Analysis of Primality Testing in Public Key Cryptosystems

Potential Outline of report:

1. Introduction
    a. Similar to brief overview from the proposal
    b. What is primality testing
    c. Why is it important in the sphere of public key cryptography
2. Methods of primality testing/Implementations
    a. This is where basically all the programming will go
        i. Fermat's Primality Test
        ii. AKS primality testing
        iii. Miller-Rabin
        iv. Ballie-PSW
    b. Comparisons and historical overview of how primality testing has evolved
        i. Compare based on Accuracy, Speed/Run time, scalability with bigger and bigger numbers, vulnerabilities etc
    c. How does each perform under controlled conditions
    d. Necessity of Accuracy for different use-cases
3. Implications of the failure of primality testing in applications
    a. example of constructing malicious Diffie-Hellman parameters.
4. Conclusive statements
    a. Re iterate importance of primality testing
    b. Summarize report
    c. Talk about the future of primality testing with the advent of quantum cryptography and its potential impacts on the framework of public key cryptography.

Introduction

Prime numbers are fundamental within cryptography, with a variety of the modern day encryption algorithms keeping our information secure relying on primality testing for their utility. Primality testing in public key encryption is the process of determining whether a number is prime, meaning it is only divisible by one or itself. Both used as a tool to validate prime parameters, or as part of the algorithm used to generate random prime numbers, primality tests are found nearly universally in cryptography. Efficient primality tests are needed for generating keys used in many modern cryptographic systems. The difficulty of the discrete logarithm problem (DLP) in many cryptographic applications is directly tied to the use of extremely large prime numbers and how they are generated. The hardness of this problem, which is crucial for public-key cryptography, is significantly reduced if the group is not carefully constructed.

In this report, we will go through a few of the most widely known primality testing algorithms, offering an overview followed by an implementation. We will compare their time complexities and accuracies and look into areas where these tests might fail and what implications that might have in the context of public key cryptography.

Methods and Implementations of Primality Testing[1]

Fermat Primality Test

The first primality test whose implementation we will look at is Fermat's Primality Test. The **Fermat primality test** is a probabilistic test to determine whether a number is a probable prime. Fermat's Little Theorem states that if p is prime and a is not divisible by p, then

$$a^{p-1} \equiv 1 \ (mod\ p)$$

If one wants to test whether p is prime, then we can pick random integers a not divisible by p and see whether the congruence holds.

Implementation (Pseudocode):

Input: Integer $n \geq 2$.
1: **for** $i = 1$ to k:
2: Choose random $a \in [2, n-1]$.
3: **if** $a^n \not\equiv a \pmod{n}$, **return** COMPOSITE.
4: **return** PROBABLY PRIME.

```
def FPT(prime,number_of_tests):
    random_number = random.sample(range(2,prime-2), number_of_tests)
                        #^ Choose random numbers to test
    for i in range(number_of_tests):
        if (pow(random_number[i],prime-1,prime) != 1):
            return True #Returns Composite
    return False        #Probably Prime
```

[1] Note that the implementations presented here are mostly pseudocode for purposes of a general outline to follow. A fleshed out version with proper syntax will be presented in the report

Miller-Rabin Test

The Miller-Rabin Test is another probabilistic primality test, however it actually improves on Fermat's test by taking advantage of the fact that if 1 has a square root other than ±1 (mod n), then n must be composite.

Implementation:

Input: Integer $n \geq 2$.

1: Choose random potential witness $a \in [2, n-2]$.
2: **if** n is even or $1 < \gcd(a, n) < n$, **return** COMPOSITE.
3: Set $a = a^q \pmod{n}$.
4: **if** $a \equiv 1 \pmod{n}$, **return** INCONCLUSIVE.
5: **for** $i = 0$ to $k - 1$:
6: **if** $a \equiv -1 \pmod{n}$, **return** INCONCLUSIVE.
7: Set $a = a^2 \pmod{n}$.
8: **return** COMPOSITE.

```
def thetest(pc,a): #pc is prime candidate | a is potential witness
    ast = a
    if (pc % 2 == 0): # check if candidate is even
        return
    d = euclidean_gcd(pc,a) # implementated but not shown
    if(d > 1 and d < pc):   #Check if the gcd is 1, composite if not
        return
    q = pc - 1              #compute pc - 1
    k = 0
    while (n1 % 2):         #Find q and k for n - 1 = 2^k * q
        n1=n1/2
        k+=1
    ast = pow(a,n1,pc)      #a <- a^q (mod pc)
    if (ast == 1):
        return              #Test Fails
    for i in range(k):      #test from 0 to k-1
        if (ast == pc - 1):
            return          #Test Failes
        ast = pow(ast,2,pc) # a <- a^2 (mod pc)

    return True             #return true for compisiteness if the cod
```

AKS Primality Test

The AKS Primality Test (Agrawal-Kayal-Saxena) is the first deterministic primality proving test that we will look at. This algorithm was the first one to determine in polynomial time whether a given number is prime or composite without relying on mathematical conjectures such as the generalized Riemann Hypothesis. It is based on the following theorem: Given an integer $n \geq 2$ and integer a coprime to n, n is prime if and only if the polynomial congruence relation

$$(X + a)^n \equiv X^n + a \,(mod\ n)$$

holds within the polynomial ring (Z/nZ)[X], where X denotes the indeterminate which generates this polynomial ring.

Implementation:

Input: Integer $n \geq 2$.

1: **if** $n = a^b$ for $a \in \mathbb{N}$ and $b > 1$, **return** COMPOSITE.
2: Find the smallest r such that $\mathrm{ord}_r(n) \not\leq \log^2 n$.
3: **if** $1 < \gcd(a, n) < n$ for some $a \leq r$, **return** COMPOSITE.
4: **if** $n \leq r$, **return** PRIME.
5: **for** $a = 1$ to $\lfloor \sqrt{\phi(r)} \log n \rfloor$:
6: **if** $(X + a)^n \not\equiv X^n + a \pmod{X^r - 1,\ n}$, **return** COMPOSITE.
7: **return** PRIME.

Baillie-PSW Test

The Baillie-PSW test is based on a combination of strong pseudoprimes and Lucas pseudoprimes. It is a probabilistic test that works similar to the other tests that we have shown: working to decide whether a given number is composite or a probable prime.

Implementation:

An implementation of the Baillie-PSW Test has been left out of this draft due to space requirements. The pseudocode that we came up with is simply too vast to fit the 2-4 page requirement intended for this 1st draft. However, a link for the pseudocode can be found here:
https://drive.google.com/file/d/18MQAgUFH2JRJrgwEaR_3bI0A4gNcsAX2/view?usp=sharing

Comparisons and Complexities

Test	Fermat	Miller-Rabin	B-PSW	AKS
Complexity*	$O(k\ log\ \ n)$	$O(k\ log^3 n)$	$O(k\ log^3 n)$**	$O(log(n)^{12})$***

* Complexity according to theory, not our implementations yet

** Note that even though Miller Rabin and B-PSW have similar running times, BPSW requires roughly three to seven times as many bit operations as a single miller rabin test

*** Can be reduced down to $O(log(n))^6$ if a well known conjecture pertaining to the distribution of Sophie Germain primes is true

- **Fermat**
  - Can give false positives for Carmichael Numbers
    - When the coprimes of a number adhere to Fermat's Little Theorem [1] $a^{-1} \equiv 1\ mod(n)$
  - Fast
- **Miller-Rabin**
  - Carmichael numbers and false positives extend to Miller-Rabin as well, since the test utilizes Fermat's Little Theorem.
  - Probability of error is 4^{-k}, where k is the number of times the test is run with a different modular base.
- **Baillie-PSW**
  - Slow
  - Believed to give no false positives
  - Known to be deterministic for values under 2^{64}
- **AKS**
  - Quite slow

- Is proven theoretically to give no false positives

[Further comparisons will be a result of further research and testing the algorithms against each other easier once they have all been implemented]

Implications of inaccurate primality tests

Primality Testing is very important in the security of public key cryptography. RSA and the Diffie-Helman key exchange rely on the underlying hard problems of Integer factoring and the Discrete Log Problem. We will analyze why having a prime modulus in both cases is necessary for security:

Diffie Helman:
The Diffie Helman Key Exchange relies on the difficulty of the Discrete Log Problem. Specifically,

$$g^x \equiv h \pmod{p}$$

for g, h in the group of units in the finite field of p prime. The Baby-Steps Giant-Steps algorithm (or any known, non-quantum algorithms) run in exponential time relative to the size of the input. This is a high enough computing time that for large enough primes p, the protocol will be secure. However, for p_* not prime, an attacker could factor

$$p^* = p_1^{e_1} p_2^{e_2} \cdots p_n^{e_n}$$

and use the Chinese Remainder Theorem to solve

$$\begin{aligned} g^x &\equiv h \pmod{p_1^{e_1}} \\ g^x &\equiv h \pmod{p_2^{e_2}} \\ &\vdots \\ g^x &\equiv h \pmod{p_n^{e_n}}. \end{aligned}$$

The order of the group of units for $\mathbb{F}^{\times}_{p_i^{e_n}}$ the finite fields generated by each $p_i^{e_n}$ in the factorization of p_* is $p_i^{e_i - 1}(p_i - 1)$ which is guaranteed to be less than the order of the group of units of the finite field of p_* elements:

$$|\mathbb{F}^{\times}_{p_*}| = \phi(p_*) = p_1^{e_1-1}(p_1-1)\, p_2^{e_2-1}(p_2-1)\cdots p_n^{e_n-1}(p_n-1).$$

Thus, breaking the protocol is reduced to solving many easier Discrete Log Problems, rather than one difficult problem.

Conclusion

Reliable primality testing is one of the strong foundations of modern cryptography. From RSA key generation to safe-prime groups and protocols, the ability to determine whether a large

integer is prime forms the basis of both security and performance. Practical requirements for primality testing are that they should be fast enough to handle hundreds or thousands of random candidates during key generation, and they must be trustworthy and accurate enough that the probability of accepting a composite is nearly zero.

Generally, mathematical guarantees and speed are inversely proportional, and are often traded off for one another across methods. Fermat's test is conceptually simple, and wonderfully fast, but can be fooled by Carmichael numbers. Miller-Rabin builds on the same principles, but results in catching far more composites. Baillie-PSW goes one step further, combining Miller-Rabin to base 2 with a strong Lucas probable-prime test. Alternatively, AKS delivers a deterministic, polynomial-time primality test, that unfortunately slows it down drastically.

We have completed the implementation of all these methods in ways that highlight the scalability of the methods. When scaling all implementations, the differences in accuracy and processing time become more apparent.

Looking ahead, quantum computing is positioned to disrupt assumptions more than algorithms. Shor's algorithm would make factoring and discrete logarithms efficient on larger quantum machines. This means primality testing itself won't be the bottleneck, rather the deeper impact is architectural, as we transition to post-quantum primitives, where many schemes no longer hinge on large random primes. Even so, primes will likely remain relevant, and will still require some form of primality testing.

Sources

1. Massimo, Jake. *An Analysis of Primality Testing and Its Use in Cryptographic Applications*, *University of London*, Royal Holloway, University of London, 2020.
2. Worthington, Riley. "Primality Testing: Theory, Complexity, and Applications." *Whitman College*, 11 May 2018, www.whitman.edu/documents/Academics/Mathematics/2018/Worthington.pdf
3. "AKS Primality Test." *Wolfram MathWorld*, mathworld.wolfram.com/AKSPrimalityTest.html
4. Nicely, Thomas R. "The Baillie-PSW Primality Test ." *TRNicely.Net*, Web.Archive.Org, web.archive.org/web/20130828131627/www.trnicely.net/misc/bpsw.html.