CSE 667 Group Project Report

MIAMI UNIVERSITY

Stream Ciphers: A Practical Solution for Efficient Homomorphic-Ciphertext Compression

Anne Canteaut, Sergiu Carpov, Caroline Fontaine, Tancrède Lepoint, María Naya-Plasencia
Journal of Cryptology '18

Authors

Mirza Sanita Haque
Md Rakibul Ahasan

Advisor

Dr. Khodakhast Bibak

Contents

1 What this paper is all about

There are three types of data, data in use, in resting condition, or transition. Data are not actively changing when they are in transition or resting condition, hence encryption and decryption of those provide the same value. On the other hand, when data is in use condition the value is changed because the purpose of cipher text is to be distinguishable from plain text considering all the mathematical operations. It is important to have a relation between plain text and cipher text so that proper key encryption and decryption are possible. Moreover, the relationship has to be hidden from an observer, else by observing the cipher text mathematical operation one can have information of the plain text and thus the encryption system is broken. Homomorphic encryption is such a way that can achieve the mutual goal of having strong encryption and mathematical operation on cipher text. The first step of homomorphic encryption for some plain text m of Alice encrypted by Bob's provided public key pk and the cipher text c will send to Charlie, $c = HE_{pk}(m)$. The purpose of this paper to find more efficient way to transfer c from Alice to Charlie. Refers to other researchers hybrid encryption provides a form of compression. For example, random key k is used for message m hence for encryption scheme E Alice sends a smaller cipher text $c^{'} = (HE_{pk}(k), E_k(m))$ and using decryption circuitt $C_{E^{-1}}$ Charlie decompressed homomorphically to original c. Here E is an additive IV based stream cipher. The authors investigate the performance of "Trivium" from eSTREAM portfolio as well as proposed a variant of 128-bit security: "Kreyvium" where homomorphic implementation constraint considered.

1.1 Encryption Scheme

Full homomorphic encryption is discovered by Gentry [1] and much exploration is done after that. Homomorphic encryption allows arbitrary computation over encrypted data and is very much useful in cloud-based services [2]. As described above the role of Alice and Charlie is distinct while transmitting c from one another even though they might be played by the same entity. However, a bottleneck of homomorphic compression expands the cipher text a lot and practically it is troublesome to transmit large c from Alice to Charlie. In [3] it was first considered the reduction of cipher text c where using an Encryption scheme E and some random key k chosen by Alice, then $c^{'} = (HE_{pk}(k), E_k(m))$ send to charlie.

$c = HE_{pk}(m) = C_{E^{-1}}(HE_{pk}(k), E_k(m)$

Charlie then recovers the message m by homorphocially evaluating decryption circuit $C_{E^{-1}}$. This can be said that in plain text m compressed encryption scheme is applied and produced compressed cipher text $c^{'}$ and the original c is recovered using cipher text decompression procedure from $c^{'}$. $|c^{'}|/|m|$, the encrypting rate is close to 1 for long message where lack of marginal improvement. However, the efficient encryption scheme will minimize the decryption overhead and should provide the same security level.

1.2 Related Work

The homomorphic implementation evaluation is done considering the optimized implementation of AES [4] or lightweight block cipher SIMON [5]. However, those lightweight block ciphers

might give a much worse homomorphic evaluation. In the general case, noise is included with the cipher text and it grows with the homomorphic operation. This increase in noise is much more affected by homomorphic multiplication then homomorphic addition, however, the noise level is determined by the multiplicative depth of the circuit. The simplicity of light weight block ciphers is maintained by increasing the multiplicative depth. In the case of low-cost homomorphic evaluation with a symmetric encryption scheme, light weight block ciphers are not appropriate [6]. The number of binary multiplication and the number of rounds is necessary to evaluate a substitution box in the encryption scheme. For low latency cipher number of rounds needs to be minimized and proper masked implementation required multiplication minimization.

LowMC [6] was the first of its type of block cipher where the above two consideration is considered with small multiplicative size and depth^2. However, there is some security issue identified in the instances of the LowMC-80 and LowMC-128. The mathematical form of encryption and decryption of the LowMC is a multivariate polynomial that has a low degree and is sparsed due to this is it prone to algebraic attacks, and variants of cube attacks. The improved version of those attacks named interpolation attacks can break LowMC. Such also yields key recovery in LowMC-128 with time complexity 2^{128} and data complexity 2^{73} due to this the designers released a tweaked version of LowMC.

1.3 Contribution

In this paper, the authors focused on the internal mode of operation of homomorphic encryption, a new design concept of the compressed encryption scheme, and consider the HE-related implementation constraint. There are two phases of homomorphic decompression are offline phase and the online phase. The offline phase is independent of the plain text and when the plain text-dependent part of compressed cipher text is received the online phase initiates. The author chooses an additive IV-based stream cipher as their encryption scheme E to make the online phase faster. However, the security level is limited to $2^{n/2}$ because the stream cipher building block is the lightweight block cipher, and n is the block size. Then less than 2^{32} is the number of blocks that can be encrypted by the same key which is not very efficient.

Therefore the author proposes the keystream generator Trivium [7] in their encryption scheme along with a new proposal of their keystream generator Kreyvium which can provide a 128-bit key and has the same internal structure. The advantage of Kreyvium over Trivium because it provides 128-bit security instead of 80-bit security with the same multiplicative depth. Both Trivium and Kreyvium are resistant to the interpolation attack on the other hand LowMC can break by interpolation attacks because permutation is not used in LowMC and attackers can compute backward. The authors instantiated and constructed their implementation using LowMC, Trivium, and Kreyvium in CTR mode and showed HE-dedicated block cipher LowMC performance can be achieved by stream ciphers.

Also, the author uses another E where multiplication depends on F_2-bilinear making the field elements homomorphically exponentiate with a log-log-depth circuit. However, this implementation remains impractical.

2 Homomorphic Encryption and Decryption

The fundamental of author's model and usual homomorphic construction is like sending the compressed cipher text from Alice to Charlie. To reiterate plain text m from charlie is encrypted by Bob's public key pk and it will send efficiently to a third party evaluator charlie.

2.1 Homomorphic Encryption

From the introduction we know HE corresponds a cipher text c with noise r which grows as long as homomorphic encryption continues. Hence efficient homomorphic decryption needs to ensure the noise r should not exceed the certain limit. When the homomorphic evaluator function is known, the purpose of the choosing system parameter such a way that noise is not exceeding the maximum bound and then we will achieve the homomorphic encryption scheme. On the other hand Gentry's *bootstrapping* procedure can provide the fully homomorphic encryption scheme (FHE) because here system parameter is independent from the evaluator function. During this procedure the decryption circuit is homomorphically evaluated to obtain FHE in the cipher text and reduced the noise smaller from the maximum bound to a state where there is the possibility of homomorphic operation. Unfortunately this procedure incur significant cost even if there are some improvement done by the researchers. Some are improving the latency of the bootstrapping procedure though it need bootstrapping after each NAND gate and homomorphically evaluated [8]. Also some did improving of the bootstrapping management, more efficient execution of bootstrapping but still the cost remains high.

The authors used the simplified setting but efficient HE schemes. For a cipher text set c_i coupled with discretized noise level $l_i = 1, 2, ...$ where 1 is the noise level of the initial cipher text and so on. Let us consider c_1 and c_2 is two cipher text and their noise level are l_1 and l_2. Then homomorphic addition $c_3 = c_1 + c_2$ will bound to the corresponding noise level $l_3 = max(l_2, l_1)$ or for homomorphic multiplication case $c_3 = c_1 * c_2$ will bound to the noise level $l_3 = max(l_2, l_1) + 1$. The authors does not consider the logarithmic increase of noise size while measuring the noise level during homomorphic addition. The multiplicative depth value of the decryption circuit define by the max value of l_i and that is the objective of the author minimizing l_i. Some other considerations are, $HE_{pk}(.)$ perform separate encryption of each plain text bit in SIMD fashion, the homomorphic evaluation latency refers the entire time of the homomorphic operation, and throughput refers to the how many blocks process in per unit time.

2.2 Different phase of Homomorphic Encryption

There are three different phases of homomorphic encryption within the evaluation of decryption circuit $C_E - 1$. The offline key setup phase uses Bob's public key and implied on all the messages before Charlie received the encrypted message with Bob's public key. If there is any component found which are independent then plain text here offline decompression will work. The online decompression phase work on aggregation of plain text dependent part and offline phase and recover the cipher text c as quickly as possible.

However this three class distinction is not available in the generic purpose formulation $c^{'} = (HE_{pk}(k), E_k(m))$, but the authors understand it is mostly relevant to a IV-based encryption.

$E_k(m) \stackrel{\text{def}}{=} (IV, E^{'}_{k,IV}(m))$

The IV can send during the offline prepossessing phase or though an alternate state which is maintained by the server. For efficient HE the latency of homomorphic decompression for charlie needs to minimize. To do that the online phase should be as minimum as possible, on way to use an additive IV-based stream cipher Z so that

$E^{'}_{k,IV}(m) = Z(k, IV) \oplus m$

Here the decompression phase depends on offline stage which refers of pk, k, IV and the plain text dependent online phase. The online phase reduced to the XOR between cipher text $E^{'}_{k,IV}(m)$ and $HE(Z(k, IV))$ keystream. Due to this reduction the decompression is free from the noise growth. It is the fact that the complex operation done on offline phase where $HE(k)$ and IV will give $HE(Z(k, IV))$.

2.3 Author Generic Construction

The author's devised a generic construction of HE, details in Fig 2.1. Here they considered the plain text $\{0, 1\}$, l_{Iv}-bit with expansion function G, a parameterized function F have input l_x, output size N, with parameter size l_k. Now let's have the understanding how the compressed encryption and cipher text decompression works.

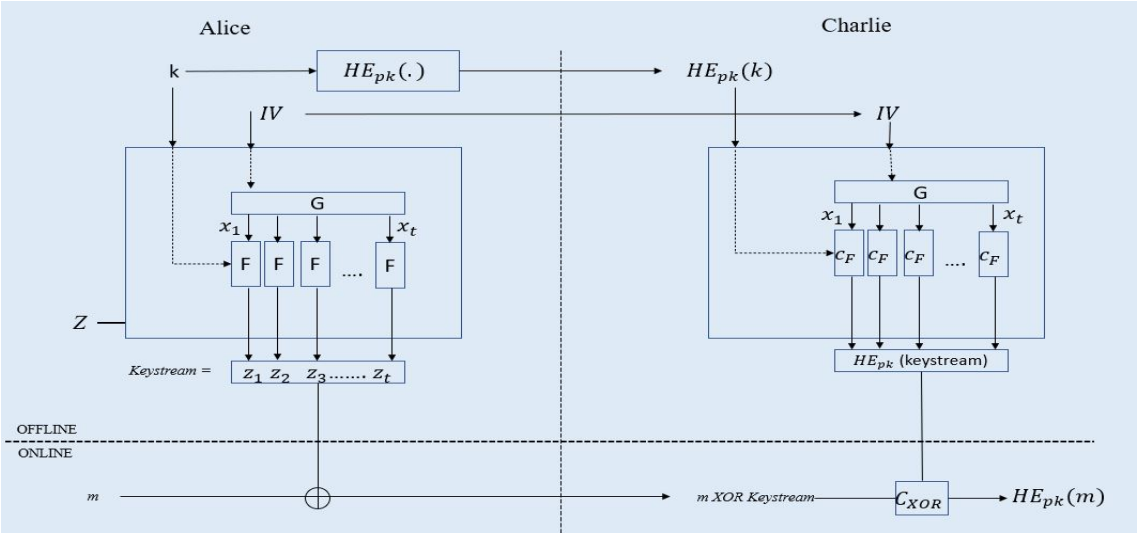

Figure 2.1: Author generic construction

Compressed Encryption: The compressed cipher text $c^{'}$ is derived using Bob's public key pk, for l_m-bit plain text m and $IV \in \{0, 1\}^{l_{iv}}$ mentioned below:

1 Define the time window $t = [l_m/N]$

2 Establishing $x_1, ..., x_t = G(IV; tl_x)$

3 Arbitrary key selection $k \leftarrow \{0, 1\}^{l_k}$

4 For a given time instance $1 \leq i \leq t$, calculate $z_i = F_k(x_i)$

5 $\oplus$ m of length l_m with keystream from left to right as $z_1, ||...||z_t$

6 Get the cipher text $c^{'} = (HE_{pk}(k), m \oplus keystream)$

Ciphertext Decompression: The decompression of compressed cipher text $c^{'}$ considering Bob's public key pk and $IV \in \{0,1\}^{l_{iv}}$ is following below steps:

1 Define the time window $t = [l_m/N]$

2 Establishing $x_1, ..., x_t = G(IV; tl_x)$

3 For a given time instance $1 \leq i \leq t$, get $HE_{pk}(z_i) = C_F(HE_{pk}(k), x_i)$ where the decryption circuit c_F.

4 From $(HE_{pk}(k), (z_1)), ..., (HE_{pk}(k), (z_t))$ get the keystream HE_{pk}

5 Get $c = HE_{pk}(m) = C_{\oplus}(HE_{PK}(keystream), m \oplus keystream)$.

In the above construction the cost of decompression is fixed and more or less equal to the C_F evaluation. Moreover the multiplicative depth also fixed and equal to the depth of circuit C_F.

The security of Compressed ciphertext Homomorphic encryption is a hybrid encryption that is a part of the key encapsulation mechanism (KEM) and data encapsulation mechanism (DEM). This framework is built on KEM which encrypts a random key that is asymmetric and DEM refers to data encapsulation with a symmetric cipher. In general understanding, the KEM-DEM framework is indistinguishable under chosen plaintext attack (IND-CPA) secure. An assumption is if KEM is symmetric secure, and DEM supports IND-CPA then the overall encryption scheme is IND-CPA secure.

The objective of the author in this paper to choose expansion function G and the function F such way that the evaluation of C_F is faster with consideration of multiplicative depth as low as possible. The fundamental approach taken not to transmit IV with compressed ciphertext $c^{'}$ as make it a constant length $IV = 0^l$ where $l = l_{IV} = l_x$. Another way make $IV \in \{0,1\}^l$ synchronized during the transmission.

The next chapter focus on using stream cipher $F_k : IV \rightarrow F_k(IV)$ and in general it is IV dependent. The process is instantiated by Trivium or author proposed Kreyvium. Now the F is a pseudorandom function if and only if the instantiated generation with a random key and along with IV is secure. Though the security of Trivium and Kreyvium is experimental however next chapter provide how those are designed and how they provide a low latency homomorphic evaluation.

The author also gives an explanation why block cipher is not a good option for function F. The lightweight block cipher Prince and Simon is limited to $2^{n/2}$ where n is the block size means the number of block encrypted by the same key is limited to $n/2$. We can see for 64-bit block like Prince and Simon the number of block encrypted by the same key will be 2^{32} that is not a big number. The suitable block cipher which is applicable is LowMC has 256-bit block size and can do encryption of 2^{128} bits under the same key.

3 Trivium and Kreyvium, Two Low-Depth Stream Ciphers

To ensure security and performance, an additive stream cipher is the first choice. For a low multiplicative depth implementation, stream cipher works well compared to other PRFs. In this section authors are focusing on keystream generation and its homomorphic evaluation. An IV-based keystream generator is deconstructed into following:

- First Sync (resynchronization function) takes the IV as input, and the key which is possibly expanded by some pre-computation phase, and n-bit initial state as output.

- The next state of the generator is computed using a transition function Φ.

- The keystream formulate from the internal state using the filtering function f.

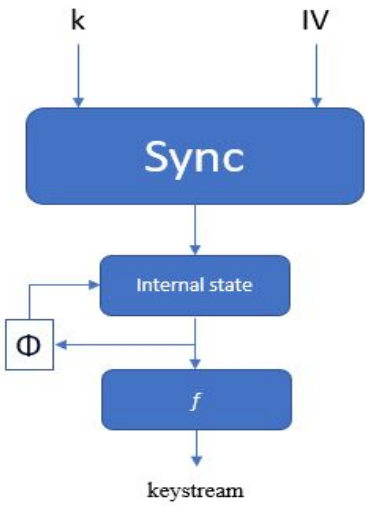

Figure 3.1: Keystream generator with Low Multiplicative Depth

3.1 Keystream Generators with a Low Multiplicative Depth

The transition function's multiplicative depth has a significant impact on the multiplicative depth of the circuit that implements the keystream generator. The homomorphic evaluation of Sync must be carried out if only the encrypted (potentially extended) key is transmitted. Consequently, a circuit with a depth of up to is needed to produce N keystream bits.

$$(depth(Sync) + Ndepth(\Phi) + depth(f))$$

Therefore, selecting a transition function with low depth is the best way to minimize this value. The extreme alternative is to select a linear function for Φ in the CTR mode where the counter mode is implemented by an LFSR, however it depends on extreme parameter selection. Hence an

alternate strategy can be select a transition which is non linear meaning depth will not increase w.r.t iteration.

For computational efficiency in HE longer key $\bar{k}$ can be transmitted before so that it can be use during the small multiplicative depth. From block cipher point of view the all round key sequence can be transmitted over server that eliminates the key scheduling task and improve homomorphic evaluation. On the other hand stream ciphers has a way of improving the encryption rate and the its throughput. The encryption ratio is define as:

$$p = \frac{|c^{'}|}{l_m} = \frac{|E_k(m)|}{l_m} + \frac{|\bar{k}|*(HE\ expansion\ rate)}{l_m}$$

Here p is the homomorphic expansion rate, also the encrypted message sent directly means the multiplicative depth is 0. The alternate scenario can be decomposed as:

1. Here the HE encrypts the secret key $\bar{k} = k$ and authors focus on rate 1 symmetric encryption scheme then

$$p = 1 + \frac{l_k*(HE\ expansion\ rate)}{l_m}$$

That is the encryption rate of l_k bit as small as possible. The l_m is limited by the length of IV, l_{iv}, $N(d)$ length of keystream for a nonce based stream cipher. The multiplicative depth is limited to $depth(d) \geq depth(Sync) + depth(f)$ and giving the minimal encryption rate with limit of length $l_m \leq 2^{l_{IV}} * N(d)$.

2. Another case is transmitting the output of the Sync function. Here with respect to size of n internal state influences the number of bits HE encrypts, while the keystream number depends on $N(d + depth(Sync))$ where d is the circuit depth for a given initial state. Then the encryption rate:

$$p = 1 + \frac{n*(HE\ expansion\ rate)}{N(d + depth(Sync))}$$

This is true for any message length l_m but the size of n is twice. So it is not interesting for the authors unless the number of plaintexts bits l_m is smaller then $2 * N(d + depth(Sync))$, encrypt by the same key.

In general the size of internal state is a bottleneck in stream cipher because it decides the storing quantity and that is related to number of gate implementation. But in author context large size of internal state is handled though it increases the non-linear operation. The fundamental of this is to use the large portion of internal state to store the secret key, this increases security without any implementation cost. The complexity of recovering internal state by is attacker is $O(2^{n/2})$ where n is size of the the secret part of the internal state. On the other hand the data complexity depend on the non-constant secret part $n^{'}$ determines the internal state collision meaning the the keystream length from the same key is bounded by $2^{n^{'}/2}$. The author also said very long keystream is require to recover but not applicable to their context because they are limiting the keystream length by the depth of circuit.

3.2 A brief of Trivium stream cipher

Trivium [7] come from eSTREAM project, that is one of the seven stream cipher from the final. The beauty of it is that during transition function it has reduce number of non linear operation. The basic construction of trivium that it has a key and IV size of 80-bit each. Three resistor 93, 84, and 111 bits are part of internal state where leftmost bit of 93-bit resister is s_1 and rightmost is s_{93}. Correspondingly for 84-bit and 111-bit those are s_{94} and s_{177}, and s_{178} and s_{288} making it a 288 bit resistor. The basic idea is illustrated below Fig 3.3.

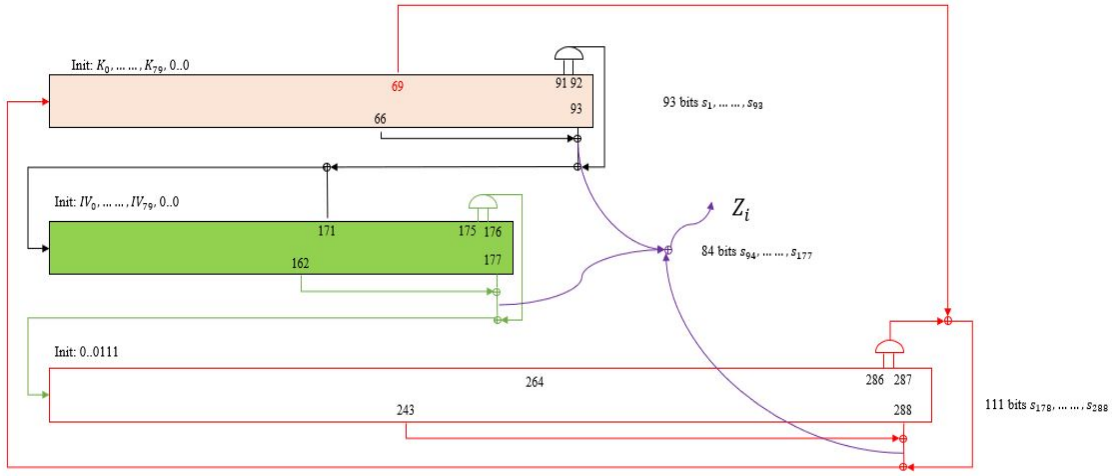

Figure 3.2: Trivium

Trivium is prone to exhaustive key search so it can be said Trivium is secure. The cube attacks and their variants recover some key bits if the initialization round reduces to 799 rounds out of 1152. The 961 is the highest round that can be attacked but that is only possible if the key is weak.

From Fig 3.3 the multiplicative depth grows slowly with the number of iteration because only the first 80-bits in register 1 is encrypted by HE and for this clear and hybrid encryption is possible like $0.[x] = 0$, $1.[x] = [x]$, $0 + [x] = [x]$, and $1 + [x] = [x]$. Here $[x]$ denotes the encrypted bit but later the homomorphic operation is avoided. This increases the number of bits generated at least 35 percent from depth 12 compare to 42 to 57. The proposition is in Trivium the production of 80-bit key after 1152 initialization rounds for multiplicative depth $d \geq 4$ gives the keystream length

$$N(d) = -1152 + 282 \times [\tfrac{d}{3}] + \begin{cases} 81 & \text{if } d \equiv 0 \, mod \, 3 \\ 160 & \text{if } d \equiv 1 \, mod \, 3 \\ 269 & \text{if } d \equiv 2 \, mod \, 3 \end{cases}$$

3.3 Kreyvium

The authors wanted to add a variant of Trivium with a 128-bit key and IV by not increasing the multiplicative depth of the corresponding circuit. The added variant has different advantages such as higher security and the number of possible IVs and the maximal length of data can be encrypted under the same key. It increases from $2^{80} N_{trivium}(d)$ to $2^{128} N_{kreyvium}(d)$. It is a challenging task to increase the key and IV size in Trivium. There are some proposals in this direction but they have seen that those proposals are less adapted than their take on Kreyvium.

Description: The proposed Kreyvium accommodates a key and an IV of 128 bits each. The main or only dissimilarity is they have added the 288-bit internal state a 256-bit part corresponding to the secret key and the IV. These two functions f and Φ depend on the key bits and IV bits, via the successive outputs of two shift registers $K*$ and $IV*$ initialized by the key and by the IV. The internal stage is comprised of different five registers of sizes 93,84, 111,128, and 128 bits. It also has an internal state size of 544 bits, among them after initialization 416 become unknown to the attacker. They have used the same notation as the description of Trivium. They have used the shift register notation as an additional register. The leftmost bit of the shift register

is denoted by $K_{127}*$ (or $IV_{127}*$) and the rightmost bit is denoted by K_0* (or IV_0*). These two registers can rotate independently from the other cipher.

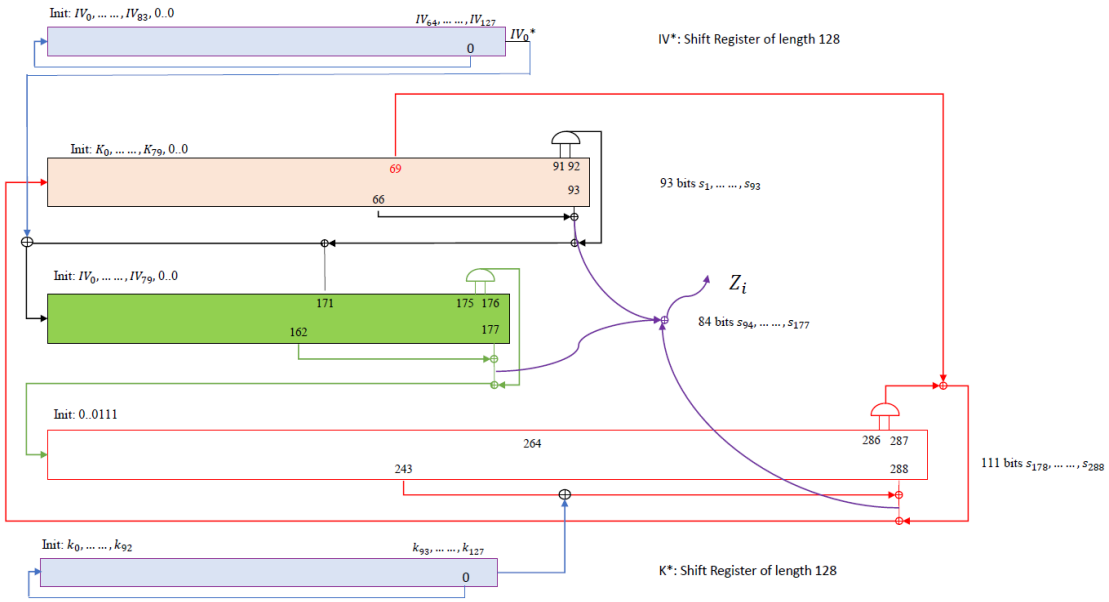

Figure 3.3: Kreyvium

In Kreyvium, the keystream length N(d) can be derived from the 128-bit key after a round of 1152 initialization with a circuit of multiplicative depth d, $d \geq 4$, is as follows:

$$N(d) = -1152 + 282 \times [\tfrac{d}{3}] + \begin{cases} 70 & \text{if } d \equiv 0 \; mod \; 3 \\ 149 & \text{if } d \equiv 1 \; mod \; 3 \\ 258 & \text{if } d \equiv 2 \; mod \; 3 \end{cases}$$

3.4 Security Analysis

There are known attacks that work on Trivium, which are also investigated on Kreyvium below:

Time-Memory-Data-Trade-Off (TMDTO): Attacks designed to restore the cipher's initial state are inapplicable because the secret part of the internal state's (416 bits) size is significantly larger than twice the key size. The size of the entire secret internal state must be taken into account, even though the additional 128-bit part corresponding to K^* is independent of the rest of the state. However, TMDTO attacks that try to recover the key are more sophisticated than exhaustive key searches since the key and the IV have the same size [9,10].

Internal State Collision: If the attacker discovers two colliding internal states, a distinguisher may be constructed because the two keystreams generated by colliding states are identical. The maximum keystream length permitted by the multiplicative depth of the circuit, which is roughly 2^{144} bits, is far greater than what is necessary to find such a collision. However, for a given key, two internal states clashing on all but IV results in two keystreams that share the identical first 69 bits since IV has an effect on the keystream only 69 clocks later. Additionally, if the leftmost bit of the difference between the two values of IV* when the rest of the state collides, this difference will impact the keystream bits $(69 + 128) = 197$ clocks later. It indicates that we can identify two similar runs of 197 consecutive bits that are equal among the roughly 2^{144} keystream bits produced from the same key. A random sequence of 2^{144} blocks are predicted

to contain substantially more collisions on 197-bit runs, hence this property is not a reliable differentiator. So, rather than limiting the number of bits created from the same key/IV pair, the birthday-bound of 2^{144} bits instead sets a limit on the number of bits produced from the same key.

Cube attacks [11,12] and Cube testers [13]: For Trivium with reduced rounds, they offer the best attacks. We can anticipate the relations to become more complex and make it difficult to use previously specified round-reduced distinguishers in our scenario since we preserve the same basic function but add two extra XORs per round, improving the mixing of the variables. In comparison to these attacks, Kreyvium is at least as resilient as Trivium, if not more so.

Conditional Differential Cryptanalysis: The attack from [14] is unquestionably interesting in our instance because of its relevance to Trivium. The largest number of blank rounds is specifically attained when certain requirements on two registers (and not just those on the register controlled by the IV bits in the original Trivium) are met at the same time. It is crucial to clarify whether an attacker might exploit our situation, where we have IV bits in two registers, by concurrently inserting disparities in the two registers. While in Trivium it is possible to inject zeros into the entire initial state in the weak-key configuration, but only in 3 bits, in Kreyvium it is considerably more difficult to use the methods from [14].

Algebraic Attacks: There are several algebraic attacks on Trivium and among them, Maximov and Biryukov [15] attacks are the most effective ones. It takes advantage of the fact that all initial state bits at indexes divisible by 3 (beginning with the leftmost bit in each register) determine the 22 keystream bits at time $3t', 0 \leq t' < 22$. Additionally, assuming that the outputs of the three AND gates at time $3t'$ are zero after all bits at location $3i$ have been determined yields three linear relationships between the internal state bits and the keystream bits. The next phase of the assault entails a thorough search for some bits at indexes that are divisible by 3. Solving the linear system derived from the keystream bits at positions $3t'$ leads to the deduction of the other bits in similar positions. The remaining 192 bits of the starting state are then derived from the other keystream equations once all of these bits have been identified. Once the outputs of the AND gates have been correctly predicted, this process must be repeated. In Trivium, the 192 linear relations involving the 192 bits at indices $3i + 1$ and $3i + 2$ require the guessing of the outputs of at least 125 AND gates. This suggests that the attack must be carried out $(4/3)^{125} = 2^{52}$ times. Inferring that only an exhaustive search with complexity 2^{32} for the additional bits at places 3i is required, we obtain several linear relations from these hypotheses that only include the bits at positions 3i. As a result, the attack's overall complexity is roughly $2^{32} \times 2^{52} = 2^{84}$. The fundamental distinction between the approach and Kreyvium is that each linear equation corresponding to a keystream bit also involves one key bit. Furthermore, any 128 consecutive output bits can be produced using independent key bits. The system now has 128 new linear equations to solve, each of which introduces a new unknown. As a result, unlike in Trivium, it is not possible to find every bit at position 3i by doing an exhaustive search on less than 96 bits. To get adequate equations on the remaining bits of the initial state, the outputs of more than 135 AND gates must also be deduced. As a result, the attack's overall complexity exceeds $2^{96} \times 2^{52} = 2^{148}$ and is significantly greater than the expense of a thorough key search. It is important to note that the attack would have been more effective if the keystream bits had not been reliant on the key, only the feedback bits.

4 Experimental Results

In the result discussion, they talked about and contrasted how useful the generic construction is when it is instantiated using Trivium, Kreyvium, and LowMC. The aforementioned algorithms are utilized to instantiate the function F, which fulfills the requirements of Propositions 1 and 2 by producing N bits of keystream per iteration, while the expansion function G implements a simple counter. Keep in mind that these claims can only be made if it is able to calculate hybrid clear and encrypted data between IV and HE ciphertexts. This explains why Tables 1 and 2 have a small variation in the number of keystream bits in each iteration (column "N") between them.

Table 4.1: Latency and throughput using HElib on a single core of a mid-end 48-core server

Algorithm	**Security Level k**	**N**	**Used × depth**	**Slots**	**Latency**	**Throughput**
Trivium-12	80	45	12	600	1414.9	1145
Trivium-12	80	45	18	720	4328	449.2
Trivium-14	80	245	14	504	1985.2	3732
Trivium-14	80	245	20	720	5276.5	2005.9
LowMC-80	80	256	14	504	1483.9	5217
LowMC-80	80	256	20	720	3690.9	2996.4
LowMC-80 [2]	80	256	14	504	1366.9	5663.5
LowMC-80 [2]	80	256	20	720	3332.6	3318.5
Kreyvium-12	128	42	12	504	1547	821
Kreyvium-12	128	42	18	756	4805.1	396.5
Kreyvium-16	128	406	16	720	5464.6	3209.6
Kreyvium-16	128	406	22	518	6667.7	1892.5
LowMC-128	128	256	16	720	4508.7	2452.9
LowMC-128	128	256	22	518	6024.7	1320.6
LowMC-128 [2]	128	256	16	720	4316	2562.4
LowMC-128 [2]	128	256	22	518	5632.6	1412.6

4.1 HE Frameworks

Two HE schemes—the BGV scheme [16] and the FV scheme [17] (a scale-invariant variant of BGV]—were taken into account in our tests. The BGV scheme is a de facto standard benchmarking library for HE applications and is implemented in the package HElib [18]. Similar to the BGV scheme, the FV scheme is one of the most effective HE schemes and has been utilized in a number of HE benchmarks [19,20,21]. We applied the FV implementation of the Armadillo compiler [19]. A C++ algorithm is converted into a Boolean circuit using this source-to-source compiler, which then produces an OpenMP parallel code that may be paired with a HE scheme.

Parameter Selection for Subsequent Homomorphic Processing: The parameters of the underlying HE scheme were typically chosen for the precise multiplicative depth necessary and not more [6,22,31,40,57] in earlier publications on the homomorphic evaluation of symmetric encryption

methods. This means that Charlie cannot do any more homomorphic computation after the ciphertext has been decompressed, which significantly reduces the significance of the claimed timings in a practical setting.

To enable Charlie to perform more homomorphic processing, we benchmarked both parameters for the precise multiplicative depth and parameters competent to handle circuits of the least multiplicative depth plus 6. They decided to add this many levels since, in practice and based on their knowledge, many applications employ algorithms with multiplicative depths lower than 7. The results we acquire using Trivium, Kreyvium, and the two variations of the LowMC cipher are compared in the sections that follow. They benchmarked both their implementation and the LowMC implementation of [6] for the original LowMC. To acquire a comparable parameterization of HElib, small adjustments to this implementation were done. The primary distinction is that the solution from [6] uses the "Method of Four Russians," an efficient technique for multiplying a Boolean vector and a Boolean matrix.

Table 4.2: Latency and throughput of our construction when using the FV

Algorithm	**Security**	**N**	**depth**	**1coreL**	**48coreL**	**Speed Gain**	**Thrpt 48 core**
Trivium-12	80	57	12	681.5	26.8	25.4	127.6
Trivium-12	80	57	18	1910.3	63	30.3	54.3
Trivium-14	80	245	14	1153.6	42.2	27.3	348.3
Trivium-14	80	245	20	2635.5	83.6	31.5	175.8
LowMC-80	80	256	14	898	106.5	8	144.2
LowMC-80	80	256	20	1787.4	179.7	10	85.5
Kreyvium-12	128	46	12	904.4	35.3	25.6	78.2
Kreyvium-12	128	46	18	2531.4	80.1	31.6	34.5
Kreyvium-16	128	407	16	2630.8	84.4	31.2	289.3
Kreyvium-16	128	407	22	5231.6	139.6	37.5	174.9
LowMC-128	128	256	16	2196	218	10	70.5
LowMC-128	128	256	22	4275.1	324.3	13.2	47.4

4.2 Experimental Results using HElib

The authors ran Trivium, Kreyvium, and LowMC in a single core using HElib, details are in Table 1. Here the time required to complete the homomorphic evaluation calls the latency and the number of blocks processed per time unit refers to the throughput. From the HElib context, the authors use the scheme where the multiplicative depth is even. Also here the number of keystream bits N is slightly higher than the theoretical value w.r.t to the proposition because HElib is in batch mode. This causes several plain texts under a single cipher text. The homomorphic operation is also being done independently but the multiplication between a clear and an encrypted data relevant to plain text polynomial while in FV mode such is not the case. And due to this, the N value obtained in FV mode is slightly larger than HElib. Also for the LowMC case, the multiplicative depth is an approximation but does not guarantee the required depth which is needed to absorb the noise generated by the homomorphic operation. Fox example 12 is the required multiplicative depth for LowMC-80 but in reality, to get a better cipher text 14 multiplicative depth is required for FV and 13 is required for the BGV scheme.

4.3 Experimental Results using FV

Table 2 has the details of kreyvium, Trivium, and LowMC comparison in the single-core and 48-core machine and this is applicable for the FV scheme. Here it is important to see in 48-core the acceleration observed is 25 percent faster for trivium and kreyvium but only 10 times for lowMC. Because multiple operations cause memory contention and memory allocation got slower.

It is observed by the author that in BGV case both Kreyvium and LowMC-128 have good performance in terms of latency but higher throughput is observed in kreyvium because in kreyvium it generates 406 bit keystream with respect to 256 bit in LowMC. For FV case kreyvium provide superior performance in terms of latency, speed gain and throughput. In overall context the trivium and kreyvium is more parallelizable then LowMC hence trivium and kreyvium can work as a HE dedicated cipher along with LowMC providing 128-bit security. Also another perspective of authors research to reduce the size of transmission between alice and charlie. Their construction supports $|c^{'}|/|m|$ which is asymptotically close to 1 for $l_m = 1GB$.

5 Conclusion

From the details work of the authors it is evident that a HE dedicated LowMC block cipher performance can also be obtained by trivium, which security is thoroughly analyzed by eSTREAM competition. Authors uses this competition experience using trivium for homomorphic encryption, moreover their proposed variant also enriched from this analysis because both architecture is essentially same.

Also in reality how much multiplicative depth is essential for reducing the noise of homomorphic encryption and not related with any performance scheme might be $|logk| + 1$ for $k - bit$ security but this approach seems impractical by the authors.

6 Contribution

We have tried to understand the paper individually and discuss it with each other. The individual contribution is given in the following table.

Table 6.1: Individual contribution details

Content	**Contribution**
What this paper is all about	MD. Rakibul Ahasan
Homomorphic Encryption and Decryption	MD. Rakibul Ahasan
Trivium and Kreyvium: Two Low Depth Cipher	Mirza Sanita Haque
Experimental Results	Mirza Sanita Haque , MD. Rakibul Ahasan
Conclusion	Mirza Sanita Haque
Reference	Mirza Sanita Haque , MD. Rakibul Ahasan
Hand Drawing Figures	Mirza Sanita Haque , MD. Rakibul Ahasan

7 References

[1] C. Gentry, Fully homomorphic encryption using ideal lattices, in STOC, (ACM, 2009), pp. 169–178.

[2] K. Lauter, A. López-Alt, M. Naehrig, Private computation on encrypted genomic data, in LATINCRYPT. LNCS (2014)

[3] M. Naehrig, K.E. Lauter, V. Vaikuntanathan, Can homomorphic encryption be practical? in ACMCCSW, (ACM, 2011), pp. 113–124

[4] J.H. Cheon, J. Coron, J. Kim, M.S. Lee, T. Lepoint, M. Tibouchi, A. Yun, Batch fully homomorphic encryption over the integers, in EUROCRYPT. LNCS, vol. 7881 (Springer, 2013), pp. 315–335

[5] T. Lepoint, M. Naehrig, A comparison of the homomorphic encryption schemes FV and YASHE, in AFRICACRYPT. LNCS, vol. 8469 (Springer, 2014), pp. 318–335.

[6] M. Albrecht, C. Rechberger, T. Schneider, T. Tiessen, M. Zohner, Ciphers forMPC and FHE, in EUROCRYPT, Part I. LNCS, vol. 9056 (Springer, 2015), pp. 430–454

[7] C. De Cannière, B. Preneel, Trivium, in New Stream Cipher Designs—The eSTREAM Finalists. LNCS, vol. 4986 (Springer, 2008), pp. 244–266

[8] L. Ducas, D. Micciancio, FHEW: bootstrapping homomorphic encryption in less than a second, in EUROCRYPT. LNCS, vol. 9056 (Springer, 2015), pp. 617–640

[9] C. De Cannière, J. Lano, B. Preneel, Comments on the rediscovery of time memory data tradeoffs. Technical report, eSTREAM—ECRYPT Stream Cipher Project (2005). www.ecrypt.eu.org/stream/pap Accessed 21 Dec 2017

[10] J. Hong, P. Sarkar, New applications of time memory data tradeoffs, in ASIACRYPT. LNCS, vol. 3788 (Springer, 2005), pp. 353–372

[11] I. Dinur, A. Shamir, Cube attacks on tweakable black box polynomials, in EUROCRYPT. LNCS, vol. 5479 (Springer, 2009), pp. 278–299

[12] P. Fouque, T. Vannet, Improving key recovery to 784 and 799 rounds of Trivium using optimized cube attacks, in FSE. LNCS, vol. 8424 (Springer, 2013), pp. 502–517

[13] J. Aumasson, I. Dinur, W. Meier, A. Shamir, Cube testers and key recovery attacks on reduced-round MD6 and Trivium, in FSE. LNCS, vol. 5665 (Springer, 2009), pp. 1–22

[14] S. Knellwolf, W. Meier, M, Naya-Plasencia, Conditional differential cryptanalysis of Trivium and KATAN, in SAC. LNCS, vol. 7118 (Springer, 2011), pp. 200–212

[15] A. Maximov, A. Biryukov, Two trivial attacks on Trivium, in SAC, vol. 4876 (Springer, 2007), pp. 36–55

[16] Z. Brakerski, C. Gentry, V. Vaikuntanathan, (Leveled) fully homomorphic encryption without bootstrapping. TOCT6(3), 13 (2014)

[17] J. Fan, F. Vercauteren, Somewhat practical fully homomorphic encryption. IACR Cryptol. ePrint Arch.2012, 144 (2012)

[18] S. Halevi, V. Shoup, Algorithms in HElib, in CRYPTO, Part I. LNCS, vol. 8616 (Springer, 2014), pp.554–571

[19] S. Carpov, P. Dubrulle, R. Sirdey, Armadillo: a compilation chain for privacy preserving applications, in ACM CCSW (2015)

[20] S. Fau, R. Sirdey, C. Fontaine, C. Aguilar, G. Gogniat, Towards practical program execution over fully homomorphic encryption schemes, in IEEE International Conference on P2P, Parallel, Grid, Cloud and Internet Computing, (2013), pp. 284–290

[21]T. Lepoint, M. Naehrig, A comparison of the homomorphic encryption schemes FV and YASHE, in AFRICACRYPT. LNCS, vol. 8469 (Springer, 2014), pp. 318–335