CS 13: Mathematical Foundations of Computer Science — Fall 2023

Cryptography: Application 00 (due Friday, April 28)

Directions: *Write up carefully argued solutions to the following problems. Your solution should be clear enough that it should explain to someone who does not already understand the answer why it works. You may use results from lecture and previous homeworks without proof. Your solutions* must *be written in LaTeX using our homework template.* ***No solution to a single part may be more than one page.***

In this application assignment, you will explore various ways that RSA can completely break if the keys aren't chosen carefully!

0. Message Size (20 points)

The first attack will be based on the *size of the message* being small. You may note that this attack is also dependent on e being small, but, in most RSA implementations e is usually chosen to be small on purpose.
Your job is to decrypt `encrypted_message` given that it was encrypted using **K0**. To do this, you will want to take the eth root of the encrypted message. Consider using the two argument `pow` function in Python and rounding the output with `roundA`.

K0:
```
N = 277570988570648267370360984740316279555330557374192811071432259821254 6
    064401130588356083212149731539713518997317662754408992082531653575866 2
    004057452835043977047864453944520289174206459357701613961369669485256 4
    637488948292528636285794081930910087953641342772995250347596991617621 6
    483016760866504584237406332611460530663403383190300943924923228331696 3
    697972003458183737076206646022007178761689286628727519302964840911264 2
    163415712581737235294914260411006413647070543258389417821463936350163 5
    355060731864097854838445218590440090318807953424713133843524131675064 4
    50707480572685136642616940620873011795462432466768967675 3

e = 3
```

```
encrypted_message = 1095999387756223995873452692127687 68
```

Note that to convert `num` (a number) to `m` (a string representing the message), you can use the following line in Python: `m = bytes([x for x in num.to_bytes(1000, byteorder='little') if x != 0])`.

(a) [10 Points] Write and submit Python code that recovers the original message in English **and** also submit the actual decrypted message.

(b) [10 Points] Write and submit an explanation of why the message has to be short for this attack to work.

1. Wiener's Attack (80 points)

The second attack will be significantly more sophisticated than the previous one (though, surprisingly not that much more code). Consider the following (weird) theorem:

Wiener's Theorem

Given an RSA key (N, e, d, p, q) with the following properties:

- $q < p < 2q$
- $d < \frac{N^{\frac{1}{4}}}{3}$

An attacker can efficiently recover the entire private key: (d, p, q) from just the public part: (N, e).

While this theorem looks scary and arbitrary, you'll investigate it in two ways in this assignment. First, you'll finish a proof of the theorem. Then, you'll implement the attack on some insecure keys.

Before we can get into either the implementation or proof of the theorem, we need make a quick digression to the land of continued fractions! Continued fractions are a way of representing a real number as a sequence of integers. Given a number $x \in \mathbb{R}$, it's continued fraction representation is of the form:

$$x = a_0 + \cfrac{1}{a_1 + \cfrac{1}{a_2 + \cfrac{1}{a_3 + \cdots}}}$$

where $a_0, a_1, a_2, a_3, \ldots$ are known as the *coefficients* of the continued fraction.

The *convergents* of a continued fraction are the best rational approximation of the number x which can be obtained by truncating the continued fraction. The n-th convergent is of the form:

$$\frac{h_n}{k_n} = a_0 + \cfrac{1}{a_1 + \cfrac{1}{a_2 + \cfrac{1}{\ddots + \frac{1}{a_n}}}},$$

Now we are ready to start proving Wiener's theorem! First, note that, by the definition of RSA, $ed \equiv_{\phi(N)} 1$. So, there is a $k \in \mathbb{Z}$ such that $ed - k\phi(N) = 1$. Dividing both sides by $d\phi(N)$, we get:

$$\frac{e}{\phi(N)} - \frac{k}{d} = \frac{1}{d\phi(N)}$$

It may also be helpful to remember that the definition of RSA gives that $e < \phi(N)$.

A Theorem By Legendre

If $\left|a - \frac{b}{c}\right| < \frac{1}{2c^2}$ and $\gcd(b, c) = 1$, then b/c appears as some convergent of the continued fraction of a.

(a) [10 Points] Use the above to prove that $a = \dfrac{e}{\phi(N)}$ with $\dfrac{b}{c} = \dfrac{k}{d}$ satisfies the preconditions of Legendre's theorem. That is, show

$$\gcd(k, d) = 1 \qquad \text{and} \qquad \left|\frac{e}{\phi(N)} - \frac{k}{d}\right| = \frac{1}{d\phi(N)} < \frac{2}{dN} < \frac{1}{2d^2}$$

You'll need to use the givens, from above: $d < \frac{N^{\frac{1}{4}}}{3}$ and $q < p < 2q$. If it helps, you may assume that p and q are large. In this case, $p, q \geq 11$ should be more than sufficient.

Unfortunately, for us, $\phi(N)$ is still something we cannot compute. Instead, we'll try switching the denominator on the left to just plain N by showing the resulting difference is still within the threshold.
That is, we want to show:

$$\left|\frac{e}{N} - \frac{k}{d}\right| < \frac{1}{2d^2}$$

But first, a lemma!

Lemma 1. $|N - \phi(N)| < 3\sqrt{N}$.

(b) [10 Points] Prove Lemma 1 using the given that $q < p < 2q$.

Now, use the first lemma to prove another lemma!

Lemma 2. $\left|\frac{e}{N} - \frac{k}{d}\right| < \frac{3k}{d\sqrt{N}}$.

(c) [10 Points] Prove Lemma 2 using Lemma 1 and the proof of Wiener's Theorem on Wikipedia. We believe the proof on there is so convoluted that it is a good usage of your time to try to understand it instead of duplicate it. Note that you must justify every step in this part (which Wikipedia does not do). You may not cite the proof on Wikipedia directly though.

And again, another lemma!

Lemma 3. $k < d$.

(d) [10 Points] Prove Lemma 3.

(e) [10 Points] Complete the proof of the new version of the claim by using Lemma 2 and Lemma 3. That is, show:

$$\left|\frac{e}{N} - \frac{k}{d}\right| < \frac{1}{2d^2}$$

Finally! Now that we've satisfied the constraints of Legendre's theorem above, we can apply it to find $\frac{k}{d}$ as a convergent of $\frac{e}{n}$.

Now, we can get to coding! You can now write a Python program to implement Wiener's Attack on some unsuspecting keys. This attack will have three pieces:

I. Find the coefficients of the continued fraction of e/N.

1. Initialize $r = e/N$. Sequentially take $i = \lfloor r \rfloor$, add i to your list of coefficients, and set $f = r - i$. If $f = 0$, stop. Otherwise, set $r = 1/f$ and repeat.
2. Use the `fractions` library in Python to wrap your r in a `Fraction` object - this will avoid compounding floating point errors and make your life much easier.

II. Find the convergents of the continued fraction of e/N.

1. Given a list of coefficients a_n for $n \geq 0$ (which you calculated in step I.), the convergents $\frac{h_n}{k_n}$ can be found according to the recurrences:

$$\begin{aligned}
h_{-2} &= k_{-1} = 0 \\
k_{-2} &= h_{-1} = 1 \\
h_n &= a_n h_{n-1} + h_{n-2} \\
k_n &= a_n k_{n-1} + k_{n-2}
\end{aligned}$$

III. For each convergent $c = a/b$:

1. Calculate $\texttt{potential\_phi} = (eb - 1)/a$. Use integer division (which is // in Python).
2. Solve this quadratic equation using the Python code below.

$$x^2 - (N - \texttt{potential\_phi} + 1)x + N = 0$$

```python
def solve(a, b, c):
    from decimal import Decimal, getcontext
    getcontext().prec = 1000
    a = Decimal(a)
    b = Decimal(b)
    c = Decimal(c)
    return (int((-b + (b**2 - 4*a*c).sqrt())/(2*a)), int((-b - (b**2 - 4*a*c).sqrt())/(2*a)
        ))
```

3. Check if the two solutions to the quadratic equation multiply together to make N. If so, we found a factorization!

(f) [30 Points] For each of the three RSA keys below, recover p and q. You should submit the actual values as well as your Python code.

K1:

```
N = 90581

e = 17993
```

K2:

```
N = 1327599385544782705790557531615292893380355971356103848695355695529552
    8774181862206805700546600906242564213981921365567995067466223388392052
    3455499163854640108988961862458566028351785784234404800665923764600071
    1273359964980769196822643351949626452954980869768799116381751066867762
    4950462967083150968059963277221081933239285021940877379067473095833217
    3836456725156352153882945661196808513029619022547193596480809570835563
    4740046281649704634417307816836168827085460709390259205853957073796135
    7033856659209349287236681426501482268591633258592399958227708167643570
    575052735251331518114924442746592771217694695504776276961

e = 6869454678986259943295672398532193670233196629741642654148482133761559
    1963906525585042771062213382688701299757720554618956684218561467630031
    0551686510418475520097270226500394372126574437409734435424145135351350
    1857878500145063148153328058395688185719694760622513640095297444895603
    1279830234286882069839601954548091097738205903547685623413058371084466
    0675709637272256599063035035258056132429807747785354614488875836317279
    5669876649383889558979157217188456756256190891713436963251646440807031
    6126918264253106051479057988714187015111852070580754712993481203973484
    2603015299505413097056641604844102873006636917657600017
```

(continued on next page)

K3:

```
N = 8812043537783992834592375402234870396641825341735701299647176256406599
    7329680517708968821609230038336072999648018850151254250723601136064813
    9710988057569803880577166678963426806064048958935291877637033951269757
    4115046707945166768935552152018424854018421507242016690189690266914976
    9371706947812076282951446003219379614266454401718124335324868348241847
    9004213375582443275235822165900630749458501399763956224105847817023488
    4138716515700636502484016416273137975716040363780457577708511916105622
    5368969778704810338946823495350702998940644191159753551005271137191558
    7581168102080247666793164559467109431802509994964838 3347

e = 3228301342303266421169117315760265758438425860657333023075829597552395
    2120177964369118842752879663904076178029387241783669421944662057284703
    7204321660893945683995480478684517557221310812943679586038984510025856
    6530684300969970499328655564039032662829299392356057113051321831341669
    2574026510492441231954878718562589098243059768965849630160029940716192
    3797664999697852829012478549822125235360847482104396890849982225289128
    0270405656437751865360460356950820053427716165084853009640829990558003
    6267076209868738737431366551802857237898014323229893926067842780598583
    2954745888548225228129691496673973128433757400155631 3361
```