NSUCRYPTO 2020: Miller–Rabin revisited

Ioan Dragomir[1], Gabriel Tulba-Lecu[2], and Mircea-Costin Preoteasa[3]

[1] ioandr@gomir.pw – Technical University of Cluj-Napoca
[2] gabi_tulba_lecu@yahoo.com – Polytechnic Univeristy of Bucharest
[3] mircea_costin84@yahoo.com – Polytechnic Univeristy of Bucharest

Table of Contents

1 Problem summary

Bob tries to improve a well known probabilistic primality test – Miller-Rabin. The following modified version of the test is proposed:

Let n be the odd number to be tested, we will write it in the form $n-1 = 2^k 3^l m$, where m is not divisible by 2 or 3.

1. Select a random integer $x \in \{2, \ldots, n-2\}$
2. $a \leftarrow x^m \bmod n$
3. If $a \equiv 1$ (mod n) then return "**probably prime**"
4. For $i = 0, 1, \ldots, l-1$ do:
  If $a + a^2 \equiv -1$ (mod n) then return "**probably prime**"
  $a \leftarrow a^3$ (mod n)
5. For $i = 0, 1, \ldots, k-1$ do:
  If $a \equiv -1$ (mod n) then return "**probably prime**"
  $a \leftarrow a^2$ (mod n)
6. return "**composite**"

Question 1. Prove the correctness of the test above (i.e. for any prime number n it will never return "**composite**").

Question 2. Does the error probability of falsely classifying a "**composite**" number n as "**probably prime**" improve when we switch from the original test, (**OT**) which has an upper bound on the error probability of $\frac{1}{4}$ [1], to the modified test (**MT**)?

2 Problem solution

2.1 Presenting the original algorithm

We will keep referencing the **OT** in this section, so having its pseudocode written here will come in handy:

Let n be the odd number to be tested, we will write it in the form $n-1 = 2^k d$, where d is odd.

1. Select a random integer $x \in \{2, \ldots, n-2\}$
2. $a \leftarrow x^d \bmod n$
3. If $a \equiv 1$ (mod n) then return "**probably prime**"
4. For $i = 0, 1, \ldots, k-1$ do:
    If $a \equiv -1$ (mod n) then return "**probably prime**"
    $a \leftarrow a^2$ (mod n)
5. return "**composite**"

2.2 Solving Question 1

We will execute the algorithm in reverse and prove that if n is prime, then the program will return "**probably prime**".

The proof is based on Fermat's little theorem: Let n be a prime number, and x s.t. $\gcd(n, x) = 1$, then $x^{n-1} \equiv 1$ (mod n).

Since $n-1 = 2^k 3^l m$, then $\dfrac{n-1}{2^i 3^j} \in \mathbb{Z} \quad \forall\, 1 \le i \le k,\ 0 \le j \le l$. From these statements, we get the following result:

$$
\begin{aligned}
& x^{n-1} \equiv 1 \text{ (mod n)} \implies \\
& x^{n-1} - 1 \equiv 0 \text{ (mod n)} \implies \\
& (x^{\frac{n-1}{2}} - 1)(x^{\frac{n-1}{2}} + 1) \equiv 0 \text{ (mod n)} \implies \\
& x^{\frac{n-1}{2}} \equiv \pm 1 \text{ (mod n)}
\end{aligned}
$$

If $x^{\frac{n-1}{2}} \equiv -1$ (mod n), then the **MT** returns at the last step of loop **5** and we're done. Otherwise $x^{\frac{n-1}{2}} \equiv 1$ (mod n) and we must go one step backwards in the algorithm.

If $k \geq 2$ we remain in the body of loop **5**, otherwise we move to loop **4**. If we remain in loop **5**, we can repeat this technique: $x^{\frac{n-1}{2}} \equiv 1 \pmod{n} \implies x^{\frac{n-1}{2^2}} \equiv \pm 1 \pmod{n}$. This process repeats until we either find $1 \leq i \leq l$ s.t. $x^{\frac{n}{2^i}} \equiv -1 \pmod{n}$ or we exit loop **5**, and move backwards to loop **4**.

The same idea can be applied to loop **4**. If $l \geq 1$ and $x^{\frac{n-1}{2^k}} \equiv 1 \pmod{n}$. For simplicity we'll note $n' = \frac{n-1}{2^k}$:

$$
\begin{aligned}
& x^{n'} \equiv 1 \pmod{n} \implies \\
& x^{n'} - 1 \equiv 0 \pmod{n} \implies \\
& (x^{\frac{n'}{3}} - 1)((x^{\frac{n'}{3}})^2 + x^{\frac{n'}{3}} + 1) \equiv 0 \pmod{n} \implies \\
& x^{\frac{n'}{3}} \equiv 1 \pmod{n} \text{ or } (x^{\frac{n'}{3}})^2 + x^{\frac{n'}{3}} = -1 \pmod{n}
\end{aligned}
$$

Like before, if $(x^{\frac{n'}{3}})^2 + x^{\frac{n'}{3}} = -1 \pmod{n}$, then the **MT** returns at the last step of loop **4** and we have again finished. Otherwise $x^{\frac{n'}{3}} \equiv 1 \pmod{n}$ and we must go another step backwards. This will repeat until we either find $1 \leq j \leq l$ s.t. $x^{\frac{n'}{3^j}} \equiv -1 \pmod{n}$ or we reach the first step of the algorithm. At the first step we have:

$$
x^{\frac{n'}{3^l}} \equiv 1 \pmod{n} \iff x^{\frac{n-1}{2^k 3^l}} \equiv 1 \pmod{n} \iff x^m \equiv 1 \pmod{n}
$$

Therefore, if we reach the second step of the algorithm going backwards, it will always return "**probably prime**". Thus, if n is prime the **MT** must return "**probably prime**" at some point in its execution, so we have proved the correctness of the algorithm.

2.3 Solving Question 2

In [1], M. Rabin proves that the upper bound for the error probability of falsely classifying a "**composite**" number n as "**probably prime**" is $\frac{1}{4}$.

This upper bound is achieved for a special class of Carmichael numbers $n = p \cdot q \cdot r$, where p, q, r are prime numbers and $p \equiv q \equiv r \equiv -1 \pmod{4}$. If there is a number n s.t. $n - 1 \not\equiv 0 \pmod{3}$, then $n - 1 = 2^k m$, and so loop **4** will have no iterations, meaning **MT** is reduced to **OT**, so the upper bound for the error probability of **MT** equals the upper bound for the error probability of **OT**, which is $\frac{1}{4}$.

In order to find such a number, we wrote a program which found a bunch of them: $10546629279551, 19177682527151, 22799069430611, 52305745012067$ etc.

Therefore, we can say that the result M. Rabin obtained is not improved by this modification.

2.4 Bonus result

As a bonus answer for **Question 2**, we also found that other Carmichael numbers of the type above have different lower bounds if they are divisible by 3. The experimental result we obtained is that if n is a Carmichael number of the type above and $n-1 = 2^k 3^l m$, for m not divisible by 2 and 3, then the upper bound for the error probability of **MT** is:

$$\frac{1}{4} \cdot (1 - \frac{1}{3} - \frac{1}{3^2} - \cdots - \frac{1}{3^l}) = \frac{1}{4} \cdot (1 - \frac{3^l - 1}{2 \cdot 3^l}) = \frac{3^l + 1}{8 \cdot 3^l}$$

A Code for searching special Carmichael numbers

```
from fractions import gcd
# http://oeis.org/A002997 for p, q, r = -1 (mod 4)
w = [8911, 1024651, 1152271, 5481451, 10267951, ...]


def is_witness_orig(p2, d, a, n):
    x = pow(a, d, n)
    if x == 1: return False
    for i in range(p2):
        if (x+1)%n == 0: return False
        x = x * x % n
    return True

def is_witness_mod(p2, p3, d, a, n):
    x = pow(a, d, n)
    if x == 1: return False
    for i in range(p3):
        y = pow(x, 2, n)
        if (x + y + 1) % n == 0:
            return False
        x = x * y % n
    for i in range(p2):
        if x == n-1: return False
        x = x * x % n
    return True

def get_p2_p3_m(n):
    p2 = 0
    p3 = 0
    x = n-1
    while x % 2 == 0:
        x //=2
        p2+=1
    while x%3 == 0:
        x //=3
        p3+=1

    return (p2, p3, x)

def count_bad_witnesses(n):
    p2, p3, d = get_p2_p3_m(n)
    cnt_mod = 0
    cnt_org = 0
    total = 0
    for a in range(2, n):
        if(a % 5000000 == 0): # early stop to get an approximation
```

```
            return cnt_mod, cnt_org, total
        if(gcd(a, n) != 1):
            continue
        total+=1
        if not is_witness_mod(p2, p3, d, a, n):
            cnt_mod += 1
        if not is_witness_orig(p2, d * 3**p3, a, n):
            cnt_org += 1

    return (cnt_mod, cnt_org, total)

for i in w:
    ce, co, t = count_bad_witnesses(i)
    if ce == i-2 and co == i-2: continue
    print("i:", i, "total:", t, "mod:", ce, "orig:", co, 100 * ce/t, 100 * co/t)
```

References

1. Michael O. Rabin *Probabilistic Algorithm for Testing Primality.* Institute of Mathematics, Hebrew University, Jerusalem, Israel, and Massachusetts Institute of Technology, Cambridge, Massachusetts 02139.