Beyond Traditional RSA: How FLT Undoes Quantum Attacks

Peter Lablans | CEO/CTO at LabCyfer | Published December 11, 2024

Why RSA often is Intractable?

RSA encryption security traditionally relies on the computationally intractable problem of factoring a large number into two (preferably) large prime numbers. Successfully attacking RSA depends on successfully factoring the RSA number *n* that is applied in the computation of *m^e* mod-*n*. Once one knows the factors of *n*, further analysis of RSA is trivial. The intractable computational problem is created by the size of *n*. For smaller values of *n*, like *n*=3233, factoring is easy and breaking RSA is a given. The *n*=3233 example is explained on Wikipedia.

RSA is Self-Validating

RSA applications, such as RSA signatures, are self-validating. In general, first a hash *H(m)* of data *m* is determined. An RSA encryption is the computation of *rsa=H(m)^e mod-n*, with *n* being the product of 2 large primes, commonly named *p* and *q*. Large in that context means *p* and *q* being primes of at least about 256-bits long. But nowadays parameter sizes of 2048 bits are recommended. A validator receives data *m* and *rsa* and determines hash *H(m)* of data *m*. As a check, *rsa^d mod-n* is computed. In order for the data to be unmodified, *H(m)* and *rsa^d mod-n* MUST be identical. The reason this works is that *m^(e*d)* mod-*n* == *m* if nothing is changed or tampered with.

It is a bit of a disappointment that such a clever validation check is ruined, when an attacker is able to factor *n*. The security of RSA is in that sense determined by the size of *n*. Theoretically, no more than *n^2* operations are required to factor *n* and practically one may achieve that even faster. The larger the value of *n*, the more difficult to factor *n=p*q*.

A Key Weakness of RSA

A key weakness of RSA, so to speak, is that everyone knows how the exponentiation *H(m)^e mod-n* works as a repeat multiplication *modulo-n*. When factors *p* and *q* of *n* are known, all computations are exposed. But what if the operation multiplication *modulo-n* is replaced by an *n*-state operation *mgn* that generates different results compared to multiplication *modulo-n*? And how many usable modifications of multiplications *modulo-n* exist, if they exist at all?

The Finite Lab-Transform (FLT) in RSA

The surprising fact is that about *(n-1)!* (factorial of *n-1*) usable different variations exist. These variations are generated by the Finite Lab-Transform or FLT. The basics of the FLT are explained on https://www.labtransform.com/. The FLT is part of a more general Computational Function Transform or CFT. For *n*=3233, the factorial *3232!* is greater than *10^9941*. The functional transformation of multiplication by *FLT* offers greater security than the assumed or advertised security of RSA. And even recently announced breakthroughs like Google's Willow quantum processor, presumably performing in 5 minutes like a supercomputer operating for 10 septillion years, barely make a dent in these super-cosmological numbers.

Big Integer Examples

Usually, toy examples are used to illustrate RSA. And production size parameters in RSA are recommended to be 2048 bits at least. We contend that 256-bit parameters with FLT will already start to be secure. In order to go beyond "bragging" and example has been worked out.

We created a big integer (256-bit) RSA example that applies an FLT. The individual steps are explained both in unmodified and in FLT transformed form. Using the following parameters:

p= 533451976602736000578437975944140937711 -(128 bits)

q= 161513927571683845503290376838074744607 -(128 bits)

*n=p*q= 86159923911985887773281189680680215084225577177354637292937695406159820174 57* -(256 bits)

The unmodified RSA encryption using this size is definitely insecure. The FLTed version of this creates a keyword with a security of close to 256 bits. The applied FLT provides the desired variation over GF(256). The security of the generated keyword is determined by the security of a random 256 bit key against brute force attack (well roughly).

Anyway, all the steps required are a bit much for this short introduction. However, it illustrates the power of Computational Function Transformation (CFT), rather than relying strictly on parameter sizes such as

of keys, alone, as applied today.

A detailed description of the above example may be obtained from this article.

Contact Us

You may contact us at info@labcyfer.com for more detailed descriptions.