Future Generation Computer Systems 72 (2017) 327–338

Contents lists available at ScienceDirect

Future Generation Computer Systems

journal homepage: www.elsevier.com/locate/fgcs

Design and implementation of a secure cloud-based billing model for smart meters as an Internet of things using homomorphic cryptography

Vu Mai [*], Ibrahim Khalil

Department of Computer Science and Information Technology, RMIT University, Melbourne, Australia

HIGHLIGHTS

- A cloud-based smart meter data processing model preserving user confidentiality.
- Aggregation of encrypted smart meter readings with encrypted fixed-point number.
- Calculating the computation time of encrypted additions performed on cloud.
- A parallel version of our billing algorithm using multi-core processors on cloud.

A R T I C L E   I N F O

Article history:
Received 16 November 2015
Received in revised form
6 April 2016
Accepted 5 June 2016
Available online 22 June 2016

Keywords:
Smart grid
IoT
Internet of things
Homomorphic encryption

A B S T R A C T

Smart grids introduce many outstanding security and privacy issues, especially when smart meters are connected to public networks, creating an Internet of things in which customer usage data is frequently exchanged and processed in large volumes. In this research, we propose a cloud-based data storage and processing model with the ability to preserve user privacy and confidentiality of smart meter data in a smart grid. This goal is achieved by encrypting smart meter data before storage on the cloud using a homomorphic asymmetric key cryptosystem. By applying the homomorphic feature of the cryptographic technique, we propose methods to allow most of the computing works of calculating customer invoices based on total electricity consumption to be done directly on encrypted data by the cloud. One of the outstanding features in our model is the aggregation of encrypted smart meter readings using fixed-point number arithmetic. To test the feasibility of our model, we conducted many experiments to estimate the number of homomorphic additions to be performed by the cloud and the computation time in different billing periods using data from the Smart project, in which smart grid readings were continuously collected from different households in every second within two months and electricity usage data collected every minute from 400 anonymous houses in one day. We also propose a parallel version of our billing algorithm to utilize the processing capability of multi-core processors in cloud servers so that computation time is reduced significantly compared to using our sequential algorithm. Our research works and experiments demonstrate clearly how cloud services can strengthen the security, privacy and efficiency of privacy-sensitive data frequently exchanged and processed in an Internet of things where smart meters communicate directly with public networks.

1. Introduction

Smart grid technologies have brought many great benefits to both providers and consumers of electricity. Governments and energy suppliers are now able to balance electricity generation with consumption through a system of billing in which customers are charged by how much energy they have consumed at different times of day using dynamic and flexible tariffs. Smart meters provide more accurate, up-to-date and fine-grained meter readings, helping energy suppliers to adjust electricity generation and prices according to demands, thereby reducing blackouts and improving the reliability of the electricity grid. Various types of energy monitor devices connected directly to smart meters via wireless links help consumers to view their usage history, the amount of electricity they are using, and the current tariff.

[*] Corresponding author.
E-mail addresses: huyvumai@gmail.com (V. Mai), ibrahim.khalil@rmit.edu.au (I. Khalil).

http://dx.doi.org/10.1016/j.future.2016.06.003
0167-739X/© 2016 Elsevier B.V. All rights reserved.

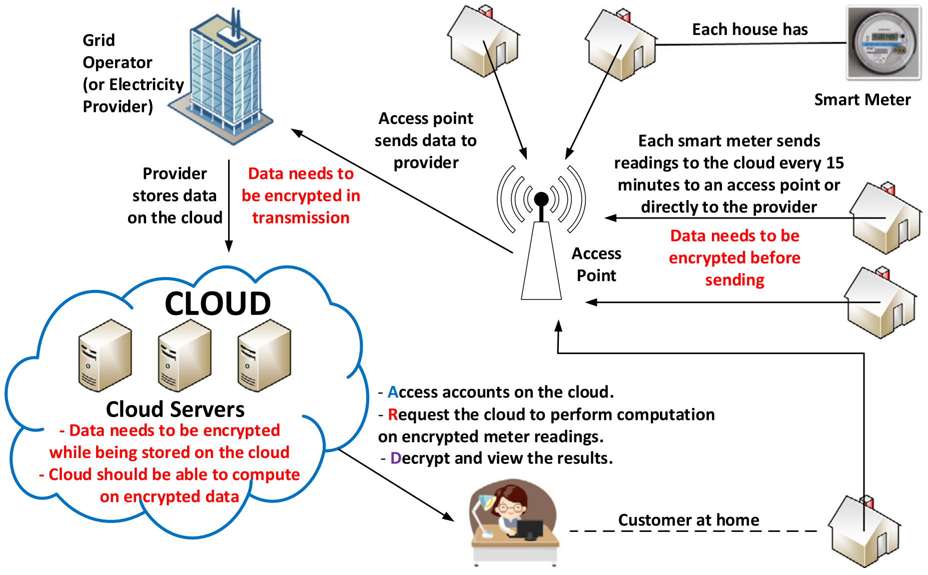

Grid Operator (or Electricity Provider)
Each house has
Smart Meter
Access point sends data to provider
Provider stores data on the cloud
Data needs to be encrypted in transmission
Each smart meter sends readings to the cloud every 15 minutes to an access point or directly to the provider
Access Point
Data needs to be encrypted before sending
CLOUD
Cloud Servers
- Data needs to be encrypted while being stored on the cloud
- Cloud should be able to compute on encrypted data
- Access accounts on the cloud.
- Request the cloud to perform computation on encrypted meter readings.
- Decrypt and view the results.
Customer at home

Fig. 1. The vision of our research, a smart grid as an Internet of things containing a large number of smart meters, access points, users and a grid operator connected to a public network with the strong support of various cloud storage and processing services working directly on homomorphically encrypted data.

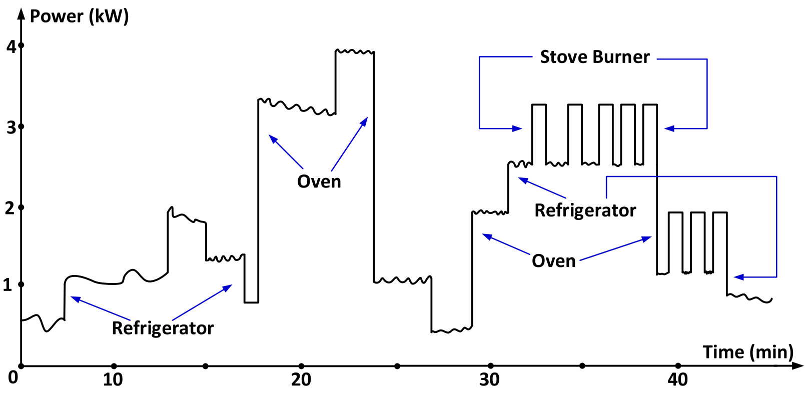


Fig. 2. Tracking usage pattern using electricity consumption data [5].

Customers now have the choice about how and when to reduce their energy consumption and take control of their electricity costs. Furthermore, smart grids also make it more efficient to manage the distributed power generation such as local solar and wind generators.

Despite the benefits to both the providers and consumers of electricity, a smart grid may also introduce many outstanding security and privacy issues, especially when smart meters are connected to public networks in which not only personally identifiable information but also energy consumption data relating to behaviors and movements of users is frequently exchanged and processed in large volumes. Fig. 1 shows the vision of our research, in which we view a smart grid as an Internet of things containing a large number of smart meters, access points, users, a grid operator and the cloud connected to a public network. The figure also shows how this Internet of things can benefit from various cloud storage and processing services as long as security measures such as data encryption is implemented.

Furthermore, as the time interval of data collected by smart meters decreases to fifteen or thirty minutes, various load monitoring techniques [1,2] can be employed to process unencrypted smart meter data to identify what electrical appliances, for example heaters, washing machines, refrigerators, air conditioners etc., are being used based on the electrical signature of those appliances [3,4]. Fig. 2 shows a power consumption trace of a customer [5]. Many statistical tools and data mining techniques can be used to extract patterns from the fine-grained electricity consumption data to build user profiles and monitor user activities such as whether someone is at home, the habits of each family member, or what they do at particular moments, etc. [6,7].

The security and privacy issues outlined above have prevented the cloud from reaching its full potential in supporting a smart grid, especially when many components of the grid are connected as an Internet of things via the public network. Smart meters are constrained devices having limited capacity and incapable of performing complex computing tasks on energy usage data. Therefore, a smart grid needs a cloud computing system with its unlimited storage and processing capabilities to support complex computing tasks such as billings and analyzing data. However, this goal cannot be accomplished fully without a suitable secure data storage and processing model providing the capability to compute on encrypted data. Therefore, we conducted the research described in this paper to find out how the smart grid, as an Internet of things can enjoy all the benefits of cloud computing while providing data owners with a strong guarantee of their privacy, the confidentiality of smart meter data and the reliability of the grid's information infrastructure.

Our contributions. In this research, we propose a cloud-based data storage and processing model applicable for a smart grid with many components connected as an Internet of things via the public network. Our model utilize the full strength of cloud computing and preserves user privacy and the confidentiality of data exchanged on the grid. This goal is achieved by encrypting smart meter data before sending to the cloud for processing and storage using a homomorphic asymmetric key cryptosystem. Each smart meter is equipped with a set of private and public keys. The grid operator is given accessed to all the private keys while each household owner can only access the decryption key corresponding to their own smart meter. Using the homomorphic feature of the cryptographic technique, we create methods to enable most of the computing works to be done directly on encrypted data by the cloud, especially, we focus on the aggregation of encrypted smart meter readings directly on the cloud. During the homomorphic computation process, the cloud is allowed to access all public keys which are required by many homomorphic computing operations. However, the cloud does not have access to the private keys, hence, no decryption would be performed and no information would be leaked during the homomorphic computation process. The prominent features of our model are summarized as follows:

- User privacy and data confidentiality are protected by means of cryptography, especially when our model can store and process data mostly on the cloud. Only parties possessing the private keys can decrypt the data. An example of such parties are the grid operator because it needs to access the fine-grained meter readings to monitor the performance of the grid. A household owner can also decrypt their smart meter readings when they want to know their total data usage and other statistics such as daily or hourly consumption. The electricity retailers, however, only need to know the total usage in a month or a quarter. Therefore, they are only given access to the data aggregation results and not the decryption keys.
- Our model allows the cloud to securely compute the aggregated energy consumption of a customer in a given period of time by performing homomorphic addition operation directly on an arbitrary number of encrypted fine-grained readings sent from the smart meter of that customer. The cloud does not need to perform any decryption during this computation process. Therefore, the privacy of the user and the confidentiality of the data are protected.
- Retailers are able to offer flexible pricing policies to consumers based on the time-varying costs of electricity procurement at the wholesale level. This feature is made possible because our model is designed so that the retailers can request the aggregated energy consumption of their customers during different time periods. The grid operator receives such requests from retailers, performs most of the data aggregation tasks directly on the cloud, then decrypts the results and sends them back to retailers.
- We propose a parallel version of our billing algorithm to utilize the processing capability of multi-core processors in cloud servers so that the computation time can be reduced significantly compared to using our sequential homomorphic addition algorithm. Using Amdahl's law and a metric called speedup, we measure how many times faster our parallel algorithm can actually run on some common processors with different number of cores and threads. We also demonstrate by many examples how an increase of speedup is determined by the size of the homomorphic expression as well as the number of threads used for the parallel computation process.

2. Related works

There have been many research works focusing on the storage and processing of smart meter data while preserving user privacy and protecting data confidentiality. However, not many of those works provide detailed solutions and practical implementation about how the data is stored and processed in an encrypted form on the cloud.

Rial et al. [8] propose a set of privacy-preserving protocols involving three parties: an electricity provider, a user agent and a simple tamper-evident meter. The data is encrypted by the smart meter using its symmetric key and stored in encrypted form in remote servers. To perform data aggregation and billing, the user will need to download the encrypted data and decrypt using a symmetric decryption key. A final bill will be sent to the provider with a zero-knowledge proof that ensures the calculation to be correct and leaks no additional information. However, this approach requires the user themselves or a third party that the user fully trusts to perform the computation. Cloud servers cannot compute on encrypted data. In a more realistic setting [9], the electricity retailer would calculate the bills first and send to customers, given only the total energy consumption in fixed period of times.

In their work, Acs et al. [7] describe a scheme allowing electricity providers to collect smart meter data periodically and derive aggregated statistics without learning anything about the activities of individual households. At each billing period, a smart meter sends to its corresponding provider the readings that are mixed with random noise and encrypted to protect the privacy of a user. The provider can still compute the total electricity consumption of that household despite the amount of noise being added to the original data. However, this research work assumes that most of the computing tasks involved in finding the total energy consumption are performed by a provider rather than using the cloud, causing limitations in cases when the provider does not have enough computing power to aggregate fine-grained readings coming from a large number of meters or when to store and secure a huge amount of data over long periods of time.

Another research work has been proposed by Ruj et al. [10] in which a decentralized security framework was designed for smart grids with the capabilities to do both data aggregation and access control. The smart grid is divided into a hierarchy comprising a home, building and neighboring area networks. Electricity usage data is collected and sent to substations along a path from lower to higher networks in the hierarchy under the monitor and control of remote terminal units. Security and user privacy is achieved by encrypting the data in the transmission process while aggregation tasks are performed on encrypted data due to the use of homomorphic encryption. While details of the access control component of the architecture are described very thoroughly, especially about how users can access the data through remote terminal units, the authors have not provided as many details about cloud-based data storage and how aggregation tasks are performed on encrypted data.

There are other research works providing different approaches to the security and privacy protection of smart meter data. In their work, Deng et al. [11] describe in detail how smart meters can register and authenticate their identities before a secure communication session can be set up with the data collector. These operations help to build a network of smart meters organized as an aggregation tree in which the data collector is the root node with information relating to the network structure and routing backbone. The author assume that smart meters are connected to one another and readings from one meter have to travel to other meters to reach the data collector. This feature is significantly different from our model in which each meter are independent and connected directly to the grid operator. In another research, Garcia et al. [12] design protocols for basic communication with E-meters using elementary cryptographic techniques such as symmetric key encryption and digital signature. Their main goal is to learn the aggregated energy consumption of N consumers without revealing

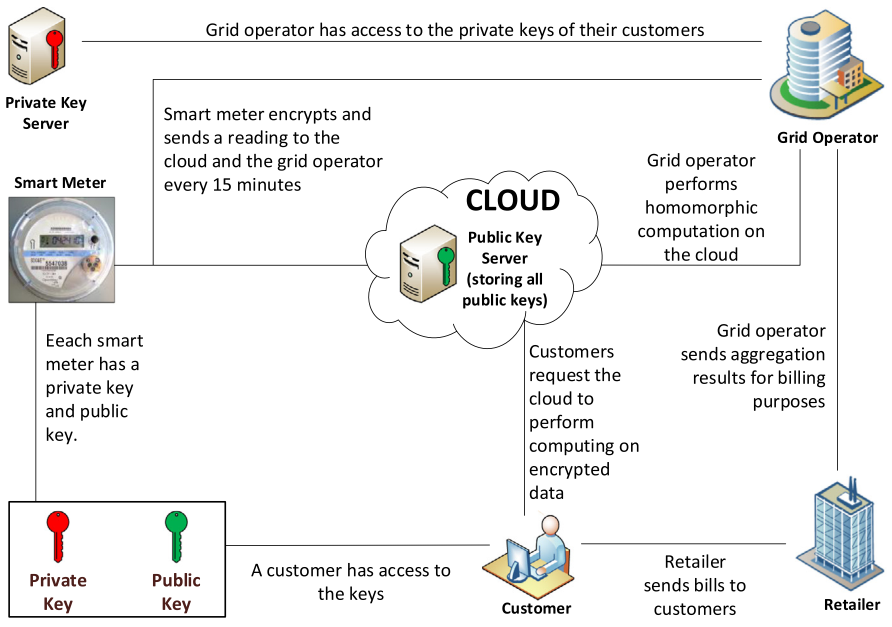

Grid operator has access to the private keys of their customers
Private Key Server
Grid Operator
Smart meter encrypts and sends a reading to the cloud and the grid operator every 15 minutes
Smart Meter
CLOUD
Public Key Server (storing all public keys)
Grid operator performs homomorphic computation on the cloud
Eeach smart meter has a private key and public key.
Customers request the cloud to perform computing on encrypted data
Grid operator sends aggregation results for billing purposes
Private Key
Public Key
A customer has access to the keys
Customer
Retailer sends bills to customers
Retailer

Fig. 3. The architecture of our model.

any information about the individual consumption of the users, even when the data collector is malicious. However, each meter M_i in their model must also know about the $N - 1$ public keys of the other meters participating in the protocol. This is a limitation because the smart meters in their model are programmed to abort the protocol when they cannot get enough meters with public keys willing to join a communication session.

In our research, we aim to complement existing literature by describing in details how data is stored on the cloud in encrypted formats. We also present a model showing how the homomorphic aggregation of encrypted smart meter readings is performed directly on the cloud. Furthermore, we implement the model and measure the computation time of our algorithm on real data sets.

3. Cloud-based smart meter data storage and processing

3.1. Architecture

In our model, each smart meter is allocated an asymmetric key pair generated by a homomorphic cryptosystem. Every fifteen minutes, a meter reading is encrypted by the smart meter using the public key and sent to the grid operator for monitoring purposes and to the cloud for storage and homomorphic computation. The grid operator can access the private keys of all smart meters in the grid and decrypt fine-grained readings to monitor the grid performance. The cloud is responsible for performing most of the homomorphic computation over encrypted smart meter data by using the public keys of those meters. Retailers are responsible for issuing bills to their customers using the total amount of energy consumed in a month or a quarter. They are not allowed to access the private keys to decrypt fine-grained readings stored on the cloud. Instead, retailers send requests for data aggregation to the grid operator, which will use the cloud to perform the homomorphic aggregation tasks, decrypt the results and send them to retailers. Customers receive bills containing the electricity cost and other statistical results from their retailers. Our model allows customers to have access to the private keys of their smart meters. Therefore, customers can check their bills and perform other statistics on their usage data by asking the cloud to perform various homomorphic operations on their encrypted fine-grained readings and decrypting the results using their private keys. Fig. 3 shows the details of our model.

The electricity consumption of each household is measured by a smart meter. Every fifteen minutes, the smart meter sends out a data package containing three types of data: a smart meter identification number, a timestamp and an encrypted reading. Each smart meter has an identification number granted by the grid operator when the smart meter is installed. This number is used to uniquely refer to the smart meter. Therefore, it must be included in each data package so that the grid operator can differentiate between data packages sent from different sources. The cloud also needs the identification number to identify smart meter data when processing queries. The timestamp indicates the moment when a reading is recorded. This value is used by the grid operator for performance monitoring purposes and for the cloud to know whether the reading was taken in the peak or off peak period. The identification number and timestamp are not encrypted. However, the smart meter reading taken every fifteen minutes is encrypted using the public key of the corresponding smart meter. Only the grid operator and the household owner of that smart meter have the private key to decrypt the readings. Encryption is required to secure the storage and processing of smart meter readings on the cloud. Fig. 4 outlines the content of a data package sent out by a smart meter.

Supporting multiple pricing policies. Our model allows a retailer to set flexible pricing policies based on different times of the day such as peak and off-peak period as shown in Fig. 5, allowing customers to save more money and increasing the efficiency of electricity usage. A typical pricing policy $\mathcal{P}$ contains a smart meter reading x which shows the number of kWhs consumed during a fixed period of time (for example 15 min) and the time t where the reading was recorded. The time information t allows a retailer to decide a rate at which a customer will be charged for their usage x. Therefore, a pricing policy $\mathcal{P}$ can be regarded as a function $\mathcal{P} : (x, t) \rightarrow p$ that takes a reading x and a recording time t and outputs a price p. After each billing period, a total fee is computed by adding the prices corresponding to the total electricity consumption in that period, according to the formula: fee $= \sum_{i=1}^{n} p_i$, in which n is the total number of readings in a billing period and p_i is the price calculated for each reading. Our model is very similar to the practical smart grid architecture described in [9] in that a retailer is not granted

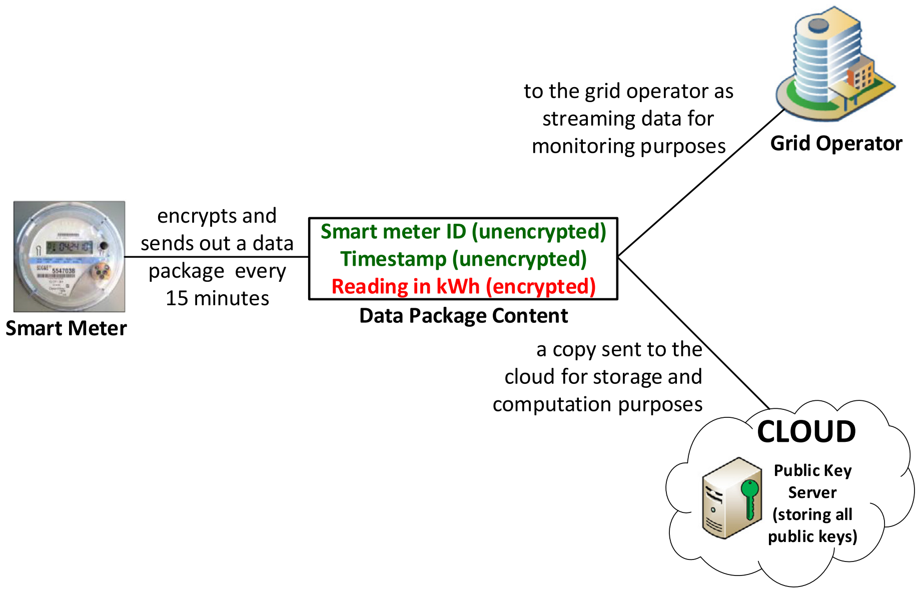
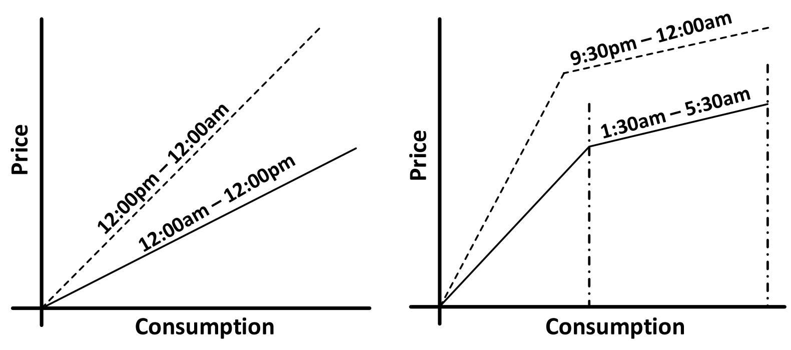

to the grid operator as streaming data for monitoring purposes
Grid Operator
Smart Meter
encrypts and sends out a data package every 15 minutes
Smart meter ID (unencrypted)
Timestamp (unencrypted)
Reading in kWh (encrypted)
Data Package Content
a copy sent to the cloud for storage and computation purposes
CLOUD
Public Key Server (storing all public keys)

Fig. 4. Content of a data package sent by a smart meter every fifteen minutes.



(a) Two linear policies. (b) Two cumulative policies.

Fig. 5. (a) A linear policy specifies the rate per unit consumption that is applied to determine the price to be paid for each measurement. (b) A cumulative policy specifies a rate per unit that is determined as a function of the consumption allowing nonlinear functions to be applied for pricing.

direct access to fine-grained smart meter readings of its customers. This design feature is reasonable because a retailer only requires the total energy consumption during a period for billing purposes while fine-grained readings are often required by the grid operator to monitor the grid performance. Furthermore, customers are far more likely to change their retailers rather than changing their grid operator because retailers constantly compete with each other to offer the best prices to customers while there are only a few grid operators which do not often deal with customers directly.

When a retailer wants to calculate a customer bill in a consumption period, it needs the total electricity figure of the customer during that period to multiply with the rates it offers. This total amount can be for a month or a quarter and can be divided into smaller sums corresponding to peak or off-peak periods as determined by various pricing policies set by the retailer. These total figures are calculated from fine-grained readings obtained from the customer's smart meter. Because the retailer cannot access those readings directly, it will send requests to the grid operator, which will instruct the cloud to perform most of the calculation tasks. This is where our model brings many benefits to customers, the grid operator and retailers in terms of data security, user privacy and efficiency. We find out a method to calculate the total figures directly on encrypted data, thereby allowing the cloud not only to store data securely but also to compute on encrypted data. Finally, the cloud outputs the total figures in encrypted forms and sends to the grid operator, which will decrypt and send to the retailer to complete the request.

Calculating total energy consumption on the cloud using encrypted data. After each billing period, retailers will need to calculate bills for each of their customers. They need the total energy consumption of each of the customer. However, retailers cannot access smart meter readings directly because they are encrypted and store on the cloud. According to our model, only the grid operator can access encrypted readings because it stores the private keys of all the smart meters. Therefore, retailers will send billing request to the grid operator which in turn will ask the cloud to perform most of the computation on encrypted data.

After being asked to calculate a total energy consumption for a customer in a period of time by a retailer, the grid operator will send a request to the cloud containing information to help the cloud to identify and calculate the data of that customer. The request will contain the identification details of the customer and the usage period during which the total energy consumption is calculated. Upon receiving the request, the cloud will use the customer identification number to locate the data related to that customer. Each smart meter reading is stored in encrypted form on the cloud and associated with a timestamp in plaintext. Based on the request usage period, the cloud decides which encrypted

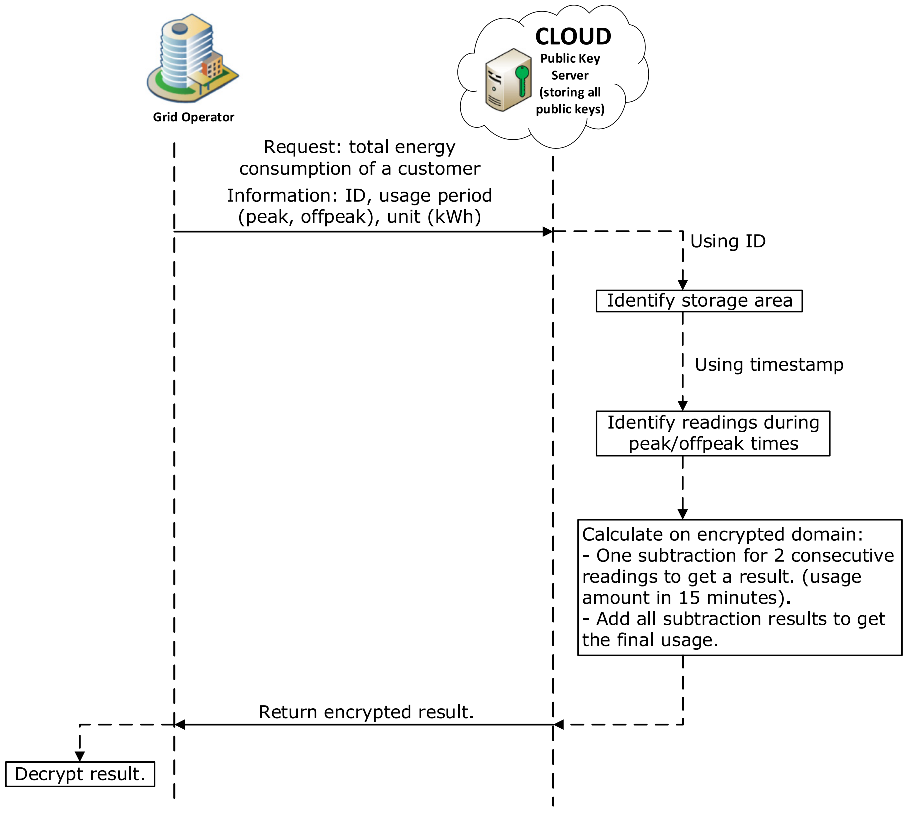

Grid Operator
CLOUD
Public Key Server (storing all public keys)
Request: total energy consumption of a customer
Information: ID, usage period (peak, offpeak), unit (kWh)
Using ID
Identify storage area
Using timestamp
Identify readings during peak/offpeak times
Calculate on encrypted domain:
- One subtraction for 2 consecutive readings to get a result. (usage amount in 15 minutes).
- Add all subtraction results to get the final usage.
Return encrypted result.
Decrypt result.

Fig. 6. Cloud-based calculation of total energy consumption on encrypted data for a customer.

reading is relevant to the calculation. In the encrypted domain, the cloud will subtract two consecutive readings to get a result, for example, the total usage in 15 min, then all such results will be added to get the final usage in encrypted form. The cloud will send this encrypted result back to the grid operator which in turn will decrypt the result using the appropriate key and send the final unencrypted result to the retailer to apply appropriate rate to the total energy consumption. Fig. 6 illustrates this process.

3.2. Homomorphic key generation, encryption and decryption

Homomorphic cryptography has long been the focus of many researchers because this technique allows specific types of computations to be carried out on ciphertext and generate an encrypted result which, when decrypted, matches the result of operations performed on the plaintext. This feature has a great potential for cloud-based applications because not only users can encrypt and store their data on the cloud, but the encrypted data can also be processed directly on the cloud without compromising data privacy.

An encryption scheme is considered to be partially homomorphic if its ciphertexts can be either added or multiplied an unlimited number of times but not both operations at the same time. There are two types of partially homomorphic cryptosystems: additively and multiplicative homomorphic. Given two plaintext integers m_1 and m_2, an additively homomorphic encryption scheme has a binary operation $\oplus$ that guarantees the following condition is true for an addition of m_1 and m_2:

$$\begin{aligned}&\textbf{Decrypt}(\textbf{Encrypt}(m_1) \oplus \textbf{Encrypt}(m_2))\\ &\quad = \textbf{Decrypt}(\textbf{Encrypt}(m_1 + m_2))\\ &\quad = m_1 + m_2.\end{aligned}$$

Similarly, a multiplicative homomorphic cryptosystem is defined as a binary operation $\otimes$ that can be applied on the ciphertexts of the two plaintext m_1 and m_2 as shown in the following equation:

$$\begin{aligned}&\textbf{Decrypt}(\textbf{Encrypt}(m_1) \otimes \textbf{Encrypt}(m_2))\\ &\quad = \textbf{Decrypt}(\textbf{Encrypt}(m_1 \times m_2))\\ &\quad = m_1 \times m_2.\end{aligned}$$

In this research, we use a somewhat homomorphic cryptosystem proposed by Smart and Vercauteren (SV) [13] because their scheme has small ciphertext and key size as well as allows both addition and multiplication on ciphertexts. At a high level, the SV scheme requires a security parameter N specified by the user. The public key consists of a prime p and an integer $\alpha \bmod p$. The private key consists of an integer polynomial $Z(x)$ of degree $N-1$ to encrypt general binary polynomials of degree $N-1$. In our model, only binary digits are encrypted and hence, the private key consists of only an integer z of our choice. Generally, a message is encrypted by first encoding it as a binary polynomial and then a randomization process occur by adding on two times a small random polynomial. Decryption is achieved when the resulting polynomial is evaluated at $\alpha \bmod p$. Therefore, the ciphertexts are integers modulo p regardless of whether bits or binary polynomials of degree $N-1$ have been encrypted. In our model, the plaintext message is a single bit. Therefore, decryption will require multiplying a ciphertext it by z and dividing the result by p. This rational number is then rounded to the nearest integer value, and subtract the result from the ciphertext. The plaintext is obtained by reducing this intermediate result modulo 2.

The SV scheme consists of five algorithms: **KeyGen**, **Encrypt**, **Decrypt**, **Add** and **Multiply**. This technique requires a set of input parameters (N, η, μ) in which $\eta = 2^{\sqrt{N}}$ and $\mu = \sqrt{N}$ and N is the security parameter specified by the user. Before each of the five algorithms can be described in details, two mathematical objects, i.e. a *polynomial ball* and a *polynomial half-ball*, are defined as follows:

$$\mathcal{B}_{2,N}(r) = \left\{ \sum_{i=0}^{N-1} a_i x^i : \sum_{i=0}^{N-1} a_i^2 \le r^2 \right\}$$
$$B_{\infty,N}(r) = \left\{ \sum_{i=0}^{N-1} a_i x^i : -r \le a_i \le r \right\}$$
$$B_{\infty,N}^{+}(r) = \left\{ \sum_{i=0}^{N-1} a_i x^i : 0 \le a_i \le r \right\}.$$

In the equations above, r is a positive integer, $\mathcal{B}_{2,N}(r)$ and $B_{\infty,N}(r)$ are polynomial balls while $B_{\infty,N}^{+}(r)$ is a polynomial half-ball. It can be seen that $B_{2,N}(r) \subset B_{\infty,N}(r)$ and $B_{\infty,N}(r) \subset B_{2,N}(\sqrt{N} \cdot r)$. In the following algorithms, the notation $a \longleftarrow b$ means assigning the value of b to a, while given a set A, writing $a \longleftarrow_R A$ means select a from the set A using a uniform distribution.

The **KeyGen** algorithm generates the public key **PK** and the secret key **SK**:

```
1. Set the plaintext space to be $\mathcal{P} = \{0, 1\}$
2. Choose a monic irreducible polynomial $F(x) \in Z[x]$ of degree $N$
3. Repeat
    ▷ $S(x) \longleftarrow_R \mathcal{B}_{\infty,N}(\eta/2)$
    ▷ $G(x) \longleftarrow 1 + 2 \cdot S(x)$
    ▷ $p \longleftarrow \mathrm{resultant}(G(x), F(x))$
4. Until $p$ is prime
5. $D(x) \longleftarrow \gcd(G(x), F(x))$ over $F_p[x]$
6. Let $\alpha \in F_p$ denote the unique root of $D(x)$
7. Apply the XGCD-algorithm over $Q[x]$ to obtain $Z(x) = \sum_{i=0}^{N-1} z_i x^i \in Z[x]$ such that
   $Z(x) \cdot G(x) = p \bmod F(x)$
8. $B \longleftarrow z_0 \pmod{2p}$
9. The public key is $\mathbf{PK} = (p, \alpha)$, the private key is $\mathbf{SK} = (p, B)$.
```

The **Encrypt** algorithm takes a plaintext message **M** and the public key **PK** as inputs to produce the ciphertext c:

```
1. Parse PK as $(p, \alpha)$
2. If $M \notin 0, 1$ then abort
3. $R(x) \longleftarrow_R \mathcal{B}_{\infty,N}(\eta/2)$
4. $C(x) \longleftarrow M + 2 \cdot R(x)$
5. $c \longleftarrow C(\alpha) \bmod p$
6. Output $c$.
```

The **Decrypt** algorithm decrypts the ciphertext **c** using the **PK** and returns the plaintext message **M**:

```
1. Parse SK as $(p, B)$
2. $M \longleftarrow (c - \lfloor c \cdot B/p \rceil) \bmod 2$
3. Output $M$.
```

The homomorphic **Add** operation can be performed given two ciphertexts $\mathbf{c_1}, \mathbf{c_2}$ and the public key **PK** as follows:

```
1. Parse PK as $(p, \alpha)$
2. $c_3 \longleftarrow (c_1 + c_2) \bmod p$
3. Output $c_3$.
```

Finally, the homomorphic multiplication operation is achieved by the performing the **Mult** algorithm that takes two ciphertexts $\mathbf{c_1}$, $\mathbf{c_2}$ and the public key **PK** as inputs:

```
1. Parse PK as $(p, \alpha)$
2. $c_3 \longleftarrow (c_1 \cdot c_2) \bmod p$
3. Output $c_3$.
```

3.3. Encoding smart meter readings

In our model, smart meter readings are encrypted using the public key of the smart meter before being sent to the cloud. We use the SV homomorphic scheme which can only encrypt binary digits. The encoding method that we use to convert the readings into binary numbers are described in this section. We assume that each reading is a decimal number with a fixed resolution ε, for example, $\varepsilon = 10^{-2}, 10^{-3}\ldots$ Therefore, each reading can be represented by a fixed-point binary number having three components: the integer part, the fractional part and a virtual binary point acting as a divider between the two parts. The position of the binary point is implicitly defined in our model and will be referred to whenever readings are converted to binary and back to decimals.

Every fifteen minutes, a smart meter reading R is converted to a fixed-point binary number Q having the format Q[QI].[QF] in which QI and QF are the number of integer and fractional bits, respectively. Let WL be the total number of bits used to represent the binary number. Hence, WL is the sum of QI and QF and is a constant set in our model. Therefore, it is very important to select an appropriate length WL when our model is used so that no overflow occurs. This can be achieved by using the formula proposed by Oberstar et al. [14], which describe how QI and QF are calculated. According to the authors, all integers α_i in a range having a minimum value $\alpha_{\min}$ and maximum value $\alpha_{\max}$ can be represented by a fixed number of bits, QI, calculated by the following equation:

$$\boxed{\mathrm{QI} = \mathrm{floor}(\log_2(\max(\mathrm{abs}[\alpha_{\max}, \alpha_{\min}]))) + 2}$$

The resolution ε of a smart meter reading is determined by the number of bits QF used to represent the fractional part of the corresponding fixed-point binary number. QF is calculated by the following equation:

$$\boxed{\mathrm{QF} = \mathrm{ceiling}\left(\log_2\left(\frac{1}{\varepsilon}\right)\right)}$$

3.4. Homomorphic addition over encrypted fixed-point binary numbers

After the encoding, each smart meter reading is represented by an array of binary digits, ready for our bitwise fully homomorphic encryption step, which is based on the approach proposed by Kaosar et al. [15]. Specifically, an n-bit binary number $X = (x_n, x_{n-1}, \ldots, x_1)$ where $x_i \in \{0, 1\}$, can be encrypted as shown in the following equation, using the somewhat homomorphic cryptosystem proposed by Smart and Vercauteren (SV) [13] with an encryption function **E** and the fully homomorphic public key **pk**:

$$\begin{aligned}\alpha &= (\alpha_n, \alpha_{n-1}, \ldots, \alpha_1) = E_{pk}(X)\\ &= [E_{pk}(x_n), E_{pk}(x_{n-1}), \ldots, E_{pk}(x_1)].\end{aligned}$$

Each ciphertext α_i generated by the SV encryption algorithm is a large integer representing an encrypted version of a binary digit. The binary number X is retrieved by using the decryption function **D** and the secret key **sk** to decrypt each ciphertext α_i to get a binary digit x_i as follows:

$$\begin{aligned}X &= (x_n, x_{n-1}, \ldots, x_1) = D_{sk}(\alpha)\\ &= [D_{sk}(\alpha_n), D_{sk}(\alpha_{n-1}), \ldots, D_{sk}(\alpha_1)].\end{aligned}$$

At the core of our model, when two binary digit $b_0 = 0$ and $b_1 = 1$ are homomorphically encrypted as $c_0 = E_{pk}(b_0)$ and $c_1 = E_{pk}(b_1)$, the addition and multiplications of c_0 and c_1 can

Table 1
Examples of homomorphic addition and multiplication operations, and the corresponding plaintext expressions.

Fully homomorphic expressions with ciphertexts	Corresponding plaintext expressions	Results
$\underbrace{c_0 + c_0 + c_0 + \cdots + c_0}_{n \text{ times}}$	$\underbrace{b_0 + b_0 + b_0 + \cdots + b_0}_{n \text{ times}}$	0
$\underbrace{c_1 \times c_1 \times c_1 \times \cdots \times c_1}_{n \text{ times}}$	$\underbrace{b_1 \times b_1 \times b_1 \times \cdots \times b_1}_{n \text{ times}}$	1
$c_1 \times c_0 + c_1 \times c_1 \times c_1$	$b_1 \times b_0 + b_1 \times b_1 \times b_1$	1

Table 2
XOR truth table.

Input		Output
A	*B*	*A* XOR *B*
0	0	0
0	1	1
1	0	1
1	1	0

Table 3
AND truth table.

Input		Output
A	*B*	*A* AND *B*
0	0	0
0	1	0
1	0	0
1	1	1

Table 4
Truth table of the full adder circuit.

Inputs			Outputs	
A	*B*	C_{in}	C_{out}	*S*
0	0	0	0	0
1	0	0	0	1
0	1	0	0	1
1	1	0	1	0
0	0	1	0	1
1	0	1	1	0
0	1	1	1	0
1	1	1	1	1

be performed an arbitrary number of times. The results of such computation, when decrypted, will be the same as performing similar computation on the plaintext b_0 and b_1. Some examples of such homomorphic computation are shown in Table 1.

The homomorphic addition and multiplication operations that can be applied on encrypted binary digits allow us to construct AND and XOR circuits with encrypted bits as inputs and outputs. From the truth tables of the XOR operation (Table 2), it can easily be seen that the addition of two binary digits without carry produces a result which is identical to the output obtained by applying these two bits as inputs to the XOR circuit. Similarly, the truth table of the AND operation (Table 3) shows that passing two bits as inputs to the AND circuit will produce an output which is the same as multiplying these two bits together. With homomorphic encryption, we can add or multiply encrypted bits, making it possible to securely perform not only the XOR and AND operations, but also other circuits as long as they are based on the XOR and AND circuits.

We can understand homomorphic addition of two encrypted fixed-point binary numbers by studying how they are added in plaintext, when the bits from each operands are added from the least significant to the most significant bit. A typical way to perform this binary addition is using a full adder circuit, which adds binary digits and accounts for values carried in and carried out. A diagram of a one-bit full adder circuit is shown in Fig. 7, its truth table is shown in Table 4, in which *A* and *B* are the operands and C_{in} is a bit carried in from the previous binary addition. The circuit produces a two-bit output, comprising of a carry out C_{out} and a sum *S*.

From the truth table of the full adder circuit, the sum S_i and the carry out bit C_i of an addition of two binary digits A_i and B_i can be described as follows (with $\oplus$, $\times$ and $+$ denote XOR, AND and OR respectively):

$$S_i = A_i \otimes B_i \otimes C_{\text{in}}$$
$$\begin{aligned} C_{\text{out}} &= (C_{\text{in}} \times (A_i \otimes B_i)) + (A_i \times B_i) \\ &= (C_{\text{in}} \times (A_i \otimes B_i)) \otimes (A_i \times B_i). \end{aligned}$$

These equations above show that the addition of two binary digits with a carry out can be performed with only XOR and AND circuit.

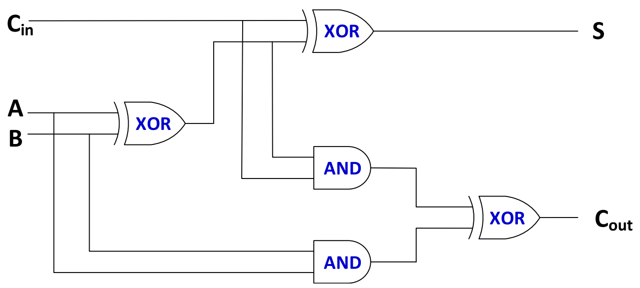


Fig. 7. The full adder circuit.

We have also shown that the XOR and AND circuit can be evaluated securely with homomorphic encryption. Therefore, the full adder circuit can also take encrypted input bits and an encrypted carry in and produce an encrypted sum and carry out bit.

Adding two encrypted fixed-point binary numbers is equivalent to adding two arrays of encrypted binary digits with all carry bits taken into account. This can be achieved by using a ripple-carry adder [16], which can combine multiple full adders to add *n*-bit numbers. Because a full adder inputs a carry in bit, which can be a carry out bit of a previous binary addition, many full adder can be chained together, so that the carry out bit from one full adder is the carry in bit of the next full adder, as shown in Fig. 8. A ripple carry adder circuit can work with encrypted inputs and produce an encrypted result because its building blocks are full adders, which can be securely evaluated. Furthermore, the fully homomorphic nature of the underlying cryptographic scheme allows the encrypted carry out bit to be passed to other full adders an arbitrary number of times without affecting the accuracy of the final result.

4. Experiments and analysis

In our experiments, we used data from the Smart* project [17] in which smart grid data were continuously collected from three households in every second within two months. The authors also collected electricity usage data every minute from 400 anonymous houses for one day. This data together with some more data generated by us allowed various application scenarios to be simulated in which a smart meter in each house hold will send a total energy consumption in 15 min, allowing us to calculate the number of homomorphic addition operations and measure

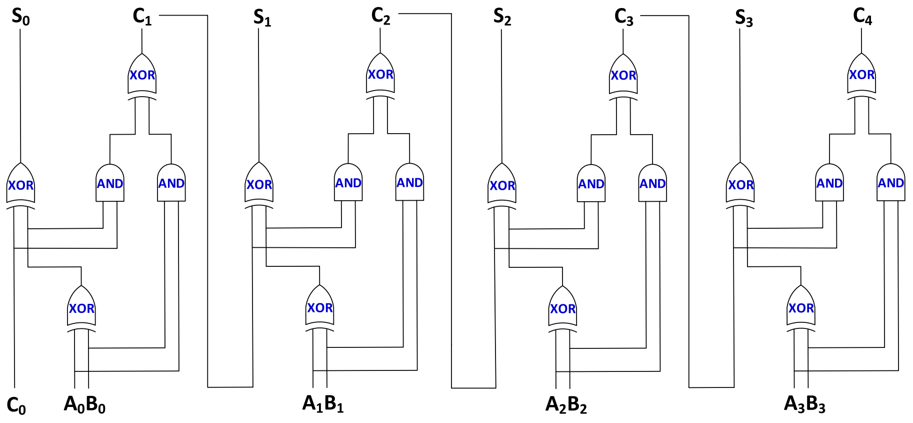


Fig. 8. A 4-bit ripple carry adder.

the time required by the cloud to calculate the total energy consumption in a given time period.

4.1. Data structure and key generation

Before conducting experiments, we assumed that each smart meter reading is represented by a decimal number with two decimal places. Hence, we wanted all computing operations to be accurate to two decimal places. This was achieved by setting the resolution $\varepsilon = 0.001$. Hence, the number of bits QF used to represent the fractional part was calculated as:

$$\boxed{\mathrm{QF} = \mathrm{ceiling}\left(\log_2\left(\frac{1}{\varepsilon}\right)\right) = \mathrm{ceiling}\left(\log_2\left(\frac{1}{0.001}\right)\right) = 10}$$

Using another 10 bits to present the integral part, we could compute with all decimal numbers from −512 to 511 up to two decimal places. We assumed that overflow would not occur by selecting numbers and calculating results within this range. This experimental setting allowed us to encode each smart meter reading as a 20-bit binary number, which was subsequently encrypted bit-by-bit using the Smart and Vercauteren [13] homomorphic encryption scheme.

To test the feasibility of our model, we wrote two software modules using the C programming language to represent a client and the cloud. The client module was responsible for encryption and decryption tasks while homomorphic computing operations were performed by the cloud module. The cloud would use public keys to perform fully homomorphic operations and return encrypted results to clients. Each smart meter was identified on the cloud using a smart meter identification number. The readings were encrypted using homomorphic encryption while the ID and timestamp were stored in plaintext on the cloud. Table 5 shows how our experiment data was stored on the cloud.

The key generation, encryption and decryption algorithms were implemented using the Scarab homomorphic cryptography library [18]. This C library is the implementation of the Smart and Vercauteren [13] homomorphic encryption scheme. Other C libraries were also used to store and process large integers such as the GNU Multiple Precision Arithmetic Library [19] and the Fast Library for Number Theory [20]. Our model and these libraries were run on a machine having 4 Gigabytes of memory and 2.4 Gigahertz quad core processor with Ubuntu Linux as the operating system.

Homomorphic public and private keys were generated on the client side. We measured the time required to generate the keys as well as the encryption and decryption times on the client, while homomorphic addition operations happened on the cloud. Table 6 shows these measurements. From this table, it can be seen that key generation took more time to execute than other operations because the key generation algorithm had to search for appropriate random numbers to construct the keys. Encryption and decryption times required to compute over 20-bit numbers were within reasonable limits. Although a homomorphic addition over 20-bit numbers took more time than encryption and decryption, this operation would be done on the cloud, which has far more computing resource than the client side.

Table 5
Encrypted smart meter data stored on the cloud.

Smart meter ID	Timestamp	Encrypted readings
SM001	03:00 am 16/04/2014	dikY3kdkzos322354
SM566	19:15 pm 17/05/2014	ieosILC23Kslskeisld
SM019	21:30 am 19/10/2013	802edisEIWDkdisle
SM005	07:45 pm 20/06/2014	987122kdiesk8392
SM302	12:15 am 19/12/2013	powodie87349874d

Table 6
Time measured for key generation, encryption, decryption and homomorphic addition operations.

Operation	Time
Key generation (client)	From 5 to 25 (s)
Encryption (20 bits, client)	244.8 (ms)
Decryption (20 bits, client)	314.4 (ms)
Homomorphic addition (20 bits, cloud)	65.6 (ms)

4.2. Homomorphic billing computation in different billing periods

In this experiment, we measured the time required to calculate electricity bills for a household in different billing periods such as in a fortnight, a month and every three months. Assuming that the smart meter would send a reading to the cloud every 15 min, we calculated the number of homomorphic additions that would happen on the cloud corresponding to each period. Table 7 presents the number of homomorphic additions corresponding to each period. The grid operator would first send a billing request to the cloud, containing the smart meter ID and the time period in which the bill must be calculated. The cloud would use the smart meter ID to identify all the readings recorded by that meter and use the timestamps to retrieve appropriate encrypted readings to perform homomorphic addition operations. Finally, the cloud would send encrypted results back to the grid operator. Table 7 also shows the time required by the cloud to perform the number of homomorphic addition operations corresponding to each billing period.

Table 7
Time taken to perform homomorphic addition operations to calculate total electricity usage for a smart meter in different billing periods, assuming that a smart meter sends four readings to the cloud in an hour.

Billing period	Number of additions	Time (s)
Fortnightly	1343	88.1
Monthly	2879	188.86
Quarterly	8639	566.72

Table 8
Time taken to perform homomorphic addition operations to calculate the total electricity usage for different numbers of smart meters in one week, assuming that a smart meter sends four readings in an hour.

Number of meters	Number of additions	Time (s)
10	6710	440.18
50	33 550	2200.88
100	67 100	4401.76

4.3. Homomorphic billing computation for different numbers of smart meters in a fixed billing period

In the second experiment, we measured the execution time of the homomorphic addition operations performed by the cloud to calculate the total electricity usage for different numbers of smart meters in one week, assuming that a smart meter sends four readings to the cloud in an hour. In this experiment, we used a different number of smart meters, i.e. 10, 50 and 100 m each time the experiment is run. Table 8 shows the numbers of smart meters used and the corresponding number of homomorphic addition operations required to calculate the bills for each group of meters in one week. For each group of smart meters, the cloud would find all identification numbers of the meters in the group, read the timestamps to retrieve appropriate encrypted readings and perform homomorphic addition operations on these encrypted data. Table 8 also shows the time required by the cloud to perform the number of homomorphic addition operations corresponding to each group of smart meters in one week. From this table, it can be seen that the more meters involved in a computation, the more time it will take. The cloud can speed up the homomorphic computation by using caching. Specifically, it can store the encrypted result of a homomorphic computation of two readings and reuse that encrypted result later when the same timestamps of those encrypted readings appear again in a computing request.

5. Performance of our algorithm on multi-core cloud servers

Our research work can be extended by creating a parallel version of our algorithm so that the calculation of total energy usage on homomorphically encrypted data can be performed more efficiently on multi-core cloud servers. In the experiments described previously, the total energy usage was calculated by adding a series of homomorphic additions sequentially on encrypted numbers. This method is not efficient because only one addition can be performed at a time. The computation time can be reduced by applying an algorithm that can compute the homomorphic sum in parallel. In this research, we use a parallel addition algorithm described by Blelloch et al. [21]. Suppose that the cloud needs to calculate the sum of a sequence S of n homomorphically encrypted numbers:

$$S[1] + S[2] + \cdots + S[n].$$

The computation can be performed in parallel by pairing and adding each element of S having an even index with the next element of S having an odd index, i.e. S[0] is paired with S[1], S[2] with S[3], and so on. The result is a new sequence of $\lceil n/2 \rceil$ numbers that sum to the same value as the sum that we wish to compute. The pairing and summing stage is repeated until, after $\lceil \log_2 n \rceil$ steps, producing a sequence consisting of a single value which is the final sum. Hence, applying this algorithm allows the execution time to be reduced significantly because many addition operations can be done in parallel. Furthermore, we want to determine how much of the computing workload can be done in parallel, especially when we increase the number of terms in a homomorphic addition expression. We demonstrate our point with the following example in which we want to add a homomorphic expression with eight terms:

$$\begin{aligned} & x_1 + x_2 + x_3 + x_4 + x_5 + x_6 + x_7 + x_8 \\ & \quad = x_{12} + x_{34} + x_{56} + x_{78} \\ & \quad = x_{1234} + x_{5678} \\ & \quad = X. \end{aligned}$$

Assume that one homomorphic expression takes a time t to complete, then the above expression will take time $7t$ to complete when the computing operations are done sequentially, i.e. one homomorphic addition at a time. However, when applying the Blelloch parallel addition algorithm described above, many homomorphic additions can be executed in parallel, for example, $x_{12} = x_1 + x_2$, $x_{34} = x_3 + x_4$ or $x_{1234} = x_{12} + x_{34}$ . . . . Hence, the total time required to calculate X is $3t$, rather than $7t$ as in the sequential case. We write a program to repeat the calculations above with an increasing number of terms in the homomorphic expression. Table 9 shows our results. From this table, when the number of terms in a homomorphic expression increases, i.e. 10, 50, 100, 500 and 1000, the fraction of computing operations that can be performed in parallel also increases significantly, i.e. 55.56%, 87.76%, 92.93%, 98.2%, 99%. When there are 1000 terms in a homomorphic addition expression, almost all of the computing workload can be done in parallel according to the algorithm we described previously. However, in reality, how much of the computing workload can be executed in parallel also depends on the number of computing threads which are allocated for that operations.

Next, we want to measure how the aforementioned parallel homomorphic addition algorithm can improve the efficiency of the computation of the total energy usage on encrypted smart meter data, especially when the algorithm is run on a multi-core cloud server. To quantify the performance enhancement, we use a metric called speedup, described in [22]. Speedup is defined by the following formula:

$$S = \frac{T_{old}}{T_{new}}$$

in which, S is the resultant speedup, T_{old} is the old execution time before any improvement is made to the algorithm, for example, when using the sequential homomorphic addition algorithm, and T_{new} is the new execution time after improvements are made over an algorithm, for example, when the homomorphic addition algorithm is parallelized.

Furthermore, according to Amdahl's law [23], the speedup S of an algorithm running on a multi-core server depends on the number of threads of execution, and most importantly, on the sequential fraction of the algorithm. Specifically, let $n \in \mathbb{N}$ be the number of threads of execution and $B \in [0, 1]$ be the fraction of the algorithm that is strictly serial, then the time $T(n)$ the algorithm would take to finish when running on n threads of execution is calculated by the following formula:

$$T(n) = T(1)\left(B + \frac{1}{n}(1 - B)\right).$$

Table 9
When the number of terms in a homomorphic expression increases, the fraction of homomorphic addition workload that can be performed in parallel also increases significantly. The symbol t represents the time required to complete one homomorphic addition operation.

Number of terms in a homomorphic expression	Time required (serial case)	Time required (parallel case)	Fraction of computing performed in parallel (%)	Fraction of computing performed sequentially (%)
10	$9t$	$4t$	55.56	44.44
50	$49t$	$6t$	87.76	12.24
100	$99t$	$7t$	92.93	7.07
500	$499t$	$9t$	98.2	1.8
1000	$999t$	$10t$	99.00	1.00

Table 10
The speedups achieved when the parallel fraction of the homomorphic addition algorithm is executed by some common processors having different number of cores and threads. These processors are usually found in desktop computers and cloud servers. These results coming from parallel performance calculations with a varying number of terms in a homomorphic addition expression, i.e. 50, 100, 500, 1000 terms as shown in Table 9. The corresponding maximum theoretical speedups are also shown for comparison purpose.

Processor	Cores	Threads	The computed speedup when using different numbers of terms in a homomorphic expression			
			50 (terms)	100 (terms)	500 (terms)	1000 (terms)
Intel Core Solo Processor U1500	1	1	1	1	1	1
Intel Core i3-4030U Processor	2	4	2.93	3.30	3.80	3.88
Intel Xeon Processor E5-2630	6	12	5.11	6.75	10.02	10.81
Sun Microsystems UltraSPARC T2	8	64	7.35	11.73	29.99	39.26
Maximum theoretical speedup			8.17	14.14	55.56	100

Hence, the speedup is calculated as:

$$S(n) = \frac{T(1)}{T(n)} = \frac{T(1)}{T(1)\left(B + \frac{1}{n}(1 - B)\right)} = \frac{1}{B + \frac{1}{n}(1 - B)}.$$

Amdahl's law also allows us to find the maximum speedup despite the maximum number of threads available. The maximum theoretical speedup is calculated by letting n, the number of threads of execution, go to infinity, as in the following equation:

$$\lim_{n\to\infty} S(N) = \lim_{n\to\infty} \frac{1}{B + \frac{1}{n}(1 - B)} = \frac{1}{B}.$$

However, the number of threads of execution n is dependent on the number of cores of a server's Central Processing Unit (CPU). Only one thread can be served by one core at a time and the CPU will switch between threads if the number of threads generated by a program is larger than the number of cores of the CPU. In the year 2002, Intel corporation introduced a technology called Hyper-Threading in its Xeon server processors and Pentium 4 desktop CPUs, allowing two threads to be run per core. Table 10 shows the speedup that we calculated when the parallel computation is performed by many processors having different number of cores and threads. This table demonstrates how an increase of speedup is determined by the size of the homomorphic expression as well as the number of threads used for parallel computation. Because of a limited number of threads used for parallel computation task, the actual speedup is always smaller than the maximum theoretical speedup. From the table, the maximum theoretical speedup when parallelizing the homomorphic addition of 1000 terms in an expression is 100 times. However, if 64 threads are used for such parallel homomorphic addition, then the actual speedup is 39.26 times. This speedup will get closer to the maximum when a larger number of threads is used.

6. Conclusion

In this research, we have designed and implemented a secure cloud-based data storage and processing model which can preserve user privacy and the confidentiality of smart meter data on a smart grid. Our research ensures that any smart grid can fully benefit from various cloud storage and processing services. This is made possible by our homomorphic computing model using a homomorphic asymmetric key cryptosystem to encrypt data, allowing the cloud to perform most of the computing works directly on encrypted data, specifically, the calculation of customer bills based on the aggregation of encrypted smart meter readings using fixed-point number arithmetics. With practical data from the Smart project, we have done many experiments to estimate the number of homomorphic additions to be performed on the cloud and measured the computation time in various billing periods. Our experiments show several factors that can influence the homomorphic computation time on the cloud such as the length of a billing period, the number of meters involved, or directly by the number of homomorphic addition operations. We also propose a parallel version of our billing algorithm to utilize the processing capacity of multi-core processors in cloud servers, reducing the majority of computation time compared to our sequential algorithm. We also demonstrate by many examples how an increase of speedup is determined by the size of the homomorphic expression as well as the number of threads used for parallel computation. In the future, we will work on more efficient methods to allow the cloud to further reduce the homomorphic computation time as well as more efficient and scalable cloud computing services to process encrypted data.

References

[1] M. Zeifman, K. Roth, Nonintrusive appliance load monitoring: Review and outlook, IEEE Trans. Consum. Electron. 57 (1) (2011) 76–84.
[2] M. Weiss, A. Helfenstein, F. Mattern, T. Staake, Leveraging smart meter data to recognize home appliances, in: 2012 IEEE International Conference on Pervasive Computing and Communications (PerCom), IEEE, 2012, pp. 190–197.
[3] S.K. Ng, J. Liang, J.W. Cheng, Automatic appliance load signature identification by statistical clustering.
[4] M. Akbar, D.Z.A. Khan, Modified nonintrusive appliance load monitoring for nonlinear devices, in: Multitopic Conference, 2007. INMIC 2007. IEEE International, IEEE, 2007, pp. 1–5.
[5] E.L. Quinn, Smart metering and privacy: Existing laws and competing policies, SSRN eLibrary.
[6] C.-I. Fan, S.-Y. Huang, W. Artan, Design and implementation of privacy preserving billing protocol for smart grid, J. Supercomput. 66 (2) (2013) 841–862.
[7] G. Acs, C. Castelluccia, I have a dream!(differentially private smart metering), in: Information Hiding, Springer, 2011, pp. 118–132.
[8] A. Rial, G. Danezis, Privacy-preserving smart metering, in: Proceedings of the 10th Annual ACM Workshop on Privacy in the Electronic Society, ACM, 2011, pp. 49–60.

[9] Powercor, Everything you should know about smart meters, https://www.powercor.com.au/media/1426/about-smart-meters-june-2012.pdf (accessed: 09.06.2014).
[10] S. Ruj, A. Nayak, A decentralized security framework for data aggregation and access control in smart grids, IEEE Trans. Smart Grid 4 (1) (2013) 196–205.
[11] P. Deng, L. Yang, A secure and privacy-preserving communication scheme for advanced metering infrastructure, in: Innovative Smart Grid Technologies (ISGT), 2012 IEEE PES, IEEE, 2012, pp. 1–5.
[12] F.D. Garcia, B. Jacobs, Privacy-friendly energy-metering via homomorphic encryption, in: Security and Trust Management, Springer, 2011, pp. 226–238.
[13] N.P. Smart, F. Vercauteren, Fully homomorphic encryption with relatively small key and ciphertext sizes, in: Public Key Cryptography–PKC 2010, Springer, 2010, pp. 420–443.
[14] E.L. Oberstar, Fixed-point representation & fractional math, Report Oberstar Consulting.
[15] M.G. Kaosar, R. Paulet, X. Yi, Fully homomorphic encryption based two-party association rule mining, Data Knowl. Eng. 76 (2012) 1–15.
[16] N. Burgess, Fast ripple-carry adders in standard-cell cmos vlsi, in: 2011 20th IEEE Symposium on Computer Arithmetic (ARITH), IEEE, 2011, pp. 103–111.
[17] S. Barker, A. Mishra, D. Irwin, E. Cecchet, P. Shenoy, J. Albrecht, Smart*: An open data set and tools for enabling research in sustainable homes, SustKDD, August.
[18] H. Perl, The scarab fully homomorphic cryptography library (Nov. 2011). URL https://hcrypt.com//scarab-library/.
[19] GNU, The gnu multiple precision arithmetic library (Mar. 2014). URL https://gmplib.org/.
[20] W. Hart, Flint: Fast library for number theory (Dec. 2013). URL http://flintlib.org/.
[21] G.E. Blelloch, B.M. Maggs, Parallel algorithms, in: Algorithms and Theory of Computation Handbook, Chapman & Hall/CRC, 2010, pp. 25–27.
[22] J.L. Hennessy, D.A. Patterson, Computer Architecture: A Quantitative Approach, Elsevier, 2012.
[23] D.P. Rodgers, Improvements in multiprocessor system design, in: ACM SIGARCH Computer Architecture News, Vol. 13, IEEE Computer Society Press, 1985, pp. 225–231.

Vu Mai is a Ph.D. Student at RMIT University. His research interests are cloud security, homomorphic cryptography and applications.

Ibrahim Khalil is a staff of National ICT Australia (NICTA), Victoria Research Laboratory, Melbourne, Australia. Ibrahim obtained his Ph.D. in 2003 from the University of Berne in Switzerland. He has several years of experience in Silicon Valley based companies working on Large Network Provisioning and Management software. He also worked as an academic in several research interests in access control, network security, scalable efficient computing in distributed systems and wireless body sensor networks.