MIPRO 2014, 26-30 May 2014, Opatija, Croatia

Homomorphic Encryption in the Cloud

Darko Hrestak and Stjepan Picek
Faculty of Electrical Engineering and Computing, University of Zagreb
Unska 3, Zagreb, Croatia
Email: darko.hrestak@fer.hr, stjepan@computer.org

***Abstract*—Since the first notions of fully homomorphic encryption more than 30 years ago, there has been numerous attempts to develop such a system. Finally, in 2009 Craig Gentry succeeded. Homomorphic encryption brings great advantages but it seems that, at least for now, it also brings many practical difficulties. Furthermore, in the last couple of years, several other fully homomorphic systems arose where each has its one advantages and drawbacks. However, with the developments in cloud computing, we need it more than ever to become practical for real-world usages. In this paper we are discussing the strengths and weaknesses of homomorphic encryption and we give a brief description of several promising fully homomorphic encryption systems. Next, we give a special attention to the homomorphic encryption systems for cloud computing. Finally, we discuss some recent developments by IBM and their open-source library for homomorphic encryption.**

I. Introduction

Today, cryptography plays important role in our lives, although many people are not fully aware of that fact. Whenever we buy something over the Internet or withdraw cash from ATM machine, cryptography protects our sensitive information.

One of the basic questions in cryptography concerns how it is possible to safely exchange information. If two parties, commonly called Alice and Bob, want to securely communicate, then they need a shared secret. This notion represents the basis of private key cryptography. In this setting Alice and Bob both have the same secret (called secret key) and they encrypt and decrypt data with it. All secure communication is limited to those that have that secret key. Of course, here we are simplifying the situation and assume that key and methods of encryption are appropriately chosen. If some third party, Eve, intercepts encrypted message it should be practically infeasible for her to decrypt it. Now, even from this abstract description, it is possible to observe one obvious drawback of private key cryptography. In order to be able to encrypt and afterwards decrypt the data, Alice and Bob must use the same key. If we add an additional condition that the key needs to be changed from time to time, question becomes, how to securely communicate (in order to exchange the key) before they can securely communicate to exchange the actual data. Almost 40 year ago, public key cryptography arose to solve that problem. In public key cryptography security relies on hard mathematical problems and each party needs a public key for encryption and a private key for decryption [1].

Additionally, Alice and Bob also wants for instance to be sure that the other party is he who says it is, and that the messages can not be changed while transmitted. For each of those constraints, appropriate solution is devised and implemented in practice. However, only a couple of months after the publication of the RSA algorithm paper [2], Rivest et al. asked the question whether it is possible to work with encrypted data, without the need for decrypting it first [3]. That question started the search for homomorphic encryption systems. It turned out that this question would pose unsolvable problem in following decades. Before going into more details about homomorphic encryption, we give a simple example why we would even need an option like that. If we have some information and we send it to the server for some additional processing, we may not want that the server can read our information. In the modern society, the need for privacy is present as never before and homomorphic encryption can help us in preserving it.

Homomorphic encryption allows us to perform operations on encrypted data without knowing the private key and without decrypting that data. When we decrypt the result of any operation, it is the same as if we carried out the calculation on the plain data. In the process of building a system with such capabilities, a plethora of systems were developed that offer partial functionalities. Those types of homomorphic encryption systems are known as partially homomorphic encryption systems. In 2009, Craig Gentry in his PhD thesis described the first fully homomorphic encryption system (FHE) [4]. From then, a number of systems were (and still are) developed that offer similar or better characteristics.

In Section 2 we give a brief introduction to the basic notions about homomorphic encryption. Several partially homomorphic encryption systems are described in Section 3. In Section 4 we describe fully homomorphic encryption as presented by Gentry and several other fully homomorphic encryption systems. Possible usages of fully homomorphic encryption with particular accent on cloud computing are discussed in Section 5. Finally, in Section 6 we give a conclusion.

II. Preliminaries

Here we give basic notions about homomorphic encryption and schemes presented. For a detailed introduction to general cryptographic notions we refer readers to [1].

A Homomorphic encryption is additive if:

$$Enc(x + y) = Enc(x) \cdot Enc(y). \quad (1)$$

A homomorphic encryption is multiplicative if:

$$Enc(x \cdot y) = Enc(x) \cdot Enc(y). \quad (2)$$

Next, we present building blocks of the homomorphic encryption system.

- Key generation is the algorithm that outputs public key P_k and secret key S_k.
- Encrypt function is the algorithm that takes public key P_k and a message m and outputs ciphertext c.
- Decrypt function is the algorithm that takes ciphertext c and a secret key S_k and outputs a message m.
- Evaluate function is an algorithm that takes evaluation key, circuit, a set of ciphertexts $c_1, ..., c_n$ and outputs a ciphertext c_e. Circuit represents a certain function realized with logical gates.

III. Partially homomorphic encryption systems

There are two different types of homomorphic encryption systems, fully homomorphic encryption systems and partially homomorphic encryption systems. Fully homomorphic encryption systems are defined over a ring and therefore support both operations, addition and multiplication. Partially homomorphic encryption systems are defined over a group and therefore support just a single operation on the encrypted data.

In last thirty or so years there have been several encryption systems that belong to the partially homomorphic encryption systems class. Those systems we can divide on a basis of the operation they support on additive homomorphic encryption systems and multiplicative homomorphic encryption systems. There are also systems that allow arbitrary many operations of one type and only a limited number (usually just one) of operations of other type.

A. Additive Homomorphic Encryption

Some well known examples of additive homomorphic encryption systems include Goldwasser-Micali system [5], Paillier system [7] and Damgard-Jurink's generalization of Paillier system [6].

The Paillier cryptosystem, invented by Pascal Paillier in 1999, is a probabilistic asymmetric algorithm for public key cryptography [7]. This cryptosystem is based on the decisional composite residuosity assumption (DCRA). DCRA assumption can be stated as the following: given a composite n ($n = p \times q$ for primes p and q) and an integer z, it is hard to decide whether z is a $n-$ residue modulo n^2 or not (i.e. it is hard to find out whether there exists y such that $z = y^n mod n^2$). Paillier cryptosystem computes encryption of $m_1 + m_2$ with public key and encryptions of m_1 and m_2. Short description of Paillier scheme is given in Algorithm 1. Implementation of that cryptosystem can be found online under the name "The Homomorphic Encryption Project" (thep) and it's main application is electronic voting where each vote is encrypted but only the sum is decrypted [8].

Suppose that we have two ciphers C_1 and C_2 such that following equations hold:

$$\begin{aligned} C_1 &= g^{m_1} \cdot r^n \ mod\ n^2, \\ C_2 &= g^{m_2} \cdot r^n mod\ n^2, \\ C_1 \cdot C_2 &= g^{m_1+m_2} \cdot (r_1 \cdot r_2)^n mod\ n^2. \end{aligned} \tag{3}$$

Algorithm 1 Paillier scheme

Key Generation KeyGen (p, q)
$Input : p, q \in P$
$Compute : n = p \cdot q, \lambda = lcm(p-1, q-1)$
$Choose : g \in Z^*_{n^2} \mid gcd(L(g^\lambda \ mod \ n^2), n) = 1 \ where \ L(u) = (u-1)/n$
$Compute : \mu = L(g^\lambda \ mod \ n^2)^{-1}$
$Output : (P_k, S_k)$
$Public \ key : P_k = (n, q)$
$Secret \ key : S_k = (\lambda, \mu)$
Encryption Enc (m, P_k)
$Input : m \in Z_n$
$Choose : r \in Z^*_n$
$Compute : c = g^m \cdot r^n \ mod \ n^2$
$Output : c \in Z_{n^2}$
Decryption Dec (c, S_k)
$Input : c \in Z_{n^2}$
$Compute : m = L(c^\lambda \ mod \ n^2) \cdot \mu \ mod \ n$
$Output : m \in Z_n$

It can be observed that Paillier cryptosystem realizes the property of additive homomorphic encryption since the product of two ciphertexts will decrypt to the sum of their corresponding plaintexts.

B. Multiplicative Homomorphic Encryption

When looking at the multiplicative homomorphic cryptosystem, good examples are previously mentioned RSA algorithm [3] and ElGamal encryption system [9]. Next we give a short description of the RSA scheme [3] as shown in Algorithm 2.

Algorithm 2 RSA scheme

Key Generation KeyGen (p, q)
$Input : p, q \in P$
$Compute : n = p \cdot q, \phi(n) = (p-1) \cdot (q-1)$
$Choose : e \mid gcd(e, \phi(n)) = 1$
$Compute : d \mid (e \cdot d) \equiv 1 \ mod \ \phi(n)$
$Output : (P_k, S_k)$
$Public \ key : P_k = (e, n)$
$Secret \ key : S_k = d$
Encryption Enc (m, P_k)
$Input : m \in Z_n$
$Compute : c = m^e \ mod \ n$
$Output : c \in Z_n$
Decryption Dec (c, S_k)
$Input : c \in Z_{n^2}$
$Compute : m = c^d \ mod \ n$
$Output : m \in Z_n$

Suppose that we have two ciphers C_1 and C_2 such that following equations hold:

$$\begin{aligned} C_1 &= m_1^e \ mod\ n, \\ C_2 &= m_2^e \ mod\ n, \\ C_1 \cdot C_2 &= (m_1 \cdot m_2)^e mod\ n. \end{aligned} \tag{4}$$

It can be observed that RSA as a homomorphic encryption system has some security weaknesses. For example, if we assume that two messages m_1 and m_2 correspond to ciphers C_1 and C_2, and if the client sends that pair (C_1, C_2) to the server, the server will perform multiplication of C_1 and C_2 and send it encrypted to the client. If the attacker intercepts ciphers C_1 and C_2 which are encrypted with the same private key, he will be able to decrypt all messages exchanged between the server and the client. This is due the homomorphic encryption multiplicativity where product of the ciphers equals cipher of the product.

C. Additive and multiplicative homomorphic cryptosystem

Each of the examples listed above allows homomorphic computation of only one operation (either addition or multiplication) on plaintexts. A step closer to the full homomorphic encryption systems represent the systems that allow arbitrary many operations of one type and limited number of operations of other type. Probably the best example of a cryptosystem which allows homomorphic computation of addition and multiplication but is not fully homomorphic systems is Boneh-Goh-Nissim cryptosystem [10] as shown in Algorithm 3. Boneh-Goh-Nissim cryptosystem supports evaluation of an unlimited number of additions but at most one multiplication. This encryption scheme is based on bilinear pairings on elliptic curves.

Algorithm 3 Boneh-Goh-Nissim scheme

Key Generation KeyGen (r)
$Input : r \in Z^+$
$Compute : \gamma(r)\ to\ obtain\ tuple\ (q_1, q_2, G, G_1, e)$
$Compute : n = p \cdot q$
$Choose : g, u \in Random$
$Compute : h = u^{q_2}$
$Output : (P_k, S_k)$
$Public\ key : P_k = (n, G, G_1, e, g, h)$
$Secret\ key : S_k = q_1$
Encryption Enc (m, P_k)
$Input : m$
$Choose : r \in Random$
$Compute : c = g^m \cdot h^r \in G$
$Output : c$
Decryption Dec (c, S_k)
$Input : c$
$Compute : m = dilog(c^{q_1})\ mod\ g^{q_1}$
$Output : m$

Here, $dilog$ represents discrete logarithm operation.

IV. Fully Homomorphic Encryption Systems

In this section we present several fully homomorphic encryption systems, starting with Gentry's systems and then covering several other constructions.

A. Gentry's FHE

The first construction of fully homomorphic scheme (based on ideal lattices which are ideals in algebraic number field) is described by Craig Gentry in 2009 [4]. This work is important not only because it represents the first fully homomorphic encryption system, but also since it served as a paradigm for later fully homomorphic encryption schemes as the one we describe in section below.

Gentry's scheme consists of several steps. First, one constructs a somewhat homomorphic encryption scheme (SWHE), which only supports a limited number of operations. In SWHE system there are usual Encrypt and Decrypt functions, but also an Evaluate function. Evaluate function conducts an operation on ciphertext where it is realized as a combinatorial circuit. However, the problem with homomorphic encryption is that ciphertext contains random noise where addition roughly doubles the noise and multiplication squares the noise. This noise becomes larger with successive homomorphic operations and only ciphertexts whose noise size remains below a certain threshold can be decrypted correctly. The origin of noise is in the probabilistic encryption process where we add some noise during the encryption, and remove that noise during the decryption. Because of this limitation, homomorphic encryption can not have arbitrary number of operations and is therefore called somewhat homomorphic encryption. Somewhat homomorphic encryption scheme is shown in Algorithm 4. To overcome this problem, Gentry devised *bootstrapping* method. In bootstrapping, Evaluate function is used to run Decrypt function. Since Evaluate works with encrypted data, the result is not the plaintext but a new encryption with reduced noise. This refreshed ciphertext can then be used in subsequent homomorphic operations. By repeated refreshing of the ciphertexts, the number of homomorphic operations becomes unlimited, and that results in a fully homomorphic encryption scheme. In order for system to work as described Decrypt function must not exceed the noise threshold. If this would not be the case, we would not be able to apply bootstrapping to the SWHE scheme. Indeed, in Gentry's original FHE scheme this condition was not satisfied. Therefore, he added a *squashing* procedure to the Decrypt function. Here, squashing means transforming SWHE scheme into one with same homomorphic capacity but with simpler Decrypt function [11]. The cost of the squashing method is that the key grows larger and more complicated.

B. FHE

After 2009 and Gentry's breakthrough result, many improvements have been made, introducing new variants, improving efficiency, and providing new features. Recently, Brakerski, Gentry and Vaikutanathan described a different framework where the ciphertext noise grows only linearly with the multiplicative level instead of exponentially, so that bootstrapping technique is no longer necessary to obtain a scheme supporting the homomorphic evaluation of any given polynomial size circuit. Currently, there are 3 main families of fully homomorphic encryption schemes:

Algorithm 4 Somewhat homomorphic encryption scheme

Key Generation KeyGen (R, B_i)
$Input : R, basis\ B_i$
$Compute : (B_{P_k}, B_{S_k}) = \ IdealGen\ (R, B_i)$
$Output : (P_k, S_k)$
$Public\ key : P_k = (R, B_i, B_{P_k}, S_{amp}$
$Secret\ key : S_k = B_{S_k}$
Encryption Enc (m, P_k)
$Input : m, P_k$
$Compute : e = S_{amp}(m, B_b i)$
$Compute : c = e\ mod B_{P_k}$
$Output : c \in \mathbb{R}\ mod\ B_{P_k}$
Decryption Dec (c, S_k)
$Input : c \in \mathbb{R}\ mod\ B_{P_k}$
$Compute : m = (c\ mod\ B_{S_k})\ mod\ B_i$
$Output : m$

- Gentry's original scheme based on ideal lattices [4].
- Van Dijk, Gentry, Halevi and Vaikuntanathan's (DGHV) scheme over the integers [12].
- Brakerski and Vaikuntanathan's scheme based on the Learning with Errors (LWE) and Ring Learning with Errors (RLWE) problems [13].

Since we already described Gentry's original scheme, now we give short descriptions of the other two schemes.

In DGHV scheme [12] a ciphertext has the form of

$$C = q \cdot p + 2 \cdot r + m, \tag{5}$$

where p is the secret-key, m is the bit plaintext (m is 0 or 1), q is a large random number and r is a small random number.

To decrypt, one calculates

$$m = (C\ mod\ p)\ mod\ 2. \tag{6}$$

It is easy to see that decryption works as long as the noise $2 * r$ is smaller than p.

Suppose that we have two ciphers C_1 and C_2 such that following equations hold:

$$\begin{aligned} C_1 &= q_1 \cdot p + 2 \cdot r_1 + m_1, \\ C_2 &= q_2 \cdot p + 2 \cdot r_2 + m_2, \\ C_1 + C_2 &= (q_1 + q_2) \cdot p + 2 \cdot (r_1 + r_2) + m_1 + m_2, \\ C_1 \cdot C_2 &= q_{12} \cdot p + 2 \cdot (2 \cdot r_1 \cdot r_2 + r_1 \cdot m_2 + r_2 \cdot m_1) \\ &\quad + m_1 + m_2. \end{aligned} \tag{7}$$

From these equations, we see that we can obtain the encryption of $m_1 + m_2$ $(mod\ = m_1 \oplus m_2)$ by computing $C_1 + C_2$. Similarly, encryption of $m_1 \cdot m_2$ can be obtained by computing $C_1 \cdot C_2$. However one gets a new ciphertext with noise roughly twice larger than in the original ciphertexts C_1 and C_2. Since the noise must remain below p, the number of permitted multiplications on ciphertexts is therefore limited. This is an example of somewhat homomorphic encryption scheme.

Scheme from Brakerski and Vaikuntanathan is known as "Fully Homomorphic Encryption without Bootstrapping" where their approach is based on Learning with Errors (LWE) and Ring Learning with Errors (RLWE) problems [13].

In RLWE problem for security parameter λ, let $f(x) = x^d + 1$ where $d = d(\lambda)$ is a power of 2. Furthermore, let $q = q(\lambda) \geq 2$ be an integer, $R = Z[x]/f(x)$ and $R_q = R/qR$. Let $\xi = \xi(\lambda)$ be a distribution over R. The RLWE problem is to distinguish two distributions, in the first distribution, one samples (a_i, b_i) uniformly from R_{q^2}. In the second distribution, one first selects $s < -R_q$ uniformly and then samples (a_i, b_i) $2R_q^2$ by sampling $a_i < -R_q q$ uniformly, $e_i < -\xi$ and setting $b_i = a_i s + e_i$. The RLWE assumption is that the RLWE problem is infeasible.

In this scheme the authors are able to obtain a direct construction of a bootstrappable encryption scheme without having to "squash the decryption circuit", and hence there is no need to apply a bootstrapping technique to achieve fully homomorphic encryption. As per Gentry's scheme, Brakerski and Vaikuntanathan started from somewhat homomorphic encryption which was constructed to work over a ring R, where R can be described as $R = Z$ for integers, or $R = Z[x]/(x^d+1)$ for polynomial ring with d being a power of 2. For integer q, R_q is used to indicate R/qR. The error distribution parameter ξ is introduced to work over the ring R, where it is set to be as small as possible. Brakerski and Vaikuntanathan somewhat homomorphic algorithms is presented in Algorithm 5.

Algorithm 5 Brakerski-Vaikuntanathan

Key Generation KeyGen (l^λ)
$Sample : S_k = s \Leftarrow R_q \mid polynomial,\ small\ coefficients \in error\ distribution\ \xi$
Encryption Enc $(S_k, \mu \in R_2)$
$Sample : a \Leftarrow R_q\ randomly, polynomiale \mid small\ coefficients \in errordistribution\ \xi$
$Compute : c = (a, a_s = 2 \cdot e + \mu)$
Decryption Dec $(c = (a, b), S_k)$
$Sample : b \Leftarrow R_q\ randomly$
$Compute : \mu' = b \cdot a_s \in R_q$
$Output : \mu = \mu' mod 2$

The key contribution of this scheme is the technique of managing noise so that it increases linearly with the multiplicative level instead of exponentially which was the case in previous FHE schemes.

V. Going to the Cloud

As Internet evolved, computing concepts changed from parallel to distributed and grid computing and recently to cloud computing. If we look for the definition of cloud computing we can find that "Cloud computing is a model for enabling convenient, on-demand network access to a shared pool od configurable computing resources that can be rapidly provisioned and released with minimal management effort or service provider interaction" [14]. Cloud computing is the most popular notion in IT today, where even an academic

report from Berkeley says “Cloud computing is likely to have the same impact on software that foundries have had on the hardware industry" [15]. They go on to recommend that “developers would be wise to design their next generation of systems to be deployed into cloud computing" [15]. Cloud computing brings many business advantages such as accessing and using configurable computing resources through sharing and virtualization techniques, and all that at a lower cost. Main disadvantages are security and privacy issues that come along with resource sharing, virtualization and possible lack of trust between business and cloud providers. Many privacy issues could be solved with using homomorphic encryption, for instance cloud user can encrypt data and store it in the cloud, where both the user and cloud provider are unable to carry on any computations on data before decrypting it. It becomes more usual even for individuals, let alone for the companies to use the computational power of cloud computing, but often they would like their data to remain private. It is not so unrealistic to envision a system that collects health information from users in real time and on a basis of that information give alerts to the users. However, in a setting like that, user privacy is important and therefore the data needs to be encrypted and the operations in the cloud must run on encrypted data. It is worth mentioning that there are many possible applications that require only a limited number of operations and therefore somewhat homomorphic encryption systems would suffice.

A. HElib

Craig Gentry was the creator of first fully homomorphic encryption scheme, which was inefficient for implementation, but in the last 4 years efficiency has rapidly improved. In 2013 IBM presented open source software library for homomorphic encryption Helib [16]. HElib is a software library that has implementation of Brakerski-Gentry-Vaikuntanathan scheme, along with many optimizations to make homomorphic evaluation faster. To be exact, in 2010 all available FHE schemes were so inefficient that they would simply not run on any hardware, for instance, at the 2012 Crypto conference Gentry, Halevi and Smart showed that it “only" takes 36 hours to evaluate AES in the encrypted domain [17]. Only one year later this was possible to do under 3 hours. HElib is written in C++ and uses NTL mathematical library, however, it does not have implementation of bootstrapping. We can say that HElib is a young library in a rather young field of cryptography.

VI. Conclusions

Homomorphic encryption systems evolved significantly in the last couple of years since the design of first fully homomorphic encryption system. However, all those systems are still to impractical for real-world applications. Nevertheless, we already know numerous practical scenarios where we could use them. Because of that, it makes it interesting problem not only from academic perspective, but also from the industrial one. This will lead to the design of more efficient homomorphic encryption systems in the following years. Naturally, since there are applications where somewhat homomorphic encryption systems would be powerful enough, it seems we are closer to that goal than it may look like.

References

[1] J. Katz and Y. Lindell, *Introduction to Modern Cryptography (Chapman & Hall/Crc Cryptography and Network Security Series)*. Chapman & Hall/CRC, 2007.

[2] R. Rivest, A. Shamir, and L. Adleman, “A method for obtaining digital signatures and public-key cryptosystems,” *Communications of the ACM*, vol. 21, pp. 120–126, 1978.

[3] R. L. Rivest, L. Adleman, and M. L. Dertouzos, “On data banks and privacy homomorphisms,” *Foundations of Secure Computation, Academia Press*, pp. 169–179, 1978.

[4] C. Gentry, “A fully homomorphic encryption scheme,” Ph.D. dissertation, Stanford, CA, USA, 2009.

[5] S. Goldwasser and S. Micali, “Probabilistic encryption,” *J. Comput. Syst. Sci.*, vol. 28, no. 2, pp. 270–299, 1984.

[6] I. Damgard and M. Jurik, “A generalisation, a simplification and some applications of paillierś probabilistic public-key system,” in *In proceedings of PKC ’01, LNCS series*. Springer-Verlag, 2001, pp. 119–136.

[7] P. Paillier, “Public-key cryptosystems based on composite degree residuosity classes,” in *Proceedings of the 17th International Conference on Theory and Application of Cryptographic Techniques*, ser. EUROCRYPT’99. Berlin, Heidelberg: Springer-Verlag, 1999, pp. 223–238.

[8] “thep - The Homomorphic Encryption Project - Computation on Encrypted Data for the Masses,” 2014. [Online]. Available: https://code.google.com/p/thep/

[9] T. El Gamal, “A public key cryptosystem and a signature scheme based on discrete logarithms,” in *Proceedings of CRYPTO 84 on Advances in Cryptology*. New York, NY, USA: Springer-Verlag New York, Inc., 1985, pp. 10–18.

[10] D. Boneh, E.-J. Goh, and K. Nissim, “Evaluating 2-dnf formulas on ciphertexts,” in *Proceedings of the Second International Conference on Theory of Cryptography*, ser. TCC’05. Berlin, Heidelberg: Springer-Verlag, 2005, pp. 325–341.

[11] V. Vaikuntanathan, “Computing blindfolded: New developments in fully homomorphic encryption,” in *FOCS*, 2011, pp. 5–16.

[12] M. van Dijk, C. Gentry, S. Halevi, and V. Vaikuntanathan, “Fully homomorphic encryption over the integers,” in *EUROCRYPT*, 2010, pp. 24–43.

[13] Z. Brakerski and V. Vaikuntanathan, “Efficient fully homomorphic encryption from (standard) lwe,” in *FOCS*, 2011, pp. 97–106.

[14] P. Mell and T. Gance, “The NIST Definition of Cloud Computing (Final) (NIST Special Publication 800-145) ,” 2011.

[15] M. Armbrust, A. Fox, R. Griffith, A. D. Joseph, R. H. Katz, A. Konwinski, G. Lee, D. A. Patterson, A. Rabkin, and M. Zaharia, “Above the clouds: A berkeley view of cloud computing,” Tech. Rep., 2009.

[16] “HElib An Implementation of homomorphic encryption,” 2014. [Online]. Available: https://github.com/shaih/HElib

[17] C. Gentry, S. Halevi, and N. P. Smart, “Homomorphic evaluation of the aes circuit,” in *CRYPTO*, 2012, pp. 850–867.