# Generating provable primes

Sigurd Meldgaard    Kasper Damgård

7. Jan.

# Introduction

- Usually primes are generated by testing random numbers the randomized Miller-Rabin test

# Introduction

- Usually primes are generated by testing random numbers the randomized Miller-Rabin test
- If we know the factorization of $p-1$ where $p$ is prime, it is possible to generate a certificate showing the primality of $p$.

# Introduction

- Usually primes are generated by testing random numbers the randomized Miller-Rabin test
- If we know the factorization of $p - 1$ where $p$ is prime, it is possible to generate a certificate showing the primality of $p$.
- This way we can generate the prime "bottom up" in a recursive way.

# Introduction

- Usually primes are generated by testing random numbers the randomized Miller-Rabin test
- If we know the factorization of $p - 1$ where $p$ is prime, it is possible to generate a certificate showing the primality of $p$.
- This way we can generate the prime "bottom up" in a recursive way.
- We have implemented an algorithm constructing primes this way, and compared it to Miller-Rabin

# Euler's theorem

$$\forall n \in N, k, gcd(k,n) = 1 : k^{\phi(n)} \equiv 1 \mod n$$

# Lucas' primality criterion

Given a base $b$ and a prime candidate $n$.
Where $gcd(b, n) = 1$ and $b$ fulfills Fermat's equation for primes:
$b^{(n-1)} \equiv 1 \mod n$
The smallest exponent where the sequence:

$$1, b^2, b^3 \ldots \mod n$$

reaches 1 is called the period of $b$ mod $n$ or $ord_p(b)$.
We know that

$$b^{\phi(n)} \equiv 1 \mod n$$

so

$$ord_p(b) | \phi(n).$$

# Lucas' primality criterion cont...

We know that

$$ord_p(b) | \phi(n).$$

And also that

$$ord_p(b) | (n-1)$$

or equivalently $n - 1 = x \cdot ord_p(b)$ for some positive integer $x$.
If we can prove $x = 1$ then

$$ord_p(b) = n - 1 = \phi(n)$$

because: $\phi(n) \leq n - 1$, and then $n$ must be prime.

# Lucas' primality criterion cont...

Assume we know the factorization of $n - 1 = q_1^{\beta_1} q_2^{\beta_2} \dots q_r^{\beta_r}$.
Then we can check that:

$$b^{(n-1)/q_i} \not\equiv 1 \mod n, i \in 1..r$$

If we raise $b$ to a power not a multiple of $ord_p(b)$ it will be different from 1, so $(n-1)/q_i \not| ord_p(b)$ for any $i$
And if that is true, all factors of $n - 1$ are not factors of $x$, and therefore $x = 1$.

# Using Lucas to generate primes

- For generating large primes we can recursively generate smaller primes, multiply them and see if the product plus one is a prime by testing for Lucas' criterion.

# Using Lucas to generate primes

- For generating large primes we can recursively generate smaller primes, multiply them and see if the product plus one is a prime by testing for Lucas' criterion.
- For the base case (primes smaller than a certain threshold) we use trial division of a random number to construct the prime.

# Let the half be random

Really we only need to know the factorization of $F$ (if $F$ is odd), generate $R$ randomly $< F$ and let:

$$n = 2RF + 1$$

Because if $F$ is odd and the test succeeds for some base $b$ the smallest possible prime factor of $n$ is $2F + 1$, and because $F > R$ $n = (2RF + 1) < (2F + 1)^2$, $n$ must be prime.

As explained in [?] almost any base will work for showing the primality of $p$, the exact proportion of good bases is:

$$\phi(F)/F \geq 1 - \sum_{j=1}^{r} 1/q_j$$

- To ensure that the primes are generated reasonably uniformly the size of the smaller primes generated in the recursive calls must be chosen properly.

- To ensure that the primes are generated reasonably uniformly the size of the smaller primes generated in the recursive calls must be chosen properly.
- In [?] a method is given for choosing the sizes from the distribution of the relative size of the largest factor of a random integer $F$. And it is argued that the conditional distribution given that $2F + 1$ is prime is almost the same.

- To ensure that the primes are generated reasonably uniformly the size of the smaller primes generated in the recursive calls must be chosen properly.
- In [?] a method is given for choosing the sizes from the distribution of the relative size of the largest factor of a random integer $F$. And it is argued that the conditional distribution given that $2F+1$ is prime is almost the same.
- Also it is noted that if $F$ has only one prime factor, we still choose from among 10 % of all primes.

# Asymptotic running time

The asymptotic estimated running time of finding a $k$-bit prime with the asymptotically best multi-precision algorithms is:

$$O(k^3 \cdot \log\log(k)).$$

For straightforward integer arithmetic the estimated running time is:

$$O(\frac{k^4}{\log(k)})$$

# The base case

PROVABLE PRIME($k$)
INPUT: a positive integer $k$.
OUTPUT: a $k$-bit prime number $n$.

1. (If $k$ is small, then test random integers by trial division. A table of small primes may be precomputed for this purpose.) If $k \leq 20$ then repeatedly do the following:
   1.1 Select a random $k$-bit odd integer n.
   1.2 Use trial division by all primes less than $\sqrt{n}$ to determine whether $n$ is prime.
   1.3 If $n$ is prime then return(n).

# The recursive case

2. Set $c \leftarrow 0.1$ and $m \leftarrow 20$.
3. (Trial division bound) Set $B \leftarrow c \cdot k^2$.
4. (Generate r, the size of q relative to n) If k > 2m then repeatedly do the following: select a random number s in the interval [0, 1], set $r \leftarrow 2^{s-1}$ , until $(k - rk) > m$. Otherwise (i.e. $k \leq 2m$), set $r \leftarrow 0.5$.
5. Compute $q \leftarrow \text{PROVABLE PRIME}(\lfloor r \cdot k \rfloor + 1)$.
6. Set $I \leftarrow \lfloor 2^{k-1}/(2q) \rfloor$.

# The recursive case 2 (testing candidates)

7. *success* $\leftarrow$ *False*.
8. While (not *success*) do the following:
   8.1 (select a candidate integer $n$) Select a random integer $R$ in the interval $[l+1, 2l]$ and set $n \leftarrow 2Rq + 1$.
   8.2 Select a random integer a in the interval $[2, n-2]$.
   Compute $b \leftarrow a^{n-1} \mod n$.
   If $b = 1$ do the following:
      Compute $b \leftarrow a^{2R} \mod n$ and $d \leftarrow gcd(b-1, n)$.
      If $d = 1$ then *success* $\leftarrow$ *True*.
9. Return(n).

# Optimizations

- Do a single Miller-Rabin test with base 2 of $2FR+1$ before actually testing for Lucas' primality criterion.
  Quickly weeds out most of the composites.

Implemented the algorithm in Python.

- Used the built-in multi-precision integers.
- Speed loss due to interpretation is negligible. (Most time is spent doing exponentiations)
- We also implemented the Miller-Rabin primality test for comparison.

- There are big deviations from the average, this is due to the algorithm depending a lot on “being lucky” when choosing the random parameters.
- For practical purposes one would want to do the Miller-Rabin test for several bases to make the probability of accepting a composite number negligibly small.

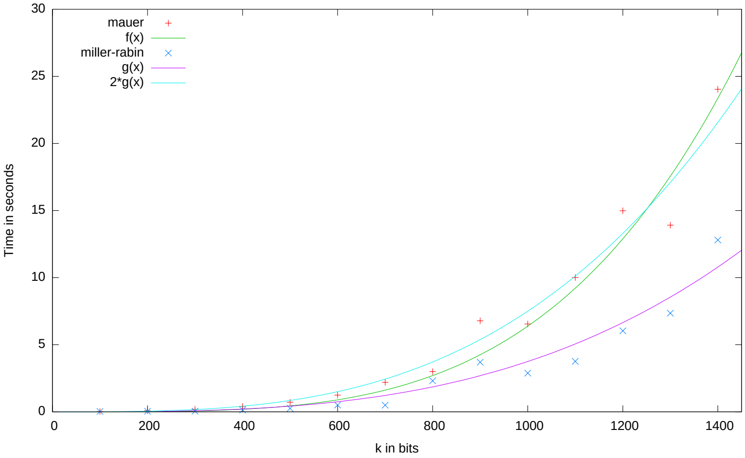


Figure: Timings of the two prime construction methods

# Conclusion

Generating provable primes for public key parameters is certainly practically possible. But Miller-Rabin test is easier to implement and can construct pseudoprimes with very high certainty in ca. the same time. And these numbers will be distributed truly uniformly among all primes.