is to perform the security verification on near real time in-order to synchronize with the processing speed of SPEs [43]. Stream data analysis performance should not degrade because of security processing time, there are several applications needs to perform data analysis on real time. According to the features of big data stream (i.e. 4 Vs) existing security solution needs huge buffer size to process security verification. Which is simple impossible to maintain such big buffer for data stream because of the continuous nature of data. Therefore, light wait security mechanism is very much important to perform security verification on near real time and reduce buffer size.

1.4. Research challenges

As discussed earlier in this section, symmetric cryptographic solution is the best way to protect data in faster processing time. Existing symmetric cryptographic based security solutions for data security are 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 the security verification time. From the required features of big data streams specified in last subsection, it is clear that security verification should be in real-time. For the dynamic key, centralizing processor rekeying and distributing keys to all the sources is a time consuming process. A big data stream is always continuous in nature and huge in size. This makes it impossible to halt data for rekeying, distribution to the sources and synchronization with DSM.

Please cite this article in press as: D. Puthal et al., A dynamic prime number based efficient security mechanism for big sensing data streams, J. Comput. Syst. Sci. (2016), http://dx.doi.org/10.1016/j.jcss.2016.02.005

Buffer size for the security verification is another major issue because of the volume and velocity of the big data stream. According to the features of big data stream (i.e. 4 Vs), we cannot halt the data for more time before performing the security verification. This leads to an allocation of bigger buffer size in SPEs and may reduce the performance of SPEs. Hence, the buffer size reduction is one of the major challenges for big data stream.

1.5. Our contributions

In order to address the challenges, we have designed and developed a Dynamic Prime-Number Based Security Verification (DPBSV) scheme. Our scheme takes in to account a typical shared key that is updated dynamically by producing synchronize prime numbers. The synchronize prime number generation at both source sensing device and DSM enables reduction of the communication overhead without compromising security. Due to the reduced communication overhead, our scheme is suitable for big data streams as it verifies the security on-the-fly (near real time) and reduce the buffer usage. Our proposed scheme uses a smaller key length (64-bit). This enables faster security processing at DSM without compromising the security. The same level of security is accomplished by changing the key progressively in a specific interval of time. Dynamic key generation is based on the random prime numbers, which are initialized and synchronized at source sensors and DSM without further communications between them after handshaking. This increases the efficiency of the solution. Based on the shared key properties, individual source sensors updates their dynamic key independently. Due to the reduced key length, our scheme is suitable for processing high volumes of data without any delay. This makes DPBSV highly efficient at DSM for processing secured big data streams.

In summary, we are proposing a scheme for big data stream security verification without the need of key exchange for rekeying. The additional benefit of this is that it reduces the communication overhead and increases the efficiency of the security verification process at DSM. Our proposed scheme is efficient in comparison to AES, as it reduces the computational load and execution time significantly compared to the original AES; furthermore, it also strengthens the security of the data, which is the main research contribution of this paper. The contributions of the paper can be summarized as follows:

- We present a secure big data stream processing architecture.
- We design and develop an efficient Dynamic Prime-Number Based Security Verification (DPBSV) scheme for big data streams.
- We evaluate our proposed DPBSV scheme in our architecture and show that our solution is efficient when applied to big data streams in comparison to AES standard.

1.6. Organization of the paper

The rest of this paper is organized as follows: related work is reviewed in the next section; Section 3 provides the background on big sensing data stream and corresponding security related work; Section 4 describes our DPBSV key exchange scheme; Section 5 presents the security analysis of our scheme formally; Section 6 evaluates the performance and efficiency of our scheme through experimental results and Section 7 concludes our work and suggests future work.

2. Related works

This section describes the related works in two broad categories: stream data processing and data security.

2.1. Stream data processing

Data streaming has become an important paradigm for the real-time processing of continuous data flows in several domains such as finance, telecommunications, large scale sensor networks which require online processing and security verification of continuous data flows. A large amount of data is collected by such applications; for example, Tien [24] measured about 4 zettabytes (or 10**21 bytes) of digital data being generated per year by everything from underground physics experiments to retail transactions to security cameras to global positioning systems.

Stonebraker et al. [20] outlined eight requirements that a system software should meet to excel at a variety of real-time stream processing applications: Keep the Data Moving, Query using SQL on Streams (StreamSQL), Handle Stream Imperfections (Delayed, Missing and Out-of-Order Data), Generate Predictable Outcomes, Integrate Stored and Streaming Data, Guarantee Data Safety and Availability, Partition and Scale Applications Automatically, and Process and Respond Instantaneously. For our purpose, we have classified some important paradigm of stream data processing in Table 1. We use the application areas, processing techniques, proposed technique and QoS as our classification properties. We next describe some of these architecture in brief.

Arasu et al. [15] proposed a Data Stream Management System (DSMS), called STREAM, for STanford stREam data Manager. The challenges in building a DSMS instead of a traditional DBMS is to handle multiple continuous, unbounded, possibly rapid and time-varying data streams. Fig. 1 shows the high level abstraction of stream data processing at DSMS. The incoming streams (on the left) produce data indefinitely and drive query processing. Processing of continuous queries typically requires intermediate states, stored as Scratch Store. This state could be stored and accessed in memory or on disk. Although

Please cite this article in press as: D. Puthal et al., A dynamic prime number based efficient security mechanism for big sensing data streams, J. Comput. Syst. Sci. (2016), http://dx.doi.org/10.1016/j.jcss.2016.02.005

Table 1
Stream processing classifications of some exiting technology.

Architecture	Focused data stream	Proposed technique	Processing technique	QoS
StreamCloud [12,14]	Deals with big data	Parallel Query processing	Parallel	Scalable and elastic with input load
Staying FIT [13]	Distributed sensor network	Load Shedding	Centralize and Distributed	–Effect of query load distribution –Effect of input dimensionality
STREAM [15]	Any	Query execution resource utilization	Centralized	Multiple continuous queries over multiple continuous data streams
Monitoring Streams [16]	Sensor data	Stream monitoring	Centralized	–Response times –Tuple drops
Aurora [17]	Other data then human data	Monitoring applications	Centralized	Resource allocation decisions
TelegraphCQ [18]	Sensor network	Continuous queries processing	Centralized	Scalable query processing
Flexible Filters [23]	Embedded application	Dynamic load balancing	Distributed	–Load balancing for stateless and stateful operators –Redistributes the load on the fly.

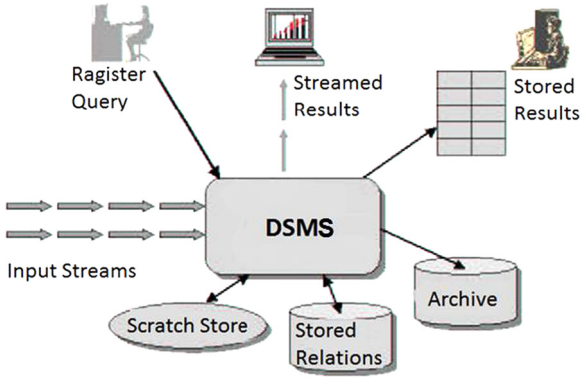


Fig. 1. A simplified view of a DSMS to process and analyze input data stream [15].

we are concerned primarily with the online processing of continuous queries, in many applications stream data also may be copied to an archive, for preservation and possible offline processing of expensive analysis or mining queries.

Gulisano et al. [12,14] presented *StreamCloud*, a large scale data streaming system for processing large data stream in cloud environments. The proposed method is a highly scalable and elastic. *StreamCloud* runs on top of a distributed Stream Processing Engine (SPE), but is made independent from it by implementing the parallelization with standard stream administrators. Tatbul et al. [13] studied the problem of load shedding in distributed stream processing and show its difference from existing centralized solutions, and they offered several new practical algorithms for addressing the problem. The presented solution is a distributed algorithm called DFIT that works by transmitting its load requirements locally to its parents. They also investigated several centralized solutions a linear programming solution (Solver), a variant on Solver that takes a workload history into account (Solver-W), and a centralized version of our distributed algorithm (C-FIT) and compares them.

Carney et al. [16] described the architecture of Aurora with the primitive building blocks for workflow processing, a DAHP system oriented towards monitoring applications. Abadi et al. [17] proposed a complete architecture of Aurora (a new model and architecture for data stream management for monitoring applications) and describe a stream-oriented set of operators. With several heuristics for optimizing a large Aurora network, they focused on run-time data storage and processing issues, discussing storage organization, real-time scheduling, introspection, and load shedding.

The TelegraphCQ, a dataflow system for processing continuous queries over data streams which supports dynamic query workloads in volatile data streaming environments is presented by Chandrasekaran et al. in [18]. The TelegraphCQ DDL supports the creation of archived and unarchived streams that are fed with external sources using stream and source specific wrapper functions. For implementation, TelegraphCQ integrated with a sensor network and for simulation freeway traffic sensors used.

In [23], overloaded operators trigger a reconfiguration of the load distribution policy with a “backpressure” message to upstream peers. However, the authors of [23] only consider stateless operators. SC provides load balancing for stateless and stateful operators and redistributes the load on the fly.

2.2. Data security

Cryptographic based security frameworks are proposed mainly in two different classes to protect data, i.e. Communication Security (data in motion) [9,19,48,49] and Server side data security (data at rest) [26,27,30,36]. Authors in [9,19] defined

Please cite this article in press as: D. Puthal et al., A dynamic prime number based efficient security mechanism for big sensing data streams, J. Comput. Syst. Sci. (2016), http://dx.doi.org/10.1016/j.jcss.2016.02.005

Table 2
Communication security threats and existing solutions.

Communication layers	Possible attacks	Security solutions
Physical Layer	• Jamming • Tampering	• Spread spectrum communication • Jamming reports • Accurate and complete design of the node physical package
Data Link Layer	• Collision • Exhaustion • Unfairness • Interrogation Attack • SYBIL Attack	• Error correcting codes • Collision detection and avoidance techniques • Rate limiting
Network Layer	• Selective Forwarding • Sinkhole • Sybil attack • Wormhole • HELLO flood • Spoofing and alternating routing information • Node capture/Node replication attack	• Link layer encryption and authentication • Multipath routing • Identity verification • Authenticated broadcast
Transport Layer	• Flooding • DE synchronization	Packet authentication including all control fields in the transport protocol header

security requirements with descriptions of individual features such as Data Confidentiality, Data Integrity, Data Freshness, Availability, and Authentication. They also classified the layer wise security threats and exiting solutions (i.e. physical layer, data link layer, network layer, transport layer) for wireless communication. They also proposed secure data collection and secure data transmission techniques in wireless networks. Both of these proposed techniques used symmetric key based cryptography.

There are several solutions proposed in the literature to protect data in wireless networks and IoT environment. We grouped the security threats and solutions for different communication layers in Table 2. For example, Jan et al. [48] proposed a novel detection scheme for Sybil attack in a centralized clustering-based hierarchical network. In [49], they have proposed a lightweight authentication scheme for IoT environment. Proposed scheme verifies the identities of the participating clients and servers in a CoAP-based and it follows a session key based solution.

Server side data security is mainly proposed for physical data centers, 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. To overcome these attacks, several solutions have been proposed such as privacy in multitenant environments, data protection from disclosure, access control, software security, service availability, access control, application security, data security (data in transit, data at rest, reminisce), cloud management control security, virtual cloud protection, hardware security, and hardware reliability [26,27,30,36]. We have classified the cloud based security threats and security requirements in Table 3. According to the recent research trend, we highlighted the cloud based security approach and the list of security requirements and threats, that are extensively from [26]. We describe a few techniques below as examples of such techniques.

Liu et al. [27] proposed an authenticated key exchange scheme, called Cloud Computing Background Key Exchange (CCBKE). This proposed method is an efficient security-aware scheduling of scientific applications in hybrid computing environments such as cloud computing and designed based on the commonly-used Internet Key Exchange (IKE) scheme and randomness-reuse strategy. Benantar et al. [30] introduced a method along with a corresponding framework for keeping up a safe relationship between a customer and a server in a distributed processing framework by registering a session identifier as a capacity of a Kerberos-based authentication ticket. The session identifier is freely inferred or confirmed by the client and the server upon first demand by the customer to the server and every consequent demand by the customer to the server is labeled with this session identifier to give a solid security affiliation. Wang et al. [51] proposed a new computing paradigm called as Cyberinfrastructure as a Service (CaaS), which provides an on demand service to building a cyberinfrastructure. Authors developed a lightweight distributed middleware namely Cyberaide Creative and demonstrated the usage via a real application, and test it with a High Performance Computing (HPC) benchmark. The authors in [50] present a parallel solution using the MapReduce paradigm in the cloud that can deliver a high-performance, fault-tolerant, and flexible solution in a water distribution system (WDS).

In the following, our focus will be on existing security solutions for data stream. They mainly focus on access control and query level security [43–45]. Nehme et al. [43] initially highlighted the need for a 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 data security, whereas query security punctuation deals with 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 [43,44]. For example, FENCE, a continuous access control framework in dynamic data stream environments, deals with both data and query

Please cite this article in press as: D. Puthal et al., A dynamic prime number based efficient security mechanism for big sensing data streams, J. Comput. Syst. Sci. (2016), http://dx.doi.org/10.1016/j.jcss.2016.02.005

Table 3
Cloud service level wise security requirements and threats.

Service level	Security requirements	Threats
Software as a Service (SaaS)	• Privacy in multitenant • Environment • Data protection from disclosure • Data access • Software security • Authentication and Authorization	• Interception • Data interruption (deletion) • Privacy breach • Impersonation • Session hijacking • Traffic flow analysis
Platform as a Service (PaaS)	• Access control • Instruction Detection • Data security, (data in transit, data at rest, remanence)	• Programming flaws • Software modification • Software interruption (deletion)
Infrastructure as a Service (Iaas)	• Virtual cloud protection • Communication security • Physical security • Environmental security • Virtualization security	• Impersonation • Session hijacking • Traffic flow analysis • Exposure in network • Defacement • DDOS
Physical Data Centre	• Legal not abusive use of cloud computing • Hardware security • Hardware reliability • Network protection • Network resources protection	• Network attacks • Connection flooding • DDOS • Hardware interruption, theft and modification • Misuse of infrastructure • Natural disasters

security restrictions [44]. It gives low overhead which is suitable for data stream environments. Similarly, ASSIST, an application system based on an effective and efficient access control framework, is proposed to protect streaming data from unauthorized access [45]. ASSIST has been implemented on top of StreamInsight, a commercial stream processing engine. This paper focuses on data security punctuation, where our security mechanism is to protect the data efficiently from potential attacks from/on untrusted intermediaries before the data reaches to the DSM. Wang et al. [46] proposed a framework named *ARTSense*, to trust without identity in participatory sensing networks. The proposed framework achieves the trust of information, reputation of participants, anonymity, and security requirements.

The question is then can we apply existing communication and server side security to the big data stream to overcome the shortcomings of current approaches. Existing solutions for communication security or server side security do not satisfy the requirements of big data stream. We propose a novel light weight security mechanism for big data stream. The preliminary version of this paper contains the stream data processing architecture, security requirements followed by proposed a mechanism to address the security verification of big data stream (e.g., integrity, and authenticity) [47]. In this paper, we propose a solution, called Dynamic Prime Number Based Security Verification (DPBSV), which is based on a common shared key that is updated dynamically by generating synchronized pairs of prime numbers for real time security verification (i.e., confidentiality, integrity and authenticity) on big data stream. We have shown the efficiency of our approach by reducing the security computation time and buffer utilization.

3. Proposed secure data stream architecture

3.1. Stream processing

Data stream processing is an emerging computing paradigm which is particularly suitable for application scenarios where huge amounts of data (Big Data) must be processed in near real-time (with small delay). Rather than processing stored data like in conventional clouds or database systems, Data Stream Manager (DSM) processes stream data on-the-fly. The needs of on-the-fly processing include the amount of input data that discourages the use of persistent storage, the requirement of providing prompt results, etc. DSM is designed to handle high-volume and bursty data streams from multiple sources. DSM handles streams of tuples similarly to the way a conventional database system handles relations. In addition, DSM needs to do the security verification of the data blocks on near real time to synchronize with stream data analysis.

Fig. 2 shows an overall architecture for a big data stream process from source sensing device to the cloud data processing center, including our proposed security framework on the data stream. It also shows the complete architecture of stream data processing in the data center of a cloud. Refer to [42] for further information on stream data processing in datacenter clouds. It starts with a three step process to reach data at DSM for stream processing. These three steps include collection, processing, and storing. All the query and security related processes are handled in DSM. It is important to note that the security verification of stream data has to be performed before query processing and it has to be done in real time (with small delay) with a fixed (small) buffer size. The processed data is stored in the cloud storage. Queries used in DSM are defined as “continuous” since they are continuously standing over the *streaming* data. Results are pushed to the user each

Please cite this article in press as: D. Puthal et al., A dynamic prime number based efficient security mechanism for big sensing data streams, J. Comput. Syst. Sci. (2016), http://dx.doi.org/10.1016/j.jcss.2016.02.005

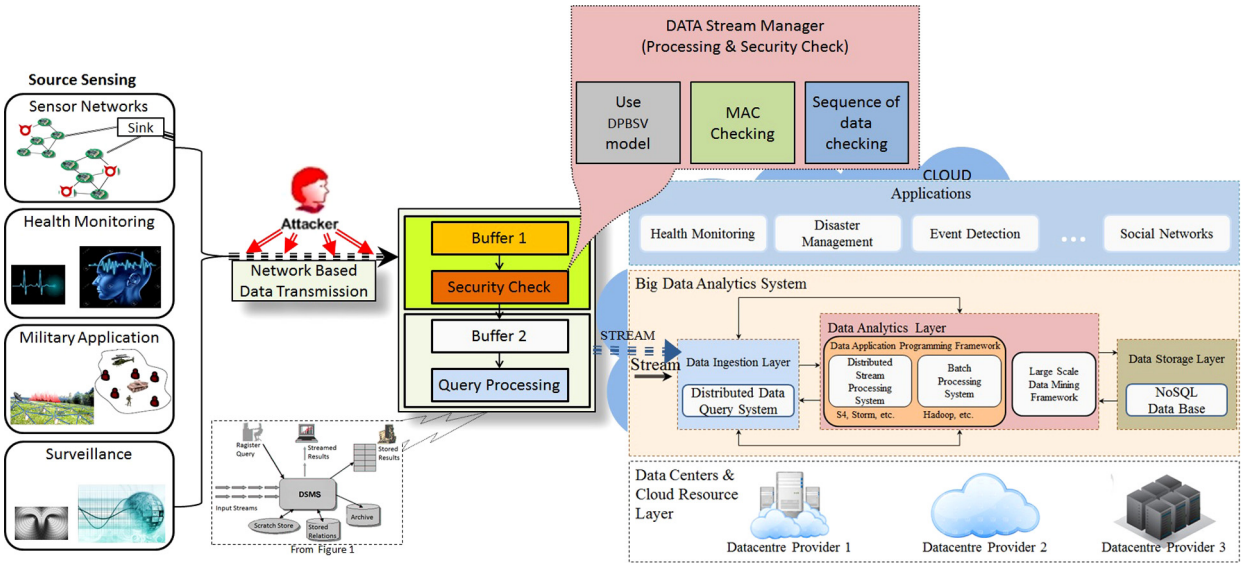


Fig. 2. Overlay of our architecture from sensing device to cloud data processing center.

time the streaming data satisfies the query predicate. The queries including security verification are defined as a directed acyclic graph where each node is an operator and edges define data flows.

A stream of data a potentially infinite sequence of tuples, denoted as $(T_1, T_2, \ldots, T_n)$. The data sources have clocks that are well synchronized with other system nodes as in [13]. A query is modeled as a network of connected operators. A connection represents a data flow. Typical query operators of DSMs are filter, map, union, join, and aggregate [12]. These operators correspond to relational algebra operators. Operators can be classified as stateless (filter, map and union) or stateful operators (join and aggregates) [14]. As the nature of the data stream is infinite, stateful operators perform their computation over sliding windows of tuples defined over a period of time (e.g. tuples received in the last hour). Cloud computing has become a platform of choice due to its extremely low-latency and massively parallel processing architecture [41]. It supports the most efficient way to obtain actionable information from big data streams [21–24].

As discussed before fixed buffer size is required for security verification, here we present procedure to compute halting time of data block in buffer. Let there are n number of sensors and each send m number of data blocks. We assume that in a DSM buffer the probability of attempt to success security verification is $(1/(n \times m))$, or delays with probability $1 - (1/(n \times m))$. We can compute Acquisition Probability as $A = \left(1 - \left(\frac{1}{(n \times m)}\right)^{((n \times m)-1)}\right)$ [34]. Based on the value of A, we can measure the halting time of the each individual data block; the halting time represented as w is $A \times (1 - A)$, where the value of w is inversely proportional to the value of A and processing time of DSM.

It is clear from the above description that security verification at DSM is one of the important features of big data stream architecture. The major concern is to process security verification in real time because of the features of big data streams. Security verification at DSM increases the stream query processing time. The major challenge of security processing time is that it should not introduce any time delay at DSM. This is critical for big data stream due to the high volume and velocity. Slow processing leads to the requirement of higher buffer size to store data before performing security verification. Hence, security verification should be done on-the-fly (with minimum delay). Motivated by this problem, this paper aims to address the challenge of real time security verification on massive data streams at DSM.

3.2. Why symmetric key cryptography?

Symmetric keys are smaller in size than asymmetric keys, so they require less computational burden. 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 [8]. Our aim is to perform security verification on-the-fly (real-time). Symmetric key cryptography becomes a natural choice for this purpose. It is mentioned with a proof that symmetric key cryptography is approximately 1000 times faster than strong public key ciphers [7]. However, it is comparatively easy for an attacker to read/modify the data as the symmetric key cryptography key length is small [7]. To overcome this problem, we use a synchronized dynamic prime number (P_i) generation algorithm at both source and DSM with equal interval of time in order to update cryptography keys dynamically to confuse malicious attackers. The procedure $Prime(P_i)$ is calculated and synchronized on both sides as shown in Fig. 3. This proposal makes the process faster and prevents potential attacks on the data streams. We explain it in detail in a later section.

Please cite this article in press as: D. Puthal et al., A dynamic prime number based efficient security mechanism for big sensing data streams, J. Comput. Syst. Sci. (2016), http://dx.doi.org/10.1016/j.jcss.2016.02.005

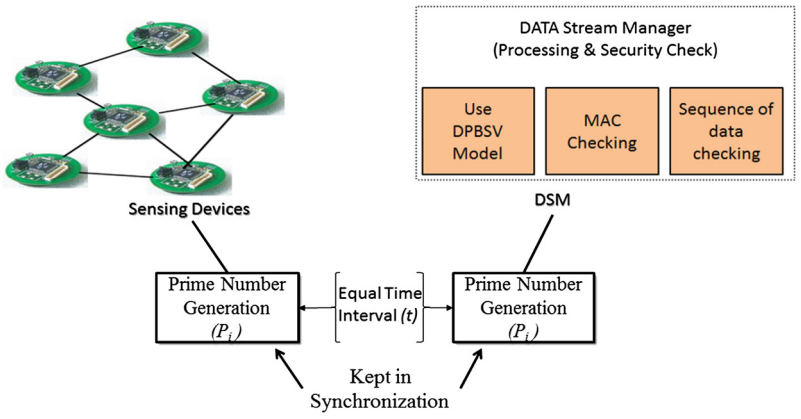


Fig. 3. Pair of dynamic relative prime number generation, one at DSM, and another in distributed sensor node, are maintained with a standard time interval. Information is communicated from the sensors to the DSM only if encrypted with the P_i based secret key.

3.3. Sensor node processing power

Power consumption of the sensor node broadly divided into three domains: sensing, data processing and communication. The processing unit of a smart dust mote prototype is a 4 MHz Atmel AVR8535 micro-controller with 8 KB instruction flash memory, 512 bytes RAM and 512 bytes EEPROM [4]. TinyOS operating system is used on this processor, which has 3500 bytes OS code space and 4500 bytes available code space [5]. In [6], the authors mentioned that the Mica2 mote – based on the Atmel ATMega 128L microcontroller – takes roughly 3.2 nJ per instruction; Texas Instruments MSP430 microcontroller corresponds to an energy consumption of roughly 750 pJ per instruction and smart dust micro architecture designed in 0.25 µm technology system consumes 12 pJ per instruction. In [1,2], Walravens et al. proved that folded tree based processing is more efficient than other traditional processing techniques. The folded tree is 8 to 10 times more efficient that MSP430 in terms of energy and 2 to 3 times faster in terms of execution time. It also reduces the processing power to 80 uW or 8 pJ/cycle.

In our proposed architecture in Fig. 2, we generate prime numbers at both sensors and DSM. We adapted the folded tree based approach as it is suitable and capable of calculating the prime number after equal intervals of time in the range up to 10^7 within 18 milliseconds [3].

We also assume that deployed source nodes operate in two modes: trusted and untrusted. 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 systems [10]. TPM provides a cost effective way of "hardening" many 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 like key generation, store, and management in hardware. The detailed architecture is at [11]. We assume that the proposed prime number generation procedure *Prime*(P_i) and *secret key calculation* operate in the trusted mode.

4. Dynamic prime-number based security verification – DPBSV

We describe our DPBSV scheme for big sensing data streams using four independent components: system setup, handshaking, rekeying, and security verification. We refer readers to Table 4 for all notations used in describing our scheme. We have made a number of sensible a practical assumptions while characterizing our scheme. We describe those assumptions where necessary. We next describe four independent components in details.

4.1. DPBSV system setup

We assume that DSM has all deployed sensors' identities (IDs) and secret keys at the time of deployment because the network is fully untrusted. We use a number of key exchanges between the sensors and DSM at the start to ensure that session key establishment process is secured. Since we are transmitting key functions such as *KeyGen* to individual source sensors later, it is important that all potential attacks are considered while establishing the session key. We also assume that each sensor node S_i knows the identity of its DSM. Further, both DSM and sensors maintain a same secret key (i.e., k) for initial authentication process. In our scheme, we also assume that sensors never communicate between each other to reduce the communication overhead. The step wise secure authentication process shown in Fig. 4.

Step 1:

In the first step, a sensor sends $\{S_i, r\}$ to the DSM, where S_i is the sensor identity and r a pseudorandom number. If there are n numbers of sensors deployed in the area such as $S_1, S_2, S_3, \ldots, S_n$, S_i denotes the id of ith sensor.

Please cite this article in press as: D. Puthal et al., A dynamic prime number based efficient security mechanism for big sensing data streams, J. Comput. Syst. Sci. (2016), http://dx.doi.org/10.1016/j.jcss.2016.02.005

Table 4
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 key calculated by the sensor and DSM.
K_{SH^-}	Previous secret shared key maintain at DSM.
K/K'	Encrypted with sensor's secret key for user authentication.
$C/C'/C''$	Calculated hash value.
r	Pseudorandom 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.
k	Initial shared key for sensor and DSM for authentication.
j	Integrity checking interval.
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.
$\oplus$	Bitwise X-OR operation.
$\|\|$	Concatenation operation.
$DATA$	Fresh data at sensor before encryption.
$RetriveKey()$	Retrieve key from DSM database by knowing specific source.
$radomdKey()$	Randomly generate the key.

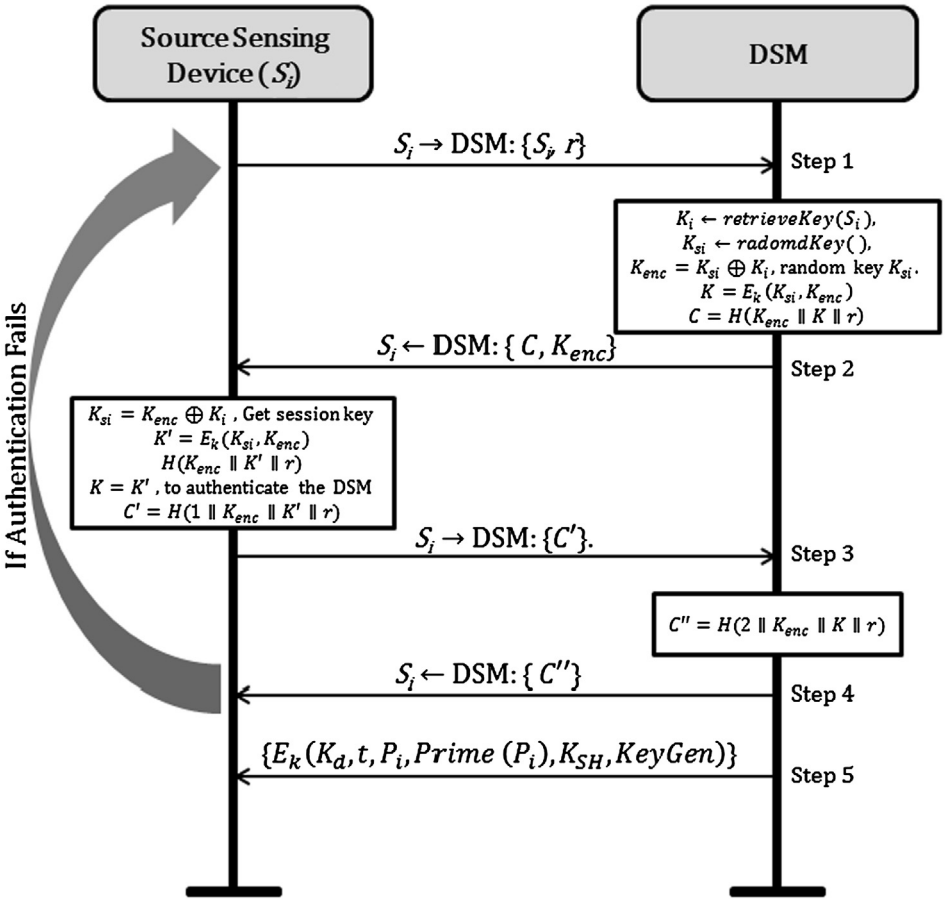


Fig. 4. Secure Authentication process between source sensing device and DSM, and Handshaking.

Please cite this article in press as: D. Puthal et al., A dynamic prime number based efficient security mechanism for big sensing data streams, J. Comput. Syst. Sci. (2016), http://dx.doi.org/10.1016/j.jcss.2016.02.005

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

Step 2:

When the DSM receives $\{S_i, r\}$ from a sensor S_i, it first retrieves S_i's secret key, i.e., $K_i \leftarrow RetriveKey(S_i)$. DSM then generates a random session key K_{si}. In order to share this with the corresponding senor (S_i), DSM generates a key using the session key and the corresponding sensor's private key (i.e., $K_{enc} = K_{si} \oplus K_i$). Then DSM encrypts the generated key with the shared key k (i.e., $K = E_k(K_{si}, K_{enc})$) and performs the hash function to generate C (i.e. $C = H(K_{enc} \parallel K \parallel r)$). Finally, DSM sends the value of C and K_{enc} to S_i. The complete computational steps is listed below.

$$\begin{aligned} &K_i \leftarrow RetrieveKey(S_i),\\ &K_{si} \leftarrow radomdKey(),\\ &K_{enc} = K_{si} \oplus K_i,\\ &K = E_k(K_{si}, K_{enc})\\ &C = H(K_{enc} \parallel K \parallel r) \end{aligned} \tag{1}$$

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

Step 3:

The corresponding sensor gets $\{C, K_{enc}\}$ from DSM and starts calculating its session key from K_{enc} based on its own secret key (i.e., $K_{si} = K_{enc} \oplus K_i$). The sensor finds out the value of K' based on the value of K_{si} and K_{enc} (i.e. $K' = E_k(K_{si}, K_{enc})$) by using the initial secret key k. It then gets the hash $H(K_{enc} \parallel K' \parallel r)$ from Equation (1) and checks whether or not it is equal to C. If the hashes are equal and $K = K'$, DSM is authenticated to the sensor S_i. 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.

$$\begin{aligned} &K_{si} = K_{enc} \oplus K_i.\\ &K' = E_k(K_{si}, K_{enc})\\ &H(K_{enc} \parallel K' \parallel r)\\ &K = K', \text{ to authenticate the DSM}\\ &C' = H(1 \parallel K_{enc} \parallel K' \parallel r) \end{aligned} \tag{2}$$

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

Step 4:

When the DSM receives C' from the sensor, it compares the value of with $H(1 \parallel K_{enc} \parallel K \parallel r)$, which is computed from Equation (2) to see whether they are equal. S_i is authenticated by DMS id the values are equal Otherwise, the protocol is terminated. After authentication of both parties, the DSM and sensors have the session key K_{si}. DSM sends $C'' = H(2 \parallel K_{enc} \parallel K \parallel r)$ to complete the protocol.

$$C'' = H(2 \parallel K_{enc} \parallel K \parallel r) \tag{3}$$

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

4.2. DPBSV handshaking

The individual session keys are established using the DPBSV system setup described in the earlier sub-section. Using the established session keys, the DSM sends its all properties (shows in step 5) to sensors $(S_1, S_2, S_2, \ldots, S_n)$. In general, if a larger prime number of secret shares is 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. Towards this, we have defined a new dynamic prime number generation function $Prime(P_i)$, which will be described later in Theorem 2. Our aim is to calculate the prime number on both sensor and DSM sides to reduce communication overhead and minimize the chances of disclosing the shared key. This is achieved by installing the same function at both sides as follows:

Step 5:

$Prime(P_i)$ computes the relative prime number on both sides with a time interval t. In the handshaking process, DSM transmits all its procedures to generate the key and prime number like $(K_d, t, P_i, Prime(P_i), K_{SH}, KeyGen)$ to individual sensors by encrypting with the initial shared key (k). These parameters and procedures are explained in a later section in details.

Please cite this article in press as: D. Puthal et al., A dynamic prime number based efficient security mechanism for big sensing data streams, J. Comput. Syst. Sci. (2016), http://dx.doi.org/10.1016/j.jcss.2016.02.005

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

In this step, DSM sends all the parameters and properties of *KeyGen* to source sensors. All of this transferred information is stored in trusted parts of sensor (e.g. TPM). Fig. 4 shows the handshaking after authentication between source sensor and DSM. It is important to note that once the function is compromised, the whole security protocol and system is also compromise. Hence, it is important to ensure that the sensors have trusted part built in.

4.3. DPBSV rekeying

We propose a novel rekeying concept by calculating prime numbers dynamically on both source sensors and DSM. Fig. 3 shows the synchronization of the shared key. In our scheme, a smaller size of the key makes the security verification faster. But we change the key very frequently in the DPBSV rekeying process to ensure that the protocol remains secure. If any types of damage happens at the source, the corresponding sensor is desynchronized with DSM. The source sensor follows Step 3 to reinitialize and synchronize with DSM. According to our assumption, we store all the secret information at a trusted part of the sensor. So the sensor can reinitialize the synchronization by sending its own identity to DSM. Once DSM authenticates the source sensor, it sends the current key and time of key generation. Authenticated sensors can update the next key by using the key generation process from a secure module of the sensor (TPM). In several situations, data blocks can arrive at DSM after rekeying process, those data blocks encrypted with 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 shared key then the DSM uses K_{SH^-} key for the security verification; otherwise, it follows the normal process. The shared key K_{SH^-} always initialize the with current K_{SH} before K_{SH} update.

Rekeying is often accomplished by running initial exchanges all over again. The following presents an alternative approach to rekeying and the corresponding analysis in terms of efficiency.

Step 6:

The above defined *DPBSV Handshaking* process makes sensors aware about the $Prime(P_i)$ and *KeyGen*. We now describe the complete secure data transmission and verification process using those functions and keys. As mentioned above, our scheme uses the synchronized dynamic prime number generation $Prime(P_i)$ on both sides, i.e., sensors and DSM as shown in Fig. 3. 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 given 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. One is 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 other part is for the authentication checking (i.e., $A_D = S_i \oplus K_{SH}$). The resulting data block is: $((DATA \oplus K_{SH} \oplus K_i) \parallel (S_i \oplus K_{SH}))$. The key generation and individual block encryption process listed are as follows.

$$\begin{aligned} K_{SH} &= H (E (P_i, K_d)) \\ I_D &= DATA \oplus K_{SH} \oplus K_i \\ A_D &= S_i \oplus K_{SH} \end{aligned} \tag{4}$$

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

4.4. DPBSV security verification

Security verification should be performed in real time (with minimal delay) based on the features of big data streams stated above. In the following step we perform the security verification of our proposed scheme. In this step, DSM verifies for authenticity in each individual data block and for integrity in specific selected data blocks. The aim is to maintain the end-to-end security of the proposed scheme.

Step 7:

The DSM verifies whether the data is modified or comes from an authenticated node. As DSM has the common initial shared key, it decrypts the complete block to find out the individual data blocks for the integrity and authenticity check. The DSM first checks for the authenticity in each data block A_D and checks for the integrity with random of interval data blocks I_D. This random value 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 not skip any 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 calculates/decrypts data and checks MAC for integrity check $DATA = I_D \oplus K_{SH} \oplus K_i$. All the security verification process from based on shared key from Equation (4). Fig. 5 shows the security verification with rekeying process.

Please cite this article in press as: D. Puthal et al., A dynamic prime number based efficient security mechanism for big sensing data streams, J. Comput. Syst. Sci. (2016), http://dx.doi.org/10.1016/j.jcss.2016.02.005

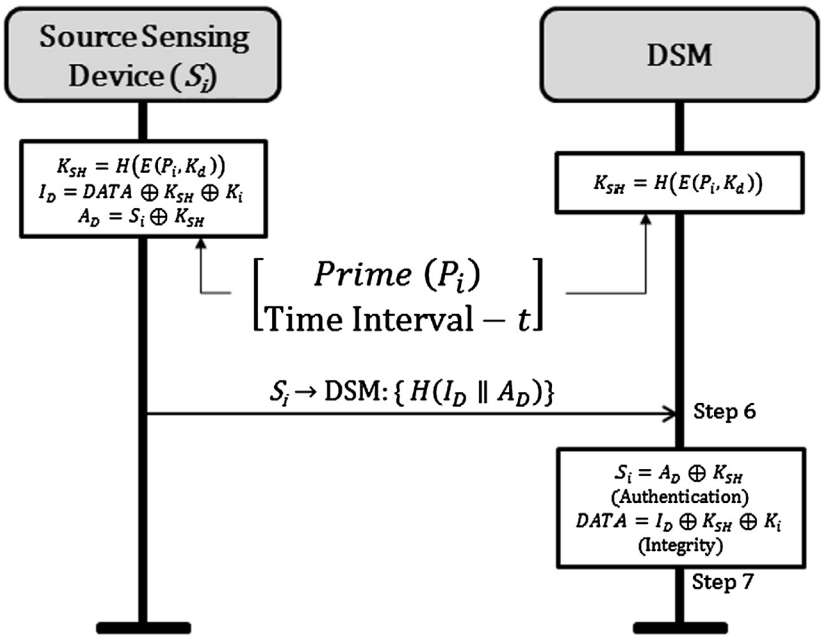


Fig. 5. Shared key update and security verification process.

$$S_i = A_D \oplus K_{SH}$$

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

The complete mechanism beginning from source sensing device and DSM authentication to handshaking, security verification mentioned in algorithmic format is shown in Algorithm 1. Algorithm 1 represents the description of the proposed mechanism in the stepwise process.

5. Security analysis of DPBSV

This section provides theoretical analysis of our scheme to show that the proposed scheme is safe against attacks on authenticity, confidentiality and integrity.

5.1. Assumptions

We have made a number of practical and realistic assumption in our scheme. In the following, we first describe those assumptions.

Assumption 1. In our scheme, the data that was encrypted by a symmetric-key algorithm cannot be decrypted by any parties, unless they have the session/shared key which was used to encrypt the data at the source (or sensor) side.

Assumption 2. DSM is fully trusted and no parties can access the DSM without proper authentication.

Assumption 3. Sensor's secret key, *Prime*(P_i) and *secret key calculation* procedures reside inside trusted parts of the sensor (like TPM) so that no one is authorized to access and manipulate them.

5.2. Threat model

We define our threat model which is similar to the most cryptological analyses to shared-key communication protocols as follows:

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 to send data in the data stream. The types of attacks possible in this category include impersonation attack, Sybil attack, and identity-based attacks [19].

Please cite this article in press as: D. Puthal et al., A dynamic prime number based efficient security mechanism for big sensing data streams, J. Comput. Syst. Sci. (2016), http://dx.doi.org/10.1016/j.jcss.2016.02.005

Algorithm 1 Security Framework for Big Sensing Data Stream.

Description Based on the dynamic prime number generation at both source sensor and DSMsides, the proposed security protocol for big sensing data streams works more efficiently without compromising security strength.

Input Prime generation process $Prime\,(P_i)$, key generation process *KeyGen*, sensor and DSM secret key, and session key K_{enc} for handshaking.

Output Successful security verification without any malicious attack and comparatively faster security verification than standard symmetric key solution (AES).

Step 1 DPBSV System setup

1.1 $S_i \rightarrow \text{DSM} : \{S_i, r\}$, ith sensor sends its random number with its identity.

1.2 $S_i \leftarrow \text{DSM} : \{C, K_{enc}\}$, DSM identifies the sensor and generates the session key for it. Then DSM encrypts and sends back to the ith sensor.

1.3 $S_i \rightarrow \text{DSM} : \{C'\}$, ith sensor identifies the DSM based on its own secret key. If sender is not authenticated then it starts authentication transaction.

1.4 $S_i \leftarrow \text{DSM} : \{C''\}$ DSM authenticates the last transaction and sends back to ith sensor with this format. Otherwise protocol terminates to start the new process.

Step 2 DPBSV Handshaking

DSM sends its properties to individual sensors based on their individual session key. It includes the prime number generation and time interval to generation etc.

2.1 $\text{DSM} \leftarrow S_i : \{E_k(K_d, t, P_i, Prime\,(P_i)\,, K_{SH}, KeyGen)\}$, for details refer Table 4.

Step 3 DPBSV Rekeying

Key updates on both source sensor and DSM and they are aware of the $Prime(P_i)$ and KeyGen. Sensors generate the shared key $K_{SH} = H\,(E\,(P_i, K_d))$ and each data block is associated with two different parts. One is encrypted i.e., $I_D = DATA \oplus K_{SH} \oplus K_i$ and another for authenticity checking i.e., $A_D = S_i \oplus K_{SH}$.

3.1 $S_i \rightarrow \text{DSM} : \{E_k\,(I_D \parallel A_D)\}$, these blocks for authentication, integration, and confidentiality checks.

Step 4 DPBSV Security Verification

The DSM checks for authenticity in each data block A_D and checks for the integrity with random interval data blocks I_D and random value is calculated based on the corresponding prime number i.e. $j = P_i \% 7$.

4.1 $S_i = A_D \oplus K_{SH}$

For the authenticity check, the DSM gets source ID. Once S_i obtained, the DSM checks source database and extracts corresponding secret key K_i for the integrity check according to the value of j.

4.2 $DATA = I_D \oplus K_{SH} \oplus K_i$

Given K_i, the DSM calculates/decrypts data and checks MAC for integrity check.

Definition 2 *(Attack on confidentiality).* A malicious attacker M_c is an unauthorized party who has the ability to access or view the unauthorized data stream before it reaches DSM. The types of attacks in this categories include phishing attack, packet sniffing, and dumpster driving [19].

Definition 3 *(Attack on integrity).* A malicious attacker M_i attack on integrity, which is an adversary capable of monitoring the data stream regularly and try to access and modify the data blocks before it reaches DSM. The types of attacks in this category includes salami attack, data diddling attacks, man in the middle attack and session hijacking attack [19].

5.3. Security proof

In this sub-section, we show that our scheme is safe against the threat model in Section 5.2 under the assumptions explained in Section 5.1. This is achieved through six theorems and corresponding proofs as follows.

Theorem 1. *The security is not compromised against the threat model by reducing the size of shared key (K_{SH}).*

Proof. We reduce the size of the prime number to make the key generation process faster and more efficient. 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. Low length of key also provides more security in a symmetric key algorithm because 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 [8]. By reducing the size of the prime number, we fixed the key length to 64-bit to make the security verification faster at DSM using the data from Table 5. From Table 5, a 64-bit symmetric key takes 3136e + 19 nanoseconds (more than a month), so we fixed interval time to generate prime number as a week (i.e. $t = 168$ hours). Dynamic shared key calculates based on the calculated prime number. Based on this calculation, we conclude that an attacker cannot calculate within the interval time t. We are changing the shared key without exchanging information between the sensors and DSM. Brute-force attack me be able to get the shared key once intruder have key length, but this possibilities also associate with 128-bit cryptographic solution. It confuses the malicious node those are listening the data flow continuously. The key has already been changed four times before an attacker knows the key and this knowledge is not known to the attackers. Which conclude that even we reduced the key size to 64 bit, we get the same security strength by changing the key in time interval t. □

Please cite this article in press as: D. Puthal et al., A dynamic prime number based efficient security mechanism for big sensing data streams, J. Comput. Syst. Sci. (2016), http://dx.doi.org/10.1016/j.jcss.2016.02.005

Table 5
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 nanoseconds)	1435.2	1e + 05	7.301e + 09	3136e + 19	5.7848e + 35

Algorithm 2 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 $m := 6k + 1$ // Next prime number.
4. If $m \geq 10^7$ then
5. $k := k/10^5$
6. GO TO: 3
7. If $S(m) = 1$ then // From Equation (5).
8. GO TO: 14 // If m is a prime number.
9. Set $m := 6k + 5$ // Next possible prime number.
10. If $S(m) = 1$ then // From Equation (5).
11. GO TO: 14 // If m is a prime number.
12. $k := \left\lfloor k^3 + \sqrt{k} \right\rfloor \ mod\ 17 + k$ // Relative number calculation.
13. GO TO: 3
14. $P_i = m$
15. Return (P_i) // Calculated new prime number.

It is important to note that a malicious attacker can find a key and decrypt the data if it can hold the data longer than the time interval t. Even in such situation, the scheme is safe from attacks on authentication and integrity, but not strong confidentiality. This means the scheme supports weak confidentiality. However, this scheme is equally safe to use in scenario where confidentiality of the data expired after certain time like in emergency management scenario (where the authenticity and integrity of the data is important). We plan to address this issue in our future work.

Theorem 2. *Relative prime number P_i calculated in Algorithm 2 synchronizes 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 this concept 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. Initially P_i is initialized by the DSM at DPBSV Handshaking process and the interval time is t seconds.

From Algorithm 2, 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 which itself is derived using P_i. The value of k is constant at source because it is calculated from the current prime number, which is initialized during *DPBSV Handshaking*. Since the value of k is the same on both sides, the procedure *Prime*(P_i) returns identical values. In Algorithm 2, the value of $S(m)$ is computed as follows.

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

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

$$S(x) = \frac{S_1(x) + S_2(x)}{2} \tag{5}$$

If $S(x) = 1$ from equation (5) then x is prime, otherwise x is not a prime.

The following procedure validates the above features

$$x \not\equiv 0\ mod\ i \forall 1 \leq i \leq x - 1, \text{ if } x \text{ is prime}$$

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

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

Same $S_2(x)$ is also 1 as shown in Equation (6) and then

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

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

Theorem 3. *Shared key (K_{SH}) is always synchronize between Source sensor (S_i) and DSM in DPBSV security mechanism.*

Proof. From our proposed mechanism K_{SH}– always initialize by the value of K_{SH}, before K_{SH} update. According to the shared key generation process (i.e. $K_{SH} = H(E(P_i, K_d))$), rekeying process need the value of dynamic prime number (P_i) and other than this value are constant. So computed value of K_{SH} is always same if rekeying process use same P_i, and from Theorem 2 we conclude that the value of P_i always same at S_i and DSM in time interval t. So there is always same shared key (K_{SH}) for S_i and DSM in the time interval. □

Theorem 4. *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*(P_i) and *KeyGen*. Considering the computational hardness of secure module (Assumption 3), 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 and process according to it, but M_a can introduce him/herself as the authenticated node to send its information. In our scheme, sensor (S_i) sends $((DATA \oplus K_{SH} \oplus K_i) \parallel (S_i \oplus K_{SH}))$, where the second part of the data block $(S_i \oplus K_{SH})$ is used for the authentication check. DSM decrypts this part of the data block for an 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. All required secured information for prime number and key generation procedure is stored at trusted parts of the sensor node (i.e., Assumption 3). According to Assumption 3, an attacker cannot get the information as discussed before. Hence, we conclude that *attacker* M_a cannot attack big data streams. □

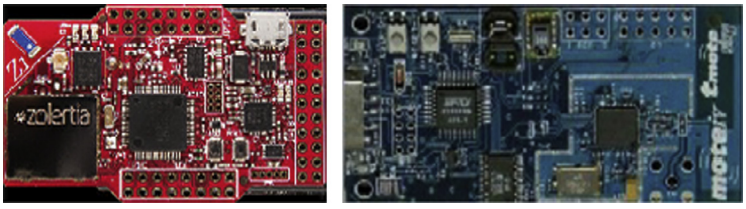
It is important to note that the proposed scheme avoids or drops the data blocks which are from malicious sources with minimum computation time by processing $(S_i \oplus K_{SH})$ only during authentication. This also addresses the attacks on availability, one of the key security features, by avoiding potential DDoS attack.

Theorem 5. *An attacker M_c cannot access or view the unauthorized data stream in our proposed DPBSV.*

Proof. It is clear from Algorithm 2 that a prime number *Prime*(P_i) is generated at sensors and DSM dynamically without any further communication. Shared secret key K_{SH} is computed using the generated prime number. Considering the Assumption 3, we know that M_c cannot get the secret information for P_i generation, K_i and *KeyGen* within the time frame. Following the Definition 2, we know that an attacker M_c can gain access to the shared key K_{SH} but no other information. In our scheme, a source sensor (S_i) sends data blocks in the format like $((DATA \oplus K_{SH} \oplus K_i) \parallel (S_i \oplus K_{SH}))$, where the first part of the data block $(DATA \oplus K_{SH} \oplus K_i)$ contains the original data. Getting the original data (*DATA*) is impossible from this because M_c does not have other information and at the same time shared key K_{SH} updates dynamically at equal intervals of time (t). As the data is protected and cannot be read within the time frame (i.e., before the update of shared key is occurred), we say that the proposed mechanism provide weak confidentiality. Though this weak confidentiality is acceptable in many applications, we plan to address this issue in our future work. □

Theorem 6. *An attacker M_i cannot read the shared key K_{SH} within the time interval t in DPBSV scheme.*

Please cite this article in press as: D. Puthal et al., A dynamic prime number based efficient security mechanism for big sensing data streams, J. Comput. Syst. Sci. (2016), http://dx.doi.org/10.1016/j.jcss.2016.02.005



(a) (b)

Fig. 6. The sensors used for experiment (a) Z1 low power sensor. (b) TmoteSky ultra low power sensor.

Proof. Following Definition 3, 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 calculates P_i randomly after time t and then uses the value P_i to generate K_{SH}. For more details on computation analysis, we refer readers to Theorem 1. □

5.4. Forward secrecy

As with other symmetric key procedures, shared keys used for encrypting communications are only used for certain periods of time (t) until the new prime number is generated. Thus, a previously used shared key or secret keying material is worthless to a malicious opponent even when a previously-used secret key is known to the attackers. This is one of the major advantages of frequent changing of the shared key. This is one of the reasons we did not choose symmetric key cryptography or an asymmetric-key encryption algorithm. However, if the attackers monitor and keep all data for a long period of time, they can find the keys to decrypt the data. This will break the confidentiality of the data, but the integrity and authenticity are maintained. However, with dynamic change of the keys and not knowing the internal process, it is always difficult to figure out which data to be kept for the potential confidentiality attacks.

6. Experiment and evaluation

The proposed DPBSV scheme is generic even though it is deployed in big sensing data streams in this paper. In order to evaluate the efficiency and effectiveness of the proposed scheme, even under adverse conditions, we observe each individual data blocks for authentication checks 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 in multiple simulation environments to validate that our security mechanism works perfectly in big sensing data streams. We first measured the performance of sensor nodes using COOJA in Contiki OS [37], then verified the security scheme using Scyther [38], and finally measured the efficiency of the scheme using JCE (Java Cryptographic Environment) [39]. We also checked the minimum buffer size required to process our proposed scheme and compared with the standard AES algorithm.

6.1. Sensor node performance

We experimented with the performance of the sensor in COOJA simulator in Contiki OS. We took the two most common types of sensor, i.e., Z1 and TmoteSky sensors, for our experiment and performance checking as shown in Fig. 6. In this experiment, we check the performance of sensors while computing or updating the shared key.

Z1 sensor nodes are produced by Zolertia, which is a low-power WSN module that is designed as a general purpose development platform for WSN researchers. It is designed for maximum backwards compatibility with the successful Tmote like family motes while improving the performance and maximum flexibility and expandability with regards to any combination of power-supplies, sensors and connectors. It supports the open source operating systems currently employed by the WSN community, like Contiki [37]. COOJA is a network simulator for Contiki, which provides real time sensor node features to simulate.

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

We successfully demonstrated in the COOJA Simulator that our key generation process works successfully in both types of sensors i.e. z1 sensor and TmoteSky sensor. These sensors support our security mechanism. The energy consumption during the key generation process is shown in Fig. 7. This shows the normal power consumption behavior for the key generation process. From this experiment we conclude that our proposed security verification mechanism DPBSV is supported by most common types of sensors and feasible for big sensing data streams.

Please cite this article in press as: D. Puthal et al., A dynamic prime number based efficient security mechanism for big sensing data streams, J. Comput. Syst. Sci. (2016), http://dx.doi.org/10.1016/j.jcss.2016.02.005

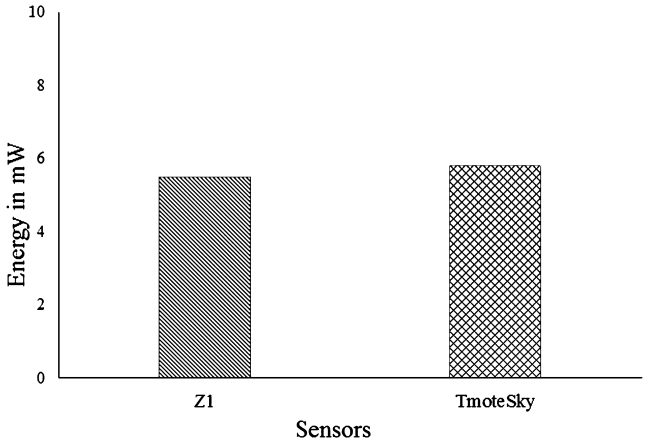


Fig. 7. Estimated power consumption during the key generation process.

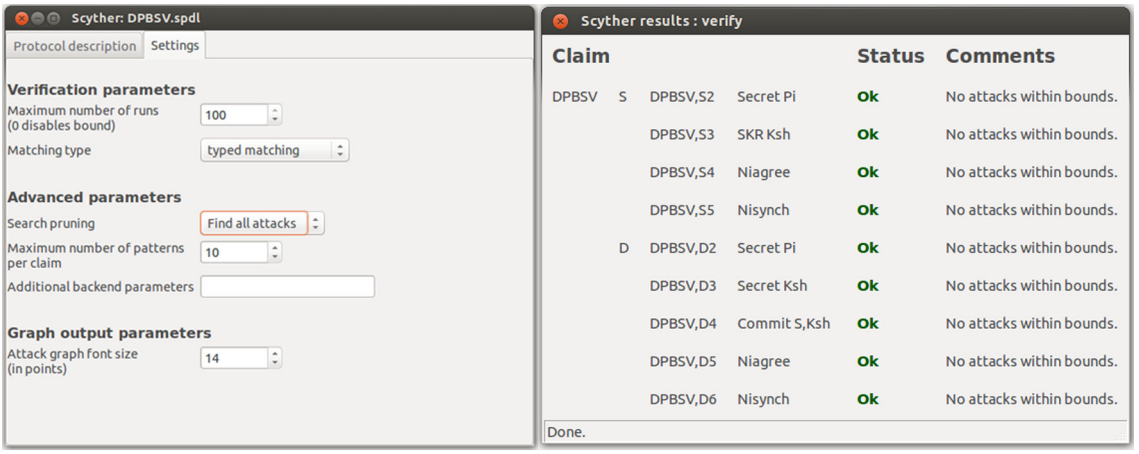


Fig. 8. Scyther simulation environment with parameters and result page of successful security verification at DSM.

6.2. Security verification

The scheme is written in the Scyther simulation environment using Security Protocol Description Language (.spdl). According to the features of Scyther, we define the role of D and S, where S is the sender (i.e. sensor nodes) and D is the recipient (i.e., DSM). In our scenario, D and S have all the required information that is exchanged during the handshake process. This enables D and S to update their own shared key. S sends the data packets to D and D performs the security verification. In our simulation, we introduce three types of attacks. The first type of attack is defined for the transmission between S and D (integrity), the second attack is defined where an adversary acquires the property of S and sends the attack data packets to D (authentication) and the third attack is defined adversary try to read the data within interval t (Confidentiality). In our experiments, we evaluated all packets at D (DSM) for security verification. We experimented with 100 numbers of runs for each claim to found out the number of attacks at D as shown in Fig. 8. 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 attacks, confidentiality attacks and authentication attacks as discussed above. In integrity attacks, an attacker can only observe encrypted data blocks/packets travelling on the network that contain information about sensed data as shown in Fig. 2. The attacker can perform a brute force attack on captured packets by systematically testing every possible key, and we assumed that he/she is able to determine when the attack is successful. In confidentiality attack, attacker continuously observe the data flow and try to read the data. In authentication attacks, an attacker can observe a 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 cases, the attacker can introduce an authenticated node and act as the original source node. In our concept, we are using trusted modules in sensors 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 at 64 bits (see Table 5). We used Scyther, an automatic security protocols verification tool, to verify our proposed mechanism.

Results: We did our simulation using variable numbers of data blocks in each run. Our experiment ranges from 100 to 1000 instances with 100 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 that information except the corresponding sensor. Without this information, attackers cannot authenticate encrypted data blocks. Hence, we did not find any attacks for authentication checks. For integrity attacks, it is hard to get the 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. As the shared key is changing with time interval t, attacker cannot read data stream within the time interval. Which conclude that our proposed mechanism provide weak confidentiality. Fig. 8 shows the result of security verification experiments in the Scyther environment. This shows that our scheme is secured from integrity, authentication and confidential (within the time interval t) attacks even after reduced key size. As we are updating the rekey process in equal interval of time, we found our scheme is secured with 64 bit key length. From the observations above, we can conclude that our proposed scheme is secure.

6.3. 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 sizes of data blocks. The results of our experiments are given below.

The Data Encryption Standard (DES) has been a standard symmetric key algorithm since 1977. However, it can be cracked quickly and inexpensively. In 2000, the Advanced Encryption Standard (AES) [31] 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 [31,32,35]. AES was introduced to replace the Triple DES (3DES) algorithm used for a good amount of time universally. 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 advanced encryption standard (AES), the standard symmetric key encryption algorithm [31,32]. Our scheme efficiency is 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), and 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 methods. We experimented with many-to-one communication. All sensor nodes communicate to the single node (DSM). All sensors have similar properties whereas the destination node has 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 the DSM node. Our experimental results are shown in Fig. 9; the result validates the theoretical analysis presented in Section 5.

Results: The performance of our scheme is better than the standard AES algorithm when different sizes of data blocks are considered. Fig. 9 shows the processing time of the proposed DPBSV scheme in comparison with base 128-bit AES and 256-bit AES for different sizes of the data block. The performance comparison shows that our proposed scheme is efficient and faster than the baseline AES protocols.

We calculated the time taken for DPBSV encryption and decryption in AMD K7-700 MHz processor and compare with standard AES-128 bit algorithm [33]. Based on our calculation DPBSV takes 3.2 microseconds and AES (128-bit) 35.8 microseconds for encryption, whereas DPBSV takes 3.3 microseconds and AES (128-bit) 36 microseconds for decryption.

6.4. Required buffer size

Experiment model: We evaluated the required buffer size for DSM by using the MATLAB as the simulation tool [40]. The buffer size is computed using the performance time shown in Fig. 9 (i.e. security verification time). We calculated the minimum required buffer size for DPBSV for high speed input data stream (i.e. MB/S). The security verification time is measured with input data size in Byte, and we scale the data rate to MB/S for buffer computation. We compared our scheme with the standard 128-bit AES and 256-bit AES. We calculated the minimum buffer size required to process security verification at DSM with various data rate starts from 50 to 200 MB/S with 50 MB/S interval.

Results: The performance of our scheme is better than the standard AES algorithm. Fig. 10 shows the minimum buffer size required to perform security verification at DSM for the proposed DPBSV scheme. We also show the compare the results with base line 128-bit AES and 256-bit AES. The performance comparison shows that our proposed scheme is efficient and required less buffer to perform security verification than the baseline AES protocols.

From the above experiments, we conclude that our proposed DPBSV scheme is secured (from authenticity, confidentiality and integrity attacks), and efficient (compare to standard symmetric algorithms such as 128-bit AES and 256-bit AES). The proposed scheme also needs less buffer than the baseline methods to perform security verification.

Please cite this article in press as: D. Puthal et al., A dynamic prime number based efficient security mechanism for big sensing data streams, J. Comput. Syst. Sci. (2016), http://dx.doi.org/10.1016/j.jcss.2016.02.005

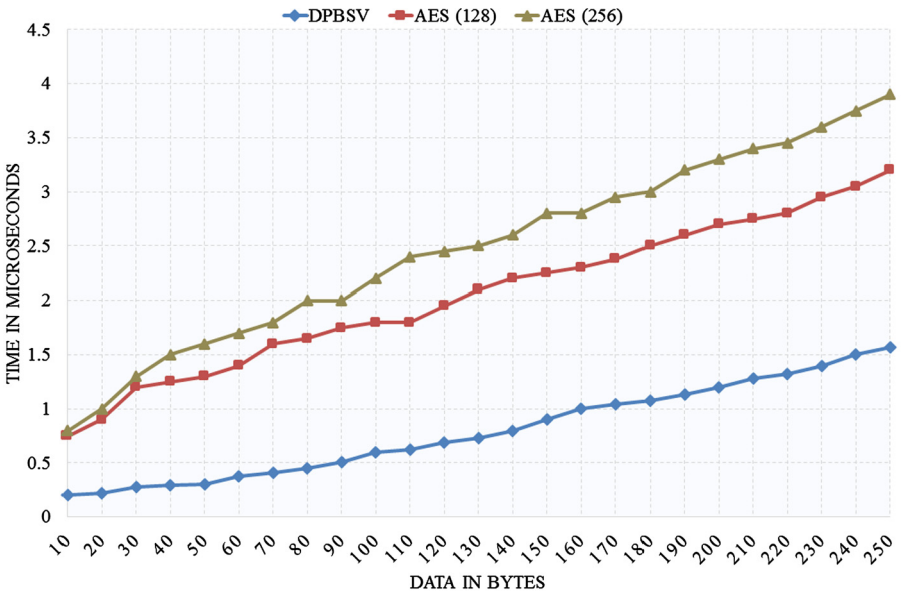


Fig. 9. Performance of our scheme compared in efficiency to 128 bit AES and 256 bit AES.

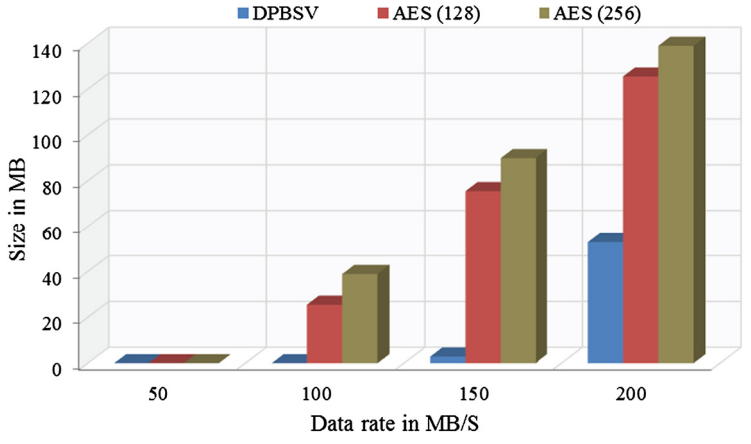


Fig. 10. Performance comparison of minimum buffer size required to process the security verification with various data rates to DSM.

7. 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 streams. Our scheme has been designed based on 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 processing time, required less buffer for processing and prevented malicious attacks on authenticity, confidentiality 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 for rekeying and decreases the communication overhead. DSM implement before stream data processing as shown in our main architecture diagram. Several applications (e.g. emergency management and event detection etc.) need to discard unwanted data and get original data for stream data analysis. Proposed security verification scheme (i.e. DPBSV) perform in near real time to synchronize with the performance of stream processing engine. Our aim is not to degrade the performance of stream processing such as Hadoop, S4, and Spark etc by verifying security on-the-fly. We plan to pursue a number of research avenues in future. The foremost is to perform a comparative study of our work with other symmetric key cryptographic techniques such as RC_5, RC_6. We will further investigate new strategies to improve the efficiency of symmetric-key encryption towards more efficient security-aware big data streams. We are also planning to investigate using the technique to develop a moving target defense strategy for the Internet of Things.

Please cite this article in press as: D. Puthal et al., A dynamic prime number based efficient security mechanism for big sensing data streams, J. Comput. Syst. Sci. (2016), http://dx.doi.org/10.1016/j.jcss.2016.02.005

Acknowledgment

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

References

[1] C. Walravens, W. Dehaene, Design of a low-energy data processing architecture for WSN nodes, in: Proceedings of the Conference on Design, Automation and Test in Europe, March 2012, pp. 570–573.
[2] C. Walravens, W. Dehaene, Low-power digital signal processor architecture for wireless sensor nodes, IEEE Trans. Very Large Scale Integr. (VLSI) Syst. 22 (2) (2014) 313–321.
[3] I. Kaddoura, S. Abdul-Nabi, On formula to compute primes and the nth prime, Appl. Math. Sci. 6 (76) (2012) 3751–3757.
[4] A. Perrig, R. Szewczyk, J. Tygar, V. Wen, D.E. Culler, SPINS: security protocols for sensor networks, in: Proceedings of ACM MobiCom'01, 2001, pp. 189–199.
[5] I. Akyildiz, W. Su, Y. Sankarasubramaniam, E. Cayirci, Wireless sensor networks: a survey, Comput. Netw. 38 (4) (2002) 393–422.
[6] M. Hempstead, M.J. Lyons, D. Brooks, G. Wei, Survey of hardware systems for wireless sensor networks, J. Low Power Electron. 4 (1) (2008) 11–20.
[7] J. Burke, J. McDonald, T. Austin, Architectural support for fast symmetric-key cryptography, ACM SIGOPS Oper. Syst. Rev. 34 (5) (2000) 178–189.
[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] TCG Trusted Platform Module (TPM) specification, https://www.trustedcomputinggroup.org/specs/tpm/ (accessed on: 04.08.2014).
[11] S. Nepal, J. Zic, D. Liu, J. Jang, A mobile and portable trusted computing platform, EURASIP J. Wirel. Commun. Netw. 2011 (1) (2011) 1–19.
[12] V. Gulisano, R. Jimenez-Peris, M. Patino-Martinez, C. Soriente, P. Valduriez, Streamcloud: an elastic and scalable data streaming system, IEEE Trans. Parallel Distrib. Syst. 23 (12) (2012) 2351–2365.
[13] N. Tatbul, U. Çetintemel, S.B. Zdonik, Staying fit: efficient load shedding techniques for distributed stream processing, in: Proceedings of International Conference on Very Large Data Bases, VLDB, 2007, pp. 159–170.
[14] V. Gulisano, R. Jimenez-Peris, M. Patino-Martinez, P. Valduriez, Streamcloud: a large scale data streaming system, in: Proceedings of 30th International Conference on Distributed Computing Systems, ICDCS, 2010, pp. 126–137.
[15] A. Arasu, B. Babcock, S. Babu, M. Datar, K. Ito, I. Nishizawa, J. Rosenstein, J. Widom, STREAM: the stanford stream data manager (demonstration description), in: Proceedings of the 2003 ACM SIGMOD International Conference on Management of Data, 2003, p. 665.
[16] D. Carney, U. Çetintemel, M. Cherniack, C. Convey, S. Lee, G. Seidman, M. Stonebraker, N. Tatbul, S. Zdonik, Monitoring streams: a new class of data management applications, in: Proceedings of the International Conference on Very Large Data Bases, 2002, pp. 215–226.
[17] D.J. Abadi, D. Carney, U. Çetintemel, M. Cherniack, C. Convey, S. Lee, M. Stonebraker, N. Tatbul, S. Zdonik, Aurora: a new model and architecture for data stream management, VLDB J. 12 (2) (2003) 120–139.
[18] S. Chandrasekaran, O. Cooper, A. Deshpande, M. Franklin, J.M. Hellerstein, W. Hong, S. Krishnamurthy, S.R. Madden, F. Reiss, M.A. Shah, TelegraphCQ: continuous dataflow processing, in: Proceedings of ACM SIGMOD International Conference on Management of Data, 2003, p. 668.
[19] D. Puthal, B. Sahoo, Secure data collection & critical data transmission in mobile sink WSN, in: Secure and Energy Efficient Data Collection Technique, LAP Lambert Academic Pubilishing, Germany, ISBN 978-3-659-16846-8, 2012.
[20] M. Stonebraker, U. Ç etintemel, S.B. Zdonik, The 8 requirements of real-time stream processing, SIGMOD Rec. 34 (4) (2005) 42–47.
[21] H. Demirkan, D. Delen, Leveraging the capabilities of service-oriented decision support systems: putting analytics and big data in cloud, Decis. Support Syst. 55 (1) (2013) 412–421.
[22] B. Albert, Mining big data in real time, Informatica 37 (1) (2013) 15–20.
[23] R.L. Collins, L.P. Carloni, Flexible filters: load balancing through backpressure for stream programs, in: Proceedings of International Conference on Embedded Software, EMSOFT, 2009, pp. 205–214.
[24] J.M. Tien, Big data: unleashing information, J. Syst. Sci. Syst. Eng. 22 (2) (2013) 127–151.
[25] M. Dayarathna, S. Toyotaro, Automatic optimization of stream programs via source program operator graph transformations, Distrib. Parallel Databases 31 (4) (2013) 543–599.
[26] D. Zissis, D. Lekkas, Addressing cloud computing security issues, Future Gener. Comput. Syst. 28 (3) (2012) 583–592.
[27] C. Liu, X. Zhang, C. Yang, J. Chen, CCBKE-session key negotiation for fast and secure scheduling of scientific applications in cloud computing, Future Gener. Comput. Syst. 29 (5) (2013) 1300–1308.
[28] A. Boldyreva, M. Fischlin, A. Palacio, B. Warinschi, A closer look at PKI: security and efficiency, in: Proceedings of the 10th International Conference on Practice and Theory in Public-Key Cryptography, PKC'07, 2007, pp. 458–475.
[29] K. Park, S. Lim, K. Park, Computationally efficient PKI-based single sign-on protocol, PKASSO for mobile devices, IEEE Trans. Comput. 57 (6) (2008) 821–834.
[30] M. Benantar, R. Jr, M. Rathi, Method and system for maintaining client server security associations in a distributed computing system, U.S. 6141758, issued October 31, 2000.
[31] PUB, NIST FIPS, 197: advanced encryption standard (AES), in: Fed. Inf. Process. Stand. Publ., vol. 197, 2001, 441-0311.
[32] S. Heron, Advanced Encryption Standard (AES), Netw. Secur. 2009 (12) (2009) 8–12.
[33] N. Penchalaiah, R. Seshadri, Effective comparison and evaluation of DES and rijndael algorithm (AES), Int. J. Comput. Sci. Eng. 2 (5) (2010) 1641–1645.
[34] R. Metcalfe, D. Boggs, Ethernet: distributed packet switching for local computer networks, Commun. ACM 19 (7) (1976) 395–404.
[35] J. Aemen, V. Rijmen, The Design of Rijndael: AES-the Advanced Encryption Standard, Springer, 2002.
[36] B. Kandukuri, V. Paturi, A. Rakshit, Cloud security issues, in: Proceedings of IEEE International Conference on Services Computing, SCC'09, 2009, pp. 517–520.
[37] Contiki operating system official website, http://www.contiki-os.org/ (accessed on: 04.08.2014).
[38] Scyther, [Online] http://www.cs.ox.ac.uk/people/cas.cremers/scyther/ (accessed on: 04.08.2014).
[39] M. Pistoia, N. Nagaratnam, L. Koved, A. Nadalin, Enterprise Java 2 Security: Building Secure and Robust J2EE Applications, Addison Wesley Longman Publishing Co., Inc., 2004.
[40] Matlab, [Online] http://au.mathworks.com/products/matlab/ (accessed on: 04.08.2014).
[41] D. Puthal, B. Sahoo, S. Mishra, S. Swain, Cloud computing features, issues and challenges: a big picture, in: Proceedings of International Conference on Computational Intelligence & Networks, CINE, 2015, pp. 116–123.
[42] R. Ranjan, Streaming big data processing in datacenter clouds, IEEE Cloud Comput. 1 (1) (2014) 78–83.

Please cite this article in press as: D. Puthal et al., A dynamic prime number based efficient security mechanism for big sensing data streams, J. Comput. Syst. Sci. (2016), http://dx.doi.org/10.1016/j.jcss.2016.02.005

[43] R. Nehme, H. Lim, E. Bertino, E. Rundensteiner, StreamShield: a stream-centric approach towards security and privacy in data stream environments, in: Proceedings of the ACM SIGMOD International Conference on Management of Data, 2009, pp. 1027–1030.
[44] R. Nehme, H. Lim, E. Bertino FENCE, Continuous access control enforcement in dynamic data stream environments, in: Proceedings of the Third ACM Conference on Data and Application Security and Privacy, 2013, pp. 243–254.
[45] J. Cao, T. Kister, S. Xiang, B. Malhotra, W. Tan, K. Tan, S. Bressan, Assist: access controlled ship identification streams, in: Transactions on Large-Scale Data-and Knowledge-Centered Systems XI, 2013, pp. 1–25.
[46] X. Wang, W. Cheng, P. Mohapatra, T. Abdelzaher, Artsense: anonymous reputation and trust in participatory sensing, in: Proceedings of IEEE Conference on Computer Communications, INFOCOM, 2013, pp. 2517–2525.
[47] D. Puthal, S. Nepal, R. Ranjan, J. Chen DPBSV, An efficient and secure scheme for big sensing data stream, in: Proceedings of 14th IEEE International Conference on Trust, Security and Privacy in Computing and Communications, IEEE TrustCom-15, 2015, pp. 246–253.
[48] M.A. Jan, P Nanda, X. He, R.P. Liu, A sybil attack detection scheme for a centralized clustering-based hierarchical network, in: Proceedings of 14th IEEE International Conference on Trust, Security and Privacy in Computing and Communications, IEEE TrustCom-15, 2015, pp. 318–325.
[49] M.A. Jan, P. Nanda, X. He, Z. Tan, R.P. Liu, A robust authentication scheme for observing resources in the Internet of things environment, in: Proceedings of 13th International Conference on Trust, Security and Privacy in Computing and Communications, TrustCom, 2014, pp. 205–211.
[50] L. Wang, D. Chen, W. Liu, Y. Ma, Y. Wu, Z. Deng, DDDAS-based parallel simulation of threat management for urban water distribution systems, Comput. Sci. Eng. 16 (1) (2014) 8–17.
[51] L. Wang, D. Chen, Y. Hu, Y. Ma, J. Wang, Towards enabling cyberinfrastructure as a service in clouds, Comput. Electr. Eng. 39 (1) (2013) 3–14.

Please cite this article in press as: D. Puthal et al., A dynamic prime number based efficient security mechanism for big sensing data streams, J. Comput. Syst. Sci. (2016), http://dx.doi.org/10.1016/j.jcss.2016.02.005