# Simulation and Comparison of Attacks on the RSA Cryptosystem

By: Heather DeVal, Jordan Fok, and Hannah Russell

# Overview

- RSA Cryptosystem
- Analysis of attacks
  - How they work
  - Difficulty for attacker
  - Time of attack
  - Reality of implementation
- Demo
- Comparison

# RSA Cryptosystem

- **Key Generation:**
  - Calculate $n = pq$, $p$ and $q$ are prime numbers
  - Compute $\phi(n) = (p-1)(q-1)$ and choose an integer e, where $\gcd(\phi(n), e) = 1$ and $1 < e < \phi(n)$
  - Calculate the integer $d$ using the equation $de\phi(n) = 1$.
  - The public key is the pair $\{e, n\}$ and the private key is the pair $\{d, n\}$.
- **Encryption:** $C = M^e(n)$
- **Decryption:** $M = C^d(n)$

# Blinding Attack

- The RSA Signing Process:
  - Allows for secure  communication between parties utilizing digital signatures.
- “Blinding” Scheme:
  - Allows the attacker to get a signature for a message from a party without the party viewing the true contents of the message.
  - Verification checks later will show that the message and signature are genuine
- Steps in Blinding Process:
  - **Blinding phase:** Attacker “blinds”  message using a random blinding factor

M’: blinded message
r: blind factor
e: public key exponent
N: product of two primes (p and q)

$$M' = r^e M \mod N$$

  - **Signing phase:** Attacker sends the blinded message to receiving party and requests them to sign it
    - Receiving party does not know better - they sign the message and send it back to the attacker

S’ = Signature sent
d = private key exponent

$$S' = (M')^d \mod N$$

  - **Unblinding phase:** Attacker removes the blinding factor from the message and obtains the signature from the original message.

$$\frac{S'}{r} = \frac{(M \times r^e)^d}{r} = \frac{(M^d \times r^{ed})}{r} = \frac{(M^d \times r^1)}{r} = M^d \mod (N)$$

# Code for Blinding

```
# Code adapted from https://asecuritysite.com/encryption/c_c2
import sys
import os
import hashlib
import libnum
import timeit

# Implementation of the Blinding Attack on RSA
# Heather DeVal, Jordan Fok, Hannah Russell
# CMSC 443 Final Project

e=79
d=1019
N=3337
r=21


def getMessage():
    message= input("What is your message? \n")
    return message

def print_values(message):
    print('== Welcome to the RSA Blinding Attack! ====')
    print('Public Key Exponent (e) =',e)
    print('Private Key Exponent (d) = ',d)
    print('Multiplication of two primes (n) =',N)
    print('Message to send =',message)
    print('Random blinding factor (r) =',r)
    print('\n=============')

```

# Code for Blinding Continued

```python
def blindingAttack():
    array = os.urandom(1 << 20)
    md5 = hashlib.md5()
    md5.update(array)
    digest = md5.hexdigest()
    M = int(digest, 16) % N
    print('The MD5 hash for your message:', digest)
    print('The value of MD5 hash mod n', M)
    signed=pow(M,d , N)
    print('The message is signed with the signature:\t',signed)
    value_sent = (M*pow(r,e, N))%N
    signed_value = pow(value_sent , d , N)
    print('You send the hacker your public signature: ',signed_value)
    result= (signed_value * libnum.invmod(r,N) ) %  N
    print('The hacker sends back your private signature of:',result)
    print('\n=== Let\'s Check If It Worked!==')
    unsigned = pow(result,e , N)
    print('The value of the unsigned message is:',unsigned)
    checkValue(unsigned, M)

def checkValue(unsigned, M):
    if (unsigned==M):
        print('*******YOU HAVE BEEN HACKED! THE MESSAGE HAS YOUR PRIVATE SIGNATURE ON IT**********')
    else:
        print('Lucky you! Something did not work out right. It does not appear you sent the other message.')

def timing():
    print("Time to execute attack: " + str(timeit.timeit(blindingAttack, number=1)) +" seconds.")

def main():
    message = getMessage()
    print_values(message)
    blindingAttack()
    timing()
main()
```

# Weiner Attack (Guess the d)

- Need to find decryption exponent **d**, given public mod **N** and public exponent **e**.
- Using quadratic equation, factoring, the Euclidean Algorithm, and continued fractions, we can break the encryption.
- Timing: polynomial
- Prevention
  - Pick a larger d. Wiener's attack probability of success is best when d < n^0.25

## Example

**Problem**

*An RSA encrpytion system uses public modulus $N = 64741$ and public exponent $e = 42667$. Find the decryption exponent $d$.*

If $\varphi(N) = 64000$, the equation

$$x^2 - 742x + 64741 = 0$$

should have integer solutions. The quadratic formula gives us $x = 641$ or $x = 101$, and we verify that $641 \times 101 = 64741$, which gives us the factorization of $N$ and the decryption exponent $d = 3$.

Remember we have $\frac{k}{d} \approx \frac{e}{N}$, so we want to find approximations to $\frac{e}{N} = \frac{42667}{64741}$. We'll use the Euclidean algorithm to find the successive convergents in the continued fraction expansion of $\frac{e}{N}$:

$$42667 \div 64741 = 0, \text{ remainder } 42667$$
$$64741 \div 42667 = 1, \text{ remainder } 22074$$
$$42667 \div 22074 = 1, \text{ remainder } 20593$$
$$22074 \div 20593 = 1, \text{ remainder } 1481$$

# Code for Weiner's Attack

```python
# Code adapted from https://pypi.org/project/owiener/
import owiener
import timeit
# Implementation of Wiener's Attack on RSA
# Heather DeVal, Jordan Fok, Hannah Russell
# CMSC 443 Final Project


def print_values(e,n):
    print("Welcome to Wiener's Attack on RSA!")
    print("The key values used in the RSA algorithm will be printed below. Note: I am unaware of the d (private key exponent) va
    print("The public exponent (e) is: ", e)
    print("The product of the two primes (n) is ", n)
    print("*************ATTACK BEGINNING****************")
    print()
    print()

def attackMode(d):
    if d is None:
        print("You are in luck!!! The attack failed!!! Your security remains intact another day.")
    else:
        print("*************UH OH*************")
        print("YOU WERE HACKED! YOUR IDENTITY IS NOT SECURE. THE D VALUE IS ={}".format(d))

def timing(e,n):
    attacking = "d = owiener.attack("+str(e)+","+str(n)+")"
    print("Time to execute attack: " + str(timeit.timeit(stmt=attacking, setup="import owiener", number=1)) +" seconds.")


def main():
    n = 10996616399290324377064345629609375913073751033373648335234548864343261420103062997020704793011565226853122207950823098
    e = 3074968630580206181633459116728403073447803142775149552792238809938192117262056931094541800746730645416001459782839070977
    d = owiener.attack(e, n)
    print_values(e, n)
    attackMode(d)
    timing(e,n)
    print()
    e = 7
    n = 77
    d = owiener.attack(e, n)
    print_values(e,n)
    attackMode(d)
    timing(e,n)


main()
```

# Factoring the Public Key

- RSA Key Generation relies on the factorization of prime integers.
- **Idea behind factorization:** If the attacker can find the prime factors used in the RSA process, they can compute the private key exponent d from any party's public key.

Factorization Process:

1. Factor n.
    - Options: difference of squares, quadratic formula manipulation, elliptic curve factorization, etc.
    - If this is done successfully, obtain p and q. P and q are the two prime integers used to compute N.
2. Compute phi.
    - $e^{-1} \bmod (p-1)(q-1) = d$. 1)
3. Compute d

** Note: This is only able to factor numbers up to 512 bits.
Defense: Use n > 2048 bits

# Code for Factorization

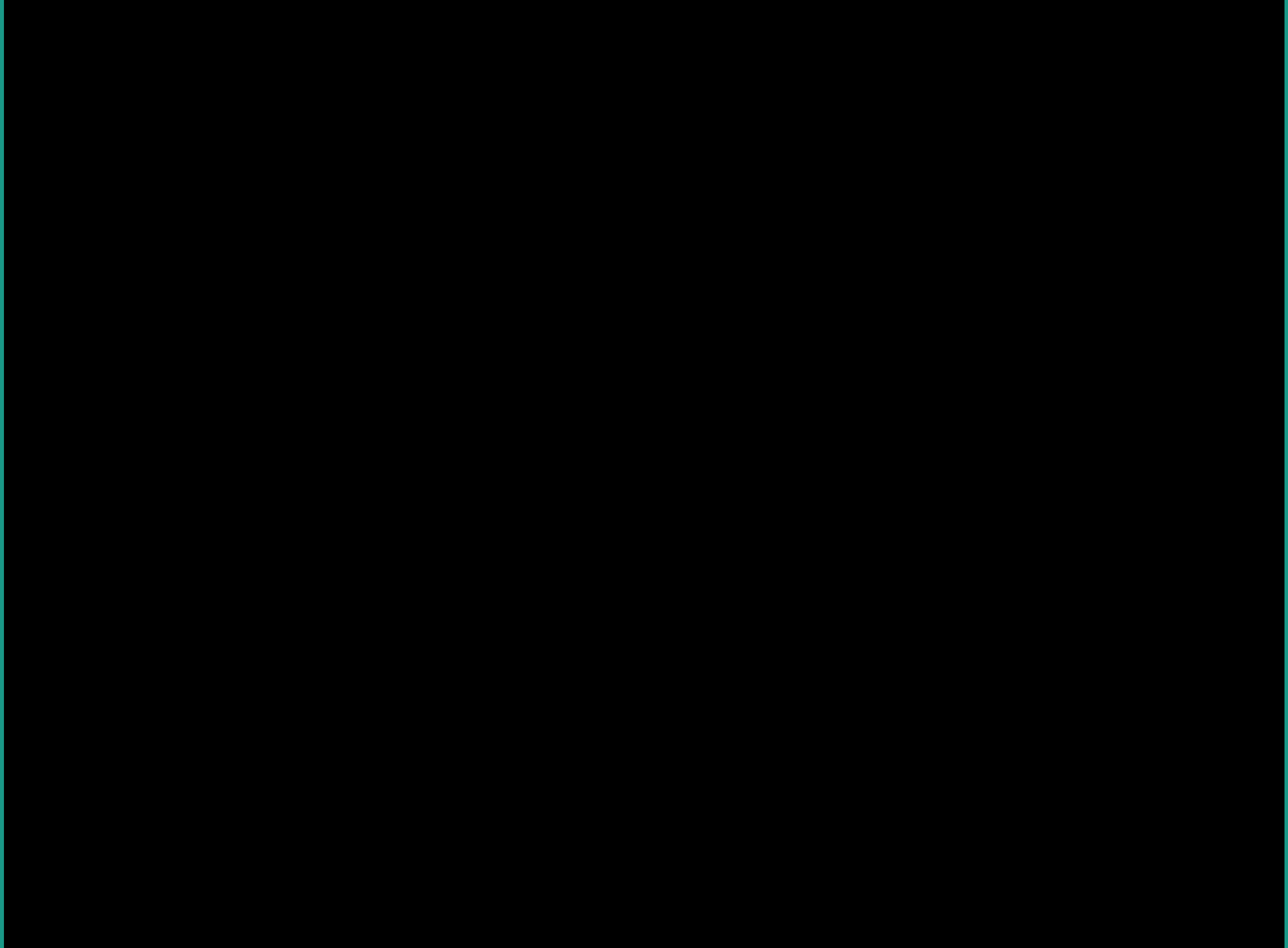
```
# Code adapted from https://asecuritysite.com/encryption/rsa12_2
# and https://medium.com/coinmonks/integer-factorization-defining-the-limits-of-rsa-cracking-71fc0675bc0e
from Crypto.Util.number import long_to_bytes
import libnum
import sys
import timeit
import math

# Implementation of Factorization Attack on RSA
# Heather DeVal, Jordan Fok, Hannah Russell
# CMSC 443 Final Project

N=36391
c1=35338

def print_values():
    print("Welcome to the Factorization Attack on RSA!")
    print("The key values used in the RSA algorithm will be printed below. Note: I only have ciphertext, and one prime number value - N")
    print("The ciphertext is: ", c1)
    print("The value of one prime (N) is ", N)
    print("*************ATTACK BEGINNING****************")
    print()
    print()

def factorTime():
    print ("Finding factors for",N)
    rtn=getfactor(N)
    print ("Factors are: ",rtn)
    p = rtn[0]
    q = rtn[1]
    n = p*q
    print("The product of the two primes is n: ", n)
    PHI=(p-1)*(q-1)
```

# Code for factorization (continued)

```python
    print("Using the two primes and Euler's law, phi is determined: ", PHI)
    e=65537
    d=(libnum.invmod(e, PHI))
    print("The private exponent d is determined using phi and the public exponent e (guessing 65537  - most commonly used)")
    print("The value of d is: ", d)
    print ("\n=== Attempting Decryption....... ===")
    res1=pow(c1,d,n)
    print ("The decrypted ciphertext is: %s" % long_to_bytes(res1))

def timing():
    print("Time to execute attack: " + str(timeit.timeit(factorTime, number=1)) +" seconds.")

def getfactor(y):
        i=0
        while True:
                val = y + i*i
#               print i,val
                sq = int(math.sqrt(val))
                if (sq*sq == int(val)):
                        print ("Factors: (",sq,"+",i,"),(",sq,"-",i,")")
                        return(sq-i,sq+i)
                i=i+1
                if (i==10000): return("Cannot find")
        return "Cannot find"

def main():
    value = N
    print_values()
    factorTime()
    timing()


main()
```

# Code Demo

# Blinding Attack



# Closer look...

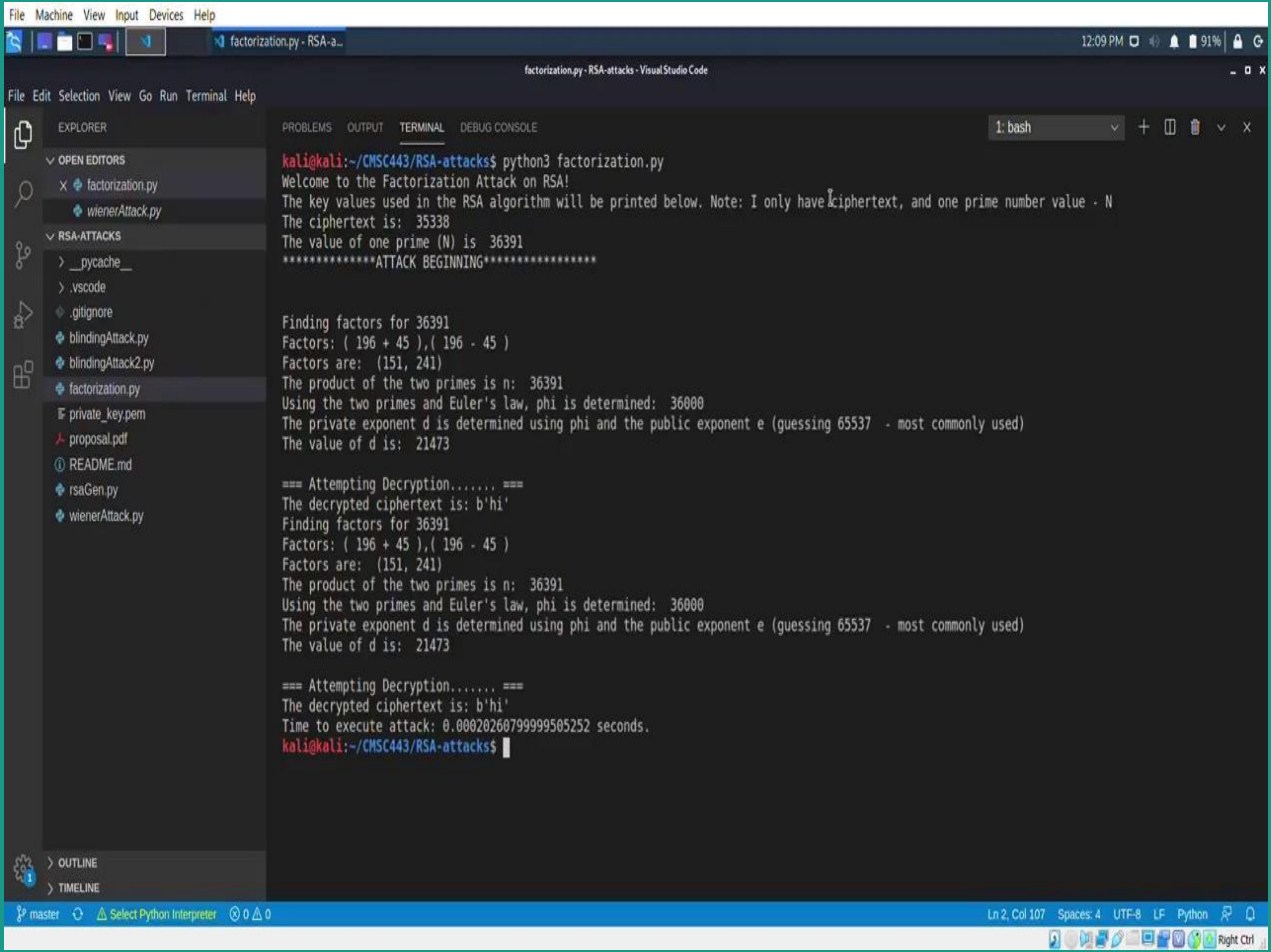
```
File   Actions   Edit   View   Help
kali@kali:~/CMSC443/RSA-attacks$ python3 blindingAttack2.py
What is your message?
Let's get hacked
== Welcome to the RSA Blinding Attack! ====
Public Key Exponent (e) = 79
Private Key Exponent (d) =  1019
Multiplication of two primes (n) = 3337
Message to send = Let's get hacked
Random blinding factor (r) = 21

=============
The MD5 hash for your message: 30e579b9deadfa105b6ffcf902c1b7b4
The value of MD5 hash mod n 2714
The message is signed with the signature:        287
You send the hacker your public signature:  2690
The hacker sends back your private signature of: 287

=== Let's Check If It Worked≠=
The value of the unsigned message is: 2714
*******YOU HAVE BEEN HACKED! THE MESSAGE HAS YOUR PRIVATE SIGNATURE ON IT**********
The MD5 hash for your message: 353f1f9c3fdec1dcb04f7eec8a97bac6
The value of MD5 hash mod n 798
The message is signed with the signature:        46
You send the hacker your public signature:  966
The hacker sends back your private signature of: 46

=== Let's Check If It Worked≠=
The value of the unsigned message is: 798
*******YOU HAVE BEEN HACKED! THE MESSAGE HAS YOUR PRIVATE SIGNATURE ON IT**********
Time to execute attack: 0.011167082000611117 seconds.
kali@kali:~/CMSC443/RSA-attacks$
```

# Wiener’s Attack

# Closer look...

```
kali@kali:~/CMSC443/RSA-attacks$ python3 wienerAttack.py
Welcome to Wiener's Attack on RSA!
The key values used in the RSA algorithm will be printed below. Note: I am unaware of the d (private key exponent) value
The public exponent (e) is:  3074968630580206181633459116728403073447803142775149552792238809938192117262056931094541800746730645416001459782839070977086157747932979394810340848949402527283447355585483504415337497855441441630501226764395783899864865110070544687597957367575767605387333733876537528353237076626094553367977134079292593746416875606876735717905892280664538346000950343671655257046364067221469807138232820446015769882472160551840052921930357988334306659120253114790638496480092361951536576427295789429197483597859657977832368912534761100269065509351345050758943674651053419982561094432258103614830448382949765459939698951824447818497599
The product of the two primes (n) is  109966163992903243770643456296093759130737510333736483352345488643432614201030629970207047930115652268531222079508230987041869779760776072105738457123387124961036111210544028669181361694095594938869077306417325203381820822917059651429857093388618818437282624857927551285811542685269229705594166370426152128895901914709902037365652575730201897361139518816164746228733410283595236405985958414491372301878718635708605256444921222945267625853091126691358833453283744166617463257821375566155675868452032401961727814314481343467702299949407935602389342183536222842556906657001984320973035314726867840698884052182976760066141
**************ATTACK BEGINNING*****************


*************UH OH*************
YOU WERE HACKED! YOUR IDENTITY IS NOT SECURE. THE D VALUE IS =4221909016509078129201801236879446760697885220928506696150646938237440992746683409881141451831939190609743447676525325543963362353923989076199470515758399
Time to execute attack: 0.002943782001239015 seconds.

Welcome to Wiener's Attack on RSA!
The key values used in the RSA algorithm will be printed below. Note: I am unaware of the d (private key exponent) value
The public exponent (e) is:  7
The product of the two primes (n) is  77
**************ATTACK BEGINNING*****************


You are in luck!!! The attack failed!!! Your security remains intact another day.
Time to execute attack: 8.687999070389196e-06 seconds.
kali@kali:~/CMSC443/RSA-attacks$
```

# Factorization Attack



# Closer look...

```
kali@kali:~/CMSC443/RSA-attacks$ python3 factorization.py
Welcome to the Factorization Attack on RSA!
The key values used in the RSA algorithm will be printed below. Note: I only have ciphertext, and one prime number value - N
The ciphertext is:  35338
The value of one prime (N) is  36391
*************ATTACK BEGINNING****************


Finding factors for 36391
Factors: ( 196 + 45 ),( 196 - 45 )
Factors are:  (151, 241)
The product of the two primes is n:  36391
Using the two primes and Euler's law, phi is determined:  36000
The private exponent d is determined using phi and the public exponent e (guessing 65537  - most commonly used)
The value of d is:  21473

=== Attempting Decryption....... ===
The decrypted ciphertext is: b'hi'
Finding factors for 36391
Factors: ( 196 + 45 ),( 196 - 45 )
Factors are:  (151, 241)
The product of the two primes is n:  36391
Using the two primes and Euler's law, phi is determined:  36000
The private exponent d is determined using phi and the public exponent e (guessing 65537  - most commonly used)
The value of d is:  21473

=== Attempting Decryption....... ===
The decrypted ciphertext is: b'hi'
Time to execute attack: 0.00029603499933728017 seconds.
```

# Comparison

### Time (in seconds) to attack:

| Blinding | Wiener's Attack | Factorization |
|---|---|---|
| .01-.02 sec | .002 sec (success)<br>9E-6 sec (fail) | .0003 sec |

- Fastest Overall Runtime: Factorization

### Implementation Difficulty:

| Blinding | Wiener's Attack | Factorization |
|---|---|---|
| - Easiest | - Moderately difficult<br>- Multiple mathematical computations<br>- Does not work in every situation. d must be less than N^(¼) | - Moderately difficult - need prime factorization method<br>- Mathematical computation and bitwise operations<br>- Only works for 512 bit numbers |

# Ethics

- It is not ethical or appropriate to hack others for personal, monetary, or political gain.
- General Rule of Thumb:
  - Do not hack people unless given explicit consent to do so.
  - Can result in jail time
- Ethical Hacking
  - Vulnerability Analysis
  - Penetration Testing
  - Very important in Defensive Security Operations

# Our Public GitHub and References Used:

- **Github Link:** https://github.com/hdeval1/RSA-attacks
- **References:**


    - B. Buchanan, “Defining the Limits of RSA Cracking“, August 2018.
    - B. Kaliski, “The Mathematics of the RSA Public-Key Cryptosystem,” *RSA Laboratories*, pp. 1–9.
    - *Blinding Attack*, asecuritysite.com/encryption/c_c2.
    - D. Boneh, “Twenty Years of Attacks on the RSA Cryptosystem,” *Stanford University*, pp. 1–16.
    - E. Milanov, “The RSA Algorithm,” *Washingtown University*, June 2009.
    - M. Wiener, “Cryptanalysis of Short RSA Secret Exponents“, *BNR*, August 1989.
    - OBE, Bill Buchanan. “Everything You Wanted To Know about Integer Factorization, but Were Afraid To Ask ..” *Medium*, Coinmonks, 30 Apr. 2020, medium.com/coinmonks/integer-factorization-defining-the-limits-of-rsa-cracking-71fc0675bc0e.
    - “Owiener.” *PyPI*, pypi.org/project/owiener.
    - *RSA Crack in 12 Lines of Python*, asecuritysite.com/encryption/rsa12_2.