**A homomorphic encryption algorithm over integers**

If a secure and efficient Fully Homomorphic Encryption (FHE) algorithm exists, it should be the ultimate solution for securing data privacy in cloud computing, where any operation can be directly applied to ciphertexts without garbling them. Though some theoretic FHE algorithms do exist, none of them is practical because of their prohibitively expensive computing cost. This research designs and analyses a homomorphic encryption algorithm over integers. Two variants of this algorithm are now under our investigation. Summary of these two variants are as follows:

* **Non-deterministic algorithm:** Both versions are non-deterministic. That is, the encryption is a one-to-many mapping. A nice feature to hide the equality relationship among ciphers from adversaries.
* **Functionality:**
  + Variant 1: Allows an unlimited number of arithmetic additions and multiplications on ciphers until the integer (application data) exceeds the encryption/decryption key size (a large prime). An equality testing algorithm was developed and can be used to test whether two ciphers (of a same integer) are equal.
  + Variant 2: Allows unlimited number of arithmetic additions and multiplications on ciphers until the integer (application data) exceeds a pre-defined size or the corresponding padded integer exceeds the encryption/decryption key (a large prime). Unlike Variant 1 ciphers, Variant 2 ciphers cannot be tested for equality.
* **Key size:** If both variants would like to support applications with the same maximum data value, Variant 2 algorithm’s key size will be bigger than that of Variant 1 algorithm. However, both algorithms’ key sizes will be in the range of several thousand bits for most applications, which is much better than that of the best implementation of the Gentry-like theoretic FHE algorithms (several megabits).
* **Ciphertext size:** The encryptions and homomorphic additions/multiplications in both variants are all modular operations. Thus, the ciphertext will be less than the modulus used.
* **Computational complexity:** Both variants are very efficient on encryption/decryption (with just a few modular multiplications) and are especially efficient on the homomorphic addition/multiplication (with a single modular addition/multiplication, respectively).
* **Security:** If an adversary knows two ciphers of a same integer, the adversary can break Variant 1, but not Variant 2. We still research on the security of these two variants to other attacks and also is trying to provide security proofs for Variant 2.

**Proposed research work:**

* Research on the security of these two variants to other attacks. Adjustment to the algorithm may be needed if a new security weakness is found. At the end, formally prove the security of both variants.
* With different functionalities of these two variants, identify suitable real world applications for each variant.

The following pages briefly describe both variants.

**Homomorphic encryption algorithm: Variant 1**

**Parameter setup:**

1. A data owner  randomly picks two large primes  and  repeatedly until  is also a prime. Compute the product . Note that  needs to choose  large enough so that all his application data . If , the cipher of *m* cannot be decrypted back to *m*.
2. The encryption key of the algorithm is the pair , which will be used by  to perform data encryption and decryption.  should keep it in secret.
3.  randomly picks another large prime  and then computes the product . The data owner  also randomly picks a set of numbers , and then computes



The set of ’s is the equality test key. The provided equality test algorithm is a probabilistic algorithm. Thus, if more  are generated by , the accuracy of the test is higher. The test algorithm will be described later.

1. The set is the operational key of the algorithm, where authorized agents (e.g., cloud servers) or the public can use *N* to perform arithmetic operations and use  to perform the equality testing.

**Encryption:** To encrypt an integer ,  picks a random number *r* and use the encryption key  to computes



**Decryption:** To decrypt a cipher ,  can use the key  to compute

****

**Homomorphic addition:** The authorized agents, with the operational key , can perform homomorphic addition directly to ciphers. That is,



Since



**Homomorphic multiplication:** The authorized agents, with the operational key , can perform homomorphic multiplications directly to ciphers. That is,



Since



**Homomorphic equality testing:** The authorized agents, with the operational key , can perform equality testing to two ciphers. For each , test whether



If any one of the ’s fails the test, then . On the other hand, if all ’s pass the test, then is most likely equal to with a neglect chance of being not equal, since



**Security weakness:** If an adversary knows a pair of ciphers of a same integer, i.e., , he can compute the encryption key  since and then the . A special case of this attack is actually a known plaintext attack. That is, if the adversary knows a plain and a cipher pair, he can perform the same gcd attack.

**Applications may be suitable for this algorithm:** Applications with highly discrete data with no repetition, e.g., high-precision scientific data.

**Homomorphic encryption algorithm: Variant 2**

**Parameter setup:** The Variant 2 algorithm has exactly the same system parameter setup as Variant except two additional parameters *w* and *z*. The selection of *w* and *z* also has impact to the selection of .

1. *w* should be selected based on the application requirement. If the maximum integer required in an application is *Imax*, then *w* must be selected big enough so that *Imax*. That is, *w* is the maximum bit size of all data in an application.
2. *z* is the number of random bits to be left padded to the data so that two encryptions of the same integer *m* are most likely to encrypt two different padded integers. Typically, *z* = 64 would be big enough for real world applications.
3. The padded integer *m′* is of the size *w+z* bits. Recall that in the Variant 1 algorithm. Thus, in Variant 2, we have . This implies that if the data owner would like to have *k* consecutive homomorphic multiplications on ciphertexts, then choosing with size at least (*k + 1*)(*w + z*) bits.

**Padding:** We use an example to show how Variant 2 algorithm works. Let and then the padded integer , where | means concatenation and  with leading 0’s to the *w*-th bit, followed by *z* padded random bits. Thus, let an integer, we have



**Encryption:** The encryption works he same as the Variant 1 algorithm, except it encrypts the padded integer *m′*, i.e.,



**Decryption:** Two mod operations are required.



**Homomorphic additions & multiplications:** Same as those in the Variant 1 algorithm.

**Equality testing:** The equality testing algorithm for the Variant 1 ciphers does not work for the Variant 2 ciphers. No known equality testing for Variant 2 ciphers.

**Security analysis:** Currently no known attacks. However, the security of this Variant 2 homomorphic encryption requires formal proofs.

**Applications may be suitable for this algorithm:** Applications with no need to perform equality matching operations.