Preliminaries to Coppersmith's Algorithm

Coppersmith's algorithm relies on a simple flaw in the RSA algorithm when messages are small compared to the public number N. Consider a message x encrypted with exponent $e = 3$ using modulus N for the public key where $x < \sqrt[3]{N}$. Then the encryption z of x can be decrypted simply by taking the cube root, because the x^3 operation never rotated x over the modulus N.

This is a highly specific case, but it can be generalized to other cases, the most interesting being Coppersmith's short pad attack. In the short pad attack the message has the same conditions as above but also a simple padding P which is known to the code-breaker. When $e = 3$ the encryption can be considered forming the polynomial $(x+P)^3 = z$. Then Coppersmith's algorithm can be applied – this will solve the polynomial, reducing the case to the simple one above.

Also of note is the Franklin-Reiter related message attack. If the user sends related messages such that the related part can be considered equivalent to "padding", the same problem arises. (An example of this might be starting a set of messages with "The password is".) The discussion here will be restricted to the short pad attack, but the fact the related message attack exists will become important later (see open problem 3).

The Algorithm Itself

The basic idea here is to find a set of polynomials with the solution we need where the linear combination is small and therefore calculable.

Let us suppose we have a polynomial $(x + P)^3$ or

$$x^3 + 3Px^2 + 3P^2x + P^3.$$

Note that if $(x + P)^3 \equiv 0 \bmod N$ then $(x + P)^{3k} \equiv 0 \bmod N^k$. We need to form the following polynomials:

$$g_{u,v}(x) = N^{m-v}x^u f(x)^v$$

for some predefined m. Here m can be considered the "expansion factor" and will be precisely defined later, u will be defined as $0, ..., m$ and v will be defined as $0, ..., d - 1$.

For example, when $m = 3$, $u = 1$ and $v = 2$, the polynomial $g_{0,0}$ will be

$$N^{3-2}x^1 f(x)^2$$

or rather

$$N(x^7 + 6x^6P + 15x^5P^2 + 20x^4P^3 + 15x^3P^4 + 6x^2P^5 + xP^6).$$

We form a lattice considering all possibilities for u and v. The lattice is designed so the first column represents the constant coefficient, the second column represents the coefficient with x^1, the third the coefficient with x^2, and so on. In the example given the rows would be $g_{0,0}, g_{1,0}, g_{2,0}, g_{0,1}, g_{1,1}, g_{2,1}$, etc. Let $a = 3P$, $b = 3P^2$ and $c = P^3$ and form the matrix starting with:

$$\begin{array}{cccccccccccc} N^3 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & N^3 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & N^3 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ cN^2 & bN^2 & aN^2 & N^2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & cN^2 & bN^2 & aN^2 & N^2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & cN^2 & bN^2 & aN^2 & N^2 & 0 & 0 & 0 & 0 & 0 & 0 \end{array} \tag{1}$$

and so on until $u = 2$, $v = 3$.

If we can find the short vectors of the matrix we will be able to find vectors less than N^m and therefore the answer we are seeking. The LLL algorithm is able to do this. We will consider it a "black box" and not go into specifics here; essentially it performs Gaussian reduction in an iterated fashion to get approximations of short vectors. In this case the approximation is good enough to solve our problem.

More specifically, the LLL algorithm will be sufficient for decryption if $m = \log N/e$. (The reasons are technical and still need exploring, see open problem 5.) This is the expansion factor discussed earlier. Once the short vectors are generated solving the polynomials will yield the answer to the problem.

Note that all the results above can be generalized for any e. However, most prior tests have been done using only $e = 3$ and considering a "high-enough" e to be out of range without giving specifics.

I tried Various values of e and m of the algorithm in an implementation using Mathematica 4.1. The results are in the Appendix. From these and other concerns there arise open problems that will be resolved in the next phase of my research:

1. A paradox of sorts may arise when N is large: the algorithm will actually work faster when e is large than when e is small. Consider a 150-digit number N (approximately RSA-512). $\log N$ is approximately 343. When $e = 17$, we need $m = 21$ to solve the equation, but when $e = 3$, $m = 115$. Judging from the results in the Appendix the first may be faster (although the conditions are stricter, since the message a needs to be less than $\sqrt[17]{N}$

rather than $\sqrt[3]{N}$). A more robust implementation needs to be made (I plan to use C code) that will allow me to test higher values of N with my current resources.

2. There is a lattice algorithm by Schnorr and one by Schönage which both boast higher running times than the typically-used LLL. I will study these in more detail and look into the possibility of combining them (which Schnorr speculates is possible, but gives no details or algorithm).

3. Note that all the issues are typically moot in practical reality: a good method of padding – for example, marking the end of the message and using random characters for the rest – will make this attack useless. However, the implementer has no control over what the user does, so an artificial situation like the Franklin-Reiter attack could *conceivably* happen in real life. It may be possible to scramble messages in such a way that even the Franklin-Reiter attack is no worry: this should be investigated, and if true, paranoia about low exponents in RSA may be unjustified.

4. Related to 3, the pseudo-random algorithm to generate the padding can be investigated, but it should be noted if one could attack the random number generator one might as well attack the generation of p and q which also rely on it.

5. I still have yet to see the original proof of the expansion factor – every summary excludes it. This proof should be verified; as well, there may be practical conditions where the expansion factor can be lower, for instance when the message size is restricted even farther than the exponent allows.