A Digitwise Primality Testing Method via Reverse Positional Multiplication

arne@hortell.se

Abstract

This paper introduces a novel, deterministic approach to integer factorization and primality testing, based on positional digit analysis and carry propagation. Unlike probabilistic or algebraic methods, this approach iteratively reconstructs candidate factors of a composite number from right to left, using only the digits of the product and the arithmetic constraints imposed by digit-wise multiplication with carries.

1 Introduction

Factorization and primality testing are central problems in number theory and cryptography. While well-known methods such as the Miller-Rabin test, AKS primality test, and elliptic curve factorization provide either fast probabilistic or algebraic tools, they are typically opaque for manual computation or pedagogical use.

We propose a new approach based entirely on base-10 digit analysis. This method, which we call the **Digitwise Reverse Multiplication Method (DRMM)**, reconstructs factor pairs from the product by analyzing the interaction of each digit's multiplication and carry propagation. It combines human intuition with a deterministic mechanism that either converges to a factorization or exhausts all possibilities.

2 Method Overview

Let N be an integer to be factored. DRMM begins with the last digit d_0 of N, and determines all pairs (a_0, b_0) such that $a_0 \times b_0 \equiv d_0 \mod 10$.

Subsequent digits $d_1, d_2, \ldots$ are analyzed recursively, using the decimal multiplication rule:

$$d_k = \left(\sum a_i \cdot b_{k-i} + c_k\right) \mod 10$$

where c_k is the carry from the previous digit multiplication.

3 Example: Composite Input

Let $N = 10999891$. DRMM sees last digit $1 \rightarrow$ try pairs $(1,1), (3,7), (7,3), (9,9)$. Try $(3,7)$: $3 \times 7 = 21$, carry $= 2$.
Next digit is 9. Try combinations of second digits (a_1, b_1) such that:

$$d_1 = a_0 \cdot b_1 + a_1 \cdot b_0 + \text{carry} \equiv 9 \mod 10$$

Try $a_1 = 1, b_1 = 2 \rightarrow$ test holds. Continue similarly. Eventually yields: $1091 \times 1009 = 10999891$.

4 Example: Prime Input

Let N = 10007, a known 5-digit prime ending in 7. DRMM begins with digit pairs whose product ends in 7: (1,7), (3,9), (7,3), (9,9).Try (3,9): $3 \times 9 = 27$, carry $= 2$ Try $(3, 9) : 3 \times 9 = 27$, carry $= 2$.
Next digit is 0. Try pairs (a,b) such that:

$$3 \times b_1 + a_1 \times 9 + 2 \equiv 0 \mod 10$$

None satisfy the equation.

All other starting pairs similarly fail to satisfy the constraints at digit 1 or 2.
This rapid exhaustion of valid paths is a strong indicator that 10007 is prime.

5 Analysis

- Time Complexity: $O(n)$ levels, each branching into 2–5 combinations.
- Termination: Always halts (complete match or path exhausted).
- Correctness: Fully deterministic. If a valid digit decomposition exists, it will be found.
- Limitation: Designed for decimal-like inputs with no noise or obfuscation.

6 Applications and Future Work

- Human-verifiable factorization method
- Use in classroom settings
- Visualizing arithmetic structures
- Base generalization
- AI-guided heuristics for branching path prioritization

7 Conclusion

DRMM is an intuitive, visual method for factorization and primality checking. It is especially effective for numbers with regular digit structure and small factor gaps.

8 RSA Vulnerability Analysis in Binary Form

In binary, all primes end with 1 RSA modulus N = pq ends in 1.Binary DRMM only starts with (1,1) as last bit pair.

Bit-by-bit, each next bit is constrained to 1–2 possible combinations.Much narrower than in base-10. For poorly generated primes, or those with known suffixes/prefixes, DRMM may succeed.

While full 2048-bit RSA remains secure under conventional assumptions, DRMM's deterministic constraints and narrow search space—especially in binary—may enable partial or full key recovery in future implementations using specialized hardware or AI-guided heuristics.

9 16-bit RSA Illustration

Example: $61 \times 53 = 3233$
In binary:

$$p = 00111101$$
$$q = 00110101$$
$$N = 110010100001$$

Apply DRMM: last bit = 1 start with (1,1) and reconstruct.

Appendix A: Reverse Positional Multiplication

For product $N = A \times B$, digit d_k is formed as:

$$d_k = \left(\sum a_i \cdot b_{k-i} + c_k \right) \mod 10$$

Appendix B: Binary DRMM Transition Table

a_i	b_i	carry_in	total	bit_out	carry_out
0	0	0	0	0	0
0	1	0	1	1	0
1	0	0	1	1	0
1	1	0	2	0	1
1	1	1	3	1	1

Appendix C: Python Implementations

```
def drmm_decimal(n_str):
    from itertools import product
    n_digits = list(map(int, reversed(n_str)))
    def search(path_a, path_b, carry, pos):
        if pos >= len(n_digits):
            a = int(''.join(map(str, reversed(path_a))))
            b = int(''.join(map(str, reversed(path_b))))
            return (a, b) if a * b == int(n_str) else None
        digit = n_digits[pos]
        for a_d, b_d in product(range(10), repeat=2):
            total = a_d * b_d + carry
            if total % 10 == digit:
                new_carry = total // 10
                res = search(path_a + [a_d], path_b + [b_d], new_carry, pos + 1)
                if res: return res
        return None

def drmm_binary(n_bin):
    n_bits = list(map(int, reversed(n_bin)))
    def search(path_a, path_b, carry, pos):
        if pos >= len(n_bits):
            a = int(''.join(map(str, reversed(path_a))), 2)
            b = int(''.join(map(str, reversed(path_b))), 2)
            return (a, b) if a * b == int(n_bin, 2) else None
        bit = n_bits[pos]
        for a_i in (0, 1):
            for b_i in (0, 1):
                total = (a_i & b_i) + carry
                if total % 2 == bit:
                    new_carry = total // 2
                    res = search(path_a + [a_i], path_b + [b_i], new_carry, pos + 1)
                    if res: return res
        return None
```

Appendix D: Lattice Multiplication Grid