Cracking an RSA Cipher Text

July 16, 2004

Dhruv Maheshwari
Adam Shapiro
Andrew Bulthaupt
Andy Ogden
Norman Yao

Our Solution

To successfully decipher the given cipher text, given the values *e*, *N*, and encrypted message *b*, with no way of communicating with Freddy or Emily, we could implement only one viable non-dictionary based brute-force technique:

- Factor *N* into *P* and *Q*.
- Use these values to solve the equation $ex = 1 \bmod (P-1)(Q-1)$.
- Take the solution *d*, and decipher by performing $a = b^{ed} \bmod N$, where *a* is the resulting clear text.

Message 1:

$N = 76130621063195078551115667942958088 3$

$e = 23$

$b = 4939632821908643988200043210176807 06$

(Clear Text: 16050114212019)

(Translation: PEANUTS)

To factor *N* into its constituent prime factors *P* and *Q*, we used Maple's ifactor() function:

```
> ifactor(761306210631950785511156679429580883);
            (777332679307424393) (979382741647047931)
```

With these values, we proceeded to solve for *x* in $ex = 1 \bmod (P-1)(Q-1)$. This was also accomplished using Maple's msolve() function:

```
> msolve(23*x = 1, (777332679307424392)*(979382741647047930));
            {x = 529604320439617936524828701547901607}
```

With the value of *d*, we moved to the final step: computing $a = b^{ed} \bmod N$:

```
>  (493963282190864398820004321017680706 ^
529604320439617936524828701547901607) mod
761306210631950785511156679429580883;
Error, numeric exception: overflow
```

Interestingly enough, the values generated were beyond the capacity of Maple's data storage capabilities. Similar results were obtained with MatLab. Mathematica and Derive were not available to attempt generation of clear text. To compute such a large number, a different approach would be needed, which was addressed when we tackled Message 2…

Message 2:

$N = 1000000000000000000000000000000000803000000000000000000000000011773$

$e = 123456789$

$b = 7201982439767275938964208220144077999154093423879102903899915907244$

(Clear Text: 1609192001030809151 9)

(Translation: PISTACHIOS)

When we began our brute-force attack by **ifactor(N)**, it became immediately clear that Maple's algorithm(s) were too slow. MatLab immediately declared that the command *factor(N)* was invalid because 2^32 was the largest accepted factorable number. We decided to do some research on factoring techniques. Our search led us to this site. It provided a variety of factoring algorithms that could be utilized. However, they relied on very large integers, not supported (to our knowledge), by Maple or MatLab.

A search for implementation using "big integer" data types led us to a cryptography library, Miracl (Multiprecision Integer and Rational Arithmetic C/C++ Library). This provided big-integer utilities and implementations for the previously mentioned factoring techniques. The zipped library can be found here (v4.82). Using the *factor* utility, designed to factor numbers up to 80 digits in length, we were able to get the two prime factors of N: 1000000000000000000000000000000061 and 1000000000000000000000000000000193. A dump of the program execution can be found here.

Once again, msolve() was used:

```
> msolve(123456789*x = 1, (1000000000000000000000000000000061-1)*(1000000000000000000000000000000193-1));
   {x = 2024457723422565283145344076622791434317638052290506275843607112509
      }
```

Yet again, we were left at a point at which computation of the clear text was virtually impossible given the current resources.

We initially tried to use the big integer library provided by the Miracl library to compute the needed values, but our C++ compiler (MSVC++ .NET) was not recognizing the proper .lib files. We then tried to use a library called FreeLip, which was originally used to crack the RSA-129 system. However, we were not able to download this library.

Finally, we resorted to using the *BigInteger* class provided by Sun's Java v1.4.2. We wrote a program that took values of *N*, *b*, and *d* from a file, and used these values to compute the clear text, and "translate" it into English. A copy of the code for this program can be found here. The input files we used for messages 1 and 2 were:

Message 1 Input

Message 2 Input