Family of PRGs based on Collections of Arithmetic Progressions

Srikanth Ch Veni Madhavan C.E. Kumar Swamy H.V.

Indian Institute of Science, Bengaluru, India - 560012
E-mail: {*sricheru, cevm, kumar*}*@csa.iisc.ernet.in*

Abstract. We propose a theory on the object: *collection of arithmetic progressions* with elements satisfying the property: j^{th} *terms of* i^{th} *and* $(i+1)^{th}$ *progressions of the collection are multiplicative inverses of each other modulo the* $(j+1)^{th}$ *term of* i^{th} *progression.* Under *certain* conditions, such a collection is uniquely generated from a pair of co-prime seed integers. In this work, we present one application of this object to construction of a new family of pseudo random number generators (PRG). The amortized cost per bit is shown to be $O(1)$ for keystream generation. This result is supported by empirical data.

One interesting aspect of the defined object is its connection to the standard Euclidean algorithm for finding the gcd of two numbers. The connection is useful in our proof of the amortized cost, which is based on results [14,15,17] concerning the average behavior of the quotient sequence of the Euclidean algorithm.

In security analysis, we study the difficulty of the inverse problem of recovering the seed pair from the keystream of the proposed method(s). Based on this study, we deduce a lower bound on the sizes for secret parameters that provide adequate security. The study of the inverse problem establishes a computational equivalence between a special case of it (defined as Problem A) and the problem of factoring integers.

We present an authenticated encryption scheme which is another application of the defined object. The present work leaves some open issues which we presently are addressing in our ongoing work.

Keywords : Arithmetic progression, sequence, pseudorandom number, factoring, Euclidean algorithm, authenticated encryption

Table of Contents

1 Introduction . . . 3
1.1 Inversive Congruential Generators . . . 3
1.2 Our Contributions . . . 4
1.3 Organization of the Paper . . . 4
2 Sequence of Arithmetic Progressions . . . 5
2.1 Groupings of $\mathfrak{S}(a_0, d_0)$ . . . 5
2.2 Pseudo-random Behavior . . . 6
2.3 Computing Specific Terms is easy . . . 6
2.4 The Inverse Computational Problem (Problem 1) . . . 8
2.5 Yield of $\mathfrak{S}(a_0, d_0)$ . . . 9
2.6 Keystream Generation . . . 10
3 Family of pseudorandom number generators . . . 11
3.1 Other Stream Generating Methods . . . 11
3.2 Refreshing Functions . . . 11
4 Security analysis . . . 11
4.1 Indistinguishability . . . 12
Proof. . . . 12
Randomness Testing. . . . 13
4.2 Computational Complexity of Problem 1 . . . 13
Nature of the Problem . . . 13
Complexity of Exhaustive search. . . . 14
5 Symmetric Key Encryption scheme . . . 15
5.1 Authenticated Encryption Scheme . . . 15
5.2 Sharing of Secret Key Pair . . . 16
6 Timing data . . . 16
6.1 Data on the first method . . . 16
6.2 Data on the second method . . . 18
7 Special Cases of Problem 1 . . . 18
7.1 Case 1 (Easy Instance) . . . 19
7.2 Case 2 (Problem A) . . . 20
7.3 Problem of finding (r, f) pairs . . . 22
8 Conclusions . . . 24
8.1 Performance Related Issues . . . 24
8.2 Security Related Issues . . . 24
A Properties of $\mathfrak{S}(a_0, d_0)$ . . . 26
A.1 Properties of terms within a grouping . . . 26
A.2 Proof of Theorem 1 . . . 27
A.3 Proof of Theorem 2 . . . 27
A.4 Proof of Lemma 1 . . . 27
B Proof of Theorem 5 . . . 28

1 Introduction

Arithmetic progressions are well-known objects in mathematics. They appear in a wide variety of theoretical and practical situations. In particular, the notion of primes in arithmetic progressions is useful in proving some number theoretical bounds applicable to cryptography. The notion of a *collection* of arithmetic progressions has been studied in the context of *covering systems* [19] for integers. In the present work, the object of our study is also a collection of arithmetic progressions (AP). But we provide a new perspectives based on a *defining invertibility property* introduced in this paper. We present applications of this object in the design of cryptographic primitives, especially in the construction of pseudorandom number generators. The progressions in our defined collection are ordered. Hence we call it a *sequence* of APs rather than a collection of APs.

The defining invertibility property binds together consecutive progressions in the sequence of APs. The property states that *j^{th} terms of the i^{th} and the $(i+1)^{th}$ progressions of the sequence are multiplicative inverses of each other modulo the $(j+1)^{th}$ term of the i^{th} progression*. This holds for any $i, j \geq 1$. Such a sequence is uniquely generated from a pair of co-prime integers under the condition that common differences of the progressions are in non-increasing order. The pair defines the first AP in the sequence. Properties of this sequence are closely related to the standard Euclidean algorithm.

The uniqueness existence of the sequence for a given seed pair along with an efficient computation of terms of its progressions and the random behavior of the terms lead to an efficient construction of a family of pseudorandom keystream generators. In this paper, we highlight one such generator and show the possibility of its variants. The main functionality of the generators is: given a seed pair (x_0, y_0) of relatively prime numbers, compute the pairs (x_i, y_i) using the invertibility property and expose only the sequence of numbers n_i, where $n_i = (x_i^{-1} \pmod{y_i}) + \alpha_i y_i$. Here, α_i are suitably chosen integers. The sequence of n_i forms the keystream. Our iterative construction of the sequence is reminiscent of the *inversive congruential generators*, which is discussed below.

1.1 Inversive Congruential Generators

The generating function of Inversive Congruential Generator (ICG) is:

$$x_0 = \text{seed},$$
$$x_{i+1} \equiv \begin{cases} ax_i^{-1} + c \pmod{m} & \text{if } x_i \neq 0, \\ c & \text{if } x_i = 0. \end{cases}$$

Here, m is a fixed prime number, and a, c are fixed integers. The properties of ICG are studied in [7,8,12]. Generalized ICGs are called *Compound Inversive Generators* (CIG). These are discussed in [9].

The fundamental difference between CIG and our method is that the modulus is not a fixed number in our case. This feature introduces a degree of freedom which precludes any natural, efficient reversibility (in other words, polynomial-time next-bit predictability). This will be be inferred from the nature of the inverse problem, defined as Problem 1 in Section 2.4. The security of the generators assumes the difficulty of solving Problem 1. Our preliminary

analysis indicate that the problem is hard. We discuss certain open issues concerning the problem.

1.2 Our Contributions

The aim of the paper is to show how the sequence of APs can be used in an efficient construction of pseudorandom number generators. Throughout the paper, we focus attention on this central theme and present our results around it. The results are listed below.

- A method for construction of a pseudorandom generator based on the sequence of APs and its performance and security analysis. We show that the sequence amortized cost per keybit is $O(1)$. The theoretical result is supported by empirical data on the performance.
- Proofs of two principal aspects of security of the proposed method:
  - (**Computational difficulty**) The difficulty of solving the inverse problem (defined as Problem 1) of recovering the seed from the given keystream. We analyze the hardness of the problem and show that existing techniques are not applicable to solving it.
  - (**Indistinguishability**) Negligible probability of success for a polynomial time adversary in differentiating the keystream produced by the generator from a random string of the same length.
- Using the sequence in additional ways of constructing generators, we show a family of pseudorandom number generators. We note that security of the variants is not dependent on Problem 1, and also indistinguishability game differs. We have not studied these issues in detail, thus leave them open.
- We show that a computational problem, defined in the present context, is equivalent to factoring integers. The problem is interesting for two reasons: (i) the study of this problem led us to formulate Problem 1 and (ii) in the proof of reduction from factoring to the problem, we define a number theoretic problem related to large divisor d of integer x bounded by $\sqrt{x}$.
- We present a method for Authenticated Encryption using the sequence of progressions. But the efficiency of the method demands fast arithmetic operations over integers: multiplication for encryption and division for decryption. One possible solution for achieving the efficiency is to employ Fast Fourier Transform (FFT) technique over very large integers. But this puts forth another problem of sharing of large secret integers. Despite this shortcomings, we still see a potential application of this method to bulk encryption.

1.3 Organization of the Paper

The rest of the paper is organized as follows. In Section 2, we present the properties of terms of the sequence. In Section 3, we discuss several ways of generating keystreams using the sequence of APs. In Section 4, we study the computational difficulty of Problem 1 and establish the indistinguishability feature of our generator. In Section 6, we present our experimental results which show that amortized cost per bit is $O(1)$. In Section 7, we study special cases of the inverse problem. One case, defined as Problem A, is shown to be equivalent to factoring integers. In Section 5, we present the Authenticated Encryption method. In Appendix A, we present the proofs of the theorems given in Section 2.

Note 1. Properties of the defined sequence of progressions are also presented in our other communications, which are under reviewing process. Presentation of these properties is important as they are essential prerequisite to study of any aspect of the sequence.

2 Sequence of Arithmetic Progressions

Let $A(a, d)$ denote an arithmetic progression (AP) with leading term a, and common difference d.

Suppose $A(a_0, d_0)$, $A(a_1, d_1)$, $A(a_2, d_2)$, ... is a sequence of progressions, in which the terms of the progressions satisfy the property:

$$(a_i + jd_i)(a_{i+1} + jd_{i+1}) \equiv 1 \pmod{a_i + (j+1)d_i},\ i, j \geq 0. \tag{1}$$

i.e., *j^{th} terms of i^{th} and $(i+1)^{th}$ progressions are multiplicative inverses of each other modulo $(j+1)^{th}$ term of i^{th} progression*. We refer to this property of the terms as Property $\mathcal{P}$.

The leading terms and the common differences of the progressions of the above sequence satisfy the properties:

$$\begin{aligned} d_{i+1} &\equiv a_i^{-1} \pmod{d_i}, \\ a_{i+1} &= d_{i+1} + \frac{a_i d_{i+1} - 1}{d_i}. \end{aligned} \tag{2}$$

From the properties (2), we observe that if the common differences are in non-increasing order then both a_i, d_i are unique for any $i \geq 1$. Thus the sequence is constructed uniquely and inductively from the starting progression $A(a_0, d_0)$. The unique sequence produced, starting from the progression $A(a_0, d_0)$, satisfying the Property $\mathcal{P}$ is denoted by $\mathfrak{S}(a_0, d_0)$.

Since the common differences are decreasing and consecutive common differences are co-prime, the sequence $\mathfrak{S}(a_0, d_0)$ eventually has a progression with common difference 1. After that point, the same progression repeats. Hence the sequence (or collection) $\mathfrak{S}(a_0, d_0)$ will have only finitely many distinct progressions. For example, consider the sequence $\mathfrak{S}(11, 25)$ with seed pair $(11, 25)$.

11, 36, 61, 86, ...
23, 39, 55, 71, ...
17, 24, 31, 38, ...
17, 22, 27, 32, ...
13, 16, 19, 22, ...
5, 6, 7, 8, ...
5, 6, 7, 8, ...

The sequence $\mathfrak{S}(11, 25)$ has 6 distinct arithmetic progressions.

2.1 Groupings of $\mathfrak{S}(a_0, d_0)$

A sub-collection $\mathcal{G}$ of consecutive distinct progressions of $\mathfrak{S}(a_0, d_0)$ is called a *grouping* if it satisfies the following two properties.

1. The difference between the common differences of any two consecutive progressions of $\mathcal{G}$ is the same
2. $\mathcal{G}$ is maximal.

We call the difference between consecutive common differences as the *second common difference* corresponding to $\mathcal{G}$. Note that any two consecutive groupings of $\mathfrak{S}(a_0, d_0)$ share a common progression.

For example, $\mathfrak{S}(11, 25)$ has two groupings:

$$\mathcal{G}_1 = < A(11, 25), A(23, 16), A(17, 7) >,$$
$$\mathcal{G}_2 = < A(17, 7), A(17, 5), A(13, 3), A(5, 1) > .$$

The second common difference corresponding to $\mathcal{G}_1$ is 9. The second common difference corresponding to $\mathcal{G}_2$ is 2. Groupings $\mathcal{G}_1$, $\mathcal{G}_2$ share the progression $A(17, 7)$. The sizes of $\mathcal{G}_1$ and $\mathcal{G}_2$ are 3, 4, respectively.

2.2 Pseudo-random Behavior

The terms of the progressions of $\mathfrak{S}(a_0, d_0)$ satisfy Property $\mathcal{P}$. The leading terms and common differences are related by inverse modular reduction as expressed below.

$$a_{i+1} \equiv a_i^{-1} \equiv a_i^{\varphi(a_i+d_i)-1} \pmod{a_i + d_i},$$
$$d_{i+1} \equiv a_i^{-1} \equiv a_i^{\varphi(d_i)-1} \pmod{d_i}.$$

Here, $\varphi(x)$ is the Euler's totient function (the number of numbers less than x that are co-prime to x). The random behavior of $\varphi(.)$ and inverse modular reduction introduce randomness in leading terms of progressions of $\mathfrak{S}(a_0, d_0)$. Hence, terms of progressions can be used to build cryptographic primitives. However, from a security point of view, certain restrictions need to be placed on the choice of such terms. We study these aspects in detail in Section 4.

2.3 Computing Specific Terms is easy

In this subsection, we discuss two properties satisfied by the sequence of $\mathfrak{S}(a_0, d_0)$. These properties lead to an efficient computation of specific terms of progressions of $\mathfrak{S}(a_0, d_0)$. Theorem 1 shows how to compute a term of a progression within a grouping given its main parameters.

Theorem 1. *Let L, D be the leading term and common difference of the first progression of a grouping $\mathcal{G}$. Let $\triangle$ be the second common difference corresponding to $\mathcal{G}$. Let $|\mathcal{G}|$ be the number of progressions of $\mathcal{G}$. Then, the leading term of j^{th} progression of $\mathcal{G}$ is given by $L + b\triangle + cV$. Here, $V = d - z$ with $d \equiv D \pmod{\triangle}$ and $z \equiv (d^{-1} \pmod{\triangle})$. The coefficients*

$$c = j - 1,$$
$$b = c\,(|\mathcal{G}| - j + 1).$$

(Note: d is the common difference of the last progression of $\mathcal{G}$).

Proof. Ref. Appendix A.2 □

Theorem 2 shows the relation between the main parameters of two consecutive groupings of $\mathfrak{S}(a_0, d_0)$. This will allow an efficient computation of main parameters of subsequent groupings from the main parameters of a grouping in $\mathfrak{S}(a_0, d_0)$.

Theorem 2. *Suppose $\mathcal{G}_i$, $\mathcal{G}_{i+1}$ are two successive groupings of $\mathfrak{S}(a_0, d_0)$. Let D_i, D_{i+1} be the common differences of the first progressions of $\mathcal{G}_i$, $\mathcal{G}_{i+1}$, respectively. Let $\triangle_i$, $\triangle_{i+1}$ be the corresponding second common differences. Then,*

$$\begin{aligned} D_{i+1} &\equiv D_i \pmod{\triangle_i}, \\ \triangle_{i+1} &\equiv \triangle_i \pmod{D_{i+1}}. \end{aligned} \tag{3}$$

Proof. Ref. Appendix A.3 □

From the definition of grouping and Theorem 1, the quantity $\lfloor \frac{d}{\triangle} \rfloor + 1$ is the number of progressions of the grouping $\mathcal{G}$. From Theorem 1 and Theorem 2, we can infer that the computation of a specific term of a progression can avoid inductive construction of all the progressions of $\mathfrak{S}(a_0, d_0)$. Let $N(a_0, d_0)$ denote number of progressions in $\mathfrak{S}(a_0, d_0)$. The Algorithm 1 computes $N(a_0, d_0)$ for given integers a_0 and d_0.

Algorithm 1 Computing $N(a_0, d_0)$

Input : a_0, d_0 with $d_0 > a_0 \geq 1$
Output : $N(a_0, d_0)$
1: $b \leftarrow a_0^{-1} \pmod{d_0}$
2: $\triangle \leftarrow d_0 - b$
3: $d \leftarrow d_0$
4: $n \leftarrow 0$
5: **while** $d > 1$ **do**
6: $\quad n \leftarrow n + \lfloor \frac{d}{\triangle} \rfloor$
7: $\quad d \leftarrow d \pmod{\triangle}$
8: $\quad \triangle \leftarrow \triangle \pmod{d}$
9: **end while**
10: $n \leftarrow n + 1$
11: Return n

The each iteration of the Algorithm 1 corresponds to a grouping. The algorithm (starting from Step 2) is an variant of the Euclidean GCD algorithm on input $(\triangle, d)$. Since the *successive iterations of the Euclidean algorithm are wrapped in a single iteration of Algorithm 1*, the number of iterations of the Algorithm 1 is half that of the Euclidean algorithm.

The average case analysis of the Euclidean algorithm [14,16,17] shows that the average number of iterations of the Euclidean algorithm on co-prime input pair $(\triangle, d_0)$, where $\triangle < d_0$, is about

$$\frac{12 \log 2}{\pi^2} \log d_0 + P + O\big(d_0^{-1/6} + \epsilon\big) \approx 0.842 \log d_0.$$

Here, P is called *Porter's constant* [17], whose value is approximately 1.4670. Hence, the expected number of groupings of $\mathfrak{S}(a_0, d_0)$ is about $0.421 \log d_0$.

Lemma 1. *The maximum number of groupings of* $\mathfrak{S}(a_0, d_0)$ *is less than* $\frac{\log_\phi \sqrt{5}d_0 - 2}{2}$. *Here,* $\phi = \frac{\sqrt{5}+1}{2}$ *is connected to Fibonacci numbers.*

Proof. Ref. Supplementary Material. □

The quotients in the Euclidean algorithm are more often small. Due to this, the expected value of $N(a_0, d_0)$ is $O\big(\log d_0\big)\big)$.

The Algorithm 1 can be modified to compute specific terms, which is given in Algorithm 2. The running time of both algorithms can be proved to be $O\big(\log^2 d_0\big)$.

Algorithm 2 : Algorithm for computing specific terms

Input a_0, d_0
Output Leading term U and common difference V of the middle progression of each grouping of $\mathfrak{S}(a_0, d_0)$

$L \leftarrow a_0$
$D \leftarrow d_0$
$\triangle \leftarrow D - L^{-1} \pmod D$
$i \leftarrow 1$
while $i \leq r$ **do**
  $\lambda \leftarrow \lfloor \frac{D}{\triangle} \rfloor$
  $Z \leftarrow \frac{L\triangle+1}{D}$
  $D \leftarrow D \pmod \triangle$
  $v \leftarrow D - Z$
  if $\lambda \geq 2$ **then**
    $U \leftarrow L + \lceil \lambda/2 \rceil (\triangle(\lambda \lceil \lambda/2 \rceil - 1) + v)$
    $V \leftarrow D + \lfloor \lambda/2 \rfloor \triangle$
    $i \leftarrow i + 1$
  end if
  $L \leftarrow L + \lambda v$
  $\triangle \leftarrow \triangle \pmod D$
end while

2.4 The Inverse Computational Problem (Problem 1)

As discussed in previous subsection, given the integers a_0, d_0, we can efficiently compute terms of the progressions of $\mathfrak{S}(a_0,\ d_0)$ using Algorithm 2. The inverse computational problem is to recover the initial parameters a_0, d_0 given a few terms of some arbitrary progressions of $\mathfrak{S}(a_0, d_0)$. From a security point of view, this reduces to the question *what is the minimum number of terms to be revealed so that computing* a_0, d_0 *remains difficult?* Special cases of the problem are:

- Given $\langle f_1, f_2, f_3 \rangle$, the leading terms of *three progressions* of a grouping $\mathcal{G}$ of $\mathfrak{S}(a_0, d_0)$, recovering the main parameters of $\mathcal{G}$ has complexity of $O\big(|\mathcal{G}|^3\big)$ where $|\mathcal{G}|$ is the number of progressions of $\mathcal{G}$. The expected size of a grouping of $\mathfrak{S}(a_0, d_0)$ is $\log d_0$.
- Given $\langle f_1, f_2 \rangle$, the leading terms of *two consecutive progressions* of grouping $\mathcal{G}$ of $\mathfrak{S}(a_0, d_0)$, computing the main parameters of $\mathcal{G}$ is referred to as **Problem A**. This problem is polynomial-time equivalent to factoring $f_1 f_2 - 1$. The reduction from Problem A to integer factoring problem is probabilistic, while the other way reduction is deterministic.

The analysis of these cases is presented in Section 7. The study of computational hardness of the cases lead to the following natural problem. The security of the proposed pseudorandom number generators assumes the computational difficulty of this problem.

Problem 1. Given the terms of some random progressions of $\mathfrak{S}(a_0, d_0)$, L_1, L_2, ..., L_t, with each term from a distinct grouping of $\mathfrak{S}(a_0, d_0)$, recover the integers a_0, d_0.

It appears that, this problem, and the case of *two non-consecutive terms* of the same grouping being given, do not admit a natural characterization as in the case of Problem A. This observation together with the analysis formulations in Section 4.2 lead us to believe that Problem 1 is computationally hard. We strengthen this claim by providing certain heuristic arguments in Section 4.2. These arguments are based on a representation of the inverse problem as one of solving a nearly-defined system of algebraic equations over integers.

2.5 Yield of $\mathfrak{S}(a_0, d_0)$

A method of generating a pseudo-random keystream based on the sequence $\mathfrak{S}(a_0, d_0)$ is to *concatenate the (leading) terms of some arbitrary progressions of the sequence*. In this paper, we focus our attention on this method, and analyze its performance and security features. Later in the paper, we note other ways of constructing a keystream generator. This allows us customization feature for the generation of keystreams. The performance and security analysis of the proposed method will be useful in understanding other generators as well.

The number of bits that can be generated by terms of the progressions of $\mathfrak{S}(a_0, d_0)$ is called *yield* of $\mathfrak{S}(a_0, d_0)$. At the same time, given the produced bits, it should be infeasible to recover the pair (a_0, d_0). In the last section, we have defined a hard problem (Problem 1). So the yield that we defined is dependent on the hardness of the problem. The required security parameter determines the balance between the yield and the hardness of the Problem 1. We note that one can can use arbitrary terms of the sequence with the condition that the terms are from distinct groupings. We consider two different ways, which are defined as follows.

- $STR_1(a_0, d_0)$ is the binary string (or keystream) that is formed by concatenating the leading terms of *first* progressions of groupings, subject to the condition that the chosen leading terms are from distinct groupings. The yield $Y_1(a_0, d_0)$ is defined as the length of $STR_1(a_0, d_0)$.
- $STR_2(a_0, d_0)$ is the binary string (or keystream) that is formed by concatenating the leading terms of *middle* progressions of groupings, subject to the condition that the chosen leading terms are from distinct groupings. The yield $Y_2(a_0, d_0)$ is the length of $STR_2(a_0, d_0)$.

The amortized cost per bit, when the first progressions of groupings are used, is $C_1 = Y_1(a_0, d_0)/C$. The amortized cost per bit, when the middle progressions of groupings are used, is $C_2 = Y_2(a_0, d_0)/C'$. The quantities C and C' are the running time complexities of producing the keystreams, respectively.

In Algorithm 2, the generation of keystream requires one inverse computation. After the computation, the steps of the algorithm are similar to the Euclidean algorithm. In addition to these computations, the algorithm performs a few additions and multiplications. Thus it can be verified that both C and C' are $O\left(\log^2 d_0\right)$.

Lemma 2. *Suppose the number of leading terms used in producing keystream $STR_1(a_0, d_0)$ is t. Then, for large enough d_0, the expected values of $Y_1(a_0, d_0)$ and $Y_2(a_0, d_0)$ are $\Theta(t \log_2 d_0)$.*

Proof. We sketch the proof for $Y_1(a_0, d_0)$. The proof for $Y_2(a_0, d_0)$ is similar.

Let the chosen leading terms be L_1, L_2, ..., L_t. Let the corresponding common differences be D_1, D_2, ..., D_t. Let the ratio $\lfloor L_i/D_i \rfloor$ be r_i. Let $|x|$ denote the number of binary bits of string (number) x.

$$Y_1(a_0, d_0) = \sum_{i=1}^{t} |L_i| \leq t|L|$$

where L is the greatest of L_i, $1 \leq i \leq t$. From Theorem 1, $L < d_0^2$. So, $Y_1(a_0, d_0) < ct \log_2 d_0$ for some constant c. The derivation of a lower-bound uses the condition of Problem 1: the chosen leading terms should be from different groupings, which means $r_{i+1} > r_i + 2$. Thus,

$$\begin{aligned} Y_1(a_0, d_0) > |\prod_{i=1}^{t} L_i| &> |\prod_{i=1}^{t} r_i D_i| \\ &\geq |2^t t! \prod_{i=1}^{t} D_i| \\ &> |\left(\frac{(2tD)}{e}\right)^t|. \end{aligned}$$

Here, D is the smallest of D_i, $1 \leq i \leq t$. The above approximation uses the Stirling's formula for the factorial of a number. If D is $O(\log_2 d_0)$, then the result follows. Indeed D is $O(\log_2 d_0)$. We see this in our experiments. □

Corollary 1. *If the quantity t (from the above lemma) is $c * \log_2 d_0$ for some constant $0 < c < 1$, then both C_1 and C_2 are $O(1)$.*

Proof. If t is $c \log_2 d_0$, then $Y_1(a_0, d_0)$ is $\Theta(\log^2 d_0)$. From the definition of C_1 and C_2, the result follows. □

2.6 Keystream Generation

By Lemma 2, the yield of $\mathfrak{S}(a_0, d_0)$ is $\Theta(\log^2 d_0)$. The main idea for generating a keystream of arbitrary length is to refresh at least one of the main parameters a_0 or d_0 to a new value so that each time a new sequence of arithmetic progressions is used. A method is to increase one of a_0, d_0 by a fixed value.

Suppose $(a_0^{(0)}, d_0^{(0)})$ is the seed pair. Using some refreshing function, distinct pairs $(a_0^{(1)}, d_0^{(1)})$, $(a_0^{(2)}, d_0^{(2)})$, ... are generated iteratively. Let KS_b denote the keystream produced by concatenating leading terms of the progressions of $\mathfrak{S}(a_0^{(i)}, d_0^{(i)})$, $i \geq 0$, one after another. For $b = 1$, the leading terms of first progressions of groupings are used. For $b = 2$, the leading terms of middle progressions of groupings are used. So, we have

$$KS_b := STR_b(a_0^{(0)}, d_0^{(0)})||STR_b(a_0^{(1)}, d_0^{(1)})||STR_b(a_0^{(2)}, d_0^{(2)})|| \ldots$$

The operator $||$ is concatenation.

3 Family of pseudorandom number generators

The above section presents two different methods of using terms of $\mathfrak{S}(a_0, d_0)$ for producing a keystream. The terms can be used in various ways in producing the keystream, thus we obtain a a family of pseudorandom number generators. The following discussion highlights this feature.

3.1 Other Stream Generating Methods

In Section 2.5, a keystream is generated by just concatenating the computed terms X_i where $X_i = Q_i^T B_i$ where $Q_i = [a_i, b_i, c_i, d_i]$, and $B_i = [L_i, \triangle_i, V_i, D_i]$. For the methods presented in the above section, we have $a_i = 1$, $d_i = 0$, $1 \leq i \leq t$, which means that X_i is the leading term of some random progression of $\mathfrak{S}(a_0, d_0)$.

One can set different values for the entries of the coefficient vector Q_i to generate a different keystream. For a new method, the computed numbers $X_i' = Q_i'^T B_i$ will not be related (directly) to $\mathfrak{S}(a_0, d_0)$. The security will not be dependent on Problem 1.

- A general way of computing the numbers X_i is by performing the operation $X_i = Q_i^T \odot B_i$. The operator $\odot$ can be term-by-term addition or XOR.

3.2 Refreshing Functions

By Lemma 2, the yield of $\mathfrak{S}(a_0, d_0)$ is $\Theta(\log^2 d_0)$. A keystream of arbitrary length is generated by refreshing at least one of the main parameters a_0 or d_0 to a new value so that each time a new sequence of arithmetic progressions is used in keystream generation.

Refreshing the main parameters (a_0, d_0) can be done in various ways. Other than the simple increment method discussed in Section 3, an example for a refreshing function is as follows. Fix a value for d_0, and set $a_0 \leftarrow a_0^z \pmod{d_0}$ for a fixed integer $z > 0$. For efficiency, the value of z is kept small. In this *powering method*, each new value of a_0 will be co-prime to d_0 which is an advantage over the simple increment method. In case of the simple increment, for a new pair (a_0, d_0), if a_0 is not co-prime to d_0, the common factor $\gcd(a_0, d_0)$ will be taken off a_0, so the sequence $\mathfrak{S}(\frac{a_0}{gcd(a_0,d_0)}, d_0)$ is used in the keystream generation.

A set of refreshing functions is based on utilizing the elements of the present sequence to update the values of the main parameters.

4 Security analysis

In this section, we analyze the keystream generated by the generator defined in the Section 3. We discuss indistinguishability property and prove that the success probability of distinguishing keystream generated by our generator from a random string by a probabilistic polynomial time adversary is negligible. We also discuss the difficulty of solving the inverse problem (defined as Problem 1) by exhaustive search method.

4.1 Indistinguishability

Let the ensemble $\mathcal{X} = \{X_n\}_{n\in\mathbb{N}}$ be the sequence of numbers produced by our method described in Section 3. By the method, $\mathcal{X}$ can be split into subsequences, $\mathcal{X}_0$, $\mathcal{X}_1$, $\mathcal{X}_2$, ..., of contiguous numbers of $\mathcal{X}$. The numbers in the subsequence $\mathcal{X}_i$ are nothing but leading terms of progressions of $\mathfrak{S}(a_0^{(i)}, d_0^{(i)})$. By the behavior of leading terms, the numbers in a subsequence diminish in size as their index increases.

Suppose $\mathcal{A}$ is a (probabilistic) polynomial time adversary. He knows the size properties of numbers within a subsequence of the ensemble $\mathcal{X}$. In other words, a random oracle $\mathcal{O}$, to which $\mathcal{A}$ queries, publishes in advance the length of the number it is going to output. After querying by $\mathcal{A}$, $\mathcal{O}$ reveals numbers in $\mathcal{X}$. The oracle now challenges $\mathcal{A}$ by providing either a random string or leading term L of the same length. The leading term belongs to $\mathcal{X}_i$ for some i. Now $\mathcal{A}$ gas to guess correctly whether the given number is random or it is the leading term belonging to $\mathcal{X}$. We argue that he will not be able to distinguish the two events if the given integer is large enough. In other words, we show the following.

$$|\Pr[\mathcal{A}(r) \to 1] - \Pr[\mathcal{A}(L) \to 1]| = \text{negligible}.$$

where r denotes a random integer of the same length as L from $\mathcal{X}$.

For proving the result, we recall the known number theoretic bounds [1,2,3] on the Euler's totient function φ.

1. For $x > 2$,
$$\varphi(x) > \frac{x}{e^{\gamma}\log\log x + \frac{3}{\log\log x}}.$$
2. As $x \to \infty$
$$\sum_{i=1}^{x}\varphi(i) = \frac{3x^2}{\pi^2} + O\left(x(\log x)^{\frac{2}{3}}(\log\log x)^{\frac{4}{3}}\right)$$

In the above formula, γ is the Euler's constant whose value is approximately 0.5772.

Proof. Let the leading term be L. Let its corresponding common difference be D. Let $\triangle$ be the second common difference corresponding to grouping where L belongs to. Let the ratio $\lfloor L/D \rfloor = k$. Here k is nothing but the index of progression used. So, for a given L, the number of distinct $(D, \triangle)$ pairs will be equal to the number of $D \in (\frac{L}{k+1}, \frac{L}{k})$ that are co-prime to L. We derive this number as follows.

For number n and constants $0 < c_1 < c_2 \leq 1$, define $B(n; c_1, c_2) = \{x \in (c_1 n, c_2 n) : gcd(x, n) = 1\}$. For large enough n, it can be proved that $B(n; c_1, c_2)$ is approximately $(c_2 - c_1)\varphi(n) + O\big(\omega(n)\big)$ where $\omega(n)$ is the number of distinct prime factors of n.

So the total number of distinct co-prime pairs covered by L is

$$B(L; 1/k+1, 1/k) = \frac{\varphi(L)}{k(k+1)} + O(\omega(L)).$$

The average order of $\omega(L)$ is $\log\log L$, which is very small compared to $\varphi(L)$, thus we omit this in subsequent expressions. By using bounds on φ (given above), we obtain

$$\frac{L}{k(k+1)(e^{\gamma}\log\log L + 3)} < B(L; 1/k+1, 1/k) < \frac{L-1}{k(k+1)}. \tag{4}$$

Let l be the length of L. Since $k < L/D < (k+1)$, the total number of possible $(D, \triangle)$ pairs is

$$\sum_{D=\frac{2^{l-1}}{k+1}}^{\frac{2^l-1}{k}} \varphi(D) = \frac{3 * 2^{2l}}{\pi^2}\left(\frac{1}{k^2} - \frac{1}{4(k+1)^2}\right) + O(l * 2^l).$$

The total number of pairs is approximately $\left(\frac{3*2^l}{2k\pi}\right)^2 + O(l * 2^l)$. The inequality (4) indicate that each leading term $L \in [2^{l-1}, 2^l - 1]$ contributes to almost the same number of $(D, \triangle)$ pairs, which can not be distinguished by polynomial time adversary $\mathcal{A}$ if L is large enough. This observation is formally given in the following probability arguments.

$$|\Pr[\mathcal{A}(r) \to 1] - \Pr[\mathcal{A}(L) \to 1]| = \frac{1}{2^{l-1}}\left(1 - \frac{2k^2\pi^2}{9e^{\gamma}(k(k+1)\log l}\right).$$

Here L is the smallest of all leading terms. If L is large enough, the probability is negligible. Note that k is $O(l)$.

Randomness Testing. We have tested the keystreams generated by our generators for randomness using the NIST Statistical suite. For different seed pairs (a_0, d_0), we have generated keystreams. These keystreams are subjected to tests in NIST suite. A brief description of the test is as follows. For each fixed value of d_0 we have used 100 different values of a_0. For each pair value of a_0, d_0, we have generated $STR_2(a_0, d_0)$. Note that before appending leading terms, we removed most significant bit (as it is always 1). We have used 100 values of d_0 of sizes from 128 to 4096 bits. The keystreams generated passed NIST tests at high significance levels.

4.2 Computational Complexity of Problem 1

Here, we discuss the algebraic nature of the problem. By the nature of the problem, it appears that the existing techniques are not applicable directly to solving the problem. We establish the complexity of exhaustive search.

Nature of the Problem From Theorem 1, each leading term used in the keystream generation is a linear combination of three unknown variables: $\langle L_i, \triangle_i, V_i \rangle$. Note that V_i can be expressed in terms of L_i, $\triangle_i$, so we have two degrees of freedom. In other words, a leading term introduces an equation in 2 unknowns. From each sequence of progressions, t leading terms are used in the key generation where t is $c * k$ with $c < 1/2$ and k is the size of secret parameter (i.e., d_0). Thus, corresponding to a sequence of progressions, we get a system of t equations in $2t$ unknowns. By Theorem 2, variables from two successive equations are related through modular multiplicative inverse and modular reduction operations. These modular operations introduce successive quadratic non-linearities in the system.

From the known keystream $KS_b := STR_b(a_0^{(0)}, d_0^{(0)})||STR_b(a_0^{(1)}, d_0^{(1)})||\ldots$ we get systems of equations $\mathcal{S}_1$, $\mathcal{S}_2$, $\mathcal{S}_3$, ... Each $\mathcal{S}_i$ is a system of t equations in $2t$ variables. We get a nearly-defined system of equations.

The security of recently proposed cryptographic systems are dependent on the difficulty of solving systems of multivariate quadratic polynomial equations. This problem is NP-hard in general. For both under-defined [5] and over-defined systems [4], efficient methods exist for finding solutions. For nearly-defined systems (i.e, number of equations is same as the number of variables), the best techniques are exhaustive search for small fields and a Gröbner base algorithm for large fields. Gröbner base algorithm is not feasible when the number of variables ≥ 15. Our present system is a nearly-defined system of equations with number of variables ≥ 60. For example, for d_0 of size at least 256 bits, the method uses leading terms from at least 30 groupings.

We derive below the complexity of an exhaustive search for finding a solution to the problem. The established complexity is useful in recommending the sizes of d_0 that provide adequate protection. The parameter a_0, which will be set to a value less than d_0, should be large enough accordingly.

Complexity of Exhaustive search. The exhaustive search is to try out all possible values for the common difference of the progression whose leading term is L_1. For each chosen value of the common difference, say D_1, the process of constructing the sequence of progressions $\mathfrak{S}(L_1, D_1)$ is easy, due to Algorithm 2. After constructing the sequence it amounts to verifying whether remaining terms L_2, L_3, ..., L_t appear as leading terms or not. The complexity of the process is $O(\log^2 L_1)$ for one value of D_1.

In the worst case, this exhaustive search tries out all possible values of the common difference D_1. From the properties of the terms of $\mathfrak{S}(a_0, d_0)$, it can be inferred that the ratio of leading term to common difference for a progression gives the index (position) of that progression in the sequence $\mathfrak{S}(a_0, d_0)$. The ratio is upper-bounded by $N(a_0, d_0)$ the number of progressions of $\mathfrak{S}(a_0, d_0)$. Let z_1 be the position of the progression whose leading term is L_1. Then, D_1 falls in the interval $(\frac{L_1}{z_1}, \frac{L_1}{z_1-1})$. Thus the number of possible values for D_1 is about $\frac{L_1}{z_1(z_1-1)}$. For each value of D_1, the method requires an inverse computation $\pmod{L_1}$. So, the total complexity of the search for solving the problem is about $\frac{L_1 * \log^2 L_1}{z_1(z_1-1)}$.

In general, the exhaustive search can be carried out on any L_i, $1 \leq i \leq t$. Let z_i be the position of the progression whose leading term is L_i. The complexity of the search at L_i is $\frac{L_i * \log^2 L_i}{z_i(z_i-1)}$.

The search should be infeasible for any L_i. Suppose the complexity of the search is minimum for L_t. Then,

$$\frac{L_t * \log^2 L_t}{z_t(z_t - 1)} > 2^{80}.$$

The index $z_t < N(a_0, d_0)$. The expected value of $N(a_0, d_0)$ is $O(\log^2 d_0)$. Let $L_t = d_0^c$ for some $0 < c < 1$. Then,

$$\frac{c^3 d_0^c}{\log^2 d_0} > 2^{80}.$$

For $c = 1/2$, the integer $d_0 > 2^{190}$ provides 80-bit security against the exhaustive search. The value of c decides the number of groupings that could be used for keystream generation. For secure against quantum search algorithm [11], the condition should be $\frac{c^3 d_0^c}{\log^2 d_0} > 2^{160}$, which

implies that $d_0 > 2^{340}$. However, our method is easily scalable to bigger values of a_0 and d_0 (see Section 6).

5 Symmetric Key Encryption scheme

In a symmetric key scheme, both parties, Alice and Bob share a secret key k. Alice first encodes a message P into a number M. A pseudorandom keystream K is generated using the shared key k. The bits of M are XORed with the bits of K to obtain cipher text C. At receiving end, Bob uses the same shared key k to generates the same keystream K. K is used decrypt the encrypted text C to obtain M. Finally, Bob decodes M to recover the plain text message P.

In our scheme, the key k shared between the Alice and Bob is a pair (a_0, d_0) of co-prime integers, with $d_0 > a_0$. Both parties will start with the sequence $\mathfrak{S}(a_0, d_0)$ to generate the same keystream. The same refreshing function is used at both ends.

5.1 Authenticated Encryption Scheme

We have seen that there are various ways of using the terms of the object $\mathfrak{S}(a_0, d_0)$ for generating keystreams, and hence there are many ways possible for encryption of messages. In the following, we describe a simple encryption algorithm, which is also an example of Authenticated Encryption.

Suppose the message (number) M is divided into smaller numbers $\langle M_1, M_2, \ldots, M_t\rangle$. The encryption algorithm $\mathcal{E}$ is defined as follows: The cipher block for M_i, $1 \leq i \leq t$, is $C_i = L_i + M_i * D_i$. In other words, C_i is M_i^{th} term of progression $A(L_i, D_i)$. At receiving end, M_i is recovered by performing $(C_i - L_i)/D_i$. We see that if cipher data is tampered, then the decryption fails. Precisely, we prove that if man-in-the-middle tampers the cipher data and the tampered data is decrypted properly, then this means that he will be to decrypt the messages by himself. The following result proves this.

Lemma 3. *If a person, called Eve, modifies cipher block C_i to C_i'. If C_i' is decrypted properly by the recipient, then this implies that Eve can actually decrypt the subsequent messages.*

Proof. The modified cipher block C_i' is decrypted properly by the recipient. This means that $C_i' = L_i + M_i' * D_i$. This could have happened only when Eve could add a multiple of D_i to C_i. This means that Eve knew D_i, so she could decrypt C_i to recover M_i. Also, she would be able to decrypt subsequent cipher blocks as well . □

The problem with the method is that cipher blocks will be of different length. This problem can be solved by having varying length message blocks. Suppose we want all cipher blocks to have same length, say ℓ bits. Then the message M will be split into $\langle M_1, M_2, \ldots, M_t\rangle$ where $|M_i| = \ell - |D_i|$. Here $|x|$ is the number of binary bits of x.

The algorithm is inefficient as the cost of multiplications employed for producing the cipher blocks (i.e., $L + M * D$) dominate the cost of producing (L, D) pairs. The difference between the two costs will be small if one uses Fast Fourier Transform (FFT) based multiplication algorithm. The algorithm is efficient on very large operands. We have not yet tested the performance of the encryption method using FFT.

Indeed, it is generally accepted that the FFT algorithms perform better than both Karatsuba and quadratic methods for operands of sizes beyond 4K bits.

Other than this performance issue, there are trivial drawbacks by which security of the scheme will easily be compromised. For example, if a message block, say M_i is power of 2, then L_i and D_i will be known from C_i the cipher block for M_i. Such attacks can be prevented by some padding technique.

5.2 Sharing of Secret Key Pair

The proposed symmetric key scheme requires a pair (a_0, d_0) of integers to be shared between the sender Alice and the receiver Bob. Sharing of secret can use the well-known public key algorithms: (i) RSA algorithm [18], (ii) Diffie-Hellman (DH) key exchange algorithm [6]. Another public key system, which is getting close to being optimal, is variants of NTRU cryptosystem [13]. Using the DH method, Alice and Bob will come to agree upon a secret integer in two message exchanges. Thus, four message exchange are required for sharing the pair: 2 message exchanges for agreeing upon a_0, and 2 exchanges for agreeing upon d_0. On the other hand, a message encrypted using the RSA algorithm is large enough to hold both a_0 and d_0.

Some other possibilities for sharing a pair are as follows. One can avoid transfer of two integers by agreeing upon only one integer, mainly d_0. After sharing the d_0, the parameter a_0 can be computed from d_0 based on any one of pre-specified protocols. One simple idea is to obtain initial value of a_0 by setting $a_0 = \lfloor \frac{d_0}{2} \rfloor$. Thus only the single secret key parameter d_0 needs to be exchanged securely as in every block cipher and stream cipher scenario. The discussion concludes that agreeing upon a seed pair is not difficult.

6 Timing data

We have conducted experiments to confirm our theoretical estimates on the performance of the keystream generator proposed in Section 3. The experiments were carried out on 2.53 GHz Intel processor, Linux OS, using GNU Multi Precision (GMP) library.

As described in Section 3, we have followed two methods of generating a keystream. In the first method, we choose leading terms of the first progression of each groupings. In the second method, we select leading terms of middle progression of each groupings. We present the summary of our experimental results of these two methods. The results indicate that the first method gives better performance.

6.1 Data on the first method

For generation of a keystream, seed pair (a_0, d_0) is chosen randomly. We used the refreshing function that updates only a_0 as $a_0 \leftarrow a_0^2 \pmod{d_0}$. For each sequence of progressions, the size of smallest leading term chosen is always greater than 100 bits. From the security analysis, we know that the size of smallest leading should be greater than 80 bits.

Table 1 gives data on the running times of producing a keystream of length 8 Giga bits from a seed pair (a_0, d_0) where d_0 is chosen randomly with size varying between 256 bits and 16k bits, and a_0 is a randomly chosen integer $< d_0$.

From the data, our observations are as follows. The running time decreases with increasing size of d_0. The yield $Y_1(a_0, d_0)$ is $c * \log_2^2 d_0$ for $c \sim 0.1$. The yield is quadratic in $\log_2 d_0$ as proved in Lemma 2. We notice that as length of d_0 doubles, the running time is reduced by about 1.5 times. The number of pairs used in generating the keystream is inversely proportional to $\log_2^2 d_0$.

Performance results are summarized in Table 2. From the data, we notice that the speed of about 0.63 Gbps is achieved for 1024-bit d_0. Equivalently, the speed is 4 cycles/bit.

Size of d_0 (bits)	Keystream (bits $\sim$ 8 Gb)	Run-time (sec)	#Pairs-used	yield/pair (bits)	time/pair (microsec)
256	8590000094	36.63	1397638	6146	26.21
512	8590000143	23.21	318103	27003	72.97
1024	8590000098	13.66	78517	109403	174.06
2048	8590000549	9.21	19651	437127	468
4096	8590001887	6.41	4924	1744517	1302
8192	8590000355	4.90	1234	6961102	3978
16384	8590006879	4.31	450	19088904	9588

Table 1. Running times of keystream generation

NOTE: Leading terms of the first progression of groupings are used The size of the leading terms is ≥ 100 bits.

Size of d_0 (bits)	Bit rate (Giga bits/second)	cycles/bit
256	0.234	10.7
512	0.370	6.8
1k	0.628	4
2k	0.932	2.7
4k	1.34	1.88
8k	1.75	1.44
16k	1.99	1.26

Table 2. Performance data

These experimental results in Table 2 (column 3) indicate that our pseudorandom generator is comparable to many ciphers discussed in [10]. Speeds of close to 1 Gbps on platforms of the type used by us are acceptable state-of-the-art, suitable for most practical applications. We mention that our method has the further advantage of exploiting the asymptotically fast multiplication algorithms such as FFT. It is known that such fast algorithms work better than Karatsuba and quadratic algorithms for sizes of operands exceeding 4k bits. Our method naturally scales to such sizes.

We have conducted experiments to study the effect of the size of smallest leading term on the running time. Table 3 presents relevant data. In the table, $T(l, c)$ denote the time taken for producing keystream of 8 Giga bits using l-bit d_0 where the size of the smallest leading term considered in the keystream generation is not less than $c * n$. If $c * n < 128$, $T(l, n)$ is marked with "-" meaning that the keystream is not generated.

By comparing the running times within Table 3 and also with that of Table 1, we observe that performance is better when chosen leading terms are of length $\geq 0.5 * \log_2 d_0$. The decrease in the performance with decreasing c is expected as the arithmetic on small operands is paid off with a small yield. At $c = 1/2$, the method achieves the speed of about 0.73 Gbps for 1024-bit d_0. Equivalently, the speed is 3.45 cycles/bit. The maximum speed is 1.18 cycle/bit for 16k-bit d_0.

l	$T(l, \frac{1}{2})$	$T(l, \frac{1}{3})$	$T(l, \frac{1}{4})$	$T(l, \frac{1}{5})$	$T(l, \frac{1}{6})$
256	37.71	-	-	-	-
512	20.19	21.2	21.98	-	-
1k	11.74	12.2	12.6	12.8	13.1
2k	7.53	7.74	7.94	8.09	8.2
4k	5.47	5.55	5.63	5.72	5.76
8k	4.47	4.47	4.52	4.58	4.63
16k	4.03	4.01	4.05	4.05	4.06

Table 3. Effect of size of leading terms on running time in seconds

6.2 Data on the second method

Table 4 presents data on running times when leading terms of middle progressions of groupings are used. Like the first method, at $c = 1/2$, the method achieves better performance. By comparing the numbers in the table with that of Table 3, we clearly observe that the generation of using leading terms of the first progressions gives better performance. This is unexpected as the leading term of middle progression of a grouping $\mathcal{G}$ is the greatest of all other leading terms within $\mathcal{G}$. This property holds for any grouping (From Theorem 1). This unexpected behavior need to be examined further.

l	$T(l, \frac{1}{2})$	$T(l, \frac{1}{3})$	$T(l, \frac{1}{4})$	$T(l, \frac{1}{5})$	$T(l, \frac{1}{6})$
256	49.5	-	-	-	-
512	25.2	27	28.1	-	-
1k	14.7	15.53	16.12	16.6	16.4
2k	9.58	9.91	10.2	10.3	10.3
4k	6.89	7.03	7.18	7.25	7.27
8k	5.61	5.68	5.75	5.76	5.77
16k	5.09	5.09	5.11	5.14	5.14

Table 4. Effect of the size of middle leading terms on running time in seconds

7 Special Cases of Problem 1

Given the leading terms of some arbitrary progressions of $\mathfrak{S}(a_0, d_0)$, the problem is to recover a pair (a_0, d_0). We present two cases of this problem. The cases deal with difficulty of recovering

the main parameters of a grouping when more than term from it are given. The study of these cases led us to finally define an hard instance, i.e., Problem 1.

Note that our study restricts only to first grouping of $\mathfrak{S}(a_0, d_0)$. The arguments are similar for other groupings as well. In the results, $\mathcal{A}(a_0, d_0)$ denotes the first grouping of $\mathfrak{S}(a_0, d_0)$. We recall the property of terms in Eq. 2 for proving the results.

7.1 Case 1 (Easy Instance)

Lemma 4. *For given f_1, f_2 and $f_3 > 0$, there exists at most one co-prime pair (a, n) such that f_1, f_2 and f_3 are the leading terms of some three consecutive progressions of $\mathcal{A}(a, n)$.*

Proof. Suppose that f_1, f_2, f_3 are the leading terms of i^{th}, $(i+1)^{th}$, and $(i+2)^{th}$ progressions of $\mathcal{A}(a, n)$, respectively, for some integer i. Then, by Theorem 1

$$\begin{aligned} f_1 &= a + \triangle(ki - i^2) + iv, \\ f_2 &= a + \triangle\big(k(i+1) - (i+1)^2\big) + (i+1)v, \\ f_3 &= a + \triangle\big(k(i+2) - (i+2)^2\big) + (i+2)v. \end{aligned} \tag{5}$$

By arithmetic on the above equations, we get $\triangle = f_2 - \frac{f_1+f_3}{2}$. Suppose u is the common difference of the progression of $\mathcal{A}(a, n)$ whose leading term is f_3. Then, due to property in Eq. 2, we have

$$f_3 = u + \frac{f_2 u - 1}{u + \triangle}.$$

So u is the positive root of polynomial $x^2 + (f_2 - f_3 + \triangle)x - (\triangle f_3 + 1)$, which shows that u, $\triangle$ are unique. From Lemma 12, the index (or position) of the progression of $\mathcal{A}(a, n)$ with leading term f_3 is equal to $\lfloor \frac{u}{f_3} \rfloor$. So, the common difference of the first progression of $\mathcal{A}(a, n)$ is $n = u + \lfloor \frac{u}{f_3} \rfloor * \triangle$, and the leading term of the first progression is $a \equiv -\triangle^{-1} \pmod{n}$. Since u, $\triangle$ are unique, so are a, n. The quantities a, n can be computed efficiently. □

Suppose given integers f_1, f_2 and f_3 are not consecutive. Suppose they are leading terms of j_1^{th}, j_2^{th}, and j_3^{th} progressions of $\mathcal{A}(a_0, d_0)$. Our aim is to compute a pair (a_0, d_0). In this case, we obtain a system of equations: $Mx = b$ where

$$M = \begin{bmatrix} 1 & kj_1 - j_1^2 & j_1 \\ 1 & kj_2 - j_2^2 & j_2 \\ 1 & kj_3 - j_3^2 & j_3 \end{bmatrix}$$

$x = [a_0, \triangle, v]^T$ and $b = [f_1, f_2, f_3]^T$. We have $1 < j_1 < j_2 < j_3 < k$. By Property in Eq. 2, $a_0 \triangle \equiv -1 \pmod{d_0}$. So, one of a_0 and $\triangle$ is at least $\sqrt{d_0 - 1}$. This shows that $f_1, f_2, f_3 > \sqrt{d_0}$. For a fixed k, an exhaustive search tries out all possible values of j_1, j_2, and j_3. So, we require roughly $O\big(k^3\big)$ matrix inversions of M where $k = \lceil \frac{d_0}{\triangle} \rceil$. The value of k (i.e., the size of grouping) is usually small. The expected value of k is $O\big(\log f\big)$ where f is the minimum of f_1, f_2 and f_3. In the worst case $k = f$ where $\triangle = 1$. For example, $\mathfrak{S}(d_0 - 1, d_0)$ has only one grouping with $\triangle = 1$.

From the above calculation, we conclude that if the size of grouping is small, an exhaustive search will compute quickly the main parameters of the grouping from given three integers. This result shows that not more than two terms from the same grouping should be chosen. However, the study of the case (below) shows that chosen two leading terms should not be consecutive.

7.2 Case 2 (Problem A)

Suppose that we are given only two numbers f_1, f_2. **Problem A** is to compute a pair (a, n) such that f_1, f_2 are the leading terms of some consecutive progressions of $\mathcal{A}(a, n)$. Unlike Case 1, we loose the uniqueness of existence on the pair (a, n). The number of solution pairs is upper bounded by the number of divisors of $f_1 f_2 - 1$. We show that computing solutions to the problem requires factoring $f_1 f_2 - 1$.

Let given f_1 and f_2 be the leading terms of two consecutive arithmetic progressions with corresponding common differences $d + \triangle$ and d. Then, we have $f_2 = d + \frac{f_1 d - 1}{d + \triangle}$. This implies that $d^2 + d\triangle + (f_1 - f_2)d - f_2\triangle - 1 = 0$. Thus, $(d, \triangle)$ is a solution pair to the equation

$$x^2 + xy + (f_1 - f_2)x - f_2 y - 1 = 0. \tag{6}$$

The following result proves that the number of integer solutions to the above equation is finite.

Lemma 5. *For any $r, s, t \in \mathbb{Z}$, with $s^2 - rs + t \neq 0$, the number of integer solutions of $f(x, y) = x^2 + xy + rx + sy + t = 0$ is $\sigma_0(s^2 - rs + t)$. Here, $\sigma_0(x)$ is the number of divisors of number x.*

Proof. The equation can be expressed as

$$y = \frac{-(x^2 + rx + t)}{x + s}.$$

Let $x = q - s$ for some integer q. Then

$$y = q + \frac{s^2 - rs + t}{q} - (2s - r).$$

y is an integer only when q divides $s^2 - rs + t$. Thus, the number of possible integer values for y is equal to the total number of divisors of $s^2 - rs + t$. Hence, the number of solution pairs is equal to $\sigma_0(s^2 - rs + t)$. □

Corollary 2. *For any divisor z of $f_1 f_2 - 1$, $(d, \triangle)$ is a solution to Equation (6) where*

$$d = f_2 + z,$$
$$\triangle = z + \frac{f_1 f_2 - 1}{z} - (f_1 + f_2).$$

Proof. Substitute $r = f_1 - f_2, s = -f_2$ and $t = -1$ in the above lemma. Then we get $s^2 - rs + t = f_1 f_2 - 1$ and $2s - r = f_1 + f_2$. So, $(d, \triangle)$ is a solution to Eq. 6 where $d = z - s = f_2 + z$, $\triangle = z + \frac{f_1 f_2 - 1}{z} - (f_1 + f_2)$. □

Equivalence to integer factoring

Let $B(f_1, f_2)$ be the set of all co-prime integer pairs (a, n) for which f_1, f_2 are the leading terms of i^{th}, $(i+1)^{th}$ progressions of $\mathcal{A}(a, n)$, respectively, for some integer $i \geq 1$.

From corollary above, if we know a divisor z of $f_1 f_2 - 1$ with $z + \frac{f_1 f_2 - 1}{z} - (f_1 + f_2) > 0$, then we can compute an element of $B(f_1, f_2)$. So, computing solutions to problem A reduces to factoring integers. We now prove that factoring reduces to problem A. The reduction is probabilistic.

Let N be the number to be factored. Suppose f_1, f_2 are integers such that $f_1 f_2 = Nr + 1$ for some $r > 0$. Suppose an algorithm $\mathfrak{A}$, on input (f_1, f_2), produces a co-prime pair $(a, n) \in B(f_1, f_2)$. Let the second common difference corresponding to $\mathcal{A}(a, n)$ be $\triangle$. From the results we know that $\triangle = zr_1 + \frac{Nr}{zr_1} - (f_1 + f_2)$ where z is a divisor of N and r_1 is a divisor of r. Then, a divisor of N can be computed as follows.

$$\begin{aligned} zr_1 + \frac{Nr}{zr_1} &= \triangle + (f_1 + f_2) \\ zr_1 - \frac{Nr}{zr_1} &= \sqrt{(\triangle + (f_1 + f_2))^2 - 4Nr}. \end{aligned} \tag{7}$$

By adding the above equations, we get

$$2zr_1 = \triangle + (f_1 + f_2) + \sqrt{(\triangle + (f_1 + f_2))^2 - 4Nr}.$$

So, the gcd of $\triangle + (f_1 + f_2) + \sqrt{(\triangle + (f_1 + f_2))^2 - 4Nr}$ and N gives a divisor of N. The following result proves that, *with some assumptions*, the gcd computation results in a non-trivial divisor of N with probability at least $\frac{1}{2}$.

Theorem 3. *(factoring ⇒ Problem A) Suppose N is an odd composite integer which is not a prime power. Let f_1, f_2 be two integers such that $f_1 f_2 = Nr + 1$ for some integer $r > 0$ co-prime to N, and the set*

$$X = \{a + b - (f_1 + f_2) : ab = Nr\}$$

consists only of positive integers. Then, using the output of the algorithm $\mathfrak{A}$ on input (f_1, f_2), we can find a non-trivial divisor of N with probability at least $\frac{1}{2}$.

Proof. Suppose $\mathfrak{A}$ produces (a, n) on input (f_1, f_2). The second common difference $\triangle$ corresponding to $\mathcal{A}(a, n)$ is in X. The gcd computation, using Eq. 7, results in a non-trivial divisor of N when N splits non-trivially between a and b, i.e., $a \equiv 0 \pmod{z}$, $b \equiv 0 \pmod{N/z}$ for a non-trivial divisor z of N. Let d_1 be the total number of divisors of N. Let d_2 be the total number of divisors of r. The size of X is $\frac{d_1 d_2}{2}$. The probability that output of $\mathfrak{A}$ fails to give

a non-trivial divisor of N is

$$\begin{aligned} P^* &= \Pr[z = 1 \text{ or } z = N] \\ &= \frac{d_2}{(d_1 d_2)/2} \\ &\leq \frac{1}{2}. \quad (\because d_1 \geq 4) \end{aligned}$$

Thus, the probability of successfully computing a non-trivial divisor of N is at least $\frac{1}{2}$. □

By running the algorithm $\mathfrak{A}$ with i different input pairs (f_1, f_2), we can compute a non-trivial divisor of N with probability at least $1 - \frac{1}{2^i}$.

The above reduction requires a few pairs (r, f) such that the set $X = \{(a+b)-(f+\frac{Nr+1}{f}) : ab = Nr\}$ consists only of positive integers. For each such pair (r, f), $(f, \frac{Nr+1}{f})$ is an input pair to Algorithm $\mathfrak{A}$. The problem is related to the largest divisor of x bounded by $\sqrt{x}$. We study this problem in the following section.

7.3 Problem of finding (r, f) pairs

The set X in Theorem 3 is redefined as follows.
Definition: For integers n, r and f,

$$X(n, r, f) = \{(u + \frac{nr}{u}) - (f + \frac{nr+1}{f}) : u | nr\}.$$

The set consists of integers only when f divides $nr+1$. Thus, we note that *whenever* $\mathcal{X}(n, r, f)$ *is used, it is implicit that* f *is a divisor of* $nr + 1$. For our convenience, we use the notation $[\mathcal{X}(n, r, f)] > 0$ to denote the fact that the set $X(n, r, f)$ consists only of positive integers.

For integer $n > 0$, define

$$G(n) = \{(r, f) : 1 \leq r < n \text{ and } [X(n, r, f)] > 0\}.$$

We reiterate the fact that in the proof of Theorem 3, we have to produce an instance of Problem A corresponding to the factoring instance n. Clearly, for each $(r, f) \in G(n)$, $(f, \frac{nr+1}{f})$ is an instance of Problem A corresponding to the factoring instance n. Thus, the aim is to find, for a given n, a few pairs $(r, f) \in G(n)$ without the knowledge of the prime factors of n.

we prove a necessary and sufficient condition for $[\mathcal{X}(n, r, f)] > 0$.

Lemma 6. *When x runs over the divisors of k, the minimum value of $g(x) = x + k/x$ occurs at $x = l(k)$ and $\frac{k}{l(k)}$.*

Proof. The result follows from the fact that g is a decreasing function in $[1, \sqrt{k}]$, is an increasing function in $[\sqrt{k}, k]$. □

Lemma 7. *If one of nr and $nr + 1$ is a perfect square, then, for any divisor f of $nr + 1$, $X(n, r, f)$ consists of negative integers.*

Proof. Clearly, only one of nr and $nr+1$ can be a perfect square. We prove the claim for both cases.

Suppose nr is a perfect square. Then, for any divisor f of $nr+1$, $\mathcal{X}(n,r,f)$ consists of integer $t = 2\sqrt{nr} - (f + \frac{nr+1}{f})$. Since $f < \sqrt{nr}$, we have $f + \frac{nr+1}{f} > 2\sqrt{nr}$. Thus $t < 0$.

Suppose $nr+1 = z^2$ for some integer $z > 0$. Then, for any divisor f of $nr+1$, $\mathcal{X}(n,r,f)$ consists of the element $s = 2z - (f + \frac{nr+1}{f})$. Since $f \leq z$, we have $f + \frac{nr+1}{f} \geq 2z$. Thus, $s \leq 0$. □

Theorem 4. *Suppose n and r are two integers such that neither of nr and $nr+1$ is a perfect square. Let f be a divisor of $nr+1$ with $1 \leq f < \sqrt{nr+1}$. Then, the following two statements are equivalent.*

- $[X(n,r,f)] > 0$
- $l(nr) < f \leq l(nr+1)$.

Proof. By Lemma 6, the least element of $X(n,r,f)$ is $l(nr) + \frac{nr}{l(nr)} - (f + \frac{nr+1}{f})$. Let the least element of $X(n,r,f)$ be m. We show that $m > 0$ if and only if $l(nr) < f \leq l(nr+1)$. Consider the product

$$\begin{aligned} f \times m &= f \times \left[l(nr) + \frac{nr}{l(nr)} - (f + \frac{nr+1}{f})\right] \\ &= (f - l(nr))\left[\frac{nr}{l(nr)} - f\right] - 1. \end{aligned}$$

Since neither of nr, $nr+1$ is a perfect square, we have $f, l(nr) < \sqrt{nr}$. Thus, $\frac{nr}{l(nr)} - f > 1$. In the above equality, if $f > l(nr)$, then $m > 0$. Also, it is clear that $m > 0$ only when $f > l(nr)$. This completes the argument. □

From the above result, computing a pair (r,f) such that $[X(n,r,f)] > 0$ is equivalent to computing integers f, r such that f is a divisor of $nr+1$ with $l(nr) < f \leq l(nr+1)$. Based on this result, an algorithm for computing pairs (r,f) is given below.

For a pair (r,f) produced by the algorithm, $f > r$. If n prime, $l(nr) = r < f$. So, $(r,f) \in G(n)$. For composite n, the following results prove the success rate of the algorithm.

Lemma 8. *For a pair (r,f) generated by the algorithm, we have $\sqrt{knr} < f < \sqrt{\frac{nr}{k}}$ where $k = f/n$ is the ratio computed in Step 11 of the algorithm. Note that $k \geq \frac{1}{2}$.*

Proof. From Step 12 of the algorithm, the integers r, f satisfy the inequalities: $kn < f < n$ and $kf < r < f$. From these inequalities, we obtain $\sqrt{k}f < \sqrt{nr} < \frac{f}{\sqrt{k}}$. Equivalently, $\sqrt{knr} < f < \sqrt{\frac{nr}{k}}$. □

From the above result, if $l(nr) < \sqrt{knr}$, then $[\mathcal{X}(n,r,f)] > 0$. The following theorem shows that, for composite n, $(r,f) \in G(n)$ with probability greater than 0.842 since $k \geq 1/2$.

Theorem 5. *The probability that $l(nr) < \sqrt{knr}$ is at least $1 + \frac{\log k}{2}$.*

Proof. Ref. Appendix B. □

Algorithm 3 : Algorithm for computing (r, f) with $l(nr) < f \leq l(nr+1)$

Input: Integer n and $k \in [1/2, 1)$
Output: A pair $(r, f) \in G(n)$ with probability $\geq 1 + \frac{\log k}{2}$

1: **if** n is prime **then**
2: Choose $f \in (1, n-1)$
3: Compute f_1 such that $ff_1 = 1 \pmod{n}$
4: Compute $r = (ff_1 - 1)/n$
5: Output (r, f)
6: **end if**
7: **if** n is composite **then**
8: Choose $f \in (kn, n-1)$ co-prime to n
9: Compute f_1 such that $ff_1 = 1 \pmod{n}$
10: Compute $r = (ff_1 - 1)/n$
11: Compute the ratio $k = f/n$
12: **if** (r is prime) or ($r > kf$) **then** output (r, f)
13: **else** jump to Step 8
14: **end if**

8 Conclusions

In this work, we proposed a new family of pseudo random number generators based on collections of arithmetic progressions with certain inverse property (Property $\mathcal{P}$). For the proposed pseudorandom generators, the generation of keystream is shown to be efficient. Our experiments confirms this feature. The construction of the generators has several interesting aspects and also leaves some open issues, which we are addressing presently in our ongoing work. The issues are described below.

8.1 Performance Related Issues

For the proposed generators, the keystream generation is efficient . For example, for 1024-bit d_0 the amortized cost of producing the keystream is 4 cycles/bit (Section 6). This result shows that the methods can be used in real-time applications. Also, there is a scope for further improvement in the performance as the methods are suitable for a parallel implementation. Our keystream generation algorithms invoke a little variant of the Euclidean algorithm, so it is amenable to hardware implementation as well for achieving further speedup. We are carrying out a new set of experiments for addressing these performance issues.

The Authenticated Encryption scheme proposed in Section 5 can make use of high speed multiplication technique such as FFT when one wants to work with very large integers. The performance of this method is yet to be tested.

8.2 Security Related Issues

In the paper, we mainly focused our attention to two keystream generating methods (Section 3). The security of the methods is dependent on the computational difficulty of solving the inverse problem: Problem 1. Solving the problem is equivalent to finding common solution to a nearly-defined system of equations. Our prelimanry analysis suggests that the existing

attacks are not directly applicable to solving the system. But, rigorous arguments are required to establish the security of the schemes.

The proof of indistinguishability assumes the fact that both a_0, d_0 are chosen randomly. However, for an efficient generation of keystream, it is suggested that only one of a_0, d_0 is refreshed and the other is fixed. The proof of indistinguishability for this case is open. The arguments indicating (or supporting) the cryptographic strength of the two methods are not applicable to other variants of keystream generation. The security analysis of these variants need to be done separately. Our present study is addressing these open issues.

References

1. E. Bach and J. O. Shallit, *Algorithmic number theory*, Vol. 1, Foundations of Computing Series, MIT Press, Cambridge, MA, 1996, Efficient algorithms.
2. J. Barkley Rosser and Lowell Schoenfeld, "Approximate formulas for some functions of prime numbers", *Illinois Journal of Mathematics*, Vol 6, (1962): pp. 64-94.
3. H. Cohen, *A course in computational algebraic number theory*, Springer, 1996.
4. N. Courtois, A. Klimov, J. Patarin and A. Shamir, "Efficient algorithms for solving overdefined systems of multivariate polynomial equations", *Advances in Cryptology, Eurocrypt*, Vol 1807, LNCS, (2000): pp. 392-407.
5. N. Courtois, L. Goubin, W. Meier and J.-D. Tacier, "Solving underdefined systems of multivariate polynomial equations", *Public Key Cryptography (PKC)*, Vol 2567, LNCS, (2002): pp. 211-227.
6. W. Diffie and M. Hellman, "New directions in cryptography", *IEEE Transactions on Information Theory*, 22(6), 1976: pp. 644-654.
7. J. Eichenauer and J. Lehn, "A non-linear congruential pseudo random number generator", *Statist. Papers*, Vol 27, (1986): pp. 315-326.
8. J. Eichenauer and J. Hermann, "Inversive congruential pseudorandom numbers avoid planes", *Math. Comp.*, Vol 56, (1991): pp. 297-301.
9. J. Eichenauer and J. Hermann, "On generalized inverse congruential pseudorandom numbers", *Math. Comp.*, Vol 63, (1994): pp. 293-299.
10. eBACS: ECRYPT Benchmarking of Cryptographic Systems, http://bench.cr.yp.to/results-stream.html.
11. L.K. Grover, "A fast quantum mechanical algorithm for database search", *Proceedings, 28th Annual ACM Symposium on the Theory of Computing*, (1996): pp. 212-219.
12. P. Hellekalek, "Inversive pseudorandom number generators: concepts, results and links", *Proceeding of the Winter Simulation Conference*, (1995): pp. 255-262.
13. J. Hoffstein, J. Pipher and J. H. Silverman, "NTRU: A Ring-based public key cryptosystem", *Algorithmic Number Theory*, LNCS, (1998): pp. 267-288.
14. D.E. Knuth, *The art of computer programming: Seminumerical algorithms*, Vol 2, second edition, (1998): pp. 354-355.
15. D.E. Knuth, "Evaluation of Porter's constant", *Computers & Mathematics with Applications*, Volume 2, (1976): pp. 137-139.
16. G.H. Norton, "On the asymptotic analysis of the Euclidean algorithm", *Journal of Symbolic Computation*, volume 10, (1990): pp. 53-58.
17. J.W. Porter, "On a theorem of Heilbronn", *Mathematika*, volume 22, (1975): pp. 20-28.
18. R. Rivest, A. Shamir, and L.Adleman, "A method for obtaining digital signatures and public-key cryptosystems", *Communications of ACM*, 21(2), (1978): pp. 120-126.
19. Zhi-Wei Sun, "Problems and results on covering systems", a survey article, http://maths.nju.edu.cn/ zwsun/Cover.pdf.

Appendix A Properties of $\mathfrak{S}(a_0, d_0)$

Proposition 1 *The leading terms a_i, and the common differences d_i of the progressions of $\mathfrak{S}(a_0, d_0)$ satisfy:*

$$\begin{aligned} d_{i+1} &\equiv a_i^{-1} \pmod{d_i}, \\ a_{i+1} &= d_{i+1} + \frac{a_i d_{i+1} - 1}{d_i}. \end{aligned} \tag{8}$$

Proof. By Property $\mathcal{P}$, $(a_i + rd_i)(a_{i+1} + rd_{i+1}) \equiv 1 \pmod{a_i + (r+1)d_i}$, for $i, r \geq 0$. This implies that

$$z_r = \frac{(a_{i+1} + rd_{i+1})d_i + 1}{a_i + (r+1)d_i} \text{ is an integer.}$$

So, for $r \geq 1$, $z_{r+1} - z_r = \frac{Cd_i}{(a_i+(r+1)d_i)(a_i+rd_i)}$ is an integer where

$$C = d_{i+1}a_i + d_{i+1}d_i - a_{i+1}d_i - 1.$$

Since d_i, $a + rd_i$, and $a_i + (r+1)d_i$ are pair-wise co-prime, we have that $a_i + rd_i$ divides C. Therefore, $C = 0$.

A.1 Properties of terms within a grouping

Let $\mathcal{G}$ be a grouping of $\mathfrak{S}(a_0, d_0)$ consisting of progressions,

$$A(a_\alpha, d_\alpha), A(a_{\alpha+1}, d_{\alpha+1}), \ldots, A(a_\beta, d_\beta),$$

for some $0 \leq \alpha < \beta$. Let $\triangle$ be the second common differences corresponding to $\mathcal{G}$. By the definition of grouping, $\triangle = d_i - d_{i+1}$, $\alpha \leq i \leq \beta - 1$.

Lemma 9. $\frac{a_{i+1}\triangle+1}{d_{i+1}} = \frac{a_i\triangle+1}{d_i} + \triangle$, $\alpha \leq i \leq \beta - 1$.

Proof. By Equation 2, we have $a_i d_{i+1} + d_i d_{i+1} - a_{i+1} d_i - 1 = 0$. Thus,

$$\begin{aligned} \frac{a_{i+1}\triangle + 1}{d_{i+1}} - \frac{a_i\triangle + 1}{d_i} &= \triangle\left(\frac{a_{i+1}d_i - a_i d_{i+1} + 1}{d_i d_{i+1}}\right) \\ &= \triangle. \end{aligned}$$

Lemma 10. *For $\alpha \leq i \leq \beta - 1$, $a_{i+1} - a_i = (\alpha + \beta - 1 - 2i)\triangle + d_\beta - z_\alpha$ where*

$$z_\alpha = \frac{a_\alpha\triangle + 1}{d_\alpha}.$$

Proof. By introducing the terms $a_{i+1}d_{i+1} + (-a_{i+1}d_{i+1})$ and $d_{i+1}^2 + (-d_{i+1}^2)$ in the equation $a_i d_{i+1} + d_i d_{i+1} - a_{i+1} d_i - 1 = 0$, we get

$$\begin{aligned} a_{i+1} - a_i &= -\frac{a_{i+1}\triangle + 1}{d_{i+1}} + \triangle + d_{i+1} \\ &= -\frac{a_{i+1}\triangle + 1}{d_{i+1}} + \triangle + (d_\beta + (\beta - i - 1)\triangle) \\ &= -(z_\alpha + (i + 1 - \alpha)\triangle) + d_\beta + (\beta - i)\triangle \\ &= (\alpha + \beta - 1 - 2i)\triangle + d_\beta - z_\alpha. \end{aligned}$$

Lemma 11. $\lfloor \frac{a_i}{d_i} \rfloor = \lfloor \frac{a_\alpha}{d_\alpha} \rfloor + i - \alpha$, $\alpha \leq i \leq \beta$.

Proof. From Lemma 9, we have $\frac{a_i \triangle + 1}{d_i} = \frac{a_\alpha \triangle + 1}{d_\alpha} + (i - \alpha)\triangle$.

Lemma 12. *Suppose* $\mathfrak{S}(a_0, d_0)$ *is the sequence of progressions* $A(a_0, d_0)$, $A(a_1, d_1)$, ... , $A(a_l, d_l)$ *where* $d_0 > d_1 > d_2 > \ldots > d_l$, *and* $d_l = 1$. *Then,*

$$\lfloor \frac{a_j}{d_j} \rfloor = j,\ 0 \leq j \leq l.$$

Proof. Since $a_0 < d_0$, $\lfloor \frac{a_0}{d_0} \rfloor = 0$. The above lemma shows that, within a grouping $\mathcal{G}$, the difference between the values of the ratio $\lfloor \frac{a}{d} \rfloor$ is one for any two consecutive progressions. Any two consecutive groupings share an arithmetic progression. Hence, the result follows.

It can be verified that leading terms and common difference of the progressions of $\mathfrak{S}(11, 25)$ satisfy the above ratio property.

A.2 Proof of Theorem 1

From Lemma 10,

$$\begin{aligned} a_i - a_\alpha = \sum_{j=\alpha}^{i-1} (a_{j+1} - a_j) &= \sum_{j=\alpha}^{i-1} \big(\triangle(\alpha + \beta - 1 - 2j) + d_\beta - z_\alpha\big) \\ &= \triangle(\alpha + \beta - 1)(i - \alpha) - \triangle(\sum_{j=\alpha}^{i} 2j) + (d_\beta - z_\alpha)i \\ &= \triangle(\beta - i)(i - \alpha) + (i - \alpha)(d_\beta - z_\alpha). \end{aligned}$$

Here, z_α is the inverse of $d_\beta \pmod{\triangle}$.

A.3 Proof of Theorem 2

Lemma 13. *Let* $\triangle'$ *be the second common difference of successive grouping of* $\mathcal{G}$. *Then,* $d_\beta \equiv d_\alpha \pmod{\triangle}$, *and* $\triangle' \equiv \triangle \pmod{d_\beta}$.

Proof. It is easy to see the first congruence. We prove the second congruence relation. From Lemma 12, $\frac{a_\beta \triangle + 1}{d_\beta}$ is an integer. This implies that $\triangle \equiv -a_\beta^{-1} \pmod{d_\beta}$. By Eq. 2, $d_{\beta+1} \equiv a_\beta^{-1} \pmod{d_\beta}$. Thus, $\triangle \equiv d_\beta - d_{\beta+1} \pmod{d_\beta}$. This proves the lemma.

A.4 Proof of Lemma 1

For $1 \leq a_0 < d_0$, the maximum value for the number of groupings of $\mathfrak{S}(a_0, d_0)$ is attained when

$$\begin{aligned} d_0 &= F_i, \\ a_0 &\equiv -F_{i-1}^{-1} \pmod{d_0}. \end{aligned} \tag{9}$$

Here, F_i, F_{i-1} are i^{th} and $(i-1)^{th}$ Fibonacci numbers, respectively. The number of iterations of the Euclidean algorithm on input pair (F_i, F_{i-1}) will be $i - 2$. So, the number of iterations of Algorithm 1 on pair (a_0, d_0) in Equation 9 is $\lfloor \frac{i-2}{2} \rfloor$. The i^{th} Fibonacci number is $F_i = \frac{(\phi^i - (-1)^i \phi^{-i})}{\sqrt{5}}$ where $\phi = \frac{\sqrt{5}+1}{2}$. So, $F_i \approx \frac{\phi^i}{\sqrt{5}}$ for large i. This proves the lemma.

Appendix B Proof of Theorem 5

For integer x, a real number $0 < k < 1$, define the set

$$\tau(x,k) = \{r \le x : l(r) < \sqrt{kr}\}.$$

We prove a lower bound on the size of τ, which will give us the success rate of the algorithm. To establish this result, we work with its complement set $\bar{\tau}$.

$$\bar{\tau}(x,k) = \{r \le x : \sqrt{kr} \le l(r) \le \sqrt{r}\}.$$

For integer $a \ge 1$, define $S(x;a,k) = \{ab \le x : a \le b \le a/k\}$.

Lemma 14. *For any* $0 < k \le 1$

$$\bar{\tau}(x,k) = \bigcup_{a=1}^{\lfloor\sqrt{x}\rfloor} S(x;a,k)$$

Proof. For $ab \in S(x;a,k)$, we have $\sqrt{kab} \le a \le \sqrt{ab}$, which satisfies the definition of $\bar{\tau}$. Thus, $ab \in \bar{\tau}$. Conversely, suppose $r \in \bar{\tau}(x,k)$. Let $r = ab$ where $a = l(r)$, $b = \frac{r}{l(r)}$. The integer r satisfies the condition $l(r) \le \frac{r}{l(r)} \le \frac{l(r)}{k}$. Hence, $r = ab \in S(x;l(r),k)$.

The number of integers in $S(x;a,k)$ depends on the value of a.

- For $1 \le a < \sqrt{kx}$, $|S(x;a,k)| = \lfloor \frac{a}{k} \rfloor - a$
- For $\sqrt{kx} < a \le \lfloor\sqrt{x}\rfloor$, $|S(x;a,k)| = \lfloor \frac{x}{a} \rfloor - a$.

From the above lemma, we have

$$\begin{aligned}
|\bar{\tau}(x,k)| &\le \sum_{a=1}^{\lfloor\sqrt{x}\rfloor} |S(x;a,k)| \\
&= \sum_{a=1}^{\lfloor\sqrt{kx}\rfloor} |S(x;a,k)| + \sum_{a=\lceil\sqrt{kx}\rceil}^{\lfloor\sqrt{x}\rfloor} |S(x;a,k)| \\
&= \sum_{a=1}^{\lfloor\sqrt{kx}\rfloor} \lfloor \frac{a}{k} \rfloor - a + \sum_{a=\lceil\sqrt{kx}\rceil}^{\lfloor\sqrt{x}\rfloor} \lfloor \frac{x}{a} \rfloor - a \\
&< \sum_{a=1}^{\sqrt{xk}} (\frac{1}{k} - 1)a + \sum_{a=\sqrt{kx}}^{\sqrt{x}} (\frac{x}{a} - a) \\
&= (\frac{1}{k} - 1) \sum_{a=1}^{\sqrt{xk}} a + \sum_{a=\sqrt{kx}}^{\sqrt{x}} (\frac{x}{a} - a) \\
&= \frac{\sqrt{xk}}{2k} - \frac{x \log k}{2} - \frac{\sqrt{x}}{2}.
\end{aligned}$$

Thus,

$$\begin{aligned}|\tau(x,k)| &= x - |\bar{\tau}(x,k)| \\ &\geq x - \frac{\sqrt{xk}}{2k} + \frac{x\log k}{2} + \frac{\sqrt{x}}{2}. \\ &= \left(1 + \frac{\log k}{2}\right)x - \frac{\sqrt{xk}}{2k} + \frac{\sqrt{x}}{2}.\end{aligned}$$

The probability that $n \in \tau(x,k)$ is

$$\frac{|\tau(x,k)|}{x} \geq 1 + \frac{\log k}{2} - \frac{\sqrt{k}}{2k\sqrt{x}} + \frac{1}{2\sqrt{x}}.$$

This proves Theorem 5.