A Key-Recovery Attack on the GTLN Knapsack Cryptosystem via Continued Fractions

Shiyi Zhou[1,2,*], Yingpu Deng[1,2], and Shi Ying[1,2]

[1]State Key Laboratory of Mathematical Sciences, Academy of Mathematics and Systems Science, Chinese Academy of Sciences, Beijing 100190, People's Republic of China

[2]School of Mathematical Sciences, University of Chinese Academy of Sciences, Beijing 100049, People's Republic of China

{zhoushiyi25,dengyp,yingshi}@amss.ac.cn

October 19, 2026

Abstract

We present a continued-fraction key-recovery attack on the knapsack-based public-key cryptosystem proposed by Gómez-Torrecillas, Lobillo, and Navarro (GTLN). The key observation is that ratios of public-key differences closely approximate rational numbers with small denominators, enabling their exact recovery by continued fractions. Divisibility relations in the recovered fractions yield multiples of two private-key integers through greatest common divisor computations. Once these two integers are identified by a bounded cofactor search, all remaining private-key integers except the modulus and the shift are uniquely determined. Further greatest common divisor computations, cofactor enumeration, and modular inversion then recover the secret modulus and shift. The above process of key recovery uses only public keys. For fixed cofactor bounds, the algorithm runs in polynomial time in the public-key bit length, while a heuristic analysis predicts a high probability of recovering the original key. In experiments with prime-based key generation, our Python implementation recovered the complete private key in all 2,400 trials across the 24 recommended parameter sets, with each attack taking less than 0.05 seconds.

Keywords: Cryptanalysis, Knapsack cryptosystems, Key-recovery attacks, Continued fractions, Convergent

1. Introduction

Gómez-Torrecillas, Lobillo, and Navarro [1] proposed a knapsack McEliece-based public-key cryptosystem, which we refer to as the GTLN cryptosystem. The scheme supports low-weight plaintext vectors over nonbinary alphabets and uses shifted modular inverses of pairwise coprime private integers as public-key entries. Encryption involves integer

*Corresponding author.

linear combinations of the public entries, while decryption relies on rational reconstruction via a truncated extended Euclidean algorithm. The construction is designed for efficient implementation using elementary integer arithmetic. The authors proposed 24 parameter sets with claimed security levels ranging from 64 to 156 bits.

Kalam and Sarkar [2] subsequently presented a key-recovery attack based on LLL lattice reduction using only the public key. Their method constructs lattices of integer relations that eliminate the secret shift. The large modulus required for correct decryption creates a gap in vector lengths, enabling LLL reduction to find short vectors whose coordinates are divisible by the corresponding private-key integers. Combining these divisibility relations across overlapping coordinate blocks through greatest common divisor computations yields multiples of a subset of the private-key integers. Cofactor enumeration and modular arithmetic then recover the complete private key. Their SageMath implementation recovered all tested keys across the 24 recommended parameter sets in less than 3 seconds per instance.

1.1. Our Contributions

We develop a key-recovery attack on the GTLN cryptosystem based on continued fractions and elementary integer arithmetic. Our starting point is that ratios of public-key differences are sufficiently close to rational numbers with small denominators to permit exact recovery by continued fractions. The recovered fractions retain divisibility information about two coordinates of the private integer vector, namely those corresponding to the smallest and largest public-key entries. Specifically, their numerators are divisible by one of these private integers, while the differences between their denominators and numerators are divisible by the other.

Taking greatest common divisors of these quantities yields multiples of the two selected private integers. A bounded cofactor search then produces candidate pairs, each of which uniquely determines all other coordinates of the private integer vector. For each admissible vector, further greatest common divisor computations yield a multiple of the secret modulus. Enumerating the remaining cofactor and computing a modular inverse then gives candidate values of the modulus and shift. The resulting candidate keys are checked against the public-key equations. This procedure uses only the public key and the public parameter bounds.

Compared to the method proposed in [2], our method replaces the lattice reduction with continued-fraction computations. Our attack searches over pairs of candidate values for two selected private-key integers, whereas the default implementation of the Kalam–Sarkar attack searches over six-tuples of candidate values for six selected private-key integers. For a common bound on the corresponding cofactors, these searches involve at most the square and the sixth power of that bound, respectively.

In terms of bit complexity, our algorithm is upper-bounded $O(nL^2 + n^2d^2)$, whereas the upper bound in [2] is $O(nm^6G^3)$. Here n denotes the length of the plaintext vector, L

represents the maximum bit-length of the public key integers, G is the bit-length of the modulus, d is the upper bound of bit length of p_is and m is the block size depending on the lattice. Both L and $d \leq G$. For the recommended parameter sets, G is substantially larger than n, making the dependence on G particularly important in comparing the complexity bounds of the two attacks. So our complexity bound has at most quadratic dependence on G, whereas the bound for the Kalam–Sarkar attack has cubic dependence on G.

We implemented our attack in Python, despite the authors' claimed security levels ranging from 64 to 156 bits in [1], our algorithm completely recovers the private key for all 24 parameter sets within 0.05 seconds. Without relying on the LLL algorithm, our method is easier to implement and the recovery time is faster than that of Abul Kalam and Santanu Sarkar on the same hardware. For the comparison of the execution times of these two methods, see table 2.

2. Key Generation of the GTLN Cryptosystem

We now briefly recall the GTLN cryptosystem. Our attack uses only the public key and the prescribed parameter bounds of the GTLN cryptosystem, without invoking its encryption or decryption algorithms. We therefore describe only the key-generation procedure here and refer interested readers to [1] for details of encryption and decryption.

The cryptosystem is parametrized by the plaintext vector length n, the Hamming weight t of the plaintext vector, and the alphabet size z, where the alphabet is given by $\{0, 1, \ldots, z-1\}$. So the plaintext space is

$$\mathcal{P} = \{e \in \{0, 1, \ldots, z-1\}^n : |supp(e)| = t\}$$

A ciphertext is a pair of integers. Set $b = \lceil \log_2 z \rceil$ with an integer $b \geq 1$, and choose an integer $\beta \geq 1$. The interval $[2^b, 2^{b+\beta})$ must contain n pairwise coprime integers. In the prime-based realization, β is chosen so that the number of primes in this interval is larger than n and not too close to n.

The private key consists of a set of pairwise coprime integers $p_1, \ldots, p_n$, a secret modulus g and a shift number u where g is larger than u and p_i. The public key consists of these $t_1, \ldots, t_n$, each obtained by computing the modular inverse of p_i modulo g and subsequently shifted by u.

Key generation.

The public keys and private keys are generated as follows.

1. Select randomly pairwise coprime integers $p_1, \ldots, p_n \geq z$ satisfying

$$2^b \leq p_i < 2^{b+\beta} = P \quad (1 \leq i \leq n). \tag{1}$$

   The realization in [1, Section 5] chooses all of them to be prime.

2. Choose randomly a positive integer g satisfying

$$\gcd(g, p_i) = 1 \quad (1 \leq i \leq n), \qquad g \geq 4\lambda_{\max}^2, \quad g > 4\omega_{\max}^2. \tag{2}$$

where $p_{n_1}, \ldots, p_{n_t}$ are the t largest integers in the set $\{p_1, \ldots, p_n\}$,

$$\lambda_{\max} = \prod_{j=1}^{t} p_{n_j}, \qquad \omega_{\max} = (z-1)\sum_{j=1}^{t} \frac{\lambda_{\max}}{p_{n_j}}. \tag{3}$$

For the recommended realization, put $d = b + \beta$ and

$$B = \max\{b + \lceil \log_2 t \rceil + (t-1)d,\ td\}, \qquad G = 2B + 3, \tag{4}$$

and select randomly a G-bit integer g coprime to all p_i.

3. Compute the least positive inverse $h_i \equiv p_i^{-1} \pmod{g}$ for every i, and choose $u \in \{1, \ldots, g-1\} \setminus \{h_1, \ldots, h_n\}$. Compute the public keys shifted by u

$$t_i \equiv h_i - u \pmod{g}, \qquad 0 < t_i < g. \tag{5}$$

4. Output the public and private keys

$$\mathrm{pk} = (t_1, \ldots, t_n), \qquad \mathrm{sk} = (p_1, \ldots, p_n, g, u).$$

3. Preliminaries on continued fractions

Let $x \in \mathbb{R}$. Write

$$x = \lfloor x \rfloor + (x - \lfloor x \rfloor).$$

Since $0 \le x - \lfloor x \rfloor < 1$, if $x - \lfloor x \rfloor = 0$, then $x = \lfloor x \rfloor$. If $0 < x - \lfloor x \rfloor < 1$, set $y = x - \lfloor x \rfloor$. Then $x = \lfloor x \rfloor + y$ and $0 < y < 1$. Write

$$y = \frac{1}{\frac{1}{y}}.$$

Set $x_1 = \frac{1}{y} > 1$. Write

$$x_1 = \lfloor x_1 \rfloor + y_1.$$

Since $0 \le y_1 < 1$. If $y_1 = 0$, then

$$x = \lfloor x \rfloor + \frac{1}{\lfloor x_1 \rfloor}.$$

If $0 < y_1 < 1$, let $x_2 = \frac{1}{y_1}$, then write

$$x = \lfloor x \rfloor + \frac{1}{\lfloor x_1 \rfloor + \frac{1}{x_2}}.$$

Here $x_2 > 1$. And so on, we obtain the continued fraction expansion of a real x. If $x \in \mathbb{Q}$, this process will terminate within finitely many steps. If $x \notin \mathbb{Q}$, we obtain an infinite continued fraction expansion of x.

$$x = a_0 + \frac{1}{a_1 + \frac{1}{a_2 + \frac{1}{a_3 + \cdots}}}$$

where $a_i \in \mathbb{Z}, \forall i \geq 0$, and $a_i > 0, \forall i \geq 1$.

$$\frac{b_i}{c_i} = a_0 + \frac{1}{a_1 + \frac{1}{a_2 + \cdots + \frac{1}{a_{i-1} + \frac{1}{a_i}}}}$$

is called the i-th convergent of the continued fraction for x. It is denoted as

$$x = [a_0, x_1] = [a_0, a_1, x_2] = [a_0, a_1, a_2, \dots].$$

For $x \in \mathbb{Q}$, the last convergent of the continued fraction is x itself.

We quote the following Legendre's convergent criterion without proof. For the complete proof of the theorem, the interested reader can refer to [3, Theorem 8.1.5, p. 84].

Theorem 1. *Suppose $x \in \mathbb{R}$, and $a, b \in \mathbb{Z}$, $b > 0$, gcd(a,b)=1. If*

$$\left| x - \frac{a}{b} \right| < \frac{1}{2b^2}, \tag{6}$$

then $\frac{a}{b}$ is a convergent of the continued fraction of x.

We put the proof of the following corollary in the appendix.

Corollary 2. *Let $Q \geq 1$ be an integer and $x \in \mathbb{Q}$. If a reduced fraction $\frac{a}{b}$ with $1 \leq b \leq Q$ satisfies*

$$\left| x - \frac{a}{b} \right| < \frac{1}{2Q^2},$$

then $\frac{a}{b}$ is the unique rational number with denominator at most Q satisfying this inequality. Moreover, $\frac{a}{b}$ can be recovered from x by computing the continued fraction expasion of x and taking the last convergent whose denominator is at most Q, or equivalently the nearest rational with denominator at most Q.

We write $\mathrm{Near}_Q(x)$ for a nearest rational with denominator at most Q, in lowest terms with positive denominator. Corollary 2 makes this rational unique whenever a fraction meets the strict bound.

4. Attack on GTLN Cryptosystem using continued fractions

Notations. We follow the key generation described in [1, Sections 2 and 5], with pairwise coprime integers $p_1, \dots, p_n \in [z, P)$, primality is not required. Let n, t, z, b, β, P be as before.

From the congruence $t_i \equiv h_i - u \pmod{g}$, there exists an integer k_i such that

$$p_i(t_i + u) = 1 + k_i g. \tag{7}$$

Dividing (7) by p_i, we obtain an equation

$$t_i = g\frac{k_i}{p_i} - u + \frac{1}{p_i}. \tag{8}$$

Define

$$q_i = \frac{k_i}{p_i}.$$

Then

$$t_i = gq_i - u + \frac{1}{p_i}, \qquad 0 < q_i < 2. \tag{9}$$

Let l be an index such that $t_l = \min_i t_i$, and let h be an index such that $t_h = \max_i t_i$. Let $I = \{1, \ldots, n\} \setminus \{h, l\}$. For any $i \in I$, define

$$x_i = \frac{t_i - t_l}{t_h - t_l} = \frac{t_i - t_l}{D} \qquad R_i = \frac{q_i - q_l}{q_h - q_l} = \frac{\frac{k_i}{p_i} - \frac{k_l}{p_l}}{\frac{k_h}{p_h} - \frac{k_l}{p_l}}. \tag{10}$$

Write R_i in lowest terms as $R_i = \frac{N_i}{M_i}$ where $M_i, N_i \in \mathbb{Z}$, $M_i > 0$ and put $a_i = t_i - t_l$, then x_i can be written as $x_i = \frac{a_i}{D}$. We will show that x_i is extremely close to a rational number R_i with small denominator, and that R_i can be recovered by using continued fractions. Define $Q = 2P^3$.

Using these notations and some direct computations, we can obtain the following lemma about some bounds on the parameters. And the proof of this lemma is put in the appendix.

Lemma 3. *The public entries are pairwise distinct, and their order agrees with the order of the q_i. For every coordinate, $0 < k_i < 2p_i$ and $\gcd(k_i, p_i) = 1$. Moreover, $D > 0$ and, for every $i \in I$, $0 < x_i, R_i < 1$ and $0 < N_i < M_i < Q$.*

4.1. Auxiliary lemmas for key recovery

In this subsection, we present three lemmas and one theorem needed for key recovery and defer their proofs to the appendix to help readers follow the main line of the argument.

Lemma 4. *If the parameters before satisfy*

$$zD > 4Q^2 + 1, \tag{11}$$

then $|x_i - R_i| < \frac{1}{2Q^2}$ for every $i \in I$.

Remark 5 (Why $zD > 4Q^2 + 1$ is reasonable in practice)**.** *The required inequality has the form $zD > 4Q^2 + 1$. This condition is automatically satisfied at the recommended modulus sizes. Indeed, set $H = k_h p_l - k_l p_h$. Because q_i have the same strict ordering as the t_i, $q_h > q_l$. H is a positive integer, so $H \geq 1$. Subtracting the two endpoints gives*

$$D = \frac{gH}{p_h p_l} + \frac{1}{p_h} - \frac{1}{p_l} > \frac{g}{P^2} - \frac{1}{z}.$$

Consequently $g \geq \frac{32P^8}{z}$ is sufficient, since then

$$zD > \frac{zg}{P^2} - 1 \geq 32P^6 - 1 > 16P^6 + 1 = 4Q^2 + 1.$$

For a G-bit modulus, $g \geq 2^{G-1}$. Using $P = 2^{b+\beta}$ and $z \geq 2^{b-1}$, a sufficient bit-length inequality is

$$G - 1 \geq 5 + 8(b + \beta) - b + 1 = 6 + 7b + 8\beta. \tag{12}$$

With G given by (4)*,*

$$G \geq 2t(b + \beta) + 3, \tag{13}$$

we only need $t > 4$ (In practice, b is typically greater than 3, we only need $t > 3$). All 24 parameter sets in [1, Table 2], with G given by (4)*, satisfy this inequality. The smallest value of $G - 1 - (6 + 7b + 8\beta)$ is 1476, attained at $(n, t, b, \beta) = (150, 20, 40, 5)$.*

If an attacker gets R_i, the endpoint integers p_l and p_h can be recovered from the numerator and denominator of R_i. Therefore the following theorem is the key step of the recovery process.

Theorem 6 (Recovery of R_i)**.** *Assume the condition* (11) *holds. Then for each $i \notin \{l, h\}$, the rational number R_i is the unique rational number with denominator at most Q that satisfies*

$$|x_i - R_i| < \frac{1}{2Q^2}.$$

Consequently, R_i can be recovered from x_i by computing the continued fraction expansion of x_i and taking the last convergent with denominator at most Q.

Proof. By discussion before, the denominator M_i of R_i satisfies $M_i \leq Q$. By Lemma 4, we have

$$|x_i - R_i| < \frac{1}{2Q^2}.$$

Thus the hypotheses of Corollary 2 are satisfied with $\frac{a}{b} = R_i$ and the given Q. Therefore, by Corollary 2, R_i is the unique rational number with denominator at most Q satisfying the inequality, and it can be recovered by continued fractions. □

Lemma 7 (Recovery of multiples of p_h, p_l)**.** *Write R_i in lowest terms as $R_i = \frac{N_i}{M_i}$, then $p_h \mid N_i$ and $p_l \mid (M_i - N_i)$ for each $i \in I$.*

Therefore, we can compute

$$\gamma_h = \gcd_{i \in I} N_i, \qquad \gamma_l = \gcd_{i \in I}(M_i - N_i).$$

In the absence of extra common factors, $\gamma_h = p_h$ and $\gamma_l = p_l$. Once p_l and p_h are known, we can recover every other p_i as follows:
Subtract the l-th identity in (8) from the i-th and h-th identities to obtain

$$t_i - t_l = g(q_i - q_l) + \frac{1}{p_i} - \frac{1}{p_l}, \qquad D = g(q_h - q_l) + \frac{1}{p_h} - \frac{1}{p_l}.$$

Multiplying the second equality by $R_i = \frac{q_i - q_l}{q_h - q_l}$ and subtracting it from the first gives

$$\begin{aligned}(t_i - t_l) - R_i D &= g(q_i - q_l) - R_i g(q_h - q_l) + \frac{1}{p_i} - \frac{1}{p_l} - R_i\left(\frac{1}{p_h} - \frac{1}{p_l}\right) \\ &= \frac{1}{p_i} - \frac{1 - R_i}{p_l} - \frac{R_i}{p_h}.\end{aligned}$$

Rearrangement proves

$$\frac{1}{p_i} = (t_i - t_l) - R_i D + \frac{1 - R_i}{p_l} + \frac{R_i}{p_h}. \tag{14}$$

Every member of the right side of the equality are known now, so is p_i.

Lemma 8 (Recovery of a multiple of g)**.** *For a tuple of pairwise coprime integers $p_1, \ldots, p_n > 1$, define*

$$A_{i,j} = p_i p_j (t_i - t_j) + p_i - p_j. \tag{15}$$

Then

$$g_0 = \gcd_{2 \le j \le n} |A_{1,j}| = \gcd_{1 \le i < j \le n} |A_{i,j}| > 0. \tag{16}$$

On the true vector $(p_1, \ldots, p_n)$, $A_{i,j} = g(p_j k_i - p_i k_j)$ and $g_0 = cg$ for a positive integer c.

4.2. The complete recovery process

We now give the complete computational procedure using these lemmas.

Step 1: Recover the exact ratios. Find the indices l, h and compute D, I, and Q from section 4. Check (11); if it fails, this procedure does not claim certified ratio recovery. Otherwise, for every $i \in I$, compute $a_i = t_i - t_l$ and the reduced fraction

$$\frac{N_i}{M_i} = \mathrm{Near}_Q\left(\frac{a_i}{D}\right).$$

Lemma 4, together with Corollary 2, show that this fraction is exactly R_i. The computation uses exact continued-fraction arithmetic. In our Python implementation, `Fraction(a_i,D).limit_denominator(Q)` finds the nearest fraction with denominator at most Q [7]. Merely choosing an arbitrary convergent below that denominator bound would not suffice. Check $0 < N_i < M_i$ and verify the strict residual condition using integer arithmetic:

$$2Q^2 |a_i M_i - N_i D| < D M_i. \tag{17}$$

This is precisely $\left|\frac{a_i}{D} - \frac{N_i}{M_i}\right| < \frac{1}{2Q^2}$ after multiplication by the positive denominator DM_i.

Step 2: Give the endpoint candidate sets. By Lemma 7, compute

$$\gamma_l = \gcd_{i \in I}(M_i - N_i) = f_l p_l, \qquad \gamma_h = \gcd_{i \in I} N_i = f_h p_h, \tag{18}$$

with unknown positive integer cofactors f_l, f_h. Fix two positive integers C_p and C_g, which bound the cofactors of endpoints and modulus, respectively. For $j \in \{l, h\}$, set

$$L_j = \max\left\{1, \left\lceil \frac{\gamma_j}{P-1} \right\rceil\right\}, \qquad U_j = \min\left\{C_p, \left\lfloor \frac{\gamma_j}{z} \right\rfloor\right\}. \tag{19}$$

The range restriction follows from $z \le p_j \le P-1$. For every integer $L_j \le f \le U_j$, first test $f \mid \gamma_j$, then retain the integer γ_j/f in $[z, P)$, obtaining a set $\mathcal{P}_j$. The true endpoint p_j is retained if $f_j \le C_p$.

Step 3: Calculate the candidate integer vectors. For every $(\widehat{p_l}, \widehat{p_h}) \in \mathcal{P}_l \times \mathcal{P}_h$, first require $\gcd(\widehat{p_l}, \widehat{p_h}) = 1$. For each $i \in I$, calculate

$$U_i^* = M_i \widehat{p_l} \widehat{p_h}, \tag{20}$$

$$V_i^* = (a_i M_i - N_i D)\widehat{p_l}\widehat{p_h} + (M_i - N_i)\widehat{p_h} + N_i \widehat{p_l}. \tag{21}$$

Discard the entire endpoint pair if $V_i^* \le 0$ or $V_i^* \nmid U_i^*$. Otherwise set $\widehat{p_i} = U_i^*/V_i^*$ and test the range $[z, P)$. Finally require all n coordinates to be pairwise coprime. This can be checked by initializing $T = 1$ and, for $j = 1, \ldots, n$, requiring $\gcd(\widehat{p_j}, T) = 1$ before updating $T \leftarrow T\widehat{p_j}$. Indeed, an integer is coprime to a product exactly when it is coprime to every factor. Since every coordinate is at least 2, this test also ensures distinctness. Discard the pair if any test fails. Thus only the two endpoints are enumerated, all other candidate coordinates are uniquely determined as $\widehat{p_l}$ and $\widehat{p_h}$ from (14).

Step 4: Recover the modulus and shift. For each retained integer vector, compute

$$g_0 = \gcd_{2 \le j \le n} \left| \widehat{p_1}\widehat{p_j}(t_1 - t_j) + \widehat{p_1} - \widehat{p_j} \right|. \tag{22}$$

Lemma 8 ensures $g_0 > 0$ for every retained vector and $g_0 = cg$ on the true vector $(p_1, \ldots, p_n)$. Since $g > t_h$, it suffices to examine

$$1 \le c \le U_g, \qquad U_g = \min\left\{C_g, \left\lfloor \frac{g_0}{t_h + 1} \right\rfloor\right\}. \tag{23}$$

For each divisor c of g_0 in this interval, set $g' = g_0/c$. If $\gcd(\widehat{p_l}, g') = 1$, compute

$$u' = \widehat{p_l}^{-1} - t_l \pmod{g'}. \tag{24}$$

Retain the tuple if $u' \ne 0$ and

$$\widehat{p_i}(t_i + u') \equiv 1 \pmod{g'} \qquad (1 \le i \le n). \tag{25}$$

Collect every retained tuple $(\widehat{p_1}, \ldots, \widehat{p_n}, g', u')$ in the output set and continue searching all admissible cofactors and endpoint pairs.

The entire computational procedure is formalized in the following Algorithm 1.

Algorithm 1: Private key recovery by continued fractions

Input: pk $= (t_1, \dots, t_n)$ with $n \geq 3$; $b, \beta \geq 1$; positive integer C_p, C_g.
Output: The set $\mathcal{K}$ of public-key-compatible candidate tuples; the empty set denotes failure.

$\mathcal{K} \leftarrow \varnothing$; $z \leftarrow 2^b$; $P \leftarrow 2^{b+\beta}$; $Q \leftarrow 2P^3$;
if *some* $t_i \leq 0$*, or the public entries are not distinct* **then return** $\varnothing$;
$l \leftarrow \arg\min_i t_i$; $h \leftarrow \arg\max_i t_i$; $D \leftarrow t_h - t_l$; $I \leftarrow \{1, \dots, n\} \setminus \{l, h\}$;
if $zD \leq 4Q^2 + 1$ **then return** $\varnothing$;
foreach $i \in I$ **do**
    $a_i \leftarrow t_i - t_l$; $N_i/M_i \leftarrow \mathrm{Near}_Q(a_i/D)$;
    if $N_i \leq 0$ *or* $N_i \geq M_i$ *or* $2Q^2|a_iM_i - N_iD| \geq DM_i$ **then return** $\varnothing$;
$\gamma_l \leftarrow \gcd_{i \in I}(M_i - N_i)$; $\gamma_h \leftarrow \gcd_{i \in I} N_i$;
foreach $j \in \{l, h\}$ **do**
    $L_j \leftarrow \max\{1, \lceil \gamma_j/(P-1) \rceil\}$; $U_j \leftarrow \min\{C_p, \lfloor \gamma_j/z \rfloor\}$;
    $\mathcal{P}_j \leftarrow \left\{ \gamma_j/f : f \in \mathbb{Z},\ L_j \leq f \leq U_j,\ f \mid \gamma_j,\ z \leq \gamma_j/f < P \right\}$;
foreach $(\widehat{p}_l, \widehat{p}_h) \in \mathcal{P}_l \times \mathcal{P}_h$ **do**
    if $\gcd(\widehat{p}_l, \widehat{p}_h) \neq 1$ **then continue with the next endpoint pair**;
    foreach $i \in I$ **do**
        $U_i^* \leftarrow M_i\widehat{p}_l\widehat{p}_h$;
        $V_i^* \leftarrow (a_iM_i - N_iD)\widehat{p}_l\widehat{p}_h + (M_i - N_i)\widehat{p}_h + N_i\widehat{p}_l$;
        if $V_i^* \leq 0$ *or* $V_i^* \nmid U_i^*$ **then continue with the next endpoint pair**;
        $\widehat{p}_i \leftarrow U_i^*/V_i^*$;
        if $\widehat{p}_i \notin [z, P)$ **then continue with the next endpoint pair**;
    if $\widehat{p}_1, \dots, \widehat{p}_n$ *are not pairwise coprime* **then continue with the next endpoint pair**;
    $g_0 \leftarrow \gcd_{2 \leq j \leq n} |\widehat{p}_1\widehat{p}_j(t_1 - t_j) + \widehat{p}_1 - \widehat{p}_j|$;
    $U_g \leftarrow \min\{C_g, \lfloor g_0/(t_h + 1) \rfloor\}$;
    for $c \leftarrow 1$ **to** U_g **do**
        if $c \nmid g_0$ **then continue with the next cofactor** c;
        $g' \leftarrow g_0/c$;
        if $\gcd(\widehat{p}_l, g') \neq 1$ **then continue with the next cofactor** c;
        $u' \leftarrow ((\widehat{p}_l^{-1} \bmod g') - t_l) \bmod g'$;
        if $u' = 0$ **then continue with the next cofactor** c;
        if $\widehat{p}_i(t_i + u') \equiv 1 \pmod{g'}$ *for every* $1 \leq i \leq n$ **then**
            $\mathcal{K} \leftarrow \mathcal{K} \cup \{(\widehat{p}_1, \dots, \widehat{p}_n, g', u')\}$;
return $\mathcal{K}$;

If the input is a valid public key with pairwise coprime private integers satisfying (11), and its true cofactors satisfy

$$f_l \leq C_p, \qquad f_h \leq C_p, \qquad c \leq C_g, \tag{26}$$

then the output contains the original private key. In fact, Lemma 7 places the true endpoints in their candidate sets; their gcd is 1, and (14) reconstructs all remaining true coordinates, which pass the pairwise-coprimality test. Lemma 8 then gives $g_0 = cg$, so the true c is examined and the original u is recovered by (24). Every returned tuple has pairwise coprime coordinates in $[z, P)$, $g' > t_h$, $0 < u' < g'$, and satisfies all public equations. This does not assert uniqueness or success for keys whose cofactors exceed the fixed C_p, C_g.

4.3. Bit complexity

Let $d = b + \beta$ and $L = \lceil \log_2(t_h + 1) \rceil$ be the the maximum bit-length of the public key integers. We use classical integer arithmetic: multiplication and division on s-bit integers, and the Euclidean and extended Euclidean algorithms, cost $O(s^2)$ bit operations [5, Chapter 4].

Theorem 9. *For positive, distinct public entries satisfying* (11)*, Algorithm 1 can be implemented with bit complexity*

$$O\Big(nL^2 + C_p d^2 + C_p^2(n(C_g + 1)L^2 + n^2 d^2)\Big). \tag{27}$$

The algorithm has polynomial bit complexity when C_p, C_g are fixed constants or polynomially bounded in the public-input length. For fixed C_p, C_g, (27) *becomes $O(nL^2 + n^2 d^2)$.*

Proof. Since $Q = 2^{3d+1}$ and $D < 2^L$, (11) implies $2^{b+L} > 2^{6d+4}$, hence $L > 6d + 4 - b \geq 5d + 4$ and $d = O(L)$. The recovered numerators, denominators, endpoint gcds and scanned endpoint cofactors have $O(d)$ bits. Apart from the prefix products used for the coprimality test below, all reconstruction, anchor-gcd and modulus-search operands have $O(L)$ bits. The natural bounds in (19) and (23) clip larger cofactors before enumeration.

Ratio recovery. Finding the endpoints is linear in n. The public entries can be checked for distinctness by sorting, using $O(n \log n)$ comparisons on L-bit integers. Since $n < 2^L$, this costs $O(nL^2)$. Each exact continued-fraction computation, nearest-rational selection and residual check costs $O(L^2)$; the accumulation of the two endpoint gcds has the same total bound. Thus this stage costs $O(nL^2)$.

Endpoint enumeration. Each endpoint interval has at most C_p integers. Divisibility, division and the interval test cost $O(d^2)$ per value. Both candidate sets together cost $O(C_p d^2)$ and have at most C_p elements each.

Reconstruction and coprimality. There are at most C_p^2 endpoint pairs. Their initial gcd test costs $O(d^2)$ per pair. Reconstructing and range-checking all coordinates costs $O(nL^2)$. To check pairwise coprimality, let $T_{j-1} = \prod_{k<j} \widehat{p}_k$; it has at most $(j-1)d$ bits. Reduce it modulo $\widehat{p}_j$, compute the gcd of the resulting two d-bit integers, and update $T_j = T_{j-1}\widehat{p}_j$. Classical division and multiplication with these operand lengths cost $O(jd^2)$, while the gcd after reduction costs $O(d^2)$. Summing over j gives $O(n^2 d^2)$ per vector. In particular, the growing prefix product must not be counted as a single d-bit operand. Computing the $n-1$ anchor inputs and their gcd costs another $O(nL^2)$ per surviving vector.

Modulus enumeration. For each surviving vector there are at most C_g cofactor values. Divisibility, the modular inverse and shift cost $O(L^2)$ per value. Checking all n public equations costs $O(nL^2)$, and writing a returned tuple costs $O(nL)$. Different endpoint pairs yield different integer vectors, while different c values yield different moduli. Thus duplicate removal requires no additional search, and this stage costs $O(nC_p^2 C_g L^2)$ in total. Adding all stages proves (27). Since $d = O(L)$, the fixed bound is also $O((nL)^2)$ and is polynomial in the public-input length. □

The bound depends on the numerical values of C_p,C_g, not merely their bit lengths. The running-time bound is deterministic; the probability that the original key lies within the bounds is analyzed separately by the heuristic model below.

5. Heuristic analysis of cofactors

We next give some results about the probability distribution of greatest common divisor of several random integers. This motivates small C_p and C_g under the heuristic assumption.

5.1. The random condition

Let $X_1, \ldots, X_m$ be independent and uniform on $\{1, \ldots, H\}$, where $m \geq 3$, and let $D_H = \gcd(X_1, \ldots, X_m)$. Let $\zeta(m)$ be the Riemann's zeta function. Nymann's theorem [4, Theorem 1] gives

$$\Pr(D_H = 1) = \frac{1}{\zeta(m)} + O\left(\frac{1}{H}\right), \qquad \lim_{H\to\infty} \Pr(D_H = 1) = \frac{1}{\zeta(m)}.$$

Proposition 10. *For every fixed positive integer d,*

$$\lim_{H\to\infty} \Pr(D_H = d) = \frac{1}{\zeta(m)d^m}. \tag{28}$$

For each integer $C \geq 1$,

$$\Pr(D_H > C) \leq \sum_{d=C+1}^{H} \left(\frac{\lfloor H/d \rfloor}{H}\right)^m \leq \frac{C^{1-m}}{m-1}, \tag{29}$$

$$\lim_{H\to\infty} \Pr(D_H > C) = \frac{1}{\zeta(m)} \sum_{d>C} d^{-m} \leq \frac{C^{1-m}}{(m-1)\zeta(m)}. \tag{30}$$

Proof. The number of tuples with gcd d equals the number of coprime tuples in $\{1, \ldots, \lfloor H/d \rfloor\}^m$. Dividing by H^m and using Nymann's limit gives (28). For the finite H, $D_H > C$ implies that some integer $d > C$ divides every X_i. The union bound and independence give the first inequality in (29); use $\lfloor H/d \rfloor / H \leq 1/d$ and integrate x^{-m} over $[C, \infty)$ to get the second. For the inequality in (30), apply (28) and the same integral estimate. □

5.2. Application to the three cofactors

For the true private key and recovered ratios, define

$$X_i = \frac{N_i}{p_h}, \quad Y_i = \frac{M_i - N_i}{p_l} \quad (i \in I), \qquad Z_j = \frac{|A_{1,j}|}{g} \quad (2 \leq j \leq n).$$

These are positive integers and

$$f_h = \gcd_{i \in I} X_i, \qquad f_l = \gcd_{i \in I} Y_i, \qquad c = \gcd_{2 \le j \le n} Z_j.$$

There are $n-2$ entries in each endpoint list and $n-1$ in the modulus list. In the following discussion, each list is treated as an independent uniform sample in the large-range limit. This is an heuristic assumption. To satisfy $m \ge 3$, we need $n \ge 5$ in the following heuristic estimate.

Applying (30) with $m = n-2$ and $n-1$ gives

$$\Pr_{\text{model}}(f_j > C_p) \le \frac{C_p^{3-n}}{(n-3)\zeta(n-2)} \quad (j = l, h), \tag{31}$$

$$\Pr_{\text{model}}(c > C_g) \le \frac{C_g^{2-n}}{(n-2)\zeta(n-1)}. \tag{32}$$

Let $\varepsilon(C_p, C_g; n)$ be the probability of omission of the original key through fixed C_p and C_g. (31) and (32) predict the following upper bound for $\varepsilon(C_p, C_g; n)$:

$$\varepsilon(C_p, C_g; n) \le \frac{2C_p^{3-n}}{(n-3)\zeta(n-2)} + \frac{C_g^{2-n}}{(n-2)\zeta(n-1)}. \tag{33}$$

For actual GTLN keys this is a heuristic estimate, not a proved failure probability. The same algebraic identities hold for composite pairwise coprime coordinates, but the model must still be assumed for their induced distributions; the experiments below use prime-based key generation only.

5.3. Choice of experimental C_p and C_g

We select

$$C_p = C_g = 2. \tag{34}$$

All 24 recommended parameter sets in [1, Section 5] have $n \ge 85$. For any of them, using $\zeta(n-2), \zeta(n-1) > 1$, (33) gives the bound

$$\varepsilon(2, 2; n) \le \frac{2 \cdot 2^{-82}}{82} + \frac{2^{-83}}{83} < 6.290 \cdot 10^{-27} < 2^{-87}. \tag{35}$$

This is based on randomness for X_i, Y_i and Z_j, which is a heuristic hypothesis of our algorithm. The right-hand side decreases as n increases. Thus these upper bound of cofactors already put the model bound below 2^{-80}, while retaining cofactor 2 or 1. They produce at most four endpoint pairs, and each candidate endpoint pair has at most two modulus-cofactor trials for each public tuple $(t_1, \ldots, t_n)$.

6. Our experiment

6.1. Protocol and implementation

We implemented the prime-based key generation from [1, Section 5] and Algorithm 1 in Python, using only its standard library. For each of the 24 parameter sets in [1, Table 2], we generated 100 keys and applied the attack, for 2,400 trials in total. All generated p_i were prime, whereas the attack accepted arbitrary pairwise coprime integer candidates in the prescribed interval. Every trial used $C_p = C_g = 2$, as justified in Section 5.3.

The attacker receives only $(t_1, \ldots, t_n), b, \beta, C_p, C_g$. The test uses t only for key generation and for checking the generated key's size conditions.

After the attack timer stops, the evaluator compares every returned tuple with the original private key . It also measures $f_l = \gamma_l / p_l$, $f_h = \gamma_h / p_h$, and $c = g_0 / g$. This use of the original key is confined to evaluation and supplies no information to the attacker. Success in Table 1 means that the complete output set contains the original private key, not merely that some compatible candidate was found.

The measurements were made on an Intel Core i7-14700 processor, with Windows 11 and 64-bit CPython 3.12.14, on October 12, 2026. The program runs sequentially in one process. Each elapsed time, measured by `time.perf_counter`, includes public-input checks, continued fractions, all cofactor scans, coprimality checks, and final verification of every candidate. It does not include key generation, evaluation against the saved private key, or file input or output. Only the key generator uses a primality test: the first twelve prime bases give an exact decision below 318665857834031151167461 [6, Theorem 1.1], which contains the entire tested range $p_i < 2^{71}$. It samples an integer uniformly and takes the strictly next prime, rejecting out-of-range values and duplicates. The attack itself has no primality-backend size restriction.

The Python implementation of the continued-fraction attack, the prime-based key generator, and the experimental data underlying Table 1 are publicly available at `https://github.com/Shiyicollab/gtln-cf-attack`. The repository includes scripts for reproducing the 2,400-trial experiment, per-trial measurements, per-parameter summaries, sample public keys, execution metadata, and usage instructions. The implementation requires only the Python standard library. The command

```
python benchmark.py --trials 100 --Cp 2 --Cg 2
  --seed 20261012 --out results_rerun
```

regenerates the experiment. This is one command split here only for typesetting. Exact mathematical outputs can be checked against the supplied hashes and records; measured times will depend on the execution environment.

6.2. Experimental results

All 2,400 trials returned exactly the original private key. There were no empty outputs or incorrect single outputs. Table 1 shows each recommended parameter tuple, its modulus bit length, and the measured attack times.

Table 1: Time to recover the private key from the public key with $C_p = C_g = 2$. Times are in milliseconds.

No.	n	t	b	β	G	Success	Mean	Median	Max
1	85	22	60	1	2695	100/100	5.560	5.475	8.296
2	90	23	60	1	2817	100/100	6.038	5.982	6.978
3	90	26	50	15	3383	100/100	7.134	7.089	8.410
4	95	22	70	1	3135	100/100	7.907	7.851	9.047
5	95	25	55	1	2811	100/100	5.877	5.831	6.863
6	100	20	50	1	2051	100/100	4.672	4.630	5.398
7	100	25	55	1	2811	100/100	6.203	6.142	7.070
8	110	25	60	1	3061	100/100	7.911	7.870	8.866
9	110	30	40	20	3603	100/100	8.494	8.468	9.417
10	125	20	60	1	2451	100/100	7.721	7.670	9.625
11	125	25	40	1	2061	100/100	4.921	4.873	7.298
12	150	20	40	5	1803	100/100	5.677	5.633	6.822
13	150	30	30	20	3003	100/100	8.795	8.737	10.207
14	150	30	50	1	3071	100/100	9.194	9.115	10.502
15	175	30	45	1	2771	100/100	9.404	9.273	12.347
16	175	40	30	20	4003	100/100	14.073	14.055	15.417
17	200	40	70	1	5693	100/100	29.094	29.014	31.532
18	200	50	45	10	5503	100/100	22.011	21.894	24.009
19	225	45	45	5	4505	100/100	20.021	19.942	21.551
20	225	55	40	10	5503	100/100	22.841	22.751	24.793
21	250	60	50	1	6133	100/100	28.594	28.530	30.044
22	250	55	30	20	5503	100/100	25.389	25.349	26.701
23	300	60	50	1	6133	100/100	34.459	34.393	36.000
24	400	50	50	1	5113	100/100	40.814	40.729	42.590

We again did the comparison experiment. The following table compares the mean key-recovery times on the 24 recommended parameter sets.

Table 2: Comparison of mean key-recovery times on the 24 recommended parameter sets.

(n, t, b, β)	CF (ms)	KS (ms)	KS/CF
(85, 22, 60, 1)	5.640	372.359	66.02
(90, 23, 60, 1)	6.113	488.854	79.97
(90, 26, 50, 15)	7.327	437.988	59.78
(95, 22, 70, 1)	7.941	494.817	62.31
(95, 25, 55, 1)	5.883	475.275	80.79
(100, 20, 50, 1)	4.794	429.756	89.64
(100, 25, 55, 1)	6.323	488.869	77.32
(110, 25, 60, 1)	8.061	607.660	75.39
(110, 30, 40, 20)	8.652	586.760	67.82
(125, 20, 60, 1)	7.993	527.772	66.03
(125, 25, 40, 1)	5.077	427.646	84.24
(150, 20, 40, 5)	5.938	561.699	94.59
(150, 30, 30, 20)	9.157	748.327	81.72
(150, 30, 50, 1)	9.495	797.197	83.96
(175, 30, 45, 1)	9.442	891.672	94.44
(175, 40, 30, 20)	14.441	938.662	65.00
(200, 40, 70, 1)	29.480	1635.992	55.50
(200, 50, 45, 10)	22.428	1432.351	63.86
(225, 45, 45, 5)	20.200	1771.395	87.69
(225, 55, 40, 10)	23.217	1802.342	77.63
(250, 60, 50, 1)	29.349	1833.374	62.47
(250, 55, 30, 20)	26.091	2420.148	92.76
(300, 60, 50, 1)	35.599	2668.089	74.95
(400, 50, 50, 1)	41.229	2157.385	52.33

In this table, CF denotes our continued-fraction attack with $C_p = C_g = 2$ and pairwise-coprimality checks. KS denotes a Python/FLINT port of Kalam–Sarkar's public code, with $m = 8$, $\kappa = 6$, $C_p = 4$, $C_g = 8192$, and LLL parameter $\delta = 0.99$. KS timings are for this port, not for the authors' SageMath implementation. For each parameter set, both attacks received the same 20 public keys, generated with prime private coordinates. Tests used an Intel Core i7-14700, CPython 3.12.14 and python-flint 0.9.0 on Windows 11, with one fixed logical CPU and alternating execution order. Timings cover the complete attack, excluding key generation, external correctness checks, I/O and warm-up. Both methods recovered all 480 original private keys; CF returned exactly one candidate in every trial. CF enumerates all candidates within its bound for cofactors; KS selects the smallest valid modulus from its first successful core tuple. KS/CF is the ratio of the unrounded mean times.

6.3. The actual size of cofactors in the experimental results

In every trial we observed

$$f_l = f_h = c = 1.$$

Thus the 4,800 measured endpoint cofactors and 2,400 modulus cofactors were all equal to 1. So precisely one endpoint pair was retained in each trial. Indeed, $\gamma_j = p_j$ was an odd prime on these samples, and $C_p = 2$ leaves only $f = 1$ as a divisor of γ_j. Each candidate vector passed the pairwise-coprimality test. For the recovered vector $(p_1, \ldots, p_n)$, every

trial also had

$$\left\lfloor \frac{g_0}{t_h + 1} \right\rfloor = 1.$$

The natural modulus bound in (23) therefore reduced the search to $c = 1$, although the fixed value $C_g = 2$. There was one modulus trial and one complete public-key validation per key. These observations explain the disappearance of branch growth in this experiment. The role of the fixed values of C_p and C_g in Theorem 9 is to control the worst-case number of trials. The observation at cofactor 1 is mainly consistent with the prediction of random condition.

7. Effect of parameter adjustments

As shown above, the 24 parameter sets originally proposed in [1] fail to realize the security claimed by the authors. A natural question is whether changing the parameter sizes can avoid this attack. Exact recovery of the ratios is certified by $zD > 4Q^2 + 1$. Under the modulus-generation rule (4), Remark 5 proves that $t > 4$ is sufficient for this condition. This conclusion applies to pairwise coprime integers; it does not depend on their primality. Someone might ask if this cryptosystem is secure if set $t \leq 4$. Recall t is the Hamming weight of plaintext. Setting $t \leq 4$ exposes the cryptosystem to combinatorial attacks discussed in [1, Section 4.2.2], under which an exhaustive search over the support of plaintext becomes possible. Changing the remaining parameters likewise fails to invalidate inequality (11). Hence the weakness of this cryptosystem lies in the design rather than the choices of parameters.

8. Conclusion

We have shown that, for fixed cofactor bounds, using public keys only, our algorithm runs in polynomial time with respect to the bit-length of the public key and its output contains the original private key whenever the separation condition (11) and the prescribed cofactor bounds hold. This cryptosystem does not possess the security claimed by the authors. In our experiments, we recovered the complete private keys for all 24 recommended parameter sets in less than 0.05 seconds. This attack exposes that the public keys t_i cannot fully conceal the information of the private keys p_i. More specifically, by simply computing the convergents of the ratios of differences of t_i, we can obtain certain multiples of p_l and p_h. The underlying flaw in the design of the public and private keys prevents the cryptosystem from avoiding continued fraction attacks.

A. Proofs of corollary 2 and lemmas 3, 4, 7 and 8

Proof of Corollary 2. First, we prove uniqueness. Suppose two distinct rational numbers $\frac{a}{b}$ and $\frac{c}{d}$ with $b, d \le Q$ both satisfy

$$\left|x - \frac{a}{b}\right| < \frac{1}{2Q^2}, \qquad \left|x - \frac{c}{d}\right| < \frac{1}{2Q^2}.$$

Then by the triangle inequality,

$$\left|\frac{a}{b} - \frac{c}{d}\right| \le \left|x - \frac{a}{b}\right| + \left|x - \frac{c}{d}\right| < \frac{1}{Q^2}.$$

On the other hand, since $\frac{a}{b} \neq \frac{c}{d}$,

$$\left|\frac{a}{b} - \frac{c}{d}\right| = \frac{|ad - bc|}{bd} \ge \frac{1}{bd} \ge \frac{1}{Q^2},$$

a contradiction. Hence at most one such rational exists.

Now let $\frac{a}{b}$ be the rational number satisfying the hypothesis. Since $b \le Q$, we have

$$\left|x - \frac{a}{b}\right| < \frac{1}{2Q^2} \le \frac{1}{2b^2}.$$

By Theorem 1, $\frac{a}{b}$ is a convergent of the continued fraction expansion of x. Since it is the unique rational with denominator at most Q within distance $\frac{1}{2Q^2}$, it must be the last convergent with denominator at most Q.

□

Proof of Lemma 3. The modulus bound (2) implies $g > p_i$ for every i, since $\lambda_{\max} \ge \max_i p_i$. If $t_i = t_j$, then (5) gives $p_i^{-1} \equiv p_j^{-1} \pmod{g}$. Multiplying by $p_i p_j$ yields $p_j \equiv p_i \pmod{g}$. Since $0 < p_i, p_j < g$, this forces $p_i = p_j$, contrary to distinctness when $i \neq j$.

Next, $0 < t_i, u < g$ and $p_i \ge z \ge 2$ give

$$0 < p_i(t_i + u) - 1 = k_i g < 2p_i g.$$

Dividing by $g > 0$ proves $0 < k_i < 2p_i$. Reducing the same equality modulo p_i gives $k_i g \equiv -1 \pmod{p_i}$. Every common divisor of k_i and p_i must therefore divide 1, so $\gcd(k_i, p_i) = 1$.

For two coordinates with $t_j > t_i$, their integer difference is at least one. Also, $p_i, p_j \ge z$ implies $|1/p_j - 1/p_i| < 1/z$. Subtracting the two identities in (8), we obtain

$$g(q_j - q_i) = (t_j - t_i) - \left(\frac{1}{p_j} - \frac{1}{p_i}\right) > 1 - \frac{1}{z} > 0.$$

Thus $q_j > q_i$. The extremal indices ℓ, h are unique, $D > 0$, and for $i \in I$ we have both $t_\ell < t_i < t_h$ and $q_\ell < q_i < q_h$. Consequently $0 < x_i, R_i < 1$.

To bound the denominator, define

$$H = k_h p_\ell - k_\ell p_h, \qquad A_i = k_i p_\ell - k_\ell p_i.$$

The just-proved order of the q_i gives $H > 0$ and $A_i > 0$. Computing the differences of the fractions $q_i = k_i/p_i$ explicitly yields

$$R_i = \frac{(k_i p_\ell - k_\ell p_i)/(p_i p_\ell)}{(k_h p_\ell - k_\ell p_h)/(p_h p_\ell)} = \frac{p_h A_i}{p_i H}. \tag{36}$$

Since $k_\ell > 0$ and $k_h < 2p_h$,

$$0 < H = k_h p_\ell - k_\ell p_h < k_h p_\ell < 2p_h p_\ell.$$

The unreduced positive denominator therefore satisfies $p_i H < 2p_i p_h p_\ell < 2P^3 = Q$. Reduction divides this denominator by a positive integer, so $M_i \le p_i H < Q$. Finally $0 < R_i < 1$ gives $0 < N_i < M_i$, as required. □

Proof of Lemma 4. For $j \in I \cup \{h\}$, put $\delta_j = \frac{1}{p_j} - \frac{1}{p_l}$. Because $p_j, p_l \ge z$, we have $|\delta_j| < \frac{1}{z}$. Subtracting the l-th identity in (8) from the i-th and h-th identities gives, respectively,

$$a_i = g(q_i - q_l) + \delta_i, \qquad D = g(q_h - q_l) + \delta_h.$$

Since $a_i = x_i D$, these equations imply

$$R_i = \frac{g(q_i - q_l)}{g(q_h - q_l)} = \frac{x_i D - \delta_i}{D - \delta_h}.$$

Subtracting this fraction from x_i and collecting terms gives

$$x_i - R_i = \frac{x_i(D - \delta_h) - (x_i D - \delta_i)}{D - \delta_h} = \frac{\delta_i - x_i \delta_h}{D - \delta_h}.$$

By the discussion before, $0 < x_i < 1$. Hence

$$|\delta_i - x_i \delta_h| \le |\delta_i| + x_i |\delta_h| < \frac{2}{z}.$$

The denominator satisfies $D - \delta_h \ge D - |\delta_h| > D - \frac{1}{z} > 0$ under (11). Combining the numerator and denominator bounds therefore gives

$$|x_i - R_i| < \frac{\frac{2}{z}}{D - \frac{1}{z}} = \frac{2}{zD - 1} < \frac{1}{2Q^2}, \tag{37}$$

where the last inequality uses $zD - 1 > 4Q^2$. □

Proof of Lemma 7. For the fixed index i, set

$$H = k_h p_l - k_l p_h, \qquad A_i = k_i p_l - k_l p_i, \qquad d_i = \gcd(p_h A_i, p_i H).$$

Then we can write R_i as

$$R_i = \frac{(k_i p_l - k_l p_i)/(p_i p_l)}{(k_h p_l - k_l p_h)/(p_h p_l)} = \frac{p_h A_i}{p_i H}. \tag{38}$$

The equation then gives

$$N_i = \frac{p_h A_i}{d_i} \qquad M_i = \frac{p_i H}{d_i}.$$

Because $\gcd(p_h, k_h) = \gcd(p_h, p_l) = 1$ as shown before, $\gcd(p_h, k_h p_l) = 1$. But any common factor of p_h and H divides $k_h p_l$, implying $\gcd(p_h, H) = 1$. So p_h and $p_i H$ are relatively prime, after reduction to lowest terms, the numerator retains p_h completely.

For the second divisibility relation, observe that

$$p_i H - p_h A_i = p_i(k_h p_l - k_l p_h) - p_h(k_i p_l - k_l p_i) = p_l(k_h p_i - k_i p_h).$$

Also,

$$M_i - N_i = \frac{p_i H - p_h A_i}{d_i} = \frac{p_l(k_h p_i - k_i p_h)}{d_i},$$

Reducing modulo p_l gives $p_h A_i \equiv -k_l p_h p_i$ and $p_i H \equiv -k_l p_i p_h$. All factors on the right are coprime to p_l. Consequently

$$\gcd(p_l, p_h A_i) = \gcd(p_l, p_i H) = 1, \qquad \gcd(p_l, d_i) = 1.$$

Because $M_i - N_i$ is an integer, d_i must divide $k_h p_i - k_i p_h$. We get $p_l \mid (M_i - N_i)$. □

Proof of Lemma 8. For each $j \neq 1$, reduction of (15) modulo p_1 gives

$$A_{1,j} \equiv -p_j \pmod{p_1}, \qquad \gcd(A_{1,j}, p_1) = \gcd(p_j, p_1) = 1.$$

Since $p_1 > 1$, this also shows that $A_{1,j} \neq 0$. Thus every anchor input is nonzero and is coprime to p_1. Writing $H_0 = \gcd_{2 \leq j \leq n} |A_{1,j}|$, we obtain $H_0 > 0$ and $\gcd(H_0, p_1) = 1$. For $2 \leq i < j \leq n$, a direct computation shows

$$\begin{aligned} p_i A_{1,j} - p_j A_{1,i} &= p_i(p_1 p_j(t_1 - t_j) + p_1 - p_j) \\ &\quad - p_j(p_1 p_i(t_1 - t_i) + p_1 - p_i) \\ &= p_1 p_i p_j(t_i - t_j) + p_1(p_i - p_j) \\ &= p_1 A_{i,j}. \end{aligned}$$

Since H_0 divides both anchor inputs, it divides $p_1 A_{i,j}$. Coprimality with p_1 implies $H_0 \mid A_{i,j}$. This also holds for pairs involving index 1 by definition. Hence H_0 divides the greatest

common divisor of all pairwise inputs. Conversely, the greatest common divisor of all pairwise inputs divides every anchor input and thus divides H_0. Both two greatest common divisors are positive, so they are equal, proving (16).

For the true vector, multiply $p_i(t_i + u) - 1 = gk_i$ by p_j and $p_j(t_j + u) - 1 = gk_j$ by p_i, then subtract. This gives

$$\begin{aligned} g(p_j k_i - p_i k_j) &= p_j(p_i(t_i + u) - 1) - p_i(p_j(t_j + u) - 1) \\ &= p_i p_j(t_i + u) - p_j - p_i p_j(t_j + u) + p_i \\ &= p_i p_j(t_i - t_j) + p_i - p_j = A_{i,j}. \end{aligned}$$

Every $A_{1,j}$ is therefore divisible by g, and $g_0 = cg$ with $c \in \mathbb{Z}_{>0}$ because $g_0 > 0$. □

Acknowledgments

The authors thank ChatGPT for assistance with some of the mathematical work presented in this paper. The authors subsequently verified the arguments and take full responsibility for the final content.

References

[1] J. Gómez-Torrecillas, F. J. Lobillo, and G. Navarro. A knapsack McEliece-based public key cryptosystem. *Designs, Codes and Cryptography*, 94, article 172, 2026. https://doi.org/10.1007/s10623-026-01913-y.

[2] A. Kalam and S. Sarkar. Breaking the Gómez-Torrecillas–Lobillo–Navarro cryptosystem: LLL recovers the secret key within seconds. Cryptology ePrint Archive, Paper 2026/2225, 2026. https://eprint.iacr.org/2026/2225.

[3] F. Beukers. *Elementary Number Theory*. Utrecht University, WISB321 lecture notes, Fall 2013 version, Theorem 8.1.5, p. 84. https://www.staff.science.uu.nl/~beuke106/getaltheorie/getalscript.pdf.

[4] J. E. Nymann. On the probability that k positive integers are relatively prime. *Journal of Number Theory*, 4(5):469–473, 1972. https://doi.org/10.1016/0022-314X(72)90038-8.

[5] V. Shoup. *A Computational Introduction to Number Theory and Algebra*. Second edition, Cambridge University Press, 2009, Chapter 4. https://www.shoup.net/ntb/ntb-v2_1.pdf.

[6] J. Sorenson and J. Webster. Strong pseudoprimes to twelve prime bases. *Mathematics of Computation*, 86(304):985–1003, 2017. https://doi.org/10.1090/mcom/3134.

[7] Python Software Foundation. `fractions`—Rational numbers, Python 3.12 documentation. `Fraction.limit_denominator`. `https://docs.python.org/3.12/library/fractions.html`.