Radium is a symmetric cryptosystem, inspired by its prequel Cyclone. With Cyclone, the $n \times n$ square matrix key functioned like a cylinder of stacked wheels. The key had to contain a permutation on each row. With Radium, however, the key must only contain each symbol from the alphabet at least once.

Instead of the composition of rows, as in Cyclone, the encoding function of Radium is "read off" a flattened version of the key matrix. This allows both the rows and the columns to be rotated, so the key is more intensely transformed with the processing of a single symbol. This leads to better results, at least in terms of the ratio of unique permutations to symbols processed as a function of key size. On the other hand, the algorithm is more complex.

Here's the encoding function:

```
function encode(p,q)
    k = copy(q)
    n = size(k)[begin]
    c = zeros(Int64,length(p))
    for i in eachindex(p)
        f = getf(k)
        g = getg(k)
        if isodd(i)
            h = composition(f,g)
        else
            h = composition(g,f)
        end
        c[i] = h[p[i]]
        f = circshift(f,p[i])
        g = circshift(g,c[i])
        for j in 1:n
            k[j,:] = circshift(k[j,:],f[j] )
        end
        for j in 1:n
            k[:,j] = circshift(k[:,j],g[j] )
        end
    end
    c
end
```

This is similar to Cyclone, but now there are two functions entwined, and it's important to note the crucial change in the **getf** and **getg** functions.

```
function getf(q)
    k = reshape(transpose(q), :, 1)
    #printvec(k,alph_numbers)
    f = zeros(Int64,size(q)[begin])
    i = 1
    for v in k
        if ! (v in f)
            f[i] = v
            i = i + 1
        end
    end
    f
end
function getg(q)
    k = reverse(reshape(transpose(q), :, 1))
    #printvec(k,alph_numbers)
    g = zeros(Int64,size(q)[begin])
    i = 1
    for v in k
        if ! (v in g)
            g[i] = v
            i = i + 1
        end
    end
    g
end
```

The decoding function is as expected.

```
function decode(c,q)
    k = copy(q)
    n = size(k)[begin]
    p = zeros(Int64,length(c))
    for i in eachindex(c)
        f = getf(k)
        g = getg(k)
        if isodd(i)
            h = composition(f,g)
        else
```

```
            h = composition(g,f)
        end
        hinv = inverse(h)
        p[i] = hinv[c[i]]
        f = circshift(f,p[i])
        g = circshift(g,c[i])
        for j in 1:n
            k[j,:] = circshift(k[j,:],f[j] )
        end
        for j in 1:n
            k[:,j] = circshift(k[:,j],g[j] )
        end
    end
    p
end
```

This is the essence of the algorithm, but, as in Cyclone, the encrypt function uses features like autospin and reversal to make an arbitrary number of rounds (hopefully) more effective.

When $n = 27$, the compositions f are almost always unique. So the key reliably wanders into a new state, meeting each symbol of plaintext with a fresh permutation. For $n \approx 10$, results are still excellent, typically just as perfect, which sets Radium apart from Cyclone in a definite way. (Things degrade slightly with $n = 9$.) It's also true that Radium has a significantly larger keyspace, since rows are not limited to permutations.

Is Radium better ? For me these systems are more like sculptures than tools, so it's not a matter of efficiency. I do take the constraint that the system be working and genuine seriously, though. So it's a bit like creating a tool that "must" be beautiful (if only to me.)

DISCLAIMER

I'm not a professional cryptographer. I just took some independent studies while in grad school. This is one of a few ideas that occurred to me at that time. If an actual cryptographer has an opinion about the security of this system, I'd be glad to hear it. To me it *seems* reasonably strong for $n = 27$, but I can't be sure.