

Discussion on the Partially Homomorphic Encryption Library

Andrew Quijano¹ and Kemal Akkaya²

¹Department of Computer Science, Columbia University, New York, NY 10027, Email: afq2101@columbia.edu

²Department of Electrical & Computer Engineering, Florida International University, Miami, FL 33174, Email: kakkaya@fiu.edu

I. HOMOMORPHIC ENCRYPTION

A. Overview

This library currently contains ElGamal, Paillier [1] and DGK homomorphic encryption systems [2] [3]. Paillier and DGK are strictly partially homomorphic. By this definition, where we define x, y as its plain-text values and $E(x), E(y)$ as its respective cipher-texts, the following statements are true

$$E(x)E(y) = E(x + y)$$

$$E(x)^y = E(xy)$$

It is defined as partially homomorphic as you can only support addition on cipher-texts but not multiplication as well. As shown in the second line, you can compute the product if one value is not encrypted. As a note, the plain-text space for Paillier is $[0, N]$ and for DGK $[0, u]$.

Note: Every Protocol discussed here will be discussed in terms of using it with Paillier encrypted values. These protocols work just as well with DGK as long as you replace N with u to account the difference in plain-text size.

You can create a subtraction protocol by just ensuring that y is negative, such that when you compute the multiplication you get $E(x - y)$. In our implementation, we do this by taking $E(y)^{-1}N^2$, then multiplying the cipher-texts.

Also, you can create a scalar division protocol by computing $y' = y^{-1} \mod N$. Then have $E(x)y' = E(x \div y)$.

Observation 1. *This scalar division only works if $y \mid x$*

ElGamal is partially homomorphic in that it can be made to either support addition over cipher-texts or multiplication but not both. This setting can be toggled in the library.

Observation 2. *If you use ElGamal to support addition over cipher-texts, it will be severely limited as it will need to pre-compute discrete logarithms to be efficient for decryption.*

With regards to ElGamal when it is homomorphic multiplicative, by definition it can support division as well.

Observation 3. *Much like with scalar division, El-Gamal division will only work if $y \mid x$*

B. Supporting Extra Operations

Mat et. al used a secure multiplication mechanism from which you can obtain the product of two Paillier cipher-texts [4]. For clarity, the algorithm is explained in Algorithm 1. **Note:** The constraint is placed there because if the product of both numbers exceed the plain-text, it will be just cut off by modulo N . With regards to division Veugen has created a protocol for encrypted division as well [5].

Algorithm 1 Secure Multiplication

Party	A	B
Input	$[x]$ and $[y]$	$K_{Paillier}$
Output	$[xy]$	
Constraint	$xy \leq N$	

- 1) A chooses $a \in [0, N)$ and $b \in [0, N)$
- 2) A computes $x' = [x + a]$ and $y' = [y + b]$ and sends both to Bob
- 3) B receives $[x']$ and $[y']$ and decrypts both and multiplies them.
- 4) B encrypts $[x' y']$ and sends it to A
- 5) A computes $[xy] = [x' y'] - [ax] - [by] - [ab]$

As for El Gamal, assuming we stick to it being only homomorphic via multiplication only, we require a secure addition protocol. It will be similar to the secure multiplication protocol, it is shown in Algorithm 2.

Algorithm 2 Secure Addition

Party	A	B
Input	$[x]$ and $[y]$	$K_{El-Gamal}$
Output	$[x + y]$	
Constraint	$x + y \leq N$	

- 1) A chooses $a \in [0, N)$
- 2) A computes $x' = [ax]$ and $y' = [ay]$ and sends both to Bob
- 3) B receives $[x']$ and $[y']$ and decrypts both and adds them.
- 4) B encrypts $[x + y]$ and sends it to A
- 5) A computes $[x + y] = [\frac{x y'}{a}]$

C. Comparing Encrypted Numbers

This library had first originally implemented the protocol by Veugen [6]. In due time Mau et al. have found a mistake and proposed a correction for Veugen's original comparison protocol [4]. They also mention that Protocol 2 would return $[x \geq y]$, not $[x \leq y]$ as stated in the original Veugen paper. Veugen has eventually submitted a correction to his original work [7]. In implementing Veugen's solution we have been made aware and confirmed with Veugen that Protocol 2 and 4 described in his paper would return $[x \geq y]$, not $[x \leq y]$. Finally, Protocol 2 would only work with Paillier encrypted values due to its much larger plain-text space than DGK or additive-homomorphic El-Gamal.

II. PAILLIER VARIATIONS IMPACT ON PERFORMANCE

Originally we have implemented Damgard-Jurik variant as described in [8], in which we encrypt Paillier using $E(m, r) =$

$(mn + 1)(r^n) \mod n^2$. According to Cao et al. this variation breaks the Paillier signature scheme and as our objective is to create a standard library, we determined it to be best we maintain the original implementation of Paillier.

Cao et al. also mentioned that there should be minimal performance gain using a different variation from the original Paillier scheme. We have tested the original Paillier implementation with 1,024-bit modulus. When we compare the results with Damgard-Jurik, the encryption and decryption time for 100,000 operations is about 188 seconds, compared to original variation which completed its task in 196 seconds. Given that on average the original variation will encrypt and decrypt 80 milliseconds slower, we can confirm Cat et al.'s claim the variations give minimal performance gain.

In addition, we found one aspect where the original Paillier scheme is superior to Damgard-Jurik. In the original scheme, it supported adding an encrypted value with a plain-text value to obtain the encrypted sum. When we tested this property on the Damgard-Jurik variant, we noticed this feature was broken. Therefore, there can be a performance gain as if you have the plain-text number you want to add with the encrypted number, you can skip the encryption step, which is an expensive operation. Furthermore, we have confirmed that DGK also shares this property with Paillier.

III. DGK SIGNATURE SCHEME

To create a standard library, we would need to support the creation of both X.509 and PKCS#8 certificates for Paillier and DGK. Paillier already has a built-in signature scheme, so to provide completion, we provide a DGK signature scheme.

First, we define $v = v_p v_q$, which is stored in the private key and we know the order of g is uv and the order of h is v . To encrypt with the private key, we define $\text{Sign}(m) = g^v h^m \mod n$, which outputs the signature s . We can verify by checking the signature through the computing the following operation, $s^u \mod n = h^{mu} \mod n$. If it is true, we can verify the message came from someone with the corresponding private key. Assuming prime factorization and computing discrete logarithm is hard, obtaining v from g^v will be difficult from the public key parameters (u, g, h, n). However, there is a potential risk that if $m = v$, then $h^v \equiv 1 \mod n$, so $g^v \mod n$ is sent. Since it is known that $g^{uv} \equiv 1 \mod n$, the attacker can exponentiate with u . If they obtain 1, the attacker knows the message congruent to v . From here, it should be easy to brute force to obtain the value of v . Therefore, it is critical for this signature scheme that $\gcd(m, v) = 1$.

IV. PERFORMANCE DISCUSSION

Overall it is clear that DGK is the fastest of all Homomorphic encryption systems.

V. FUTURE WORK

- Complete ElGamal Signature
- Complete Goldwasser-Micali implementation
- Support Creating certificates for all Homomorphic Encryption systems
- Submit it to Bouncy Castle Library

ACKNOWLEDGEMENTS

Andrew Quijano was supported by the US NSF REU Site at FIU under the grant number REU CNS-1461119. The work is also supported in part by a grant from Cisco Silicon Valley Foundation. We would also like to thank Dr. Samet Tonyali for his help in understanding the details of DGK.

REFERENCES

- [1] P. Paillier *et al.*, "Public-key cryptosystems based on composite degree residuosity classes," in *Eurocrypt*, vol. 99. Springer, 1999, pp. 223–238.
- [2] I. Damgård, M. Geisler, and M. Krøigaard, "Efficient and secure comparison for on-line auctions," in *Australasian Conference on Information Security and Privacy*. Springer, 2007, pp. 416–430.
- [3] I. Damgård, M. Geisler, and M. Kroigard, "A correction to efficient and secure comparison for online auctions," *Int. J. Appl. Cryptol.*, vol. 1, no. 4, pp. 323–324, Aug. 2009. [Online]. Available: <http://dx.doi.org/10.1504/IJACT.2009.028031>
- [4] B. V. Mau and K. Nuida, "Correction of a secure comparison protocol for encrypted integers in ieec wifs 2012 (short paper)," in *International Workshop on Security*. Springer, 2017, pp. 181–191.
- [5] T. Veugen, "Encrypted integer division," in *2010 IEEE International Workshop on Information Forensics and Security*, 2010, pp. 1–6.
- [6] T. Veugen, "Improving the dgk comparison protocol," in *2012 IEEE International Workshop on Information Forensics and Security (WIFS)*, Dec 2012, pp. 49–54.
- [7] —, "Correction to" improving the dgk comparison protocol". *IACR Cryptology ePrint Archive*, vol. 2018, p. 1100, 2018.
- [8] Z. Cao and L. Liu, "The paillier's cryptosystem and some variants revisited," *arXiv preprint arXiv:1511.05787*, 2015.

APPENDIX

TABLE I: 100,000 Homomorphic encryption operations with 1024-bit keys on 15-bit random plain text numbers

Time in Seconds	Paillier	DGK	El Gamal
Encryption	189	17	116
Decryption	185	3	67
Addition	1	0	1
Scalar Multi.	4	1	2
Scalar Addition	4	1	N/A
Signature			

TABLE II: 100 operations with 1024-bit keys on 15-bit random plain text numbers, d = 1000 for division

Time in Seconds	Paillier	DGK	ElGamal
Encrypted Mult.	11	11	9
Encrypted Division	16	22	19
Protocol 1	19	19	18
Protocol 2	34	N/A	N/A
Protocol 3	18	19	18
Modified Protocol 3	18	18	18
Protocol 4	34	29	23

TABLE III: 100,000 Homomorphic encryption operations with 2048-bit keys on 15-bit random plain text numbers

Time in Seconds	Paillier	DGK	El Gamal
Encryption	1396	63	803
Decryption	1390	9	425
Addition	2	1	4
Scalar Multi.	15	4	8
Scalar Addition	15	5	N/A
Signature			

TABLE IV: 100 operations with 2048-bit keys on 15-bit random plain text numbers, $d = 1000$ for division

Time in Seconds	Paillier	DGK	ElGamal
Encrypted Mult.	17	9	13
Encrypted Division	13/23	8/17	13/23
Protocol 1	21	21	21
Protocol 2	41	N/A	N/A
Protocol 3	21	21	21
Modified Protocol 3	20	20	20
Protocol 4	40	30	28