Mar 20, 2016 at 1:27pm

[Quote](https://cicada3301.boards.net/post/24/quote/19)[like](https://cicada3301.boards.net/thread/19/checking-key-sequences-practical-guide)

Post Options

**Post by mortlach on Mar 20, 2016 at 1:27pm**

Introduction  
  
Keys generated from a crib for a particular algorithm should be checked against known sequences. This is necessary as page 56 was encoded with a stream cipher with the key as prime[n]-1 where n is the position of the rune in the text giving the n’th prime minus 1.  
  
  
**Example**Let’s assume we have a trial key generated from some encryption algorithm and crib (cf our trial key will \*always\* be mod 29 ):  
  
  
    *trialKey*= 22, 24, 13, 15, 8, 10, 22  
  
  
Our task is to find if *trialKey*is part of a regular, simple, sequence, e.g. the sequence of primes. We also want to be able to check against sequences with a constant offset (such as the prime[n]-1 from page 56). This is easily achieved using a similar idea to “*shift independent text analysis.*” We take the difference between successive elements in *trialKey*, *delta\_trailkey*and then modulus 29:  
  
 *delta\_trailkey\_mod29* = 2, 18, 2, 22, 2, 12  
  
  
*delta\_trailkey\_mod29* is the pattern to search for. Clearly, the sequences we want to search for the pattern *delta\_trailkey\_mod29*must also be the differences mod 29 of the original sequences. For example, here are the 1000’th to 1011’th primes, primes -1, the differences and the differences mod 29:  
  
  
                                              primes(n) = 7919, 7927, 7933, 7937, 7949, 7951, 7963, 7993, 8009, 8011, 8017  
  
                                           primes(n)-1 = 7918, 7926, 7932, 7936, 7948, 7950, 7962, 7992, 8008, 8010, 8016  
  
                         primes(n) - primes(n-1) = 8, 6, 4, 12, 2, 12, 30, 16, 2, 6  
  
                    primes(n) - primes(n-1) - 2 = 8, 6, 4, 12, 2, 12, 30, 16, 2, 6  
  
        (primes(n) - primes(n-1) ) mod 29  = 8, 6, 4, 12, 2, 12, 1, 16, 2, 6  
  
   (primes(n)-1) - primes(n-1)-1) mod 29  = 8, 6, 4, 12, 2, 12, 1, 16, 2, 6  
  
  
        where n=1000 to 1011  
  
  
Hopefully, it is clear how searching for the differences between elements allows all constant offsets of sequences to be checked simultaneously.  
  
  
**Practical Considerations: Some Implementation Ideas I use.**  
**Search strings not lists.**  
  
I map the elements of delta\_trailkey\_mod29 and all *delta\_sequence\_mod29* to a base 29 system (you could use the runes! But to avoid confusion I use the following (yes, I code in c++ :P  ):  
  
  
    const std::map<unsigned char, std::string> sequenceChecker::num29ToString {{0 ,"0"},{1 ,"1"},{2 ,"2"},{3 ,"3"},{4, "4"},{5, "5"},{6 ,"6"},{7 ,"7"},{8 ,"8"},{9 ,"9"},{10,"a"}, {11,"b"},{12,"c"},{13,"d"},{14,"e"},{15,"f"},{16,"g"},{17,"h"},{18,"i"},{19,"j"},{20,"k"},{21,"l"},{22,"m"},{23,"n"},{24,"o"},{25,"p"},{26,"q"},{27,"r"},{28,"s"}};  
  
    in this case *delta\_trailkey\_mod29* will become the string:  
 *delta\_trailkey\_mod29* = “2i2m2c”  
  
    and many library routines for searching and comparing strings become available. For example, I once found it particularly useful to search for patterns that contained wildcards without having to write my own implementation.  
  
Which Sequences to search?  
  
This is difficult. The number of potential sequences is practically infinite. Here is how I cut the parameter space.  
  
Each section of runes is no more than a few thousand characters, so each sequence I have generated only goes to its 4000’th element.

* Wikipedia has a list of different primes sequences, I have pre-computed many of these. [en.wikipedia.org/wiki/List\_of\_prime\_numbers](https://en.wikipedia.org/wiki/List_of_prime_numbers)
* Prime sequences generated from OEIS sequences that match “*message.asc*” and “*the planned parenthood message 2015*”
* The runes from the Liber Primus, cipher text and decrypted text.
* Every first character of each word, sentence and line.
* Prime factors of each rune-word-total
* Sequences are searched from start-to-finish and finish-to-start.
* Adding more lists is very easy to do…

Searching strings as described above can be made practically fast enough for all searches I have tried. However, one issue is that you may get many “*matches*” (due to the large number of sequences in the search-space). In this case I try and cut the number of matches by checking the position in the sequence. For example, still using:  
  
    *trialKey*= 22, 24, 13, 15, 8, 10, 22      *delta\_trailkey\_mod29* = 2, 18, 2, 22, 2, 12 =“2i2k2c”  
  
Gives the following matches from the Twin Primes Sequence, primes\_2, ordered first to last, where I give the positional indices in the sequence and the values:

|  |  |
| --- | --- |
| Positional Index | prime value |
| 800, 801, 802, 803, 804, 805 | 23831, 23833, 23909, 23911, 24107, 24109 |
| 942, 943, 944, 945, 946, 947 | 30389, 30391, 30467, 30469, 30491, 30493 |
| 2314, 2315, 2316, 2317, 2318, 2319 | 93809, 93811, 93887, 93889, 93911, 93913 |
| 3488, 3489, 3490, 3491, 3492, 3493 | 154079, 154081, 154157, 154159, 154181, 154183 |
| 3754, 3755, 3756, 3757, 3758, 3759 | 169241, 169243, 169319, 169321, 169691, 169693 |

Given these matches I would make a cut based on the position of the rune in cipher text (remember to consider positions from start-to-end and end-to-start!). Therefore, If the crib was close to the 800th rune, OR the 942’nd rune, OR the 2314’th rune etc. then this match would be worth investigating further.  
  
  
Summary

* Any *trialKey*can be checked against any pre-computed sequence simply and efficiently by calculating *delta\_trailkey\_mod29* and using the above method.
* The method naturally includes all constant offsets of the sequences
* Wildcard characters can be used in a *delta\_trailkey\_mod29*if necessary.
* The number of potential keys is still huge, but (I have found) can be managed in a reasonable and practical manner
* Comparing the positional index of any matches with the positional index of the cipher text that generated *trilaKey*is a simple way to reduce the number of hits