Anonymous Reporting by Smart-Meters using Homomorphic Encryption

Michael Dubell
Emelie Widegren

Group 2

November 18, 2016

ABSTRACT

The deployment of the smart grid has led to a big number of advantages in terms of the ability of monitoring, controlling and predicting energy use. Nonetheless, along with the benefits of the smart grid comes the drawbacks related to the privacy of the customers. The ability to monitor the customers' energy usage might reveal their personal habits and behaviour including for example which electrical appliances they are using and whether or not they are home. In this report, we present an approach to protect the customers' privacy using homomorphic encryption. The use of homomorphic encryption allows the energy supplier to perform operations on the data even though it is still encrypted to preserve the customers' privacy. In addition to earlier work we will investigate the feasibility of using these techniques on platforms with less resources than a regular computer by using Raspberry Pi's.

Contents

Chapter 1

Introduction

Using smart-meters can help many people better understand their energy consumption in order to reduce cost. However when sending the energy reports to the energy supplier, the energy supplier has the ability to create patterns of usage for people by mapping their use of electricity throughout the day and which appliances they are using at any given time [1], [2]. The energy consumption can also reveal if anyone is at home at any moment which for example, can be used for break-ins by burglars'. Other information that can be revealed by the costumers' power consumption includes the number of residents in a household, the location of a resident inside the home and even identifying the TV channel or movies being watched [3].

The ability to better understand customers' energy consumption is one of the major benefits provided by the smart grid but at the same time one of the biggest concerns from a privacy viewpoint. The more data the users are willing to share the smarter decisions can be made for optimising profits. However, more accessible information usually means more privacy leaks. Balancing these interest is not an easy task. Additionally, energy suppliers may outsource the information management to a third party which may result in some loss of control of ensuring information confidentiality and integrity.

In order to address these privacy issues of smart meters, some different approaches have been proposed [4], [5], [6], [7], [8], [9]. The approach proposed by Li et al. [4] constructs a spanning tree rooting at the collector device to perform data aggregation at all smart meters by combining child node packets and sending the resulting packet to its parent. Furthermore, homomorphic encryption is used to protect the privacy of the data so that any intermediate or final results are not revealed. However, according to Baumeister [10] there are no evidence that indicates that this method would yield better results than not performing data aggregation. Baumeister claims that even though the use of homomorphic encryption reduces the computation load on the devices in the network it is at the cost of aggregation efficiency.

In this report, we present an additional approach to address this issue by using homomorphic encryption to preserve the customers' privacy.

1.1 Scope

Our approach to this problem will assume that both the network and all smart meters are 100% reliable. The scope of our study is to investigate the technique and resources needed for it, and we therefore do not need the real world environment or data. Furthermore, our system is implemented over wired TCP/IP communication.

1.2 Outline

First, we will present some background information in Chapter 2. Then we describe the system design in Chapter 3. In Chapter 4, we present our implementation. In Chapter 5, we discuss our implementation and in Chapter 6 we present some concluding remarks and mention possible future work.

Chapter 2

Background

This chapter introduces some background knowledge of the smart grid as well as different encryption techniques and cryptosystems.

2.1 Smart Grid

When we consider privacy issues in the smart grid, there are some important aspects that need to be considered and that cannot be overseen. One of which is that the electricity producer needs accurate data about how much electricity to produce at which moment, and how much capacity it needs to keep in reserve. Additionally, the electricity producer needs aggregate usage data of the individual customers for billing purposes. Furthermore, the grid operator, i.e. the company that controls the infrastructure for the distribution and transportation of electricity, also needs accurate data about electricity flows and status information about essential grid components in order to optimise their networks.

Another important stakeholder is the actual consumer. We want the consumers to be informed and aware of their energy consumption and its costs with the hope that this might lead to energy savings.

The main question here is how much information must be revealed in order to achieve the needs of the electricity producer, grid operator and the consumer? We do not want to leave out more information than necessary to the different parties. As an example, some consumers might want to follow their energy consumption in order to lower their billing but they may not want their electricity producer to follow their habits and behaviours.

In order to fulfill the different needs of the parties, we therefore present *homomorphic encryption*, introduced in Chapter 1 and explained more thoroughly in Section 2.4, that allows operations on ciphertexts, without having to decrypt them [11].

2.2 Public-Key Encryption

Public-key encryption, also known as asymmetric encryption, requires two separate keys: one public key and one private key. The encryption and decryption rely on different keys. If the goal of using public-key encryption is to ensure confidentiality so that only the appointed receiver can read the message, the plaintext shall be encrypted with the receiver's public key and the ciphertext shall be decrypted with the receiver's private key. Furthermore, public-key encryption can also be used to guarantee authenticity of the message that the sender cannot deny having sent the message.

Public-key encryption is based on hard to solve mathematical problems such as factorisation of large integers or computing a discrete logarithm in a large group [12]. It should be computationally infeasible for an adversary to determine the private key knowing the public key, or to recover the plaintext knowing the public key and the corresponding ciphertext.

2.3 Threshold Encryption

In modern cryptography, most schemes have been developed for a scenario with one sender and one receiver. These types of schemes are built upon the assumption that the sender and receiver trust each other. In threshold encryption, we do not fully trust a unique person, but possibly a pool of individuals.

In threshold public-key cryptosystems, this is done by sharing the decryption key corresponding to a public key among a set of n users. A ciphertext can then only be decrypted if at least t users cooperate. No information about the plaintext is leaked below this threshold t [13].

2.4 Homomorphic Encryption

In contrary to ordinary encryption, where a receiver needs to decrypt a message in order to perform operations on it, homomorphic encryption is a form of encryption that allows computations directly on ciphertexts. When decrypting this ciphertext the resulting plaintext will match the result of performing the computation on the plaintexts.

The most common definition is the following: Let M denote the set of the plaintexts and C denote the set of the ciphertexts. An encryption scheme is said to be homomorphic if for any given encryption key k the encryption function E satisfies

$$\forall m1, m2 \in M,\ E(m1 \odot_M m2) \leftarrow E(m1) \odot_C E(m2)$$

for some operators $\odot_M$ in M and $\odot_C$ in C, where $\leftarrow$ means "can be directly computed from", that is, without any intermediate decryption [12].

2.4.1 Somewhat/Fully Homomorphic Encryption

A homomorphic encryption scheme can also be categorised into somewhat homomorphic or fully homomorphic based on the range of functions it can be applied to.

A somewhat homomorphic encryption scheme, is a scheme that support only a limited number of homomorphic operations, e.g., only additive homomorphism, or multiplicative homomorphism up to a certain degree. Fully homomorphic encryption schemes do not have any constraint regarding the circuit depth and can evaluate arbitrary functions [11].

When considering this property of fully homomorphic encryption it should make it a powerful tool in constructing various privacy preserving systems but in reality it comes with a heavy overhead which makes it unpractical [14]. With regard to the efficiency aspect, somewhat homomorphic encryption, can be much faster and more compact than fully homomorphic encryption.

Some well-known homomorphic encryption schemes include: RSA [15], ElGamal [16], Paillier [17], Naccache-Stern [18], Boneh-Goh-Nissim [19], etc.

In this work, the additive homomorphic property is desirable and that is why we adopt the Elliptic Curve ElGamal cryptosystem [20].

2.5 ElGamal Encryption Scheme

The ElGamal cryptosystem is an asymmetric key encryption algorithm for public-key cryptography which is based on the Diffie–Hellman key exchange [21]. It can be defined over any cyclic group G and its security depends upon the difficulty of a certain problem in G related to computing discrete logarithms.

Let $y = g^x$ be the intended recipient's public key, where x is a private key and g is an element of G, then a plaintext m is encrypted by multiplying it by y^k mod p where k is randomly selected by the sender. The sender transmits this product $c2 = my^k$ mod p and also $c1 = g^k$ mod p to the recipient who uses her private keys to compute

$$h = c_1^x \equiv g^{kx} \equiv y^k \pmod{p}, \text{ and then}$$
$$c_2 \cdot h^{-1} \equiv (my^k)(y^k)^{-1} \equiv my^k \cdot y^{-k} \equiv m \pmod{p}.$$

An eavesdropper who wishes to recover m needs to calculate y^k mod p. The encryption algorithm is described in Algorithm 1 and the decryption algorithm in

Algorithm 2 [22].

Algorithm 1 Basic ElGamal encryption

```
INPUT: DL domain parameters ($p$,$q$, $g$), public key $y$, plaintext $m \in [0, p-1]$.
OUTPUT: Ciphertext ($c_1$, $c_2$).
  1. Select $k \in_R [1, q-1]$.
  2. Compute $c_1 = g^k$ mod $p$.
  3. Compute $c2 = m \cdot y^k$ mod $p$.
  4. Return($c_1$, $c_2$).
```

Algorithm 2 Basic ElGamal decryption

```
INPUT: DL domain parameters ($p$,$q$, $g$), private key $x$, ciphertext ($c_1$, $c_2$).
OUTPUT: Plaintext $m$.
  1. Compute $m = c_2 \cdot c_1^{-x}$ mod $p$.
  2. Return($m$).
```

2.6 Elliptic Curve

An elliptic curve is the set of points that satisfy a specific mathematical equation. More formally, let p be a prime number, and let $\mathbb{F}_p$ denote the field of integers modulo p. An elliptic curve E over $\mathbb{F}_p$ is defined by an equation of the form

$$y^2 = x^3 + ax + b,$$

where $4a^3 + 27b^2 \neq 0$ [23]. All points (x, y) which satisfies the above equation plus a point at infinity, ∞, lies on the elliptic curve.

For example, if E is an elliptic curve over $\mathbb{F}_7$ with defining equation

$$y^2 = x^3 + 2x + 4,$$

then the points on E are

$$E(\mathbb{F}_7) = \{\infty, (0,2), (0,5), (1,0), (2,3), (2,4), (3,3), (3,4), (6,1), (6,6)\}.$$

Adding two elliptic curve points will involve taking the line passing between them and finding the third point of intersection with the elliptic curve [22]. This concept is shown in Figure 2.1. In Figure 2.1 we have two points, P and Q, on an elliptic curve E: $y^2 = x^3 - x + 1$ and $L(x)$ is the line between them, connecting P and Q. A third intersection point of L with E is computed which we call R. Note that R cannot be the same point as P or Q. To get the final point $P + Q$, R is reflected across the x-axis.

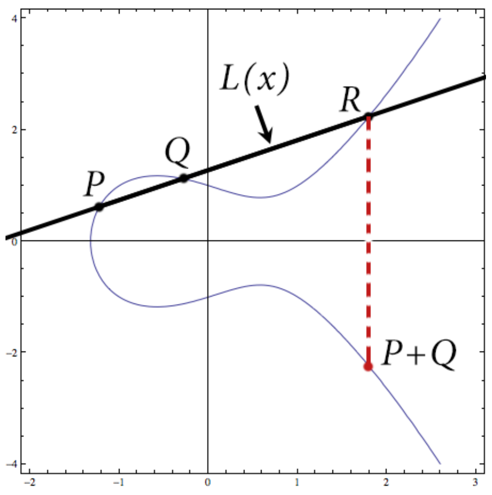

Figure 2.1: Adding the points P and Q on an elliptic curve.

2.7 Elliptic Curve Encryption

The encryption and decryption procedures for Elliptic Curve ElGamal is presented in Algorithms 3 and 4. In the algorithm, a plaintext m is first represented as a point M, and then encrypted by adding it to kQ where k is a randomly selected integer, and Q is the intended recipient's public key. Then the sender transmits the points $C_1 = kP$ and $C_2 = M + kQ$ to the recipient who uses her private key d to compute

$$dC_1 = d(kP) = k(dP) = kQ,$$

which will give her $M = C_2 - kQ$. An eavesdropper who wishes to recover M needs to calculate kQ.

Algorithm 3 Basic ElGamal elliptic curve encryption

```
INPUT: Elliptic curve domain parameters ($p$, $E$, $P$,$n$), public key $Q$,
plaintext $m$.
OUTPUT: Ciphertext ($C_1$,$C_2$).
  1. Represent the message $m$ as a point $M$ in $E(\mathbb{F}_p)$.
  2. Select $k \in_R [1, n-1]$.
  3. Compute $C_1 = kP$.
  4. Compute $C_2 = M + kQ$.
  5. Return($C_1$,$C_2$).
```

Algorithm 4 Basic ElGamal elliptic curve decryption

```
INPUT: Domain parameters ($p$, $E$, $P$,$n$), private key $d$, ciphertext ($C_1$,$C_2$).
OUTPUT: Plaintext $m$.
  1. Compute $M = C_2 - dC_1$, and extract $m$ from $M$.
  2. Return($m$).
```

The Elliptic Curve ElGamal cryptosystem's homomorphic property can be shown by letting $C_1 = (A_1, B_1)$ and $C_2 = (A_2, B_2)$ be two ciphertexts encrypting M_1 and M_2. They are then aggregated by computing,

$$C = C_1 + C_2 = (A_1, B_1) + (A_2, B_2) = (K_1P, M_1 + K_1Q) + (K_2P, M_2 + K_2Q) = [(K_1 + K_2)P, (M_1 + M_2) + (K_1 + K_2)Q] = (A_1 + A_2, B_1 + B_2).$$

The decryption of C will provide $M_1 + M_2$ as a result. The original ElGamal, described in Section 2.5, is multiplicatively homomorphic while Elliptic Curve ElGamal is transformed to additive group and hence is additively homomorphic.

Chapter 3

System design

This chapter covers a very high-level description of the system design and all of its components. Chapter 4 will go into detail about the implementation of the system and the algorithms used.

3.1 Assumptions

The system assumes an honest-adversary model, meaning that there is no malicious entity trying to sabotage the system by inserting false data or modifying the application code. Therefore the system does not contain any security aspects such as confidentiality and integrity. Furthermore, the system assumes there are no process interruptions, node or network failures.

3.2 Software

The application is written in Python 2.7 and uses a number of libraries and modules for different puroses. The main library is the petlib library [24] which provides the cryptographic operations.

3.3 Hardware

The application is executed on three different Raspberry Pi devices and one laptop. The following Raspberry Pi models are used: `Two Raspberry Pi 2 model B` and `One Raspberry Pi 1 model B`. In addition to that, one Macbook Pro (2011) is used as a fourth node. All of the computer devices are connected by a network switch, the network can only transmit data at 100 Mbit/s because the Raspberry Pi's on-board ethernet port only supports a maximum of 100 Mbit/s.

3.4 Application design

The system is divided into a client and server model as can be seen in figure 3.1. The server node, or the collector, is responsible for computing a global group key which is based on all the public keys generated by the clients. When the group key has been created the server will begin to distribute the key to all clients which will use this key to encrypt data. The server is also in charge of co-ordinating decryption of messages. In order to decrypt messages the server needs to collaborate with *all* clients to successfully decrypt a message. This is called threshold decryption and eliminates a central entity which holds the private key used for decryption. Instead, the system requires collaboration between nodes which improves the privacy for all the users.

The clients are responsible for generating *(fake)* energy readings every n seconds, then encrypt the readings with the group key computed by the server, and finally send the data to the server. In turn, the server will sum all encrypted readings for each client, and after a specified number of received readings per client the server will request decryption of aggregated result. As mentioned before, the clients needs to collaborate with the server in order to decrypt a message. A complete visualisation of the system can be found in the appendix.

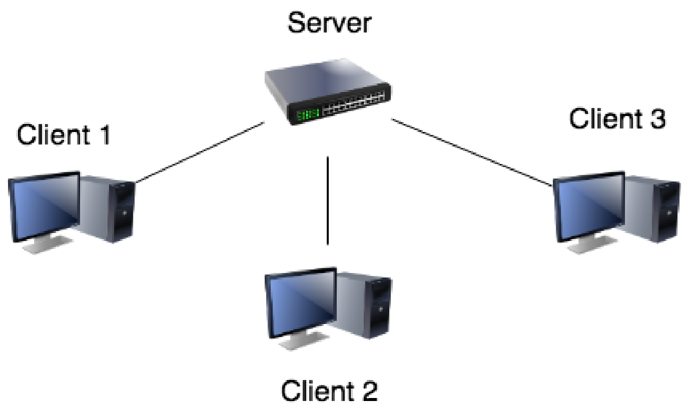

Figure 3.1: Illustration of the client-server model used.

3.4.1 Communication

There is no built-in message reliability which means nodes assume that messages sent are actually received. Each message is structured as showing in listing 3.1.

```
{"ID":[node_id], "OPERATION": [Operation to perform], "GROUP_KEY":[group_pub],"
    PUB_KEY":[node's public key], "READING": [encrypted_reading]}
```

Listing 3.1: Message structure

Chapter 4

Implementation

As mentioned in Chapter 3, the system is connected by a network switch and each node knows the IP address of every other node in the network. This is done by hard coding the IPs of every node in a special file called `node_list.txt`, which every node has access to. At system boot, each node will begin to compute their public and private key pair. Following this operation, every client will request the server to compute the group key. This is done by sending the message seen in listing 4.1.

```
{"ID":node_id, "OPERATION": "GROUP_KEY_CREATE", "PUB_KEY":node_pub}.
```

Listing 4.1: Clients requesting the group key

The client specifies the operation `GROUP_KEY_CREATE` and appends its own public key. This is necessary in order for the server to create the group key. The group key is a combination of the public keys from all the clients in the system. As mentioned in Chapter 3, all the clients need to be a part of the decryption process. Therefore it is necessary for clients to provide their public key to the server in order to create a correct group key.

Once the group key has been computed by the server, the server will broadcast a message to all clients which includes the newly created group key as well as what operation to perform. The message can be seen in listing 4.2.

```
{"ID":node_id, "OPERATION": "RECEIVE_GROUP_KEY", "GROUP_KEY":group_pub}
```

Listing 4.2: Server informing clients about the new group key

When all clients have received the new group key, they will update their local group key variable to include this key. At this point, the clients can begin generating *(fake)* energy readings, which is done every n seconds. The readings are then encrypted with the group key and sent to the server. The server will track each client's readings and create an aggregated result by simply adding each encrypted reading. In order to perform additions of ciphertexts, an additive homomorphic encryption scheme is used which is provided by petlib [24]. After the server has received a specific number of readings from a given client, e.g. 10 readings, the server will send a decryption request to the first client specified in the `node_list.txt`.

```
{"ID":given_node_id, "OPERATION": "DECRYPT_GROUP_MSG", "GROUP_KEY":group_pub, "
    READING": encrypted_reading}
```

Listing 4.3: Server requesting clients to decrypt a message

Once the first client has decrypted their part of the message, the client will forward the message with the new reading value to the next client in the list. This operation will continue in a ring like manner until the last client has received the message. During this stage, the client will decrypt its part of the message as previous clients did, and send a `DECRYPT_GROUP_FINAL` message to the server which indicates that it is the last node to decrypt. Now the server can access the decrypted aggregated value for the specific client. This operation is done for every client. Once they have been performed, a new period (e.g. a month) starts where the server will track new readings.

Chapter 5

Discussion

When designing and building systems, engineers should strive for implementing privacy enhanced technologies into their design from the beginning. As technology progresses and we start to rely more and more on digital systems in our daily lives, making sure our information is private is becoming an important aspect. As seen in the smart grid, detailed information about customers' energy consumption can be exposed which can reveal patterns of daily activities such as when someone is home. Protecting this information can be done using encryption technologies. However the collector or the energy supplier needs to be able to do different computations on the aggregated result. Therefore classical encryption algorithms such as AES is not suitable. This is what homomorphic encryption solves, it enables mathematical operations on ciphertexts. In the system presented in this paper, a partial homomorphic encryption scheme with an additive encryption property is used in order to sum aggregated energy readings. For more complex operations that requires addition, multiplication and other operations, a fully homomorphic encryption scheme (FHE) is needed. However using FHE schemes are not practical due to significant processing and encryption/decryption time. As shown in this paper, partial homomorphic encryption schemes are better suited for practical use. The Raspberry Pis used to run the application did not experience any performance issues and could handle encryption/decryption without delay. Implementing the application in a low level programming language such as C would most likely provide better performance, than using an interpreted language such as Python.

There are some limitations in the system design presented in this paper. All clients directly communicates with the server when providing energy readings. This can create a large network and local computation overhead for the server. Once the server wants to decrypt an aggregated result, the server needs to send a decryption request along with the encrypted result in a ring like manner, for each aggregated result. This too introduces large network overhead. A system design by [4] solves much of this overhead by computing a BFS tree where each node encrypts its readings with the public key of the collector and receives encrypted readings from its children. Once all readings have been received, the node sums the encrypted readings and sends the result to its parent who will perform the same action until the collector is reached. This design limits the overhead in the network and at the collector while achieving the same goal, protecting user's privacy. However the design discussed in this paper introduces a new component called *threshold decryption.* This forces the collector to collaborate with all clients in order to decrypt readings. This system can be modified to include a set of special intermediate nodes which the collector needs to collaborate with, instead of all the clients. The intermediate nodes can perform different distributed algorithms such as voting to decide whether the collector is malicious in its requests. This operation can enhance users' privacy by detecting malicious requests in the network. Of course it is possible that one of the intermediate nodes becomes malicious, in that case there needs to be a sufficient amount of intermediate nodes that can successfully handle n faulty nodes.

Chapter 6

Conclusion

Protecting users' privacy in the smart grid should be a priority by energy companies and system designers in general. As shown in this report, it is possible to limit the information about customers energy consumption and still allow the energy supplier to create accurate bills. Using homomorphic encryption schemes, we can compute data on ciphertexts to accurately compute the aggregated result without needing access to individuals' energy consumption every 10 or 15 minutes. To perform more complex operations than what partial homomorphic encryption provides, fully homomorphic encryption (FHE) is needed. However FHE requires more research until it can be used in any practical applications.

6.1 Future work

This report presents one approach for preserving privacy in the smart grid by using homomorphic encryption. It would be interesting to take this further and compare different system designs and the use of different cryptosystems. By comparing different system designs it is possible to draw more conclusions about the characteristics of our system. Furthermore, it would be interesting to see how well different cryptosystems perform depending on variables such as the number of smart meters in the neighbourhood.

Bibliography

[1] L. Yu, H. Li, X. Feng, and J. Duan, “Nonintrusive appliance load monitoring for smart homes: recent advances and future issues,” *IEEE Instrumentation Measurement Magazine*, vol. 19, pp. 56–62, June 2016.

[2] G. W. Hart, “Nonintrusive appliance load monitoring,” *Proceedings of the IEEE*, vol. 80, no. 12, pp. 1870–1891, 1992.

[3] I. Rouf, H. Mustafa, M. Xu, W. Xu, R. Miller, and M. Gruteser, “Neighborhood watch: Security and privacy analysis of automatic meter reading systems,” in *Proceedings of the 2012 ACM Conference on Computer and Communications Security*, CCS ’12, (New York, NY, USA), pp. 462–473, ACM, 2012.

[4] F. Li, B. Luo, and P. Liu, “Secure information aggregation for smart grids using homomorphic encryption,” in *Smart Grid Communications (SmartGridComm), 2010 First IEEE International Conference on*, pp. 327–332, IEEE, 2010.

[5] H. Li, R. Mao, L. Lai, and R. C. Qiu, “1 compressed meter reading for delay-sensitive and secure load report in smart grid.”

[6] A. Rial and G. Danezis, “Privacy-preserving smart metering,” in *Proceedings of the 10th annual ACM workshop on Privacy in the electronic society*, pp. 49–60, ACM, 2011.

[7] C. Efthymiou and G. Kalogridis, “Smart grid privacy via anonymization of smart metering data,” in *Smart Grid Communications (SmartGridComm), 2010 First IEEE International Conference on*, pp. 238–243, IEEE, 2010.

[8] G. Kalogridis, C. Efthymiou, S. Z. Denic, T. A. Lewis, and R. Cepeda, “Privacy for smart meters: Towards undetectable appliance load signatures,” in *Smart Grid Communications (SmartGridComm), 2010 First IEEE International Conference on*, pp. 232–237, IEEE, 2010.

[9] F. D. Garcia and B. Jacobs, "Privacy-friendly energy-metering via homomorphic encryption," in *International Workshop on Security and Trust Management*, pp. 226–238, Springer, 2010.

[10] T. Baumeister, "Literature review on smart grid cyber security," *Collaborative Software Development Laboratory at the University of Hawaii*, 2010.

[11] C. Gentry, "Computing arbitrary functions of encrypted data," *Communications of the ACM*, vol. 53, no. 3, pp. 97–105, 2010.

[12] C. Fontaine and F. Galand, "A survey of homomorphic encryption for non-specialists," *EURASIP Journal on Information Security*, vol. 2007, no. 1, pp. 1–10, 2007.

[13] I. Damgård and J. B. Nielsen, "Universally composable efficient multiparty computation from threshold homomorphic encryption," in *Annual International Cryptology Conference*, pp. 247–264, Springer, 2003.

[14] V. V. Kristin Lauter, Michael Naehrig, "Can homomorphic encryption be practical?," tech. rep., May 2011.

[15] R. L. Rivest, A. Shamir, and L. Adleman, "A method for obtaining digital signatures and public-key cryptosystems," *Communications of the ACM*, vol. 21, no. 2, pp. 120–126, 1978.

[16] T. ElGamal, "A public key cryptosystem and a signature scheme based on discrete logarithms," in *Workshop on the Theory and Application of Cryptographic Techniques*, pp. 10–18, Springer, 1984.

[17] P. Paillier, "Public-key cryptosystems based on composite degree residuosity classes," in *International Conference on the Theory and Applications of Cryptographic Techniques*, pp. 223–238, Springer, 1999.

[18] D. Naccache and J. Stern, "A new public key cryptosystem based on higher residues," in *Proceedings of the 5th ACM conference on Computer and communications security*, pp. 59–66, ACM, 1998.

[19] D. Boneh, E.-J. Goh, and K. Nissim, "Evaluating 2-dnf formulas on ciphertexts," in *Theory of Cryptography Conference*, pp. 325–341, Springer, 2005.

[20] N. Koblitz, "Elliptic curve cryptosystems," *Mathematics of computation*, vol. 48, no. 177, pp. 203–209, 1987.

[21] W. Diffie and M. Hellman, "New directions in cryptography," *IEEE transactions on Information Theory*, vol. 22, no. 6, pp. 644–654, 1976.

[22] D. Hankerson, A. J. Menezes, and S. Vanstone, *Guide to elliptic curve cryptography.* Springer Science & Business Media, 2006.

[23] M. Anoop, "Elliptic curve cryptography," *An Implementation Guide*, 2007.

[24] G. Danezis, "A python library that implements a number of privacy enhancing technolgies." `https://github.com/gdanezis/petlib`, 2014. [Online; accessed 31-Oct-2016].

Appendix A

System design visualisation

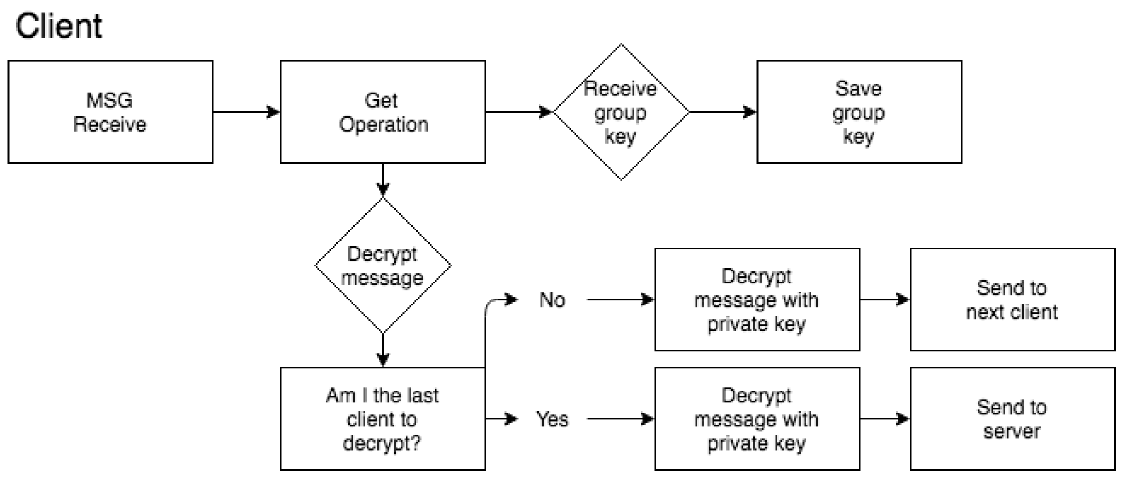

Figure A.1: Client design for decrypting messages.

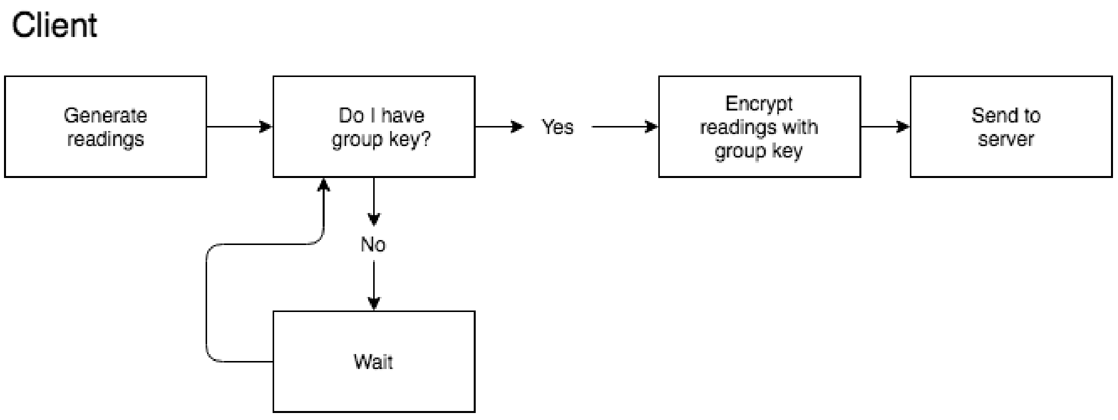

Figure A.2: Client generating energy readings.

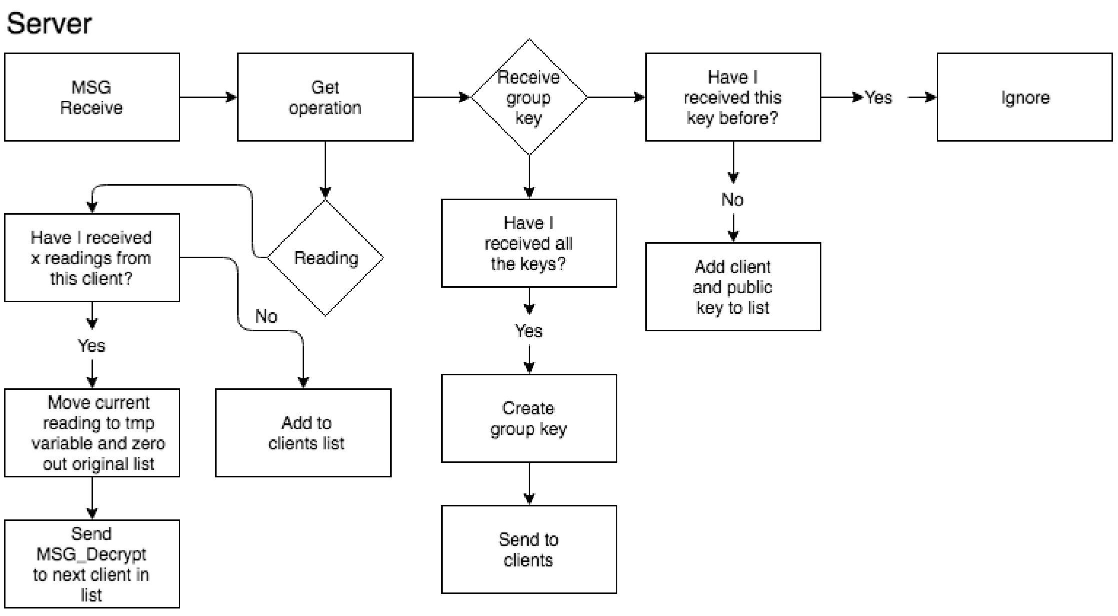
Server
MSG Receive
Get operation
Receive group key
Have I received this key before?
Yes
Ignore
No
Add client and public key to list
Have I received all the keys?
Yes
Create group key
Send to clients
Reading
Have I received x readings from this client?
No
Add to clients list
Yes
Move current reading to tmp variable and zero out original list
Send MSG_Decrypt to next client in list

Figure A.3: Server design.