Number engines

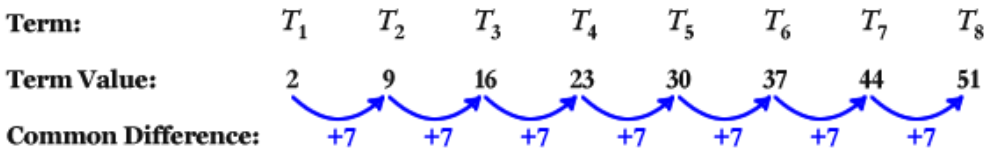
License management and software security are two different things. The concepts overlap, naturally, since the whole purpose of a license system is to enable or disable features based on the premise of bought access. But there are differences that are important to understand.

Ultimately license management is about just that: to manage licenses. Which in practical terms can be reduced to: being able to produce serial numbers that are non-linear yet can be validated against its origins.

This is where license management and cryptography differs greatly. Being able to produce a non-linear serial number has really very little to do with certificates, private and public keys, checksum validation and all the intricate aspects that software security entails.

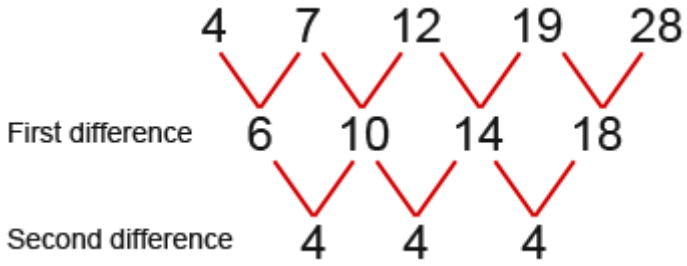
Linear vs. non linear

A linear number is a normal integer value that grows in a linear, forward fashion. Let' say you have a serial number that is "1001". Now that's a very bad serial number, because the first thing a hacker is going to do is to try "1002". Perhaps you have been clever about it and added 28 or 64 for each valid number — but a brute force password finder will cut through "schemes" like this in milliseconds. It's not even worthy of debate.



Linear numbers should never be used as the basis for a license management system

Non-linear numbers are more complex: first of all they consist of more parts. Not just a single value, but several values. Each of which must match or have a range of matches in order to validate. It is very hard to use traditional hacking tools to figure out a non-linear number, because you have to find out the valid number ranges for each part of the whole.



Non-linear means that more factors must match in order to produce a valid range. The more factors, the harder it is to reverse engineer

Also, if you don't know the number that was used to start with, the root key, then trying to reverse engineer it is indeed tricky.

One way of thinking about this is a bicycle lock: you know, those small code wheels you have to turn to input the code? Now non-linear number generators can be thought of as a collection of such wheels, but with 256 (0..255 = one byte) possible settings per wheel. Not just 8 or 10.

This gives you: 12^256 different combinations (roughly 1.8 billion). And out of those the cipher allocates a percentage or range that is deemed valid within the scope of each byte. HexLicense allows for a maximum of 64 out of 256 per segment to be valid, which gives you 4.7 million possible combinations. This is further reduced by the way numbers are grown. The gaps as a number advances non-linearly are invalid and unused, and the total they represent must further be subtracted from the total of possible combinations.

HexLicense has been tested with up to 100.000 generated serial numbers from a single root-key. This is a very low number compared to the possible combinations involved, but it also helps limit brute force attacks. Besides, minting a new root-key after having sold 100.000 copies of your program is a luxury problem, not a technical problem.

Number engines and matrixes

There really is no "proper" way when it comes to creating a number generator. You can't go to school and take a course in generating unique license strings. It all comes down to the formula and how you deploy it.

Another fact to remember is that number engines and random number generators is not the same thing. The latter being a field within mathematics and science; The first belonging more in the realm of natural mathematics, esotericism and number games. Despite this, the way scientific random number generators work is surprisingly low-tech until you enter the realm of physics.

But its important not to mix these topics up, because science is about availing something, while the other is about masking and hiding something.

The majority of engines that generate unique licenses today either use encryption to mask data, which is not what HexLicense is about; or they rely on alternative mathematics. Hexlicense uses quite ancient number sequences, like the Lucas numbers

(https://en.wikipedia.org/wiki/Lucas_number), the Fibonacci numbers (https://en.wikipedia.org/wiki/Fibonacci_number) and sepsephos (https://en.wikipedia.org/wiki/Isopsephy)gematria.

We are adding support for more formulas in order to bring a greater diversity to the available serial lists. As of writing the following ancient formulas have been added to the library:

- Beatus gematria matrix
- Hebrew gematria matrix
- Sepsephos greek gematria matrix
- Latin common gematria matrix
- Egyptian gematria matrix (*)

(*) A study of numbers (https://www.amazon.com/Study-Numbers-Constant-Creation-Universe/dp/0892811129/ref=sr_1_sc_1?ie=UTF8&qid=1475768207&sr=8-1-spell&keywords=A+study+of+numbers+R.A+schwaller+de+lublics), R.A Schwaller de lubics

I hope this little article has helped provide some clarity into the concepts involved.