Secure Management and Processing of Metered Data in the Cloud

Bokun Zhang, Pirathayini Srikantha(✉), and Deepa Kundur

University of Toronto, Toronto, ON, Canada
{bokun.zhang,pirathayini.srikantha,dkundur}@mail.utoronto.ca

Abstract. The recent cyber-physical integration in the electric power grid provides unprecedented insights and management capabilities to the grid operator. Devices such as smart meters have been widely deployed in urban cities and produce granular user consumption information that can be used for many useful applications such as demand response. Efficiently managing and processing this expansive sensitive data in a secure manner is a challenging task. We address this in this paper by leveraging on light-weight encryption and the cloud services. Results indicate that our proposed solution is economical and eliminates many of the issues associated with relying on third party services for data management.

Keywords: Cloud · Homomorphic encryption · Security · Demand response

1 Introduction

The advanced metering infrastructure (AMI), a component of an Electric Power Utility (EPU), is comprised of smart meters equipped with bi-directional communication capabilities [1]. Smart meters residing at the local premises of thousands of energy consumers generate and transmit local energy consumption data to the EPU at a daily basis. This process generates vast amounts of data which is typically used by the EPU for efficient and convenient billing. As many valuable insights can be drawn from this data, the EPU and many third party solution providers can utilize this information for many other useful applications. One particular example of such an application is real-time demand response. Statistics such as average energy consumption extracted from this metered data can be used in real-time demand response (DR) for reducing peak aggregate power consumption in the system. Real-time applications such as DR require these statistics from metering data generated at high frequencies [2]. Hence, an extremely efficient data management system equipped with significant storage and computational capabilities is imperative to enable these real-time applications.

Managing and performing computations on data at such a large scale in an economical manner is not an trivial task. The cloud provides vast amount of storage and powerful computational resources on-demand with no need for

A. Leon-Garcia et al. (Eds.): Smart City 2015, LNICST 166, pp. 362–373, 2016.
DOI: 10.1007/978-3-319-33681-7_30

advanced commitment. This inherent flexibility renders the cloud to be a practical and viable option for meter data management [3]. However, the cloud has many security issues that must be carefully dealt with by the EPU due to the revealing nature of meter data. Techniques based on energy signatures can be used to infer the daily activities of a consumer from his or her metered data [4].

As the EPU will not be able to ascertain the security statuses of physical servers used in the cloud, sensitive data can be exposed to data leakage and loss of integrity due to unresolved vulnerabilities in the cloud environment [7]. In order to overcome these issues, it is necessary to apply additional processing to ensure that this data cannot be compromised while residing on a third party infrastructure like the cloud. One approach will be to store data in encrypted form at all times on the cloud. However, typical encryption techniques impose difficulties with being able to leverage the vast amount of computational resources available in the cloud for data analytics.

In the existing literature, many proposals discuss potential solutions for cloud security issues. Reference [13] suggests processing sensitive data via homomorphic encryption technology to promote security. However, this solution supports limited functions, such as data aggregation. Reference [14,15] propose schemes that utilize the Paillier scheme to obtain sums of meter measurements at the neighborhood level in order to address privacy issues. Again, this imposes limitations on the processing capabilities on encrypted data. In [16], Goh's encryption scheme is leveraged to perform homomorphic operations for statistical data analysis in smart grid system. This is associated with possible efficiency and functionality issues. All of these results indicate that although homomorphic encryption consists of many attractive properties for ensuring security and integrity, there are unresolved issues associated with efficiency and functionality.

In this paper, we present a cloud-based solution that focuses on the use of Paillier encryption for storing and processing meter data in a secure and an efficient manner while also addressing limitations in basic computational features supported by this encryption scheme. Simulations are used to infer the efficiencies of these three methods. Then, we address potential issues with metered data synchronization in the cloud due to latencies in data transmission.

2 Homomorphic Encryption

In this section, we provide a brief background on homomorphic encryption. Homomorphic encryption allows basic mathematical operations to be performed directly on cipher-text. Even though these operations are applied on cipher-text, these are also reflected in the plain-text [6]. For example, suppose that an addition operation on two cipher-text units which are the encrypted versions of two numbers, say 3 and 4. When the resulting cipher-text is decrypted, the value 7 will be obtained. This is extremely advantageous as there is no need to decrypt cipher-text for computations that involve elementary mathematical operations. This homomorphic property can be expressed as:

$$f(Encode(m)) = Encode(f(m)) \tag{1}$$

where $Encode(x)$ is the homomorphic encryption function, $Decode(x)$ is the homomorphic decryption function, m is the plain-text and $f(.)$ represents the function that performs a mathematical operation on *(.)*.

The various types of mathematical operations $f(.)$ that allow the homomorphic property in Eq. 1 to hold depends on the type of homomorphic encryption utilized. There are two main types of homomorphic encryption schemes widely used in the literature. These are the fully homomorphic encryption (FHE) and Paillier encryption (PE) schemes.

2.1 Fully Homomorphic Encryption

FHE scheme supports a broad range of mathematical operations that include division, etc. Although this scheme supports all of the mathematical operations required for our purposes, major issues that render it unsuitable for real-time large data sets include: slow processing speed, complex cipher-text noise reduction functions and tremendous memory storage space [6,8,9,11]. FHE supports many additional features that are typically not necessary for energy applications.

2.2 Paillier Encryption Scheme

The PE scheme is a light-weight factoring based, asymmetrical encryption technology used typically in electronic money and voting system applications [10]. It supports homomorphic addition and a relative homomorphic multiplication property. These PE homomorphic operations are expressed as follows:

$$\text{Addition: } Add(c1,\ c2) = Encode(m1 + m2,\ pk) \tag{2}$$

$$\text{Multiplication: } Mult(K,\ c1) = Encode(K * m1,\ pk) \tag{3}$$

where pk is the public key, $m1$ and $m2$ are plain-text values, $c1$ and $c2$ are cipher-text values corresponding to the plain-text, and K is an integer. The relative multiplication property, as it is evident from 3, is restrictive. One operand must be an integer (i.e. both operands cannot be cipher-text values and the non-ciphertext operand cannot be a fraction). Hence, it is not possible to perform averaging, which is a very common computation in energy applications such as DR, using this relative multiplication operation. As the PE scheme is light-weight, it is well-suited for processing vast amounts of data in the cloud. In order to overcome the limitation of not being able to use the division operation, in Sect. 3, we propose three methods that enable division possible. These can be applied to obtain general statistical metrics such as averaging via the basic operations supported by the PE scheme.

3 Extended Cipher-Text Operations for PE

Here, we propose the following three methods that extend aggregation operations in PE: direct send back, data amplification-grouping and multiplier amplification. Although our focus is on averaging operations, these methods can be easily

extended to other operations such as division on cipher-text, etc. Our primary goal is to design a light-weight scheme for averaging computations on large data sets in a quick and efficient manner. To assess this, we also include comprehensive results that compare key metrics such as latency and complexity of our proposed extensions.

3.1 Direct Send Back

For an averaging operation, many additions must be performed and only one division operation is required. The direct send back method involves a two step process. First, the *Add* operation in the PE scheme can be applied within the cloud to the cipher text corresponding to the data set of interest. Then information including this result and the number of entries that have been added can be sent to the client. The client can then decrypt the cipher-text and directly apply division operation to obtain the average value. Although this is a viable option as data is not decrypted on the cloud, there are two main issues associated with this approach. Firstly, the computational process is partially completed by the client and this is not desirable as division is a computationally intensive task. Cloud resources can more effectively perform this operation than the client. Also, the aggregated value can be very large and communicating this to the client can cause issues such as transmission delays. Moreover, there are security concerns with this approach. Revealing the size of the data set to the client can expose demographic information.

3.2 Data Amplification and Grouping

This is our second proposal which utilizes the modular inverse concept in lieu of the division operation for averaging. This allows us to use the relative multiplication and modular operations supported by PE to perform division indirectly. However, modular inverses are associated with some limitations and we show that with additional processing on data via grouping and amplification, these can be overcome.

As mentioned in the background section, PE does not support direct division or relative multiplication of a cipher-text with a float value. Suppose that b and N are relatively prime. Then, the modular inverse $modInv(b)$ of b with respect to N satisfies $b * modInv(b) = 1(mod\ N)$. $modInv(b)$ is an integer. A division operation $\frac{a}{b}$ can be effectively replaced by the modular inverse as follows if certain conditions are met:

$$\frac{a}{b} \equiv\ a\ modInv(b)\ (\text{mod}\ N)$$

The first condition is that $N \geq \frac{a}{b}$ and we satisfy this condition by setting N to be the public key pk used to encrypt the data set (as $pk \gg \frac{a}{b}$). The next requirement is the existence of the modular inverse. The modular inverse of b modulo pk exists if and only if

$$gcd(b,\ pk) = 1$$

In PE, the public key pk is the product of two non-even prime numbers. If b is an even number, then the above condition is satisfied for $N = pk$. However, it is not possible to force b to be an even number as for our averaging application, b represents the total number of data values composing the data set that we are averaging. In order to resolve this issue, we group the data values in the data set so that the number of data values composing each subset is even. This guarantees the existence of the modular inverses required to compute the averages of data values in these groups. Next, we discuss how we propose to divide the data set into groups containing even number of data values.

Let b be the total number of data values composing the data set of interest. b can be represented as a binary number as follows:

$$b = \sum_{i=0}^{\lfloor log_2 b \rfloor} c_i * 2^i$$

where $c_i \in \{0, 1\}$ is the bit corresponding to the i^{th} significant digit of the binary representation of b. The data set originally containing b data values can be considered to be a composition of several smaller groups where the i^{th} group (which exists if $c_i = 1$) consists of 2^i data values. In Fig. 1, an example of this data grouping is illustrated for a data set comprising of 1173 elements.

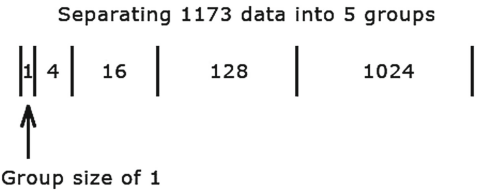

Fig. 1. Grouping of data values in a data set

The averaging of each group i can be applied directly on the cipher-text by first summing all cipher-text values in this group to obtain S_i. Before applying $mult(S_i, modInv(2^i))$, it is necessary to check if S_i is exactly divisible by 2^i. Otherwise, the modular division will fail to produce the correct value. In order to ensure that this condition always holds, an amplification is applied to S_i by multiplying this value with 10^i so that now $S_i * 10^i/2^i$ will always be exact. $A_i = S_i * 10^i/2^i$ can now be obtained from the cipher-text values in the cloud via these homomorphic encryption operations: $A_i = Mult(Mult(S_i,\ 10^i), modInv(2^i))mod(pk)$. Averages computed for all groups are compiled into a list $L = \{[A_0, Encode(0), c_0], \ldots, [A_i, Encode(i), c_0], \ldots, [A_{\lfloor log_2 b \rfloor}, Encode(\lfloor log_2 b \rfloor), c_0]\}$. This list is then sent to the client. The client will decrypt L and perform a weighted average $\sum_{i=0}^{\lfloor log_2 b \rfloor} \frac{c_i * A_0}{5^i}$ to recover the final average.

With this particular proposal, we have demonstrated how the traditional division can be replaced with the modular division after some additional processing. Bulk of resource intensive operations are performed on the cloud. However, there still exists the issue of exposing the total number of value points in the data set to the client as the binary representation of the size of the data set is appended to L.

3.3 Multiplier Amplification

Another simpler alternative that does not require exposing the total number of data points in the set is presented next. Suppose the sum of all data points in the set is S, then the average of this set is $\frac{S}{b}$. Amplifying $\frac{1}{b}$ by 10^n, setting $P_n = \lfloor \frac{1}{b} * 10^n \rfloor$ preserves a precision of n digits. Now, the relative multiplication operation can be applied to S and P_n which results in T_n. The value pair $[T_n, n]$ is then sent to the client who can now recover the final average by applying $decode(T_n, n)$ and dividing this value by 10^n. Although this is much more simpler and maintains privacy, the cipher-text transmitted to the client will typically represent a large value.

3.4 Time Latencies of PE

In order to establish the performance of each of the above methods, the homomorphic encryption project (THEP), made available by Pwnhome Research [12] is utilized in our test cases implemented in Java. In our algorithm implementations, we use Java's building-in BigInteger class to store and process the plain and cipher text. Our test client consists of a CPU i5-4200U that operates at 1.6GHz. Before characterizing the performance of our proposed algorithms, we first present the latencies for various PE functions.

Key Generation. First, we assess the time required to generate a pair of keys of constant length. This generation process is repeated 20 times and the average of this time values are presented in Table 1.

Table 1. Average time for key generation

Key length	256 bit	800 bit	1024 bit
Time consumption	125 ms	188 ms	216 ms

It can be concluded from the above set of results that the time complexity for key generation increases with the size of the key.

Encryption Speed. Next, we assess the complexity of encrypting data with PE. Since a longer key translates to greater security, we use a key length of 1024 bit for testing purposes in the remainder of this section. In the set of

Table 2. Encryption speed of data

Message length	32 bit	64 bit	256 bit	512 bit
Time consumption	13.0 ms	14.7 ms	15.9 ms	17.6 ms

results in Table 2, 100 randomly generated plain-text data of the same size are encrypted.

It is evident from the above set of results that even though the message size is doubling, the time required for encryption increases only slightly.

Decryption Speed. Here, decryption is applied on 100 randomly generated plain-text messages as in the above. The average time required for decryption is presented in Table 3.

Table 3. Decryption speed of data

Message length	32 bit	64 bit	256 bit	512 bit
Time consumption	12.3 ms	12.1 ms	12.5 ms	11.4 ms

The time required for decryption remains almost constant regardless of the size of the data being decrypted.

Proposed Algorithm. Finally, we present the time complexity of applying our three proposed algorithms on the client side for computing the average of a data set containing 750,000,000 values where each is of length 256 bits. This data set consists of 12 groups where the i^{th} group is of size 2^i. The average time required for the client to recover the final averaged value is listed in Table 4.

Table 4. Client side latency for recovering final average

Method	Direct send back	Data grouping and amplification	Multiplier amplification
Time consumption	24.46 ms	156.08 ms	24.37 ms

It is clear that the data grouping and amplification method results in the most computational latency on the client side. The direct send back and multiplier amplification methods have similar performances. Although the data grouping and amplification method makes an interesting connection to modular mathematics and is tailored around the PE scheme, it requires excessive resources at the client side. As our goal is to minimize resource allocation on the client side, this will not be a suitable algorithm. Since, the multiplier amplification is the least revealing and the fastest, it is most suited for our purpose.

4 Data Transmission Synchronization

Data generated from the metering infrastructure is encrypted via PE at the smart meter and is transmitted to the cloud in the encrypted form. This start-to-end encryption has been proposed in works such as [14–16]. For applications such as real-time demand response, averaging operations are performed on meter data transmitted in the order of minutes [2]. Communication latencies can cause this encrypted data to be unordered. Hence, when these arrive at the cloud, it is not possible to identify precisely the time at which data has actually been generated at the smart meter as this information is encrypted. An example of this issue is illustrated in Fig. 2. In this figure, although the data is generated by the slow and fast client at the same time, the data from the fast client always arrives much earlier than the slow client. If the averaging window is small, then the information from the slow client will not be representative of the averaging window.

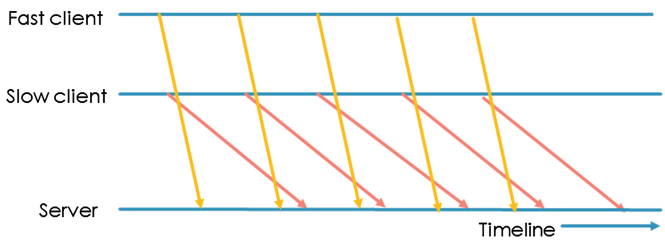

Fig. 2. Example of desynchronization of data values

We define a bucket to be a period of time in which we take the average of data points from a particular meter and an example is presented in Fig. 3. Hence, incorrect ordering within this time frame is tolerated.

We explore the tolerance to error of the averaging function for various averaging bucket sizes. Load profiles of 30 homes are generated over a 24 h period. All load generation parameters are obtained from [5] for Ontario. Figures 4 and 6

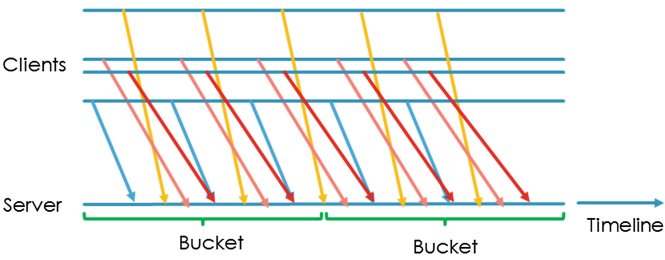

Fig. 3. Illustration of a bucket

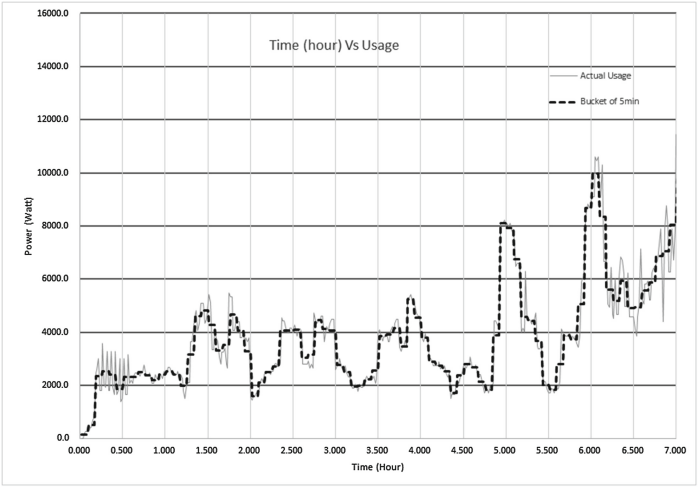

Fig. 4. Comparison of averaging during off-peak hours

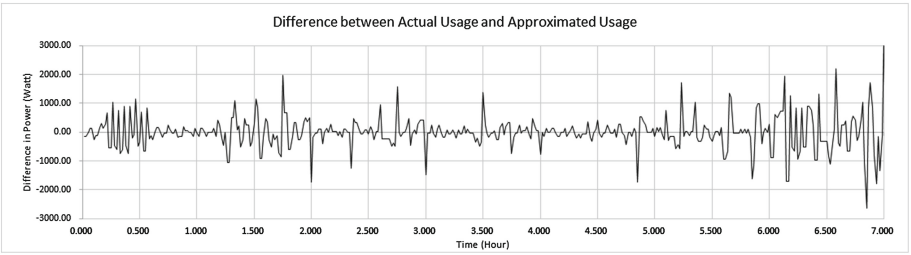

Fig. 5. Error between actual average and approximated average during off-peak hours

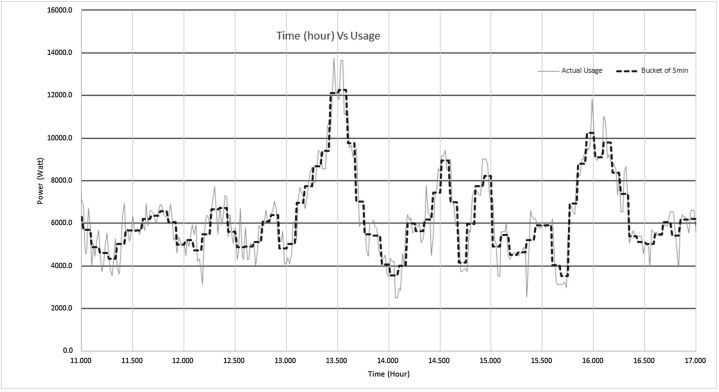

Fig. 6. Comparison of averaging during on-peak hours

illustrate the average load profiles of 30 homes for off-peak and on-peak periods respectively. Each one of these graphs contains the actual average load profile obtained from perfectly synchronized data points and the approximated average using a bucket that is 5 min in length. Errors resulting from the approximation is illustrated in Figs. 5 and 7.

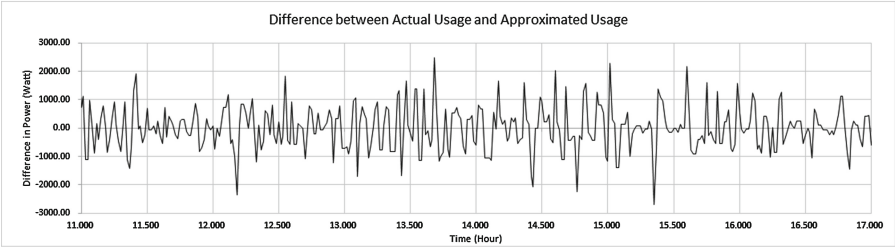

Fig. 7. Error between actual average and approximated average during on-peak hours

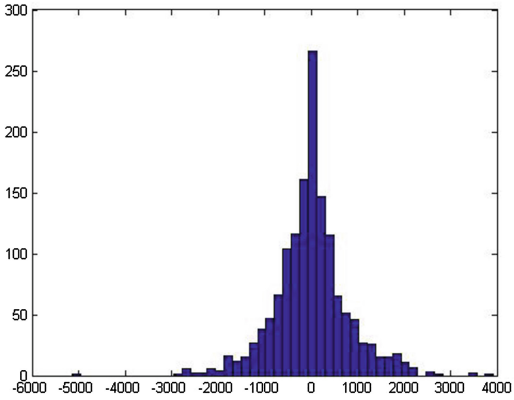

Fig. 8. Error distribution

The distribution of error over a day is illustrated in Fig. 8. This distribution curve is very similar to a normal distribution with a mean of approximately 0 W and a standard deviation of 799 W. Next, Fig. 9 presents a scatter plot containing the standard deviation of error for various bucket sizes. A line of best fit reveals that the general trend of error standard deviation is that it is logarithmically increasing with the bucket size. This log function has a coefficient of determination of 0.98.

All results presented in this section provide interesting insights on the impact of the bucket size on errors introduced into the computations. Smaller the bucket size, the smaller is the variation in error around the mean 0.

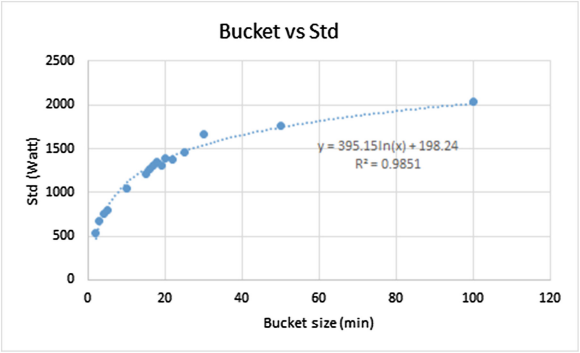

Fig. 9. Best-fit curve for error deviation for various bucket sizes

5 Conclusions

In this paper, we explore how sensitive meter data can be stored and processed securely in the cloud for applications such as demand response that perform basic analytics on this data in real-time. As vast amounts of meter data are generated and processed daily for these applications, it is necessary to utilize a light-weight encryption scheme that protects the data from issues such as leakage and integrity which are especially of concern when a third party infrastructure is used for managing this data. Homomorphic encryption is a suitable candidate for this as it enables the utilization of powerful computational capabilities of the cloud without exposing actual data to the external environment. PE is a light-weight scheme but with limited functionality. As the averaging operation is a common computation applied to metered data, we presented three methods that leverage on existing features of the PE scheme to support this. Of these methods, we have identified the multiplier amplification method to be the least intrusive for clients. Next, we investigate how latency introduced in the transmission and storage of encrypted data can affect the actual representation of the system state. In order to reduce the impact of incorrect ordering, the data values from a single source are averaged over a time window (i.e. bucket). We show that the error of the approximated system represents a normal distribution and more specifically, the error deviation increases logarithmically with the bucket size.

References

1. BC Hydro's Smart Metering Program - Program Overview for Key Account Customers. BC Hydro PDF, British Columbia
2. Srikantha, P., Kundur, D.: A novel evolutionary game theoretic approach to real-time distributed demand response. In: IEEE Power Engineering Society General Meeting (PES GM), July 2015
3. Rong, C., Nguyen, S.T., Jaatun, M.G.: Beyond lightning: a survey on security challenges in cloud computing. Comput. Electr. Eng. **39**(1), 47–54 (2013)

4. Ruzzelli, A., Nicolas, C., Schoofs, A., OHare, G.: Real-time recognition and profiling of appliances through a single electricity sensor. In: Seventh IEEE Communications Society Conference on Sensor Mesh and Ad Hoc Communications and Networks, pp. 1–9 (2010)
5. Srikantha, P., Rosenberg, C., Keshav, S.: An analysis of peak demand reductions due to elasticity of domestic appliances. In: Proceedings of the 3rd International Conference on Future Energy Systems: Where Energy, Computing and Communication Meet. ACM (2012)
6. Craig, G.: Fully homomorphic encryption using ideal lattices. In: STOC, vol. 9 (2009)
7. Murrill, B.J., Liu, E.C., Thompson, R.M.: Smart meter data: privacy and cybersecurity. In: Congressional Research Service, Library of Congress (2012)
8. Halevi, S., Shoup, V.: Bootstrapping for HElib. In: Oswald, E., Fischlin, M. (eds.) EUROCRYPT 2015. LNCS, vol. 9056, pp. 641–670. Springer, Heidelberg (2015)
9. van Dijk, M., Gentry, C., Halevi, S., Vaikuntanathan, V.: Fully homomorphic encryption over the integers. In: Gilbert, H. (ed.) EUROCRYPT 2010. LNCS, vol. 6110, pp. 24–43. Springer, Heidelberg (2010)
10. Paillier, P.: Public-key cryptosystems based on composite degree residuosity classes. In: Stern, J. (ed.) EUROCRYPT 1999. LNCS, vol. 1592, p. 223. Springer, Heidelberg (1999)
11. Wang, W., et al.: Exploring the feasibility of fully homomorphic encryption. IEEE Trans. Comput. **64**(3), 698–706 (2015)
12. Pwnhome. The Homomorphic Encryption Project. Vers. 0.2. Computer software. Pwnhome(2011)
13. Tebaa, M., El Hajji, S., El Ghazi, A.: Homomorphic encryption applied to the cloud computing security. In: Proceedings of the World Congress on Engineering, vol. 1 (2012)
14. Garcia, F.D., Jacobs, B.: Privacy-friendly energy-metering via homomorphic encryption. In: Cuellar, J., Lopez, J., Barthe, G., Pretschner, A. (eds.) STM 2010. LNCS, vol. 6710, pp. 226–238. Springer, Heidelberg (2011)
15. Li, F., Luo, B., Liu, P.: Secure information aggregation for smart grids using homomorphic encryption. In: First IEEE International Conference on Smart Grid Communications, pp. 327–332 (2010)
16. He, X., Pun, M.O., Kuo, C-C.J.: Secure and efficient cryptosystem for smart grid using homomorphic encryption. In: Innovative Smart Grid Technologies (ISGT), IEEE PES. IEEE (2012)