A Quick Example of Homomorphic Encryption Using the Paillier Cryptosystem: Supplemental Note

Nitin Kohli

UC Berkeley Center for Effective Global Action

May 2024

Abstract

This note provides supplemental information to CEGA's blog post on homomorphic encryption. We present the mathematical details underpinning the Paillier cryptosystem using a simple example from banking.

1 Homomorphic Encryption

Homomorphic encryption enables computations on encrypted data without the need to decrypt it first. Homomorphic encryption is not a single technique; rather, it encompasses a class of algorithms (often called *schemes* or *cryptosystems*) that facilitate computations on encrypted data. While homomorphic encryption can be used for a wide array of computational tasks – ranging from machine learning [1, 2] to auctions [3] – we focus on the simple setting of adding individual's data using the Paillier cryptosystem [4] to help the reader develop intuition more broadly. For more information on the topic of homomorphic encryption in general, we point the reader to *A Guide to Homomorphic Encryption* by Will and Ko [5].

2 Running Example from CEGA's Blog Post

Consider a bank that stores the number of transactions customers initiated in the last quarter using homomorphic encryption. The bank wishes to compute the total number of transactions computed across all customers.

To ease the mathematics presented (and build intuition for the general approach), we consider the case where the bank computes over two customers data. Suppose the two customers made 4 and 7 transactions, respectively. When computing, the bank should conclude 11 transactions were made in total.

3 Paillier Cryptosystem in Action

Homomorphic cryptosystems incorporate a set of procedures to ensure the encryption, computation, and decryption processes ultimately return the correct answer. Many homomorphic cryptosystems can be decomposed into four steps.

1. Generate public keys and private keys to aid in encryption and decryption.
2. Given a datapoint m, generate a random value r and encrypt the data as $e(m;r)$. Then, store it in a database.
3. Perform encrypted computations using "adjusted computations."
4. To decrypt the output v of an encryption computation, run the decryption process to return the real output $d(v)$.

Through the careful introduction of algebra and probability in Steps 1 - 4, the Paillier cryptosystem does this as follows.

Step 1: Generate public and private keys to facilitate secure computations.

This is done in Step 1 using the 5 sub-steps (a) - (e), described below.

(a) Randomly pick 2 large prime numbers p and q independently of one another, such that the greatest common divisor of pq and $(p-1)(q-1)$ is 1.

- For simplicity, suppose that $p = 3$ and $q = 5$ are selected. In reality, a practitioner would never choose such small numbers (which is why it's important to keep in mind this is just a simple example – and not advice – and experts should be consulted for implementation), but 3 and 5 will keep the math relatively simple.
- Observe that $\gcd(pq, (p-1)(q-1)) = \gcd(15, 8) = 1$, so the primes are acceptable.

(b) Set $n = pq$ and λ equal the least common multiple of $p-1$ and $q-1$.

- Assign $n = pq = 15$ and $\lambda = \text{lcm}(p-1, q-1) = \text{lcm}(2, 4) = 4$.

(c) Select an integer g from $\{1, ..., n^2\}$ uniformly at random.

- Suppose we randomly sampled $g = 4$.

(d) Check that n, g, and λ "behave nicely": that is, verify that $\left(\left(g^\lambda \bmod n^2\right) - 1\right)/n$ has an inverse mod n. If so, set μ equal to this inverse. Otherwise, return to (a).

- We compute $\left(g^\lambda \bmod n^2\right) - 1 = \left(4^4 \bmod 15^2\right) - 1 = 30$, so dividing by 15 yields the value of 2. The multiplicative inverse of 2 is 8 mod 15, and thus exists.
- Hence we assign $\mu = 8$.

(e) Assign public key = (n, g) and private key = (λ, μ).

- Therefore, in our example, we have public key = $(15, 4)$ and private key = $(4, 8)$.

Step 2: Encrypt all the data you want to store in a database using the public keys.

Encode data as an integer $m \in \{0, ..., n-1\}$. For every datapoint m you wish to encrypt,

(a) Select r uniformly at random from the set $\{1, ...n-1\}$ where the greatest common divisor of r and n is 1.

(b) Define our encryption function $e(m;r) = g^m r^n \bmod n^2$.

In our setting, the encrypted values of 4 and 7, denoted $e(4;r_1)$ and $e(7;r_2)$, would be placed in the database. For simplicity, suppose the first random number is $r_1 = 1$ and the second is $r_2 = 8$. Then $e(4;1)$ and $e(7;8)$ would be placed in the database. Using the definition of $e(m;r)$, we see that

$$e(4;1) = 4^4 1^{15} \bmod 225 = 256 \bmod 225 = 31$$

and

$$e(7;8) = 4^7 8^{15} \bmod 225 = 212$$

So, the values of 31 and 212 are stored in the database instead of 4 and 7.

Step 3: Perform "adjusted computations."

Now, when the bank submits their query, the computation of "4+7" will involve 31 and 212. However, the Paillier cryptosystem will not use addition, but rather multiplication. At a conceptual level, this occurs because the values of 4 and 7 were used in the exponents when encrypting the data in Step 2; since multiplying numbers with exponents requires us to add the exponents, multiplying 31 and 212 will encode the addition of 4 and 7 (very sneakily in the exponents). This gives us the encrypted value $v = 31 \times 212 = 6{,}572$.

Step 4: Decrypt computations using the private key.

A careful reader will note that 6,572 does not equal 11 – this is expected, as the value 6,572 is in "encrypted space" and not in the original "data space." To get an answer in the original data space, we have to use a decryption function $d(\cdot)$. Let v be the value we wish to decrypt. We do so using the decryption function

$$d(v) = \left(\left((v^\lambda \bmod n^2) - 1\right)\mu/n\right) \bmod n$$

In this case, the system will return $d(6572)$, which reduces to

$$d(6572) = \left(\left((6572^4 \bmod 225) - 1\right) \times 8/15\right) \bmod 15 = 11$$

This is exactly the answer we wanted!

4 Conclusion

In the above example, when the analyst tries to compute the sum of transactions last quarter, instead of examining the values of 4 and 7 and adding them, the Paillier cryptosystem:

1. Generates public and private keys (15,4) and (4,8) respectively to facilitate secure computations.
2. Examines the values of encrypted values $e(4;r_1) = 31$ and $e(7,r_2) = 212$ using a public key (15,4).
3. Multiplies 31 and 212 to get 6,572 (to numerically encode the addition operation in the encrypted computation).

4. And runs the decryption process to map 6,572 to 11 using a private key (4,8).

Other homomorphic cryptosystems deviate from the specific implementation details of the Paillier cryptosystem, but at their core they all enable computations over encrypted data through the careful utilization of algebraic and probabilistic operations.

References

[1] X. Sun, P. Zhang, J. K. Liu, J. Yu, and W. Xie, "Private machine learning classification based on fully homomorphic encryption," *IEEE Transactions on Emerging Topics in Computing*, vol. 8, no. 2, pp. 352–364, 2018.

[2] J. E. Blumenstock and N. Kohli, "Big data privacy in emerging market fintech and financial services: A research agenda," *arXiv preprint arXiv:2310.04970*, 2023.

[3] M. Pan, J. Sun, and Y. Fang, "Purging the back-room dealing: Secure spectrum auction leveraging paillier cryptosystem," *IEEE Journal on Selected Areas in Communications*, vol. 29, no. 4, pp. 866–876, 2011.

[4] 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.

[5] M. A. Will and R. K. Ko, "A guide to homomorphic encryption," in *The cloud security ecosystem: technical, legal, business and management issues*, Elsevier, 2015.