Cryptography, Nathan McNew
Notes 04.16.2019

Benjamin Pecson

April 16, 2019

1 Miller-Robin Primality Test

Compositeness Test
Test if n is prime

Write n-1 as $m2^k$, where m is odd. Choose a randomly in 2 $\leq$ a $\leq$ n-1

Compute: $b_0 = a^m$ (mod n)

if $b_0 = \pm$ 1
Return probably prime

for i from 1 to k-1: $b_i \equiv b_i$ (mod n)

if $b_i \equiv$ -1(mod n)
return probably prime

if $b_i \equiv$ 1 (mod n)
return composite

If during the for loop $\pm$ 1 is never produced
return composite

Example,Given:

$b_k \equiv b_0^{2^k}$
$b_k \equiv a_0^{m \times 2^k}$
$b_k \equiv a^{n-1}$(mod n)
if $a^{n-1} \not\equiv 1$(mod n)
then n is composite by the Fermat Test

Another Example, Given:

a,b(mod n)
where a $\not\equiv$ b (mod n)
and a $\not\equiv$ -b(mod n)
but $a^2 \equiv b^2$ (mod n)
then n is composite and gcd(a-b,n) is a non-trivial factor of n

2 Proof of the Factory Trick

Suppose a $\not\equiv$ b (mod n)
and a $\not\equiv$ -b (mod n)
but $a^2 \equiv b^2$ (mod n)

Goal N has to be composite and a factor of n
Then a^2 - b^2 *equiv* 0 (mod n)
(a+b)(a-b)$\equiv$ 0 (mod n)
so, n divides (a+b)(a-b)

Proof by Contradiction:
Suppose GCD is a non-trivial factor of n
GCD(a-b,n)
Then gcd(a-b,n)$\equiv$1, or gcd(a-b,n)=n
if gcd(a-b,n)$\equiv$1
or gcd(a-b,n)=n
if gcd(a-b,n)=n
then n divides a-b
so, a-b $\equiv$0(mod n)
a$\equiv$b(mod n) ** Not allowed
to the gcd(a-b,n)=1

n has to divide a+b,
but if this is true a+b $\equiv$ 0 (mod n),
so a $\equiv$ -b (mod n)

How is this relevant to the Miller-Robin Test?

Suppose somewhere in the for loop the following:
$b_i \equiv b_{i-1}^2$(mod n) which yields 1
Note: $b_{i-1} \not\equiv \pm 1$(mod n) Note: $1^2 \equiv 1$(mod n)
The following is found: $(b_{i-1})^2 \equiv 1 \equiv 1^2$(mod n)
Note: $(b_{i-1}) \not\equiv \pm 1$ (mod n)
So, n has to be composite.

If n is composite, then at least 3/4 of the choice for a can be proven composite

Example: Miller Robin
Test n = 25
Find m and k
m = n - 1 = 24 = 8*3 = 2^3*3
m and k are 3
Pick a, a = 7, "Random" **For the purpose of demonstration
$b_0 = a * m(mod n) \equiv a^3(mod 25)$
$\equiv 7^3$(mod 25)
$\equiv 7^2$*7(mod 25)
$\equiv$ (-1)*7(mod 25)
b_0=18

For i in 1 to 2
Compute $b_i \equiv b_{i-1}$(mod n)

If $b_i \equiv 1$(mod n)
return composite

If $b_i \equiv -1$(mod n)
return probably prime

$b_i \equiv 18^2$ (mod 25)
$\equiv -7^2$ (mod 25)
49=-1(mod 25)
$b_i \equiv -1$(mod 25)
$b_i \equiv -1$
return "probably prime" according to Miller-Robin

Try a=4
$b_0 \equiv 4^3$(mod 25)
$\equiv$64(mod 25)
$\equiv$14(mod 25)
$b_1 \equiv 14^2$(mod 25)$\equiv 196(mod 25) \equiv 21(mod 25)\}$

$b_2 \equiv 21^2(mod 25)$
$\equiv$ 16(mod 25)
End of the loop

Composite confirmed by Miller Robinson Test

For i in 1 to 2
Compute $b_i \equiv b_{i-1}$(mod n)
Return Composite

If b_i ≡-1 mod n
Return probably prime

If the for loop finishes then n is composite

3 RSA and Miller Robinson

RSA uses Miller Robinson
Breaking RSA to find factors of n?
What is the best way to factor n = pq?
Idea: Trial division, divide n by numbers ¡ $\sqrt{n}$ until a factor is found.
How long will it take to check all the numbers of the square root of n?
For a computer the size of a number of bits required to write it down: The size of n is $\lceil log_2 n \rceil$
Suppose, x = log_2n
2^x=n

The run-time of Trial Division is O($\sqrt{2^x}$)
This is an example of exponential run-time.
Goal: Find algorithm with Big O ¡ trial Division
Can the factory trick make factoring faster?

If a$\not\equiv$b (mod n),
with $a^2 \equiv b^2$ (mod n)

Idea 2:
Pick a random a with a square root of n $\leq a \leq$n-1
Compute A $\equiv a^2$ (mod n)
If A $\equiv b^2$(mod n) for some b factor n

Example: 91
Pick a = 10
Compute $a^2 \equiv$ 100=9=3^2
Here, $10^2 \equiv 3^2$ (mod 91)
but 10 $\not\equiv$ 3,10 $\not\equiv -3$ (mod 91)
So here,
gcd(91, 10-3)=7

Since a is random
$A \equiv a^2$(mod n) is essentially a random number mod n

What is probability that A ≡ ¡something¿ (mod n)
How many squares are there less than n?
The floor function of $\sqrt{n}$ many squares
Probability that we get a square is the $\sqrt{n}/n = 1/\sqrt{n}$

Probability of success is $1/\sqrt{n}$

This means the run-time ends up being O($\sqrt{n}$)
This run time is also exponential.

How can an exponential run-time for factoring be beat?
Dixon's factorization algorithm has quadratic run-time, which is faster then exponential, but slower than polynomial run-time.
Dixon's factorization algorithm has an approximate run time of $O(e^{\sqrt{x \ln x}})$