GANITA,Vol.74(2), 2024, 375-385

A COMPLETE BREAK OF AN HOMOMORPHIC ENCRYPTION ALGORITHM BASED ON POLYNOMIAL RINGS

YASH MAKWANA and ANUPAMA PANIGRAHI ✉

Abstract

With the rise of cloud computing, there is a growing demand for cryptographic algorithms that not only ensure data security but also support specific operations on encrypted data. Homomorphic encryption schemes, which enable computations such as addition and multiplication over ciphertexts, have been proposed to meet this need. However, the practical applicability of these schemes depends on their efficiency and security. In this paper, we analyze a specific homomorphic encryption algorithm and demonstrate its vulnerabilities. We achieve a complete break of the scheme by employing a known plaintext-ciphertext attack, recovering the secret key within fractions of a second. This analysis underscores the importance of thoroughly evaluating the security of homomorphic encryption algorithms before deployment in real-world applications.

2010 *Mathematics subject classification:* primary 68P25; secondary 68P27, 946A0.
Keywords and phrases: Cryptanalysis, Homomorphic Encryption.

1. Introduction

Homomorphism, a function that preserves mathematical operations between a domain and its range, is a fundamental concept in abstract mathematics. While traditional cryptography addresses the challenge of securely transmitting messages over unsecured channels, the rise of cloud computing has highlighted the need for performing computations directly on encrypted data. This capability, enabled by homomorphic encryption, has become increasingly significant in ensuring data privacy while supporting operations such as addition and multiplication of encrypted information.

Homomorphic encryption paves a way for solving this problem. Its importance can be understood by example of our data being uploaded to cloud platforms. If our data in the cloud is encrypted and the cloud platforms want to perform some operations on our data to offer a more personalized experience, they need access to our data. Some platforms may compromise our data security and use the information for their computations. This is where homomorphic encryption plays its role. If the data was encrypted using such an algorithm, the cloud platforms could perform

This research was supported by the University Grants Commission (UGC) through JRF with Ref. No. 211610136084.

required operations on our data without gaining direct access, and it would be a win-win situation for both parties.

In cryptology, homomorphism was first used and introduced by Rivest et al. [9] as a means to perform computations without the need of plaintext. Yao et al. introduced two party computation by a protocol called Yao's Garbled circuits but its implementation was an overhead [10]. A remarkable work by Gentry [4] supported all operations and was, in sense, a fully homomorphic encryption scheme but lacked practical applications [5, 6]. Since then following the idea a lot of papers have tried to develop new fully homomorphic schemes [7, 8].

Broadly, such schemes can be into three classes based on the number of operations they preserve and the number of times these operations can be used, Partially, Somewhat, and Fully Homomorphic Encryption schemes. When we can perform an operation a limited number of times and the number of such operations is also limited, it is called Somewhat Homomorphic encryption Algorithm. [2] proposed a SWHE, which can then be transformed into FHE using a refresh mechanism. It is based on polynomial rings $\mathbb{F}_2[x]$ and makes use of homomorphisms, $\phi : \mathbb{F}_2[x] \to \mathbb{F}_2[x]$.

Boneh et al. in 2005 [1], introduced a scheme that can perform addition unlimited number of times but multiplication only once. Polly-Cracker Scheme was one another algorithm introduced by Koblitz and fellows [3], and was later improved by various researchers. However, upon rigorous cryptanalysis they could not withstand new attacks. Similarly, analyze the algorithm introduced in [2] and prove it to be insecure for any application.

2. Preliminaries

To provide a clear understanding of the concepts discussed in this paper, we begin by introducing essential definitions and notations related to homomorphic encryption and cryptographic attacks.

Definition 2.1 (Partially Homomorphic Encryption). Partially homomorphic encryption refers to an encryption scheme that can perform computations on encrypted data using only a single type of mathematical operation, such as addition or multiplication. Although this restricts the range of possible applications, designing partially homomorphic encryption schemes is comparatively straightforward.

Definition 2.2 (Somewhat Homomorphic Encryption). Somewhat homomorphic encryption is an encryption scheme capable of evaluating two types of mathematical operations, but only for certain circuits. Its limitation arises when the ciphertext accumulates excessive noise, which increases computational overhead. As the noise grows, the performance of the somewhat homomorphic encryption scheme degrades, making it slower.

Definition 2.3 (Fully Homomorphic Encryption). Fully homomorphic encryption (FHE) enables the evaluation of any circuit composed of multiple types of operations with unlimited depth. This makes FHE the most powerful among the three types of

homomorphic encryption. It is suitable for complex data security scenarios and can address challenges associated with mobile networks.

Definition 2.4 (Integer Factorization Problem). Let n be any positive integer, the problem is to find distinct $p_1, p_2 \dots p_r$, such that $n = p_1^{e_1} p_2^{e_2} p_3^{e_3} \dots p_r^{e_r}$

Definition 2.5 (Homomorphism). Let A and B be any two algebraic structures with operation $\star$ and $\circ$ respectively. Then $\phi : A \to B$ is called an homomorphism if for any $a_1, a_2 \in A$,

$$\phi(a_1 \star a_2) = \phi(a_1) \circ \phi(a_2)$$

3. Design of Homomorphic Encryption

Dasgupta et al.[2] developed a new method to encrypt message, such that one can perform addition and multiplication on two encrypted messages and get the encryption after the respective operation of the two plaintext messages. Its security is claimed to be equivalent to RSA as it is dependent on the Integer factorization problem. Hoewever, upon analyzing we find that the cipher can be easily broken using some elementary arithmetic operations.

3.1. Key Generation A large random prime number k is generated and is used as a secret key in further encryption and decryption functions with binary length $l_k = \log_2(k)$. An random even integer z of binary length $\log_2(l_k)$ and the refresh key is given by $r = z * k$.

3.2. Encryption and Decryption Let $m = [m_1, m_2, \dots m_n]$ be the plaintext in the binary format that is to be encrypted and k, z, r be as defined in sec 3.1. Now, a polynomial $p(x)$ is chosen of degree n such that its coefficients p_i satisfy $m_i \equiv p_i \pmod{k}$ and a random polynomial $r(x)$ of degree n is chosen, which has coefficients of length l_k^a with a as any positive integer. The encryption of $p(x)$ is given as,

$$c(x) = p(x) + k * r(x) \tag{3.1}$$

Now, to decrypt this ciphertext, we perform a single operation.

$$\begin{aligned} d(x) &\equiv c(x) \pmod{k} \\ &\equiv p(x) + k * r(x) \pmod{k} \\ &\equiv m(x) \end{aligned} \tag{3.2}$$

As, $m_i(x) = p_i \pmod{k}$, the coefficients of $d(x)$ is equal to plaintext bits.

3.3. Claimed Security To achieve a general break of this cipher, an attacker must factorize the coefficients of $c(x)$ to identify the secret common factor k. Once k is determined, the attacker can proceed to apply the decryption operation defined in sec 3.2.

4. Cryptanalysis

We observe that after applying encryption operation the ciphertext satisfy,

$$\begin{aligned} m_i &\equiv c_i(x) \pmod{k} \\ m_i &\equiv p(x) + k * r(x) \pmod{k} \end{aligned} \tag{4.1}$$

Considering known plaintext-ciphertext attack, we assume that we know some pairs. We define c_i' as,

$$c_i' = \begin{cases} c_i & \text{if } m_i = 0 \\ c_i - 1 & \text{if } m_i = 1 \end{cases} \tag{4.2}$$

Now, all c_i are divisible by the secret prime number k. As stated, integer factorization is a hard problem; finding gcd(greatest common divisor) is computationally easy using the Extended Euclidean Algorithm. To attack the cipher we employ Algorithm1.

Input: c,m, t(size of m)
Output: Secret key k

if $m_i = 1$ **then**
  $c_i \leftarrow c_i - 1$
end
$d \leftarrow c$;
$zero_c \leftarrow [0, 0 \ldots 0]$ (same size as of c) ;
$k' \leftarrow c_0$;
$i \leftarrow 1$;
while $d \neq zero_c$ **do**
  $k' \leftarrow \gcd(k', c_i)$;
  $d \leftarrow d \pmod{k'}$;
  $i \leftarrow i + 1$;
  if $i > t$ **then**
    break
  end
end
if d=m **then**
  return *k' is the secret key*
else
  return *Insufficient data to retrieve secret key*
end

Algorithm 1: Finding secret key

The code and data can be accessed from `https://github.com/ymakwanamath/homo_enc_break` or by clicking here

Example 1. Consider an instance of encryption with parameters $l_k = 11, a = 3$. Let $k = 1657$ and plaintext m be

```
In this paper, we shall learn to break cryptosystems.
```

The binary data for this text is,

[0, 1, 0, 0, 1, 0, 0, 1, 0, 1, 1, 0, 1, 1, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 1, 1, 0, 1, 0, 0, 0,
1, 1, 0, 1, 0, 0, 0, 0, 1, 1, 0, 1, 0, 0, 1, 0, 1, 1, 1, 0, 0, 1, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1,
1, 1, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 1, 0, 1, 1, 1, 0, 0, 0, 0, 0, 1, 1, 0, 0, 1, 0, 1, 0, 1, 1,
1, 0, 0, 1, 0, 0, 0, 1, 0, 1, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 1, 1, 0, 1, 1, 1, 0, 1, 1, 0,
0, 1, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 1, 1, 0, 0, 1, 1, 0, 1, 1, 0, 1, 0, 0, 0, 0, 1, 1, 0, 0,
0, 0, 1, 0, 1, 1, 0, 1, 1, 0, 0, 0, 1, 1, 0, 1, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 1, 0, 1, 1,
0, 0, 0, 1, 1, 0, 0, 1, 0, 1, 0, 1, 1, 0, 0, 0, 0, 1, 0, 1, 1, 1, 0, 0, 1, 0, 0, 1, 1, 0, 1, 1, 1,
0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 1, 1, 0, 1, 0, 0, 0, 1, 1, 0, 1, 1, 1, 1, 0, 0, 1, 0, 0, 0, 0, 0,
0, 1, 1, 0, 0, 0, 1, 0, 0, 1, 1, 1, 0, 0, 1, 0, 0, 1, 1, 0, 0, 1, 0, 1, 0, 1, 1, 0, 0, 0, 0, 1, 0,
1, 1, 0, 1, 0, 1, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 1, 1, 0, 1, 1, 1, 0, 0, 1, 0, 0, 1,
1, 1, 1, 0, 0, 1, 0, 1, 1, 1, 0, 0, 0, 0, 0, 1, 1, 1, 0, 1, 0, 0, 0, 1, 1, 0, 1, 1, 1, 1, 0, 1, 1,
1, 0, 0, 1, 1, 0, 1, 1, 1, 1, 0, 0, 1, 0, 1, 1, 1, 0, 0, 1, 1, 0, 1, 1, 1, 0, 1, 0, 0, 0, 1, 1, 0,
0, 1, 0, 1, 0, 1, 1, 0, 1, 1, 0, 1, 0, 1, 1, 1, 0, 0, 1, 1, 0, 0, 1, 0, 1, 1, 1, 0]

Now, $p(x)$ is a polynomial that satisfy $p_i = m_i \pmod{k}$ and its coefficients are given as, $p =$

[14913, 16571, 3314, 11599, 8286, 14913, 4971, 9943, 1657, 14914,
13257, 8285, 6629, 14914, 9943, 4971, 6628, 1657, 8286, 4971, 16570,
6628, 14913, 13256, 8285, 9943, 3315, 4972, 6628, 8286, 9942,
13256, 8285, 6629, 1658, 13256, 13257, 16570, 8285, 14913, 6628,
3315, 8286, 16570, 3315, 13256, 9942, 11600, 16570, 4972, 1658,
14914, 11599, 4971, 16571, 1658, 14913, 9942, 8286, 11599, 6628,
16570, 1657, 4971, 4971, 1658, 1658, 11600, 8285, 8285, 14913,
13256, 16570, 8286, 4972, 4971, 9942, 13256, 6628, 6629, 14913,
14914, 1658, 16571, 13256, 1657, 9942, 8285, 1657, 3315, 16571,
3314, 8285, 4972, 11599, 4972, 3314, 13257, 3315, 4972, 14913,
8285, 3315, 8285, 6628, 9942, 3315, 4971, 1658, 14914, 14913,
14913, 1657, 14913, 6629, 16570, 13256, 1657, 1657, 9942, 4971,
8286, 9943, 3315, 8285, 9943, 14914, 9943, 14913, 8286, 1658,
1657, 6628, 8286, 13256, 3315, 3314, 9942, 4972, 11599, 3314,
13256, 1657, 1657, 1657, 14914, 9943, 4972, 1657, 4971, 16571,
16571, 16570, 9943, 14914, 14913, 13257, 14913, 9942, 8285, 4971,
11600, 14914, 13256, 3314, 4971, 9942, 16571, 6628, 11600, 6629,
9942, 14914, 14914, 11599, 4971, 9942, 1658, 13257, 1657, 11600,
4972, 11599, 1657, 6628, 9942, 9943, 4971, 4971, 1657, 1657,
9942, 8285, 9943, 13257, 8285, 6629, 4972, 11599, 13256, 9942,
11600, 11600, 16570, 16570, 4972, 3314, 4972, 9942, 9943, 9943,

8285, 13256, 11599, 3314, 6629, 3314, 14914, 9943, 9943, 1657,
11599, 6629, 1657, 16570, 11600, 6629, 16570, 9943, 13257, 11600,
9942, 13256, 4971, 11600, 1657, 14913, 13256, 11599, 1657, 8285,
11600, 11600, 16571, 9942, 9943, 16570, 8285, 16570, 6629, 16571,
1657, 1658, 1658, 9943, 9943, 14913, 13256, 6629, 8285, 4971,
1657, 9942, 4971, 4971, 11600, 8286, 1657, 11599, 3314, 9943,
16570, 9942, 3315, 9943, 8286, 1657, 3314, 13257, 4971, 3314,
16571, 3315, 9942, 6628, 9943, 3314, 9943, 3314, 13257, 11600,
16570, 1657, 3314, 6628, 6629, 11599, 11600, 16571, 3314, 6629,
16570, 9943, 16571, 11599, 9942, 8286, 16570, 6628, 9942, 13256,
4971, 3314, 4972, 3315, 9942, 4971, 13256, 3315, 4972, 1657,
16571, 14914, 1658, 4971, 6628, 3315, 16570, 3314, 3315, 6629,
16571, 13257, 13256, 11599, 4972, 9942, 4972, 3315, 4972, 9942,
13256, 9942, 8285, 1657, 9943, 1658, 3315, 16570, 1658, 9942,
6628, 13256, 3315, 4972, 6628, 13257, 6629, 9943, 4972, 14913,
8286, 6629, 8286, 16570, 1657, 4972, 11600, 4971, 14914, 14914,
11600, 8286, 16570, 8285, 1658, 13256, 6629, 4972, 13257, 14913,
13256, 3315, 4972, 13256, 13257, 3315, 11600, 11599, 16571, 11599,
16570, 6628, 9943, 16571, 3314, 8285, 4972, 6628, 8286, 3314,
3315, 8286, 14913, 6629, 6629, 8285, 1658, 8285, 9943, 3315, 1658,
8285, 3314, 3315, 13257, 16570, 6628, 14914, 6628, 8286, 9943, 4972, 1657]

Now, after chosing any random polynomial $r(x)$ and performing $c(x) = p(x) + k * r(x)$, we get coefficients of c as

[1879297570707, 3125669456221, 3073905251779, 3291614498351,
2457796542436, 3239320106934, 2702910842413, 2748386658947,
3320185627396, 2526350039228, 2119752737970, 2585351631930,
3052952055960, 3343640076316, 2260570708986, 2342026169500,
2258088218097, 2868993654152, 3363796018185, 3390662458769,
2915768552057, 1965624637734, 3211416226934, 2057272159433,
3152920058999, 2954866045351, 2193471120332, 1976967667689,
3497984350427, 1876183875496, 3508670928348, 3370569008998,
2198747828546, 1854843236622, 3151339933858, 2188595026663,
2908956262175, 2676229424105, 1948713661931, 2613564597111,
2234400763495, 1983980414804, 1846474543209, 2581501951999,
2125268579454, 2952006459373, 2821788198053, 2122414082124,
1851258800965, 2741190828245, 3193080183079, 2757672791835,
2316884039569, 2809105829912, 1803696042643, 2234189497653,
3217227614252, 1972086334586, 2244101873807, 2445937976699,
1828926480731, 2590836006984, 1975264861580, 2402910399207,
2974571576508, 2269502712805, 2277411065106, 3171812178800,
2060726033431, 1920769669275, 3406623236207, 2245125091190,

2252720033540, 3290299473326, 2851771104355, 3168201279196,
1839938849707, 3095121443325, 2363695015335, 2716932803920,
3226894205939, 2045697368204, 2416261091787, 2151359209325,
2327847802107, 2829176403141, 2268099742503, 2144201037261,
2218813076177, 3111625912290, 2117306089649, 3054146694964,
2969259415121, 3415502636905, 1908423905771, 3270500260359,
3404772569361, 2219250772728, 2022443298638, 3274083302478,
2867940804637, 3082873573724, 3063354584313, 2017628790688,
2733141279659, 2911401046370, 2220972925968, 1893433160988,
2020519796700, 2176098376740, 2417166085534, 3514662315576,
2183610894938, 3160289329548, 2006343039911, 2724181135009,
3497530551151, 2017323221661, 2439376636152, 2777664538259,
2527254981608, 2304303025148, 3221546134049, 2242999325891,
2445816818516, 1810755102908, 2851891765438, 1983976350183,
3063306102149, 2908828235727, 2795971362578, 2111288879732,
1862689613808, 2753765595777, 1841561059335, 2904554479786,
2575264153958, 1873924261049, 1940008097105, 3507781909737,
3034105859914, 3272255679530, 1991317756619, 3040412897357,
2846209642346, 2098967543723, 2337522110444, 2753290116313,
3133513235231, 2672471245371, 3425573268158, 3462356833859,
3266705443697, 2949082179146, 3303773549439, 2143097405667,
2513842872325, 2569553542066, 2027802111202, 2297354042706,
2801528938920, 3493942112184, 1905461608993, 2354368097049,
3320991840746, 2142850694937, 2432366619773, 2031764899611,
2279019490150, 3456612034743, 3425016451535, 2967968857357,
2976008303214, 2114766924390, 3044954397986, 2767162493800,
3352110340514, 2217760147127, 3420853898521, 3369380076701,
3334083141575, 2760993194483, 3253184018809, 3165490283037,
1952523394906, 3475734485827, 2754871666475, 1961842575002,
3208263832487, 3137943450083, 2422319232916, 2663468842250,
2008821431380, 2529762632385, 3319268277400, 2748150319379,
2045230788487, 3494403081299, 1928563125196, 2668991515545,
2041259935459, 2930651911642, 1833752375585, 2985608164196,
3412427491797, 1923209045024, 2216857925539, 2991069495352,
1844652217690, 1900749579866, 3217692251966, 2160715413428,
2464296274065, 2406177353663, 2871102009353, 2973100455455,
3115190352926, 2086864656651, 3476703138202, 3442924507204,
3350501429968, 2731710584434, 3196860096682, 2860103348738,
2743204533948, 2830464377643, 2945011009682, 2217490798462,
2470243512186, 1926147578766, 2610834848684, 3108764473786,
1979122675724, 2734997184282, 2673914974559, 2233699139985,
2020419702300, 2710028603560, 3526517235912, 2645229240765,
3263211019595, 3350177993511, 3259128342267, 2753977277527,

3132755820531, 2720847330546, 3480448478499, 1918822412585,
3367677136376, 2763538588405, 1879606670802, 3113319117739,
2969176442504, 2756526895142, 3486812031242, 2384996840642,
2071568313014, 2362746800399, 1779904581465, 2131797215791,
2794930126975, 3477511464226, 2108010037958, 2284319311858,
1887197417627, 1787296728556, 3037231995883, 1896109157993,
2248684531001, 2111162689240, 2147768523465, 2832625170265,
2921322739835, 2461738964657, 3457704289375, 1934595803208,
2897415184266, 2792347970851, 3125997379835, 2697700555043,
3329673427126, 2506724511344, 3320779267531, 1905334437556,
2343398599634, 1831049372794, 2033309009857, 1993990186104,
2193223468425, 2388948532121, 2087906738980, 2660690934774,
3191572604710, 2274761826993, 2617071375805, 2425609867560,
3065741138214, 2492045340556, 3298776401979, 3411379558601,
3074359808305, 2123346736172, 2864292995360, 2975200895167,
1923509995991, 2142618174755, 2052938822744, 2796622745847,
1817803778753, 3258945850228, 2340879482418, 2308515508542,
1810822388706, 3100956807434, 3200810588493, 1993223374556,
3419024482699, 2294524935587, 1890348446707, 2359585277540,
2675275924996, 2596556443230, 2653358222617, 2657043408844,
2547590739460, 3523092037956, 2070859160097, 2341997949133,
3072251456417, 1920007043311, 2605865228965, 2018111042312,
2064095572758, 1817627863348, 1834141662879, 2964760776112,
2125886299111, 2312723696994, 2666328038833, 2011210092987,
2204847641569, 2757290684320, 2007210578830, 3010547831925,
1909904635768, 1828793912447, 2735143414533, 2208523651331,
2560315284879, 2037889796300, 2897820017535, 1945283328932,
2160900056252, 2990095000396, 1902603361872, 2013845233342,
3542043710337, 3100924674890, 2060406915116, 3182454774970,
2624450682803, 2953045181307, 1984026648418, 3451634970123,
3531767553088, 3168187191382, 2685863102802, 2275732699748,
2188000443696, 2923456140592, 2135760600720, 1877983808316,
1931898339768, 1988373928762, 2128684204920, 3488125205399,
3516170202146, 3061738280233, 3435866797394, 3509402437511,
2985579970341, 2710754856718, 1967920984557, 1839872168714,
3134576387972, 3123732335400, 1983143008429, 2169440621991,
1804615717410, 2811802282583, 2778167001968, 2527449763615,
2350554445269, 2419477933592, 1965904468581, 1800867732540,
3280894472228, 3343784069616, 1963005313443, 3216022957026,
2621966009646, 3094743219820, 1798602802439, 2440283870164,
3493375161349, 1810777220544, 3292356526149, 3113112413618,
1932768163690, 2202088769710, 1788971141969, 2036897561502,
3131266687887, 2722066070615, 3270776664529, 2104188944592,

3032365975117, 1907774182815, 3291792624195, 1808051107573,
2770225814556, 3415855067549, 2815432189633, 3275682012531]

Now, to break this ciphertext, we employ Algorithm 1 and get the result as:

```
for i=1
kdash is 1879297570707
cdash is 3125669456220
kdash is
4971
for i=2
kdash is 4971
cdash is 3073905251779
kdash is
1657
Break Successful
Secret key is 1657
```

We can verify these results from source code given at `https://github.com/ymakwanamath/homo_enc_break`

5. Time Analysis of Break

We perform several experiments by changing the size of key, l_k and parameter a and find that time taken to break the cipher is even less than that of encryption. All the experiments were performed on i5-10210U Processor with 8 GB of RAM.The results can be verified from GitHUb Repository. The time is given in microseconds in the tables.

Size of k	Encryption Time	Break Time	Decryption Time
10	1144.17	438.69	85.830
11	1164.9131	223.398	85.592
12	1112.93	192.16	82.96
13	1098.63	160.455	73.671
14	1048.08	148.77	69.37
15	1144.88	155.21	70.095
16	936.508	259.87	87.02
17	910.75	185.96	86.78
18	900.745	231.98	69.61
19	903.36	317.81	76.05
20	1004.934	140.190	67.94

TABLE 3. Comparison of time taken with $a = 3$ and time is given in microseconds. (1 second=1000000 microseconds)

Size of k	Encryption Time	Break Time	Decryption Time
10	905.03	127.55	70.09
11	890.01	287.29	63.65
12	833.03	133.03	65.08
13	827.78	131.13	65.80
14	868.08	179.05	73.43
15	880.47	142.81	65.08
16	850.20	129.69	65.32
17	1026.15	149.01	73.90
18	1012.08	139.7	71.04
19	954.38	227.92	69.61
20	970.84	179.05	81.53

TABLE 4. Comparison of time taken with $a = 4$ and time is given in microseconds. (1 second=1000000 microseconds)

Size of k	Encryption Time	Break Time	Decryption Time
10	958.68	135.18	65.80
11	931.50	135.18	66.99
12	890.25	274.89	67.94
13	891.20	133.03	66.75
14	923.871	157.11	66.04
15	895.97	132.32	68.187
16	875.71	144.004	66.99
17	881.43	175.71	66.51
18	1019.23	161.88	72.47
19	1056.43	140.19	71.04
20	959.39	145.43	74.38

TABLE 5. Comparison of time taken with $a = 5$ and time is given in microseconds. (1 second=1000000 microseconds)

6. Conclusion

The idea of homomorphic encryption is to preserve mathematical operations under the encryption function, and we found a serious vulnerability in a homomorphic encryption cipher that helped us recover secret keys in a few microseconds for small-size keys. Even if we increase the size of key to a great extent, the break is considerably faster than the encryption. We have provided source code for the cipher, its break, and time comparison of encryption, break, and decryption function. For future direction, the attack can be made efficient by different implementation techniques, and programming languages. Moreover, using similar ideas from these papers, a strong

homomorphic cipher can be constructed whose encryption and decryption time is also reduced.

7. Material and Methods

The experiments given in the section 4 and 5 were obtained by using personal codes in python without using any extra precision. All computations were performed on the Linux Ubuntu 24.04.1 LTS Machine (Intel® Core™ i5-10210U × 8 and 8 GB of RAM).

Author contributions:
Conceptualisation: Y. Makwana, A. Panigrahi; *Software:* Y. Makwana ; *Writing-Original Draft:* Y. Makwana, A. Panigrahi

Conflicts of Interest: The authors declare no conflict of interest.

References

[1] Dan Boneh, Eu-Jin Goh, and Kobbi Nissim. Evaluating 2-dnf formulas on ciphertexts. In *Theory of Cryptography: Second Theory of Cryptography Conference, TCC 2005, Cambridge, MA, USA, February 10-12, 2005. Proceedings 2*, pages 325–341. Springer, 2005.

[2] Smaranika Dasgupta and SK Pal. Design of a polynomial ring based symmetric homomorphic encryption scheme. *Perspectives in Science*, 8:692–695, 2016.

[3] Michael Fellows and Neal Koblitz. Combinatorial cryptosystems galore! *Contemporary Mathematics*, 168:51–51, 1994.

[4] Craig Gentry. *A fully homomorphic encryption scheme*. Stanford university, 2009.

[5] Craig Gentry. Toward basing fully homomorphic encryption on worst-case hardness. In *Annual Cryptology Conference*, pages 116–137. Springer, 2010.

[6] Craig Gentry and Shai Halevi. Implementing gentry's fully-homomorphic encryption scheme. In *Annual international conference on the theory and applications of cryptographic techniques*, pages 129–148. Springer, 2011.

[7] Adriana López-Alt, Eran Tromer, and Vinod Vaikuntanathan. On-the-fly multiparty computation on the cloud via multikey fully homomorphic encryption. In *Proceedings of the forty-fourth annual ACM symposium on Theory of computing*, pages 1219–1234, 2012.

[8] Michael Naehrig, Kristin Lauter, and Vinod Vaikuntanathan. Can homomorphic encryption be practical? In *Proceedings of the 3rd ACM workshop on Cloud computing security workshop*, pages 113–124, 2011.

[9] Ronald L Rivest, Len Adleman, Michael L Dertouzos, et al. On data banks and privacy homomorphisms. *Foundations of secure computation*, 4(11):169–180, 1978.

[10] Andrew C Yao. Protocols for secure computations. In *23rd annual symposium on foundations of computer science (sfcs 1982)*, pages 160–164. IEEE, 1982.

Yash Makwana, Department of Mathematics, University of Delhi, India
e-mail: yash9281@gmail.com

Anupama Panigrahi, (Corresponding Author)Department of Mathematics, University of Delhi, India
e-mail: anupama.panigrahi@gmail.com