STRONG RSA AND EL GAMAL KEY GENERATION GUI

Michael Baker
With enough time and effort, all keys can eventually be cracked.

Encryption is a trade off between time and security.

Encryption keys should be viewed as secure only over a period of time.
Why Stronger Keys?

- Can’t simply compute any key.
- Weak keys make data communication unsafe.
- Average user does not want to think about cryptography.
Software designed to generate strong keys.

User does not need knowledge of cryptography.

Supports the time vs. security trade-off
How CryptoDefend works

- Two modes, RSA and El Gamal
- Generates keys using defenses against attacks.
- Allows the user to select the time investment when making a key.
Select two primes $p$ and $q$

Calculate $n = p \times q$ and $\phi_n = (p-1)(q-1)$

Select $e$ such that $\text{gcd}(\phi_n, e) = 1$

e and $n$ are the public key

Calculate private key $d = 1/e \mod \phi_n$
RSA Basics

- Encryption
  \[ E = M^e \mod n \]

- Decryption
  \[ M = E^d \mod n \]
RSA Defenses

- Choose large prime numbers
- Don’t let your private key get out
- Change keys regularly based on the size.
Initial Segment Attack

- When one prime contains a large number of zeros and the other prime is small.

- A multiple of the small prime can appear in the resulting n value.

- CryptoDefend:
  - Tests against the Attack and reports failing results.
Fermat’s Attack

- Based on odd integers being the difference of two squares.
- Relies on $p$ and $q$ being close together.

CryptoDefend
- Creates $p$ and $q$ values that will not be close enough to make this attack fast.
- Runs Fermat’s Defense algorithm to report weaknesses due to small prime number sizes.
Wiener’s Attack

- Can be a problem if the private key $d$ is too small.
- Requires $p$ and $q$ to be close together. Provides the requirement $q < p < 2q$.

- CryptoDefend
  - Generates keys outside this requirement to prevent Wiener’s attack.
  - Generates $e^{1/3}$ the bit selection to ensure a larger $d$. 
Pollard’s p-1

- It is based on Fermat’s Little theorem:
  - If $a < p - 1$ and $p > 2$ and $p$ is a prime number then $a^{(p-1)} \mod p = 1$

- CryptoDefend:
  - Uses Pollard’s defense algorithm to show max iterations until factorization of $p$ and $q$ is found.
  - Using a large bit size can help as well.
Pollard’s Rho

- Based on Floyd’s cycle-finding algorithm
- Good at factoring composite numbers with small factors.

CryptoDefend:
  - Keep primes as large as possible in the given bit size
Select a prime $P$ and a primitive root of $P$, $g$.

Select a value $x$ such that $0 < x < P-1$

Calculate $b = g^x \mod P$

Public key is $P$, $g$ and $b$

Private key is $x$
El Gamal Basics

- Encryption
  
  Select a random number $r$ such that $1 < r < P-1$

  Calculate
  
  $y_1 = g^r \mod P$ and $y_2 = M \cdot b^r \mod P$

- Decryption
  
  $M = y_2 \cdot (y_1)^{-x} \mod P$
El Gamal defenses

- Select a large prime for P
- Select a large value x.
A time-memory trade off algorithm

Works well against small values of \( \sqrt{P} \).

CryptoDefend:
- Ensure large value for \( \sqrt{P} \), warn if value is low.
Pollard’s Rho for Logs

- Similar to BSGS, but uses less memory.
- Works well when $\sqrt{p-1}$ is small.

- CryptoDefend
  - Ensure large value for $\sqrt{p-1}$, warn if value is low.
Very efficient when $p-1$ is smooth (has small prime factors)

CryptoDefend
- Factors $p-1$ and warns when all factors are small.
Implementation in Java

- General Benefits
  - Cross platform
  - NetBeans IDE for development
  - Tools are free
Implementation in Java (Benefits)

- Java.security.SecureRandom
  - Provides cryptographically strong random numbers
  - Uses pseudo-random algorithms with truly random seed values.
BigInteger Class

Breaks large number down and stores in an array.

Provides all basic arithmetic functions.

Provides modular arithmetic functions such as modPow and modInverse
Implementation in Java (Drawbacks)

- BigInteger limitations

- Limited functionality available for more advanced mathematics.

- Equations can be hard to read:
  - a.add(b).multiply(c.subtract(d))
CryptoDefend Demo