Predicting Prime Factor Sums of Odd Semiprimes via Modular Residues: A Conjecture on Prime Factor Sums

Chandhru Srinivasan[*]
Molecular Oncology Laboratory, Department of Biochemistry,
School of Life Sciences,
Bharathidasan University,
Tiruchirappalli, Tamil Nadu, 620 024, India.
Email: Chandhrusrinivasan@bdu.ac.in
ORCID: 0009-0004-5621-825X

Abstract

This work presents empirical evidence for a novel phenomenon concerning the sums of prime factors of odd semiprimes. For a semiprime $N = p \cdot q$, where p, q are odd primes, this work investigates the modular residue properties of the sum $s = p + q$ modulo a dynamically chosen modulus m. We conjecture that $s \pmod{m}$ lies within a small subset $R_m \subset \mathbb{Z}_m$ whose size grows sublinearly relative to m. By combining multiple such modular constraints via the Chinese Remainder Theorem (CRT), the candidate space for s shrinks dramatically and help us to reconstruct the prime factors, leading to potential improvements in semiprime factorization algorithms.

Keywords: *Semiprimes, modular arithmetic, factorization, number theory.*

1. Introduction:

The factorization of composite integers, particularly semiprimes, is a classical problem in number theory with deep theoretical and practical significance (Crandall and Pomerance, 2001). Semiprimes — numbers that are the product of exactly two prime factors — play a central role in cryptography, computational number theory, and the study of arithmetic structures (Rivest et al., 1978). Most known factoring methods rely on probabilistic searches or smoothness methods (Brent, 1980; Lenstra, 1987; Pollard, 1975). In contrast, this study explores whether intrinsic modular structures in semiprimes encode deterministic information about their factors. Despite extensive research, the development of efficient deterministic methods for factorization remains a major challenge (Bressoud and Wagon, 2000).

In this note, this work presents the conjecture on the behavior of the sum of semiprime factors, $s = p + q$, in modular arithmetic. Specifically, this work observed that for any odd semiprime N and a chosen modulus m or any combinations of moduli, the value $s \bmod m$ is constrained to a surprisingly small subset of $\mathbb{Z}_m$. Empirical experiments confirm this behavior across a wide range of semiprimes, and the reduction in candidate values for s enables a significant narrowing of the search space for factorization.

2. Conjecture: Semiprime modular restriction Conjecture

Let $N = pq$, where p and q are distinct odd primes, and define $s = p + q$. Then for any modulus $m \in \mathbb{N}$, there exists a set $R_m \subset \mathbb{Z}_m$ such that:

1. $s \bmod m \in R_m$
2. $\mid R_m \mid \ll m$, with empirical evidence suggesting:

$$\frac{\mid R_m \mid}{m} \to 0 \text{ as } m \to \infty$$

3. Multiple such congruences:

$$s \equiv r_i \bmod m_i, r_i \in R_{m_i}$$

can be combined via CRT to isolate $s \bmod M$, with $M = \operatorname{lcm}(m_1, \ldots, m_k)$.

4. Once $M > s$, the value of s is fully recovered and thus p and q are efficiently computed.

2.1 The key properties of this system are:

i. Self-Contained Construction: The calculation of each residue set $\{R_m\}$ requires only the value of N.

ii. Guaranteed Constraint: The true factor sum is guaranteed to satisfy the congruence $S_N\ (mod\ m) \in R_m$ for every or all chosen moduli.

3. Methodology:

To evaluate the proposed modular constraint behaviour on semiprime factor sums, we implemented a custom Python script using only standard libraries.

3.1. Precomputation of Modular Signatures

The core of this method is a one-time overhead to generate and store a database of all possible modular restrictions on the factor sum, $S_N = p + q$. This process is performed once for a selected set of moduli or any moduli $\{m1, m2, \dots, mk\}$.

3.1.1. Choice of Moduli: The system's effectiveness is enhanced by using a diverse set of moduli. This implementation includes primorials (e.g., 30,210,2310,30030), which are products of the first primes, as well as smaller semiprimes (e,g., 13,17,19,23) and a few composites (e.g., 100,300,360). This variety is lower the computation burden and the explosion of CRT combinations.

3.1.2. The Signature Database: For each modulus m, I generate a key-value map, which I term the modular signature. This map is stored as a JSON file or a similar dictionary structure.

The key is a possible residue of a semiprime modulo m, i.e., $k = N(modm)$.

The value is the restricted set of allowed residues for the factor sum, $Rm = \{r1, r2, \dots\}$, where one of $r_j \equiv (p + q)(modm)$.

3.2. Algorithm for Signature Generation

The modular signature for each modulus is generated using the following algorithm:

1. Potential Factor Residues: Since is an odd semiprime, its factors and must be odd. Therefore, the set of all possible factor residues modulo m is defined as $\{1,3,5, \dots, k\}$, where k is the largest, odd integer less than m.

2. Generate Semiprime Residues: All possible residues of $N(mod\, m)$are computed by taking the outer product of the factor residue set with itself: $N_{res} \equiv p \cdot q(mod\, m)$ for all pairs $(p, q) \in P_m \times P_m$.

3. Map to Factor Sum Residues: A dictionary is initialized. For each computed semiprime residue N_{res} , compute the corresponding factor sum residue $S_{res} \equiv p + q(mod\, m)$ and add it to a set associated with the key N_{res}.

4. Store the Signature: After iterating through all pairs, the resulting dictionary, which now represents the complete modular signature for m, is saved to a file.

The python code and example modular signature files for the computations are available in the following Github repository: https://github.com/chandhruyuva006/Modular_constraint_based_integer_factorisation

3.3. Factorization via Signature Lookup and CRT

Once the database is precomputed, factoring any new large semiprime N becomes an efficient lookup-based process:

3.3.1. Residue Calculation and Lookup: For each modulus in our precomputed set, I calculate the residue $k_i = N(mod\, m_i)$. I then perform a lookup in our database to fetch the corresponding restricted set of factor sum residues, R_{m_i}.

3.3.2. Construct System of Congruences: This lookup process yields a system of congruences for the true factor sum S_N :

$$\begin{cases} S_N\ (mod\ m_i) \in R_{m_i} \\ S_N\ (mod\ m_i) \in R_{m_i} \\ S_N\ (mod\ m_i) \in R_{m_i} \end{cases}$$

3.3.3. Solve for S_N: The **Chinese Remainder Theorem (CRT)** is used to solve this system. By combining the constraints from each modulus, the set of possible candidates for is drastically reduced and the modulus (M_{eff}) might become larger based on the chosen $\{m_i, m_i, \dots m_i\}$ and I have a larger but sparse set of residues $R_{eff}\ where\ S_N\ \in \{r_i, r_i \dots \dots r_i\}$. An efficient search through this small solution space reveals the true value of S_N.

3.3.4. Final Factor Recovery: upon searching through the current constrained space the S_N the factors and are found by solving the quadratic equation $x^2 - S_N x + N = 0$, whose roots are the desired factors.

3.4. Illustrative Example: Factoring N = 72011

Here I demonstrate the entire process with the semiprime $N = 72011$. For this example, I will use our precomputed modular signatures for two small moduli: a primorial modulus $m_1 = 30$ and a prime modulus $m_2 = 17$.(The choice of the selection of the moduli is solely on the empirical evidence and I have observed that combining a primorial, small prime which is not the factor of the primorial and a composite moduli gave optimal search space reduction. But the stacked use of prime moduli can lead to a CRT combination explosion and may increase the memory usage and the moduli should be carefully stacked with CRT rather than combined at once. The example is chosen simply for demonstrating the CRT combination explosion vs search space reduction tradeoff).

(For the reader's reference, the true factors are 107 and 673, and the true factor sum is $107 + 673 = 780$)

Step 1: Signature Lookup

First, I calculate the residue of N for each selected modulus and perform a lookup in our precomputed signature database.

1. **For** $m_1 = 30$ **:**
   - I. Calculate $N\ (mod\ 30) = 72011\ (mod\ 30) = 11$.
   - II. Perform a lookup in our signature file. The precomputed signature reveals the restricted set of possible factor sum residues: R_{30}: $\{0, 12, 18\}$.
   - III. This gives us our first congruence: $S_N = 0\ mod\ (30)\ or\ 12\ mod(30)\ or = 18\ mod\ (30)$.

2. **For** $m_1 = 17$ **:**
   - I. calculate $N\ (mod\ 17) = 72011\ (mod\ 17) = 16$.
   - II. look up the key 17 in our file we have previously generated from the precomputation of modular signatures algorithm. Let's assume the precomputation yielded the set: R_{17}: $\{0, 2, 5, 6, 8, 9, 11, 12, 15\}$.
   - III. This gives us our second congruence: $S_N \in R_{17}$: $\{0, 2, 3, 7, 8, 9, 10, 14, 15\}$.

(Note: we can verify our true factor sum $S_N = 780$ satisfies these constraints: $0\ (mod30)$ and $15\ (mod\ 17)$ or I can say that $S_N \in\ R_{30}\!: \{0, 12, 18\}\ and\ R_{17}\!: \{0, 2, 3, 7, 8, 9, 10, 14, 15\}$

Step 2: Solving the Congruence System:

As 30 and 17 are coprime I can easily solve the CRT by creating all the possible combinations $R_{eff}\ = \{0, 42 \ldots 270, . . 480\}\ (3 * 8 = 24)$. But now the M_{eff} is also increased to 510. I can see that the R_{eff} is increased by a factor of 8 but the M_{eff} is increased by a factor of 17.

S_N is bounded by $2 *\ \sqrt{N}\ to\ N$. The start of $S_{N\_start}\ \approx 536$. From the R_{eff} I serve the possible candidates for the perfect square check and in less than 20 iterations I found the true $S_N = 780$ which satisfies the $S_N = 270\ mod(510)$.

Step 3: Final Factor Recovery

With the candidate value $S_N = 780$, I can find the factors by solving the quadratic equation $x^2 - 780 + 72011 = 0$. Solving the quadratic equation yields the unknown factors $p = 107, q\ = 683$.

3.5 Empirical testing:

The conjecture was tested for N ranging from $1\ -\ 10^{18}$. The code for this testing and example modular signature files for the computations is available in the following Github repository: https://github.com/chandhruyuva006/Modular_constraint_based_integer_factorisation

4. Conclusion

This work presented a novel modular residue framework that significantly constrains the possible values of the factor sum $s = p + q$ for odd semiprimes $N = pq$. Our experiments show that, for suitable moduli m, the true value of $s \bmod m$lies within a small, predictable subset of $\mathbb{Z}_m$, allowing for substantial reductions in the factor search space.

While the primes p and q may be randomly distributed, the semiprime N they form is not: it leaves a modular trace of its structure. Although reaching the exact factor sum remains computationally intensive for large N, our method reliably narrows the region where it must lie, revealing an underlying regularity in semiprime construction. We state this behavior as a conjecture and invite further mathematical investigation into its proof, properties, and potential cryptographic implications.

Acknowledgements

I sincerely thank Mr. Indhirakumar Murugesan, 12-B class members, PSGCAS class members, mathisfun.com members, Mr. Dario Alpern and open-source Python community for the standard library tools that enabled this implementation.

Supplementary information and data availability:

The code for this implementation and example modular signature files for the computations is available in the following Github repository: https://github.com/chandhruyuva006/Modular_constraint_based_integer_factorisation

References:

Brent, R.P., 1980. An improved Monte Carlo factorization algorithm. BIT 20, 176–184. https://doi.org/10.1007/BF01933190

Bressoud, D.M., Wagon, S., 2000. A course in computational number theory. Key College Publishing, in cooperation with Springer, New York.

Crandall, R., Pomerance, C., 2001. Prime Numbers. Springer New York, New York, NY. https://doi.org/10.1007/978-1-4684-9316-0

Lenstra, H.W., 1987. Factoring Integers with Elliptic Curves. Ann. Math. 126, 649. https://doi.org/10.2307/1971363

Pollard, J.M., 1975. A monte carlo method for factorization. BIT 15, 331–334. https://doi.org/10.1007/BF01933667

Rivest, R.L., Shamir, A., Adleman, L., 1978. A method for obtaining digital signatures and public-key cryptosystems. Commun. ACM 21, 120–126. https://doi.org/10.1145/359340.359342