Verifiable Delegation of Computation through Fully Homomorphic Encryption

Alpana Vijay
M. Tech (p)
Department of Computer Science
Rajasthan Institute of Engineering & Technology
Jaipur, India

Vijay Kumar Sharma
Assistant Professor
Department of Computer Science
Rajasthan Institute of Engineering & Technology
Jaipur, India

ABSTRACT
Delegating or outsourcing of computation is a prominent feature of cloud computing. It can be made secure through fully homomorphic encryption which allows processing of encrypted data. Verifiability of results is essential where a service-provider cannot be trusted. Since it is mostly used by lightweight devices, so mechanisms are required to verify the results of computations efficiently. In this paper protocols for VDoC are discussed, which focus on verification of aspects of computation other than the results, namely depth and complexity of delegated function which are useful when a user wants to verify amount charged by the service provider. A symmetric-key homomorphic scheme for encryption is being used. The protocols have a real-world relevance and the runtime are small enough for practical feasibility.

General Terms
Cryptography, outsourcing computation.

Keywords
Homomorphic encryption, symmetric FHE, cloud computing, verifiable delegation of computation, outsourcing computation.

1. INTRODUCTION
Cloud Computing with pay-per-use as its prominent feature is an emerging paradigm in modern computing. Its popularity grows as companies and users reduce their computing assets and turn to weaker computing devices; thereby delegating a number and variety of computations to cloud service providers. The major risk here is that the business critical computations are being performed remotely by untrusted parties that may be error-prone or even malicious. This motivates exploring methods for delegating computations reliably: a weak client delegates his computation to a powerful server. After the server returns the result of the computation, the client should be able to verify the correctness of that result using considerably less resources than required to actually perform the computation from scratch. Many solutions have been proposed for this purpose, recent ones are [1,2,3,4 and 5].

Due to increasing misuse of private data, IT security solutions for protection of sensitive data are indispensable, e.g. encryption. Storing private data on a Cloud in encrypted form is best solution only until one wishes the cloud to compute over the data. In that case sharing of keys may become mandatory. Ideally it is required that the cloud should be able to compute on private data without seeing (decrypting) it - referred to as secure outsourcing of computation. This is possible only through Fully Homomorphic encryption. Until 2009, FHE was a much desired but not yet achieved methodology among cryptographers. With the breakthrough work of Gentry [6], much research has been made in this direction to come up with practically feasible FHE schemes. Yet, the problem is not completely solved. The open issue is of verifying these computations. There exist a few methods of verifying the results of outsourced computations which combine computational proofs with homomorphic schemes[1,7,8 and 9]. Still, the "money" factor is not resolved. Consider a situation where an online consultancy service charge for the suggestions provided based on the claim that it is performing some popular complex computations over the user's input information. How will user be assured that the returned decisions are outcome of a complex operation or a simple calculation? It might be possible to verify the result, but what about the cost charged towards client? The "money" factor here should depend on the actual computation performed and not only on the correctness of result.

In this paper it is presented here some basic terminology to build up the background and proceed to identification of the problem and justification of motivation towards it. Further it describes few protocols for such verification for delegated computations.

2. BACKGROUND
In this section a brief introduction of Verifiable computation and homomorphic encryption is given. With evolution of new applications depending on verifying the results of a computation, delegated to other party, new definitions and concepts have emerged in the field of Verifiable Delegation of Computation (VDoC). Similarly, fully homomorphic encryption has also received recent attention.

2.1 Verifiable Computation
A verifiable computation scheme VC = (KeyGen, ProbGen, Compute, Verify) consists of the four algorithms -

1. KeyGen (F, λ) $\rightarrow$ (PK, SK): Based on the security parameter, the randomized key generation algorithm generates a public key that encodes the target function F, which is used by the worker to compute F. It also computes a matching secret key, which is kept private by the client.

2. ProbGen_{SK} (x) $\rightarrow$ (σ_x , τ_x): The problem generation algorithm uses the secret key SK to encode the function input x as a public value which is given to the worker to compute with, and a secret value which is kept private by the client.

3. Compute_{PK} (σ_x) $\rightarrow$ σ_y: Using the client's public key and the encoded input, the worker computes an encoded version of the function's output y = F(x).

4. Verify_{SK} (τ_x , σ_y) $\rightarrow$ y $\cup$ $\perp$: Using the secret key SK and the secret "decoding", the verification algorithm converts the worker's encoded output into the output of the function, e.g.,

y= F(x) or outputs ⊥ indicating that does not represent the valid output of F on x.

An essential property of a VC scheme is correctness. A scheme is correct if the problem generation algorithm produces values that allow an honest worker to compute values that will verify successfully and correspond to the evaluation of F on those inputs. All VC schemes should be secure, that is a malicious worker cannot manipulate the verification algorithm to accept an incorrect output. The efficiency condition required from a VC scheme is that the time to encode the input and verify the output must be smaller than the time to compute the function from scratch. Input privacy is defined based on a typical indistinguishability argument that guarantees that no information about the inputs is leaked. Public delegation allows decoupling the party who provides the function to be evaluated and the party who has the input for the computation. That is, the delegator chooses from the functions which are made publicly available by a third party. Public verifiability enables anyone to verify the correctness of the returned results. That is, the role of prover can be separated from the role of delegator, without sharing additional secret information. A recent survey of VC approaches and open problems are described in [10].

2.2 Fully Homomorphic Encryption

The aim of homomorphic cryptography is to ensure privacy of data in communication, storage or in use by processes with mechanisms similar to conventional cryptography, but with added capabilities of computing over encrypted data, searching an encrypted data, etc. Formally, homomorphic encryption scheme is defined as a quadruple of following algorithms:

Keygen –The key generation algorithm takes input the security parameter λ and outputs keys (pk, sk, ek), where *pk* is public key, *sk* is private key and *ek* is evaluation key.

Enc – The encryption algorithm converts plaintext to ciphertext using the public key.

Dec – The decryption algorithm converts ciphertext to plaintext using the private key.

Eval – The homomorphic evaluation algorithm evaluates the result of a computation *f* on ciphertexts c_1, c2,…, c_l using evaluation key *ek* and/or public key *pk*.

3. PROPOSED PROTOCOLS AND SCHEME

Verifiable delegation of computation has immense application in today's cloud computing demands. Emerging new areas call for newer implications of verifiability. Existing definitions are insufficient to address requirements of certain applications. Consider a situation where a user U delegates computation *f* over inputs m_i to cloud P. For security reasons m_i are encrypted into *c*i. P evaluates *f'* over *c*i (homomorphic to *f* over plaintext), returns result y, which could later be decrypted by U to obtain x (the expected result). Assuming that correctness of the returned result y is ensured through the primitives of homomorphic encryption and some one-way functions, one could think that the purpose of VDoC has been achieved. But, in context of computation-as-a-service, it should be noticed that the service charge (bill amount) depends not on the complexity of *f* but of *f'*. U is aware only of *f* and has no way to judge the complexity of *f'*. Hence, a different perspective of verification is required. So some new definitions are proposed:

Definition 1: Complexity verifiability: A computation is said to be complexity-verifiable if the number of gates in corresponding circuit and hence complexity of the computation can be deduced from the output.

Definition 2: Depth verifiability: A computation is said to be depth-verifiable if the gate-depth of corresponding circuit can be deduced from the output.

3.1 Symmetric FHE scheme

The following symmetric fully homomorphic scheme is used, based on the symmetric somewhat homomorphic scheme of [11]. Let λ be the security parameter in context of an equivalent computational effort of 2^{λ} cycles. Message to be encrypted is an integer from Z_N, here N is η bits long. The relationship between these two parameters are $\eta=O(\log \lambda)$.

The scheme consists of following primitives:

Key generation: Generate the secret key p, a prime numbers of length λ bits. Select q as refresh key such that q = p*s, s is any random number of length kη bits and s = kN for some small k. Thus, the refresh key is some multiple of the secret key, thus containing the secret key without revealing it. Also, the condition s = kN implies q has essentially more than two factors making it difficult to factorize.

Encryption: Encryption is simply a mapping Enc: $Z_N \rightarrow N$, thus converting the message into an integer. The mapping is not deterministic or one-to-one, that is a message may have several encryptions. This is possible through a random number involved in the encryption process. Encryption involves following steps:

Step 1: Choose m' such that m≡m' mod N

Step 2: Choose a random number r of length λ^2 bits

Step 3: Output c = m' + pr

The larger size of the random number r as compared to the key p is necessary for the homomorphism in operations.

Decryption: The decryption needs to be inversion of the encryption process, a mapping Dec: $N \rightarrow Z_N$. It is a very simple and efficient operation producing output, the message as m = c mod p mod N.

Homomorphic Operations: Addition of two messages in Z_N, that is m1+m2 mod N is homomorphic to direct addition of two ciphertexts (integers). Multiplication of two messages in Z_N, that is m1*m2 mod N is homomorphic to integer multiplication of two ciphertexts.

Refresh procedure: The noise in ciphertext grows with the number of operations applied to it. To make it suitable for decryption even after several operations, it is required to refresh the ciphertext, thus reducing noise in it. The refresh procedure is very simple and efficient one-step procedure. The refreshed ciphertext is outputted as c' = c mod q.

To include the depth of circuit for verification, it is considered that the data objects to be computed over as tuples containing the ciphertext and a number indicating current depth level. The initial data begins at depth 0, which is then cascaded as input to next levels. At every level the depth of all input objects are compared, selecting highest, it is forwarded after incrementing, as shown in Fig 3. Let the actual circuit to be computed be converted into that consisting only of AND and XOR gates, as in Fig 1. For homomorphic calculation using the proposed scheme, corresponding homomorphic circuit as shown in Fig 2 can be drawn, by replacing AND operations

with multiplication and XOR operations with addition. For calculation complexity for verification, the tuple contains ciphertext and a number indicating number of operations it has already been processed through. The initial value of this number is 0. At every operation the numbers of operations of input data tuples are added together. This is incremented and forwarded as input to next operation. The final complexity count output from the function represents the total number of operations performed over the input data. The procedure is illustrated in Fig 4 over an example circuit.

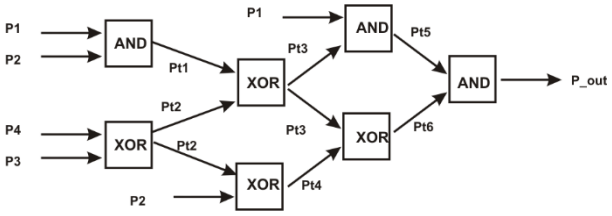

Figure 1: Circuit containing only XOR and AND gates, operating on plaintexts

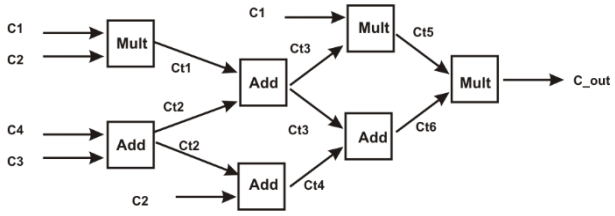

Figure 2: Circuit homomorphic to circuit of Figure 1, operating on ciphertexts

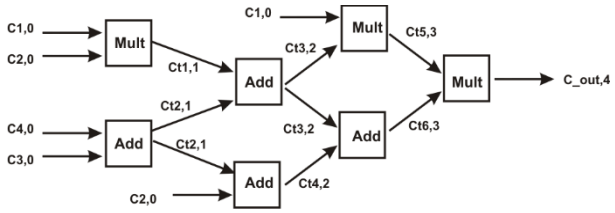

Figure 3: Circuit of Figure 2 modified to calculate depth along with computation

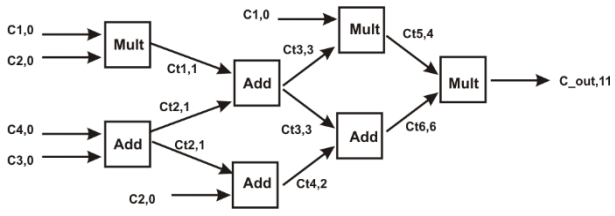

Figure 4: Circuit of Figure 2 modified to calculate complexity along with computation

3.2 Protocol for Depth Verification

For depth verification a non-interactive protocol is suggested:

Step 1: Delegator selects evaluation function f and prepares its equivalent function g with known depth d.

Step 2: Delegator sends encrypted input, function g and the refresh key to the worker as $\langle c_{in}, g, rk \rangle$.

Step 3: Worker computes the function and returns the calculated result and depth as $\langle g(c_{in},), d' \rangle$.

Step 4: Worker raises the bill with charged amount Amt.

Step 5: Verify $d'=d$. If yes, accept Amt else reject.

The preparation of function g can be done either by the delegator itself, or by a third party in case of limited resources.

3.3 Protocol for Complexity Verification

For complexity verification a non-interactive protocol is suggested:

Step 1: Delegator selects evaluation function f and prepares its equivalent function g with known complexity τ.

Step 2: Delegator sends encrypted input, function g and the refresh key to the worker as $\langle c_{in}, g, rk \rangle$.

Step 3: Worker computes the function and returns the calculated result and complexity as $\langle g(c_{in},), \tau' \rangle$.

Step 4: Worker raises the bill with charged amount Amt.

Step 5: Verify $\tau'= \tau$. If yes, accept Amt else reject.

The preparation of function g can be done either by the delegator itself, or by a third party in case of limited resources.

3.4 Modifications required in FHE scheme

Modified Evaluation algorithms for depth verifiable computation:

ADD ($\langle c_1, n_1 \rangle$, $\langle c_2, n_2 \rangle$): Takes input ciphertexts along with a number associated to it, which is by default 0. Outputs ciphertext as sum of the input ciphertext and a number associated to it.

Step 1: c = c1 + c2

Step 2: If n1 > n2, n = n1 + 1 else n = n2 + 1

Step 3: Output $\langle c, n \rangle$

MULTIPLY ($\langle c_1, n_1 \rangle$, $\langle c_2, n_2 \rangle$): Takes input ciphertexts along with a number associated to it, which is by default 0. Outputs ciphertext as sum of the input ciphertext and a number associated to it.

Step 1: c = c1 * c2

Step 2: If n1 > n2, n = n1 + 1 else n = n2 + 1

Step 3: Output $\langle c, n \rangle$

Modified Evaluation algorithms for complexity verifiable computation:

ADD ($\langle c_1, n_1 \rangle$, $\langle c_2, n_2 \rangle$): Takes input ciphertexts along with a number associated to it, which is by default 0. Outputs ciphertext as sum of the input ciphertext and a number associated to it.

Step 1: c = c1 + c2

Step 2: n = n1 + n2 + 1

Step 3: Output $\langle c, n \rangle$

MULTIPLY ($\langle c_1, n_1 \rangle$, $\langle c_2, n_2 \rangle$): Takes input ciphertexts along with a number associated to it, which is by default 0. Outputs

ciphertext as sum of the input ciphertext and a number associated to it.

Step 1: c = c1 * c2

Step 2: n = n1 + n2 + 1

Step 3: Output ⟨*c, n*⟩

3.5 Results and Implementation

The implementation of the protocol was done in the form of functions in Java on a 3.40 GHz Intel Core i3-2130 processor. For comparison of run-times, each function is designed in four versions:

1. Function f – this is the plain function as known to the delegator. It operates on plaintexts.

2. Function f' – This is homomorphic to f, operating on ciphertexts and gives the result as ciphertext only. This corresponds to input/output verifiable computation.

3. Function f'' – This is f' modified to include calculation of depth of the circuit. This corresponds to depth verifiable computation.

4. Function f''' – This is f' modified to include calculation of complexity of the circuit. This corresponds to complexity verifiable computation.

Table 1 shows the run-time for different parameters (depth and complexity) of the circuits which consist only of XOR gates and Table 2 shows the same for circuits with only AND gates. Fig 5 and Fig 6 show the graph derived from the values of Table 1 and Table 2 respectively. Here it can be seen the growth of cost of modified homomorphic versions for depth and complexity verification is parabolic, that is $O(d^2)$, while the growth of simple homomorphic function and the plain function is only a degree lower, that is O(d). It can easily be observed that time taken for AND operations is much larger than XOR operations. Hence, some arbitrary circuits are required that contain both XOR and AND gates to see the effect of FHE and Verifiable computation. The time of execution recorded for arbitrary circuits of various depths and complexity is shown in Table 3, for all four versions. The growth of verifiable computations against depth of circuit is shown in Fig 7. Since the roles of delegator and worker are implemented as separate threads of same Java program, the recorded run-times do not include any communication cost, and hence indicate purely computational cost of the algorithms and protocols proposed.

Table 1. Performance over Circuits consisting only XOR gates

Circuit Parameters		Time of Execution (in milliseconds)			
Depth	**Complexity**	**Plain Computation**	**Homomorphic computation**	**Depth verifiable computation**	**Complexity verifiable computation**
1	1	0.604	5.13	5.98	5.888
2	3	0.905	6.035	11.165	11.16
3	5	1.207	6.639	13.278	12.998
4	8	1.207	7.544	14.183	14.786
5	10	1.509	7.544	16.295	17.544
6	14	1.509	8.147	17.804	18.204
7	18	1.508	9.354	20.52	20.632
8	21	1.509	10.561	25.649	25.647
9	26	1.81	11.768	29.874	29.968
10	30	1.809	13.579	35.608	35.812

Table 2. Performance over Circuits consisting only AND gates

Circuit Parameters		Time of Execution (in milliseconds)			
Depth	**Complexity**	**Plain Computation**	**Homomorphic computation**	**Depth verifiable computation**	**Complexity verifiable computation**
1	1	1.508	4.526	10.863	10.854
2	3	1.509	5.431	13.881	13.578
3	5	1.81	7.242	17.582	17.589
4	8	1.811	9.053	21.727	21.689
5	10	2.414	11.165	25.951	25.458
6	14	2.414	13.579	29.271	29.91
7	18	3.017	14.786	35.608	34.927
8	21	3.32	15.088	41.039	41.067
9	26	4.527	17.804	43.454	43.821
10	30	4.828	20.52	52.205	52.187

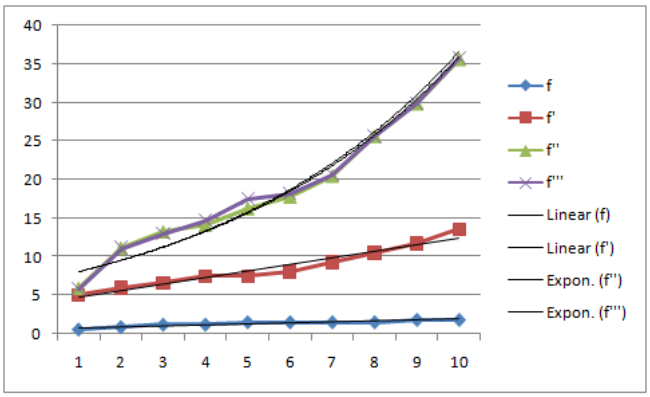

Figure 5: Growth of run-time for all four versions with varying depth of circuits containing only XOR gates

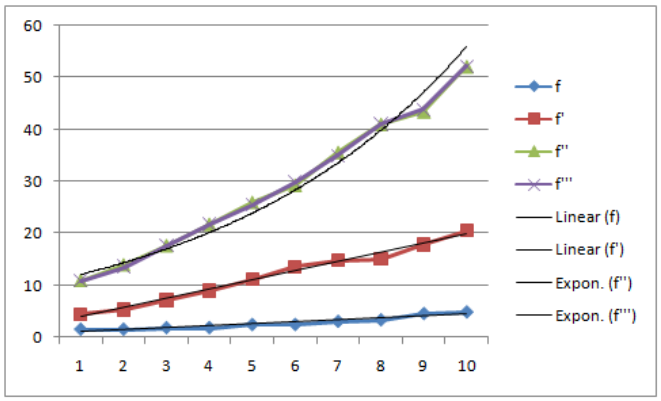

Figure 6: Growth of run-time for all four versions with varying depth of circuits containing only AND gates

Table 3. Performance over arbitrary circuits

Circuit Parameters		Time of Execution(in milliseconds)			
Depth	**Complexity**	**Plain Computation**	**Homomorphic computation**	**Depth verifiable computation**	**Complexity verifiable computation**
1	1	1.056	4.828	8.472	8.371
2	3	1.207	5.733	12.568	12.401
3	5	1.561	6.952	15.752	15.284
4	8	1.602	8.259	18.034	18.237
5	10	1.904	9.357	21.264	21.501
6	14	1.962	10.863	23.547	24.001
7	18	2.263	12.07	28.011	27.799
8	21	2.428	12.875	33.523	33.357
9	26	3.154	14.786	37.162	36.891
10	30	3.326	18.742	42.961	43.015

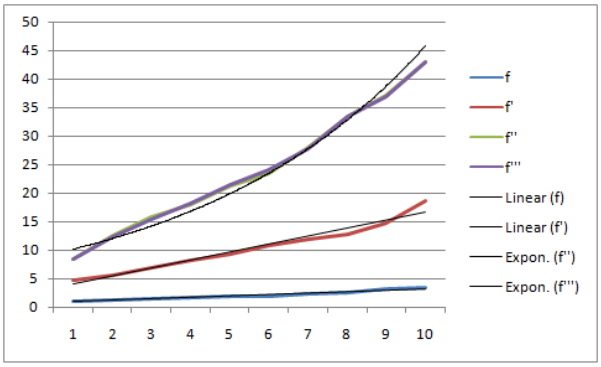

Figure 7: Growth of run-time for all four versions with varying depth of arbitrary circuits

4. CONCLUSION

Cloud computing has presented an easy means for delegating computationally heavy tasks over to the service providers. Increasing use of lightweight devices is making such delegation more essential part of upcoming applications. What makes FHE schemes a winner in context of cloud computing is, that it allows one to compute over encrypted data. But such easiness of computation makes it more difficult to verify, since the function delegated to worker for computation and its homomorphic equivalent may differ in terms of depth or complexity. Thus, actual computational effort put in by a worker cannot be measured directly or clearly. Proposal given in this paper includes three such protocols which imply input/output verifiability, depth verifiability and complexity verifiability for delegated computations. The proposal also includes these definitions as was identified by us a requirement in new applications of VDoC. The results obtained through implementation of the scheme and protocols vouch for its efficiency and feasibility in practical applications.

The proposed protocols work only for those computations which can be converted into Boolean circuits. Extending it to work for arithmetic circuits could be a possible scope. Many delegation applications include processes like compression, file conversion etc, which cannot be immediately seen as an arithmetic computation. Verifiability in context of such applications needs to be defined and explored appropriately.

5. REFERENCES

[1] R. Gennaro, C. Gentry, and B. Parno. "Non-interactive verifiable computing: Outsourcing computation to untrusted workers." In Tal Rabin, editor, CRYPTO, volume 6223 of Lecture Notes in Computer Science, pages 465-482. Springer, 2010.

[2] S. Goldwasser, H. Lin and A. Rubinstein. "Delegation of Computation without Rejection Problem from Designated Verifier CS-Proofs." IACR Cryptology ePrint Archive, 2012:455, 2012.

[3] D. Fiore and R. Gennaro. "Publicly verifiable delegation of large polynomials and matrix computations, with applications." IACR Cryptology ePrint Archive, 2012:281, 2012

[4] Ran Canetti, Ben Riva, Guy N. Rothblum: Two Protocols for Delegation of Computation. Information Theoretic Security - 6th International Conference, ICITS 2012, Montreal, QC, Canada, August 15-17, 2012. Proceedings. Springer 2012 Lecture Notes in Computer Science. pg 37-61

[5] M. Backes, D.Fiore and R.M. Reischuk. "Verifiable Delegation of Computation on Outsourced Data." IACR Cryptology ePrint Archive, 2013:469, 2013.

[6] C. Gentry. "A Fully Homomorphic Encryption scheme." Dissertation. Sep 2009. Available at https://crypto.stanford.edu/craig/craig-thesis.pdf.

[7] K.-M. Chung, Y. Kalai, and S. P. Vadhan. "Improved delegation of computation using fully homomorphic encryption." In T. Rabin, editor, Advances in Cryptology – CRYPTO 2010, volume 6223 of Lecture Notes in Computer Science, pp 483–501, Springer.

[8] Fangyuan Jin; Yanqin Zhu; Xizhao Luo, "Verifiable Fully Homomorphic Encryption scheme," Consumer Electronics, Communications and Networks (CECNet), 21-23 April 2012 2nd International Conference Proceedings, pp.743,746.

[9] M. Barbosa and P. Farshim. "Delegatable Homomorphic Encryption with Applications to Secure Outsourcing of Computation." Proceedings of The Cryptographers' Track at the RSA Conference 2012, San Francisco, CA, USA, February 27 – March 2, 2012. pg 296-312.

[10] A. Vijay, T Sharma and R Modi. "Verifiable Delegation of Computation to Untrusted Clouds: Approaches and Open Problems." 1st Interntaional Conference on Emerging Trends in Engineering and Applied Science, 2013.

[11] M. van Dijk, C. Gentry, S. Halevi, and V. Vaikuntanathan. "Fully homomorphic encryption over the integers", Proceedings of Eurocrypt-10, Lecture Notes in Computer Science, vol 6110,. Springer, pp 24-43, 2010.