# Rivest-Shamir-Adleman Algorithm (aka RSA Encryption)

The RSA algorithm, originally published in 1978, belongs to a class of algorithms called ‘public key encryption.’ A greatly simplified description of the algorithm is as follows: Bob wants to send a secure message to Alice. In order to do this, Alice must generate a private- and public-key pair. She does this by generating two *prime integers*, *p* and *q*, and then computes the *composite key* from:

Using *n*, Alice can then generate two other integers, *d* and *e* that are used for the algorithm itself. Alice gives Bob her *public* *key* pair consisting of two integers: (*n, e*). In fact, Alice can give her public key to anyone. When Bob wants to send her a message *M*, he uses Alice’s key and *modular exponentiation* to create the cryptographic message, *C*:

Later, when Alice receives the message, she can decrypt the message using her *private key* (*n, d*):

The strength of the RSA algorithm lies the in the “trap-door” function of the *modulus,* the size of *n,* and the fact that *d* is very hard to recover from *n* and *e* alone. As we will see in this lab, a small *n* (e.g. 32-bit) can be broken on a regular PC. Key sizes today are considerably larger, often 2048- or even 4096-bits.

## Attacking RSA

There is no known fast method for determining the private key, *d* from the public key (*n,e)*. In fact, the only known attack against RSA is called a *brute-force* attack, which tries every possible *d* in the range of 3 to *n*. Of course, with a 32-bit *n*, there are 4 billion keys to check. With a 4096-bit *n*, there are keys to check.

for d in 3 to by 2 do

if(M is a valid message) then  
 private\_key = d

If we could do a peta-key checks per second ( ), it would take seconds, or centuries. It would take more resources than exist in the universe (time and Joules) to break one 4096-bit key. Clearly, we aren’t going to try to brute-force such a large key. Even if we could, how would we know if the message was the right message?

There is one weak spot – if the messages are predictable, we can . In our project, all of Alice’s messages are HTML, and are wrapped with “<h1>*message*</h1>”. So, all we need to do is look for a *d* that gives back a string that matches this pattern… easy, right?

# The Project Challenge

The purpose of this project is not to make you learn too much about the RSA algorithm, but to explore making a single-threaded and slow program faster. I’ve posted a bunch of files to this site on D2L:

* rsa.c – My implementation of the RSA algorithm (don’t use for NSA work)
* rsa.h – The header file for my RSA algorithm
* find-keys.c – Brute-force code to find the keys
* make-test.c – Generates a set of keys and writes an encrypted file
* Makefile – a make-file to build all of the code
* public-*n*.txt – public keys for Alice, *32, 34, … 50* bits
* encrypted-*n*.txt – encrypted messages using *32*, *34, …* *50-*bit keys

Download the files and experiment. You’ll need to do a “make” to build the programs. They all build and run fine on WSL and UNIX. You may need to “apt-get install libgmp-dev”

There are many ways to make this project run faster. Some are simple: take a look at the Makefile and look around at the C code to see if there are some optimizations you can find. Some are algorithmic improvements. They can potentially improve results more than the threading techniques.

The main way to make this run faster is to look to use pthreads or you can fork and use shared memory. You can do this in many different ways, and the ways that you choose are up to you. For example, there is an embarrassingly easy way to multithread this, but it won’t be optimal. There are harder ways to multithread this which may be better but will take more planning. You can use multiple computers – but that’s going to take more effort to design. You can use CUDA or OpenCL but then you’ll really need to put some thought into what parts are done on the CPU and what is done on GPU. You can even do some research and look for better algorithms – while not instantaneous, there are some that will speed-up the search space.

You can even mix and match – you could use multiple computers with multiple threads with the GPUs. You get to decide.

You will get more points for larger keys broken. You will get more points for faster times on each of the keys – i.e. you’ll get more points if you get the 32-bits faster than any one else; you’d also get more points if you get greater key lengths than anyone else. I’ll personally be surprised if someone breaks 40-bit keys or higher.

If you’ve been looking for a reason to get your parents to sport you a new nVidia 3080 series card, now you can honestly tell them you need it for class 😊.

## Deliverables

The main deliverable for this project is a report that describes your methods and results. You should collect data from the baseline code, and then make incremental changes and describe if there was improvement and by how much. For example:

“We looked at the C code and decided that the variable names were too long. So, we went through and changed all of the variable names to single letters or single letters followed by a number. For example, the variable “encrypted\_text” was renamed “x32”. We re-ran the 32-bit key and found that the time was 325.7 seconds, and the speedup was 325.8/325.7 seconds = 1.000. We found it surprising that didn’t actually help since the variable names were shorter.“

If you successfully break a key, you will need to document what the clear-text message was and what the *d*-value that you found. You should also look to document important things like: how much time it took, what resources it took, etc.

Your grade will be determined by the effort you put into tackling this problem, the quality of your experimental approach, and your ability to document that experiment and results.

There is an old saw, “Great claims in science must be backed up by even greater evidence.” I reserve the right to make you demonstrate your results to me in person. If I suspect that results are fabricated (and its surprisingly easy to do) then we’ll have that discussion and that would end poorly for you.

It is very hard to give a quantitative way to judge your grade, but I can give the subjective evaluation as:

* An “A” looks like you tried a number of really innovative methods, you maybe pulled from the research community, and did not give up when it got difficult; you were able to break some of the higher-length keys, and your report does a great job documenting your project.
* A “B” looks like you tried a number of different methods, but your implementation was hampered by ability, and/or your report is generally good.
* A “C” looks like you tried some basic techniques, you wrote a decent report, but its nothing outstanding.
* A “D” looks like you tried to wrap a “pthread\_create” around my code and didn’t go beyond that. You documented things in your report but it wasn’t clear that you had a plan or took good notes along the way. Your report was not convincing.
* An “F” looks like you ran the sample code, maybe successfully. The best thing about your report was that it was brief.

If you can break keys > 64 bits, I’ll consider awarding some bonus points based on the level of difficulty that you applied to the problem. **But, to meet the standards for the course, you must use threads for at least part of the problem.**