Encryption using cellular automata chain-rules

Andrew Wuensche[1,2]

[1] Discrete Dynamics Lab. (www.ddlab.org)
[2] University of Sussex, United Kingdom

Abstract. Chain rules are maximally chaotic CA rules that can be constructed at random to provide a huge number of encryption keys — where the the CA is run backwards to encrypt, forwards to decrypt. The methods are based on the reverse algorithm and the Z-parameter [5].

1 The CA reverse algorithm and basins of attraction

In the simplest cellular automata [4], each cell in a ring of cells updates its value (0,1) as a function of the values of its k neighbours. All cells update synchronously — in parallel, in discrete time-steps, moving through a deterministic forward trajectory. Each "state" of the ring, a bit string, has one successor, but may have multiple or zero predecessors.

A book, "The Global Dynamics of Cellular Automata" [5] published in 1992 introduced a reverse algorithm for finding the pre-images (predecessors) of states for any finite 1d binary CA with periodic boundary conditions, which made it possible to reveal the precise topology of "basins of attraction" for the first time — represented by state transition graphs — states linked into trees rooted on attractor cycles, which could be drawn automatically, as in Fig. 1. The software was attached to the book on a floppy disk — the origin of what later became DDLab [12].

As state-space necessarily includes every possible piece of information encoded within the size of its string, including excerpts from Shakespeare, copies of the Mona Lisa, and one's own thumb print, and given that each unique string is linked somewhere within the graph according to a dynamical rule, this immediately suggested that a string with some relevant information could be recovered from another string linked to it in some remote location in the graph, for example by running backwards from string A (the information) to arrive after a number of time steps at string B (the encryption), then running forwards from B back to A to decrypt (or the method could be reversed) — so here was a new approach to encryption where the rule is the encryption key.

Gutowitz patented analogous methods using dynamical systems, CA in particular [2], but these are different from the methods I will describe, where its crucial to distinguish a type of CA rule were the graph linking state-space has the appropriate topology to allows efficient encryption/decryption.

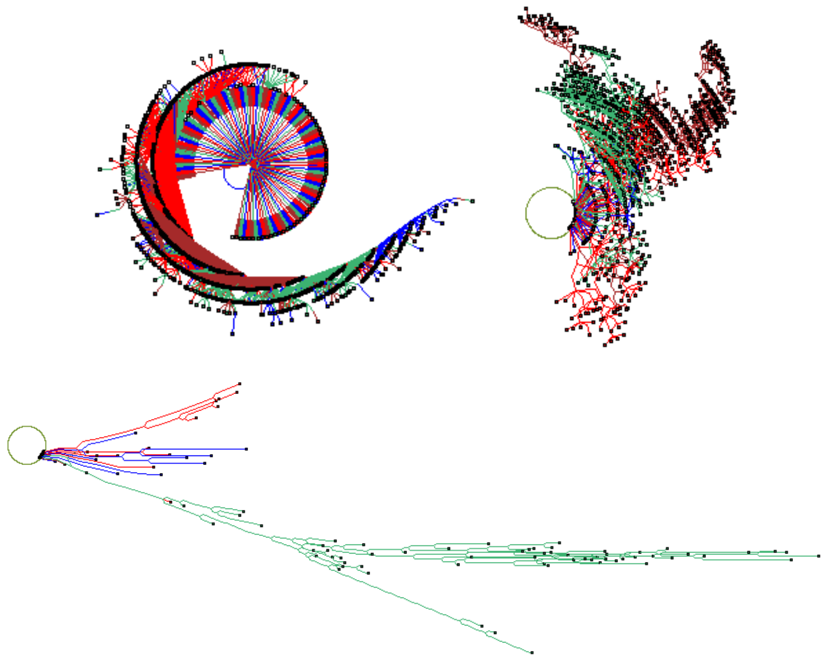
Fig. 1. Three basins of attraction with contrasting topology, n=15, k=3. The direction of time flows inward towards the attractor, then clockwise. One complete set of equivalent trees is shown in each case, and just the Garden-of-Eden (leaf) states are shown as nodes. Data for each is provided as follows: attractor period=p, volume=v, leaf density=d, longest transient=t, max in-degree=P_{max}.
topleft: rule 250, Z_{left}=0.5, Z_{right}=0.5, too convergent for encryption, p=1, v=32767, d=0.859, t=14, P_{max}=1364,.
topright: rule 110, Z_{left}=0.75, Z_{right}=0.625, too convergent for encryption, p=295, v=10885, d=0.55, t=39, P_{max}=30,.
bottom: rule 30, a chain rule, Z_{left}=0.5, Z_{right}=1, OK for encryption, p=1455, v=30375, d=0.042, t=321, P_{max}=2,

2 The Z-parameter

Many fascinating insights emerged during the genesis of [5], among them a refinement of Langton's λ-parameter [3]. This "Z-parameter" arose directly from the reverse algorithm, which computed the next unknown bit (say, from left to the right, giving Z_{left}) of a partly known pre-image, or conversely from right to left. The Z-parameter, by analysing just the lookup-table (the CA rule) gave the probability that this next bit was uniquely determined, and being a probability the value of Z_{left} ranged between 0 and 1. The converse direction gave Z_{right}, and the greater of the two was deemed to be the final Z-parameter.

Z did a good job of predicting the bushiness or branchiness of subtrees in basins of attraction — their typical in-degree (or degree of pre-imaging), which related to the density of end states without pre-images (Garden-of-Eden states) but lets call them "leaves" for short. A branchier tree (low Z) has more leaves, and shorter branches (transients) because all those pre-images and leaves use up a finite state-space. Conversely, sparsely branching trees (high Z) have fewer leaves, and longer branches.

Figure 1 gives three examples with contrasting topology.

Low Z, a low probability that the next bit was determined, meant the next bit would probably be both 0 and 1, equally valid, so more pre-images and branchiness, or that there was no valid next bit, more leaves. High Z, a high probability that the next bit was determined, meant it would probably be either 0 or 1, not both, so fewer pre-images, less branchiness, but more chance of a valid pre-image, so fewer leaves.

This nicely tied in with the behaviour of CA when run forward. Low Z, high branchiness results in ordered dynamics. High Z, low branchiness, results in disordered dynamics (chaos), behaviour that could be recognised subjectively, but also by various objective measures, in particular "input entropy" [6, 7]. The entropy stabilises for both order (at a low level) and chaos (at a high level); entropy that does not stabilise but exhibits variance over time is a signature of complexity, with intermediate Z.

3 Limited pre-image rules

Rules in general, even with high values of $Z<1$, can generate huge numbers of pre-images from typical states, which would slow down the reverse algorithm. A branchy (convergent) graph topology has many leaves. As strings become larger, the leaf density increases taking up almost all of state-space, as in Fig. 2 (rule 250).

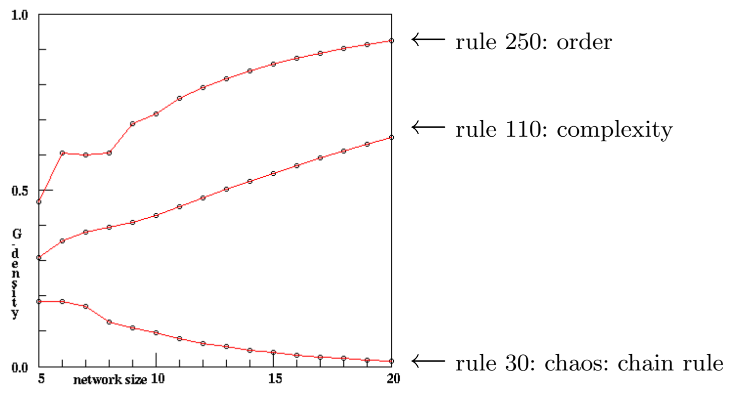

Fig. 2. A plot of leaf (Garden-of-Eden) density with increasing system size, n= 5 to 20, for the three rules in Fig. 1. The measures are for the basin of attraction field, so the entire state-space. For rule-space in general, leaf density increases with greater n, but for chain rules leaf density decreases.

These leaf states cannot be encrypted by running backward. The alternative, running forwards to encrypt, then backwards to decrypt, poses the problem of selecting the correct path out of a multitude of pre-image branches at each backward time-step. Running forward continually loses information on where you came from; CA are dissipative dynamical systems.

But there is an solution! When Z=1, however large the size of the lattice, the number of pre-images of any state is strictly limited. These Z=1 rules come in two types [5] as follows, (note that k is the effective-k, some rules have redundant inputs [5]),

– "two-way limited pre-image rules": Z_{left} and Z_{right} both equal one, where the in-degree must be either 2^{k-1} or zero,
– "one-way limited pre-image rules": Z_{left}=1 or Z_{right}=1, but not both, where the in-degree must be less than 2^{k-1}.

Limited pre-imaging (in-degree) appears to produce a promising topology on which to implement encryption, because we would usually need a long string to encode information, but the "two-way" Z=1 rules still suffer from some of the same problems as rules in general, too branchy and a high proportion of leaves.

"One-way" Z=1 rules on the other hand, seem to provide the ideal graph topology. They have an unexpected and not fully understood property: that although the maximum number of pre-images (P_{max}) of a state must be less than 2^{k-1}, experiment shows that the actual number is usually much less, and decreases as the system size increases; consequently the leaf-density also decreases as in Fig. 2 ($rule30$).

For large strings of 1000+, P_{max}=1, except for very rare states where the in-degree=2 in transients, or where transients joint the attractor cycle, which may be extremely long. However, this branching is not a problem when running forward to decrypt, because forward paths converge, and the original pattern will be found. Because the vast majority of state-space occurs in long chains, I renamed the "one-way limited pre-image rules" — "chain-rules".

In Figs. 3 and 4, where k=7, P_{max} must be less than 2^6=64, but the basin of attraction field (for a chain rule constructed at random) has P_{max}=5 and usually less, as shown in the in-degree histogram Figs. 3 ($right$), though there is a small basin where P_{max}=18.

As n increases P_{max} decreases. In Fig. 5 where n=400, P_{max}=2, but 97% of states have an in-degree of one. As n increases further, in-degrees of 2, and leaves, become exceedingly rare, becoming vanishingly small in the limit.

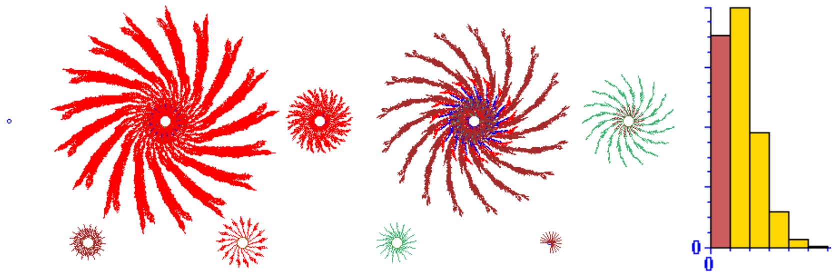

Fig. 3. $\underline{left}$: The basin of attraction field of a chain rule, showing all 9 basins of attraction (state transition graphs) for the k=7 rule (hex)879ac92e2b44774b786536d1d4bb88b41d. Note there is a tiny attractor (top left) consisting of just one state, all-0s; the last basin (bottom right) has the all-1s point attractor. The chain rule (Z_{left}=0.59, Z_{right}=1) was constructed at random. The string length n=17, state-space=2^{17}=131072, leaf density=0.345, P_m for each basin of the 9 basins is [1,5,5,5,3,4,4,3,18].
$\underline{right}$: The in-degree histogram, from 0 to 5, showing in-degree=1 as the most abundant.

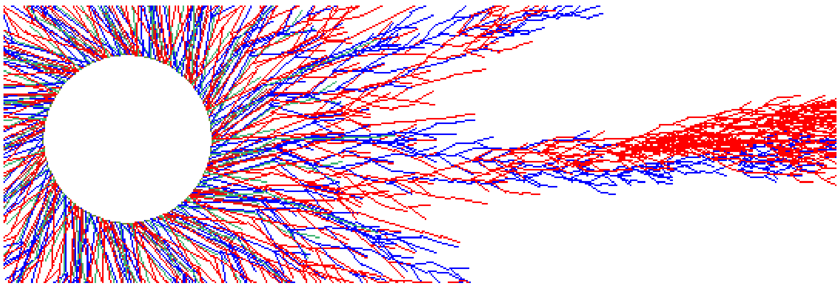

Fig. 4. A detail of the largest basin in Fig. 3, attractor period=357, basin volume=91868 70.1% of state-space, leaf density=0.345, max levels=119, P_m=5.

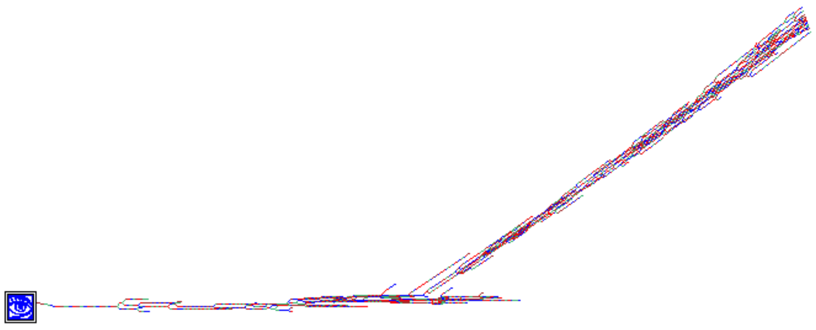

Fig. 5. The subtree of a chain rule, n=400, where the root state is shown in 2d (20×20), with the same chain rule as in Figs. 3 and 4. Backwards iteration was stopped after 400 time-steps. The subtree has 3346 states including the root state. There are 109 leaves (leaf density = 0.0326). P_{max}=2 and the density of these branching states is 0.035.

4 Constructing chain rules at random

The procedure for finding the Z-parameter [5–7] (its first approximation) from a rule-table (lookup-table) is as follows: Consider pairs of neighbourhoods that differ only by their last right bit, so the k-1 bits on the left are the same. Then look at the outputs of these pairs of neighbourhoods in the rule-table to see if they are the same (00 or 11) or different (01 or 10). Z_{left} is the fraction of *different* pairs in the look-up table. If all the pairs are different then Z_{left}=1. Z_{right} is given by the converse procedure.

There are refinements if effective-k in parts of the rule-table is less than k, but we will not bother with that because the pairs procedure gives the most chaotic dynamics.

To assign a chain rule at random, first pick Z_{left} or Z_{right} at random, then randomly assign different pairs of outputs (01 or 10) to the pairs of neighbourhoods defined above. Check the Z-parameter (with the full algorithm). If Z_{left}=1 or Z_{right}=1 but not both, we have a chain rule.

From experiment, the lesser value should not be too low, 0.5 or more [9]. This is to avoid a gradual lead-in and lead-out of structure in the space-time pattern when decrypting. Ideally the message, picture, or information, should pop out suddenly from a stream of chaos then rescramble quickly back into chaos, as in Fig. 7. This is accompanied by a lowering blip in the high input-entropy, the duration of the blip needs to be minimised to best "hide" the message.

DDLab [12] can assign a chain at random as above, instantaneously, with a key press, see the DDLab manual [9], section 16.7 and elsewhere.

5 How many chain-rules?

How many chain-rules, C, are there in a rule-space S = 2^{2^k}?

The number of ways (say Z_{left}=1) "pairs" can be assigned is $2^{2^{k-1}} = \sqrt[2]{S}$. Adding the same for Z_{left}=1, the number of chain rules C=2($2^{2^{k-1}}$), but we must subtract the number of rules where both Z_{left}=1 and Z_{right}=1, which is about $2^{2^{k-2}}$, because "pairs" need to be assigned to half of the rule-table, and their compliment to the other half. Subtracting also cases where the lesser value is less than 0.5, a round estimate for the number of acceptable chain rules is $2^{2^{k-1}}$, or the square root of rule-space.

This is sufficiently large to provide an inexhaustible supply of keys, for example for k=5: 2^{16}, k=6: 2^{32}, k=7 :2^{64}, k=8: 2^{128} etc. A chain-rule constructed randomly in DDLab will very probably be a unique key.

6 Encryption/decryption with chain rules

The CA reverse algorithm is especially efficient for chain rules, because the rules-tables are composed purely of "deterministic permutations" — they lack the "ambiguous permutations" that can slow down the reverse algorithm [5].

Many experiments have confirmed that chain rules make basin of attraction topologies that have the necessary properties for encryption. Nearly all states have predecessors and are embedded deeply within long chain-like chaotic transients.

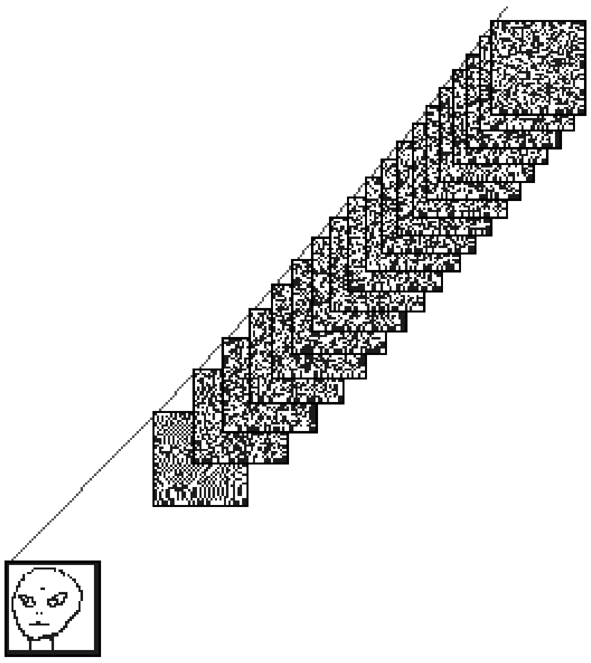

Fig. 6. A 1d pattern is displayed in 2d (n=1600, 40×40); the "alien" seed was drawn with the drawing function in DDLab. The seed could also be an ASCII file, or any other form of information. With a k=7 chain rule constructed at random, and the alien as the root state, a subtree was generated with the CA reverse algorithm; note that the subtree did not branch, and branching is highly unlikely to occur. The subtree was set to stop after 20 backward time-steps. The state reached is the encryption. This figure was taken from [8, 10].

There will still be leaves, and states close to the leaves, patterns that could not be encrypted by that particular chain rule because a backwards trajectory would just stop prematurely. However, for big binary systems, like 1600 as in Figs. 6 and 7, the state-space is so huge, 2^{1600}, that to stumble on an unencryptable state would be very unlikely, but if it were to happen, simply construct a different chain rule.

Encryption/decryption has been available in DDLab since about 1998. To encrypt, select a chain rule (and save it). Select a large enough 1d lattice (which can be shown in 2d). Select the information to be encrypted by loading an ASCII file (for text), or a DDLab image file as the seed, or create a picture with DDLab's drawing function. Select a subtree, and

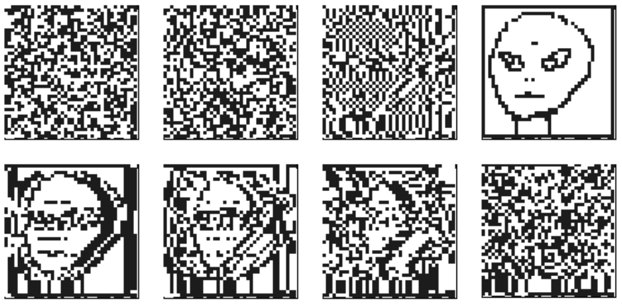

Fig. 7. To decrypt, starting from the encrypted state in Fig. 6 (n=1600, 40×40), the CA with the same rule was run forward by 20 time steps, the same number that was run backward, to recover the original image or bit-string. This figure shows time steps 17 to 25 to illustrate how the "alien" image was scrambled both before and after time step 20. This figure was taken from [8, 10].

set it to stop after say 20 backwards time-steps. The subtree is unlikely to branch, but if it does, no worries. Save the (encrypted) state reached. Fig. 6 and 8 show examples.

To decrypt, reset DDLab to run forward. Keep the same rule or load it. Load the encrypted state as the initial state. If decrypting a picture, set the presentation for 2d. Run forward, which can be done with successive key-press to see each time-step at leisure. At the 20th time-step, the image will pop out of the chaos. To fully observe the transition, continue stepping forward until the pattern returns to chaos. There will be some degree of ordering/disordering on either side, as in Figs. 7 and 9.

7 Generalising the methods to multi-value

In 2003, all functions and algorithms in DDLab were generalised from binary, v=2 (0,1), to multi-value. The "value-range" v (number of colours) can be set from 2 to 8, i.e. v=3 (0,1,2), v=4 (0,1,2,3), up to v=8 (0,1,2,..7). This included the reverse algorithm, the Z-parameter, chain-rules, and encryption. Details of the new algorithms will be written up at a later date.

Any available value-range can be used for encryption, but for efficiency's sake, not to waste bits, v=2, v=4 or v=8 are preferable.

The examples in Figs. 8 and 9 shows the encryption of a portrait, with v=8, k=4, on a 88×88 lattice (n=7744), but as v=8, the size of

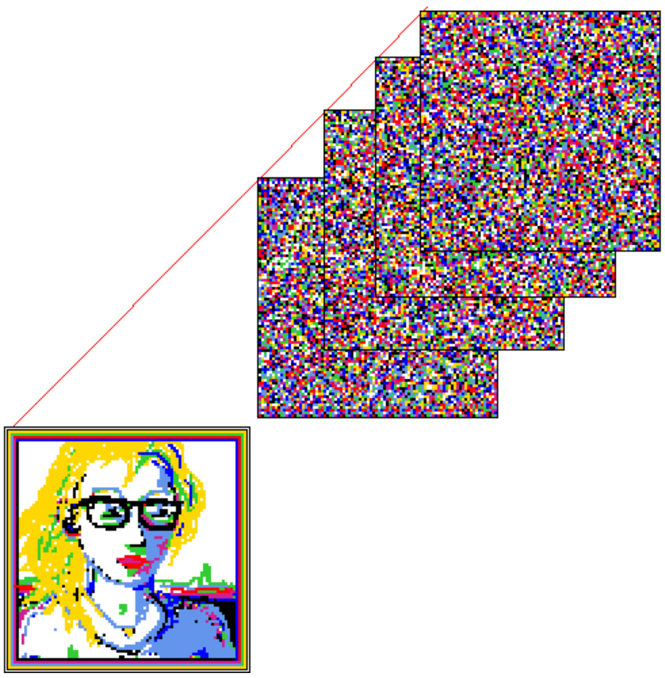
Fig. 8. A 1D pattern is displayed in 2*2* (n=7744, 88×88). The “portrait” was drawn with the drawing function in DDLab. With a v=8, k=4 chain rule constructed at random, and the portrait as the root state, a subtree was generated with the CA reverse algorithm. The subtree was set to stop after 5 backward time-steps. The state reached is the encryption.

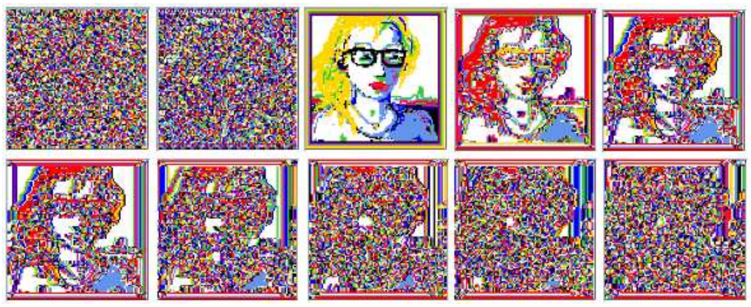
Fig. 9. To decrypt, starting from the encrypted state in Fig. 8 (n=7744, 88×88), the CA with the same rule was run forward to recover the original image. This figure shows time steps -2 to +7.

the binary string encoding the lattice is 61052. The v=8 chain rule was constructed at random; Z_{left}=0.4, and Z_{right}=1. When decrypting, the portrait pops out suddenly from chaos, but it takes about 50 time-steps to fully restore chaos. This is because k=4 is a small neighbourhood, and the chaotic pattern moves into areas of order at its "speed of light", set by k.

8 Possible drawbacks

Here are some possible drawbacks of the encryption method.

Chain rules usually result in attractor cycles with very long periods, though this is relative — the fraction of state-space made up of attractor cycles is probably small. If a state to be encrypted happens to be on an attractor cycle, running it backward may arrive at a point where a transient joins the attractor. In this case the backwards trajectory will branch.

Note also that there is an effect called "rotational symmetry" (and also "bilateral symmetry") that is inescapable in classical CA, where states with greater symmetry must be downstream of states with lesser symmetry, or with none [5]. This means that the uniform states, all-0s and all-1s, must be downstream of all other states in the dynamics, and the states like 010101.. downstream of states like 00110011.., which are downstream of the rest, etc. However, these highly ordered states hold little or no information, so are irrelevant for encryption.

A consequent effect is that "rotational symmetry" must stay constant in an attractor cycle, so in binary systems each uniform state must be one of the following: a point attractor; a transient state leading directly to the other's point attractor; part of a 2-state attractor with the other. In multi-value networks things get more complicated.

The key itself (the chain-rule) must be transmitted somehow — and with perfect accuracy. Noise in transmission of the encryption will spread during decryption, but only at the "speed of light" depending on k and the number of forward time-steps, so the process is relatively robust to this extent.

9 A bit of history

The encryption method described in this paper relies on a number of ideas first published in 1992 [5] and developed from work started in about 1988. The reverse algorithm for running 1d CA backwards made it possible to invent the Z-parameter, to reveal the topology of basins of attraction, and begin to study and understand these objects - how they relate to rule-space.

The "limited pre-image rules" in [5] were renamed "chain-rules" for brevity in about 1998, but the principle and possibility of encrypting by running CA backward, decrypting by running forward, using "limited pre-image rules", was well know to the authors of [5] at the time of writing. The method was made to work within DDLab in about 1998.

I've often described this encryption method in talks and lectures, including live demos of encryption with DDLab in many venues in a number of countries. The first time for a big audience was at the SFI summer school in Santa Fe in 1999. Up till now I have written only brief accounts specifically on this encryption method, both published [9, 11] and unpublished [8, 10].

I'm relating this bit of history because I have just now read a new paper by Gina M. B. Oliviera and others [1], sent to me for review, presenting their independent discovery of the same encryption method; except they used a genetic algorithm to evolve the rule-tables (instead of constructing them) to produce the Z-parameter property: Z_{left}=1 or Z_{right}=1 but not both (with some refinments) — the exact definition of chain rules.

Other than spurring me on to write this paper (for which I am most grateful, and also for their citations) I must say that as far as the science goes, I have not been influenced by their paper in any way.

10 Conclusions

I have described a method of encryption where the key is a chain-rule, a special type of maximally chaotic 1d CA rule.

Information is encrypted by using a key to run the CA backward in time. A secret message can be transmitted openly. The receiver has the same key, and uses it to decipher the message by running the CA forward in time by an equal number of steps. Anyone could construct their own unique key instantaneously from a virtually unlimited source — its size is about the square root of rule-space.

What is important to someone trying to crack an intercepted encrypted message, with DDLab available? The key itself is vital; data on the CA, its neighbourhood k, and value-range v, is important — not obvious from the key itself. The number of time-steps are useful, to know when the forward run should stop.

Suppose both the raw message and the encryption where known, could the key be deduced? I do not see how if the two are separated by a number of time-steps, without knowing the intervening steps.

In other security measures, the key itself could be encrypted. A message could be doubly or multiply encrypted with more than one key.

Although these methods are available in DDLab, dedicated software and hardware could be developed for the whole procedure to be fast, hidden and automatic, and also to handle data streams in real time.

References

[Note] For publications by A.Wuensche refer to www.cogs.susx.ac.uk/users/andywu/publications.html where most are available for download.

1. Oliveira, G.M.B, H.B. Macódo, A.B.A. Branquinho, and M.J.L. Lima., A cryptographic model based on the pre-image calculus of cellular automata, to appear in the proceeding Automata-2008.
2. Gutowitz, H., Method and Apparatus for Encryption, Decryption, and Authentication using Dynamical Systems, U.S. patent application (1992)
3. Langton, C.G., Computation at the edge of chaos: Phase transitions and emergent computation", Physica D, 42, 12-37, 1990.
4. Wolfram, S., Statistical mechanics of cellular automata, Reviews of Modern Physics, vol 55, 601–644, 1983.
5. Wuensche, A., and M.J. Lesser. The Global Dynamics of Cellular Automata; An Atlas of Basin of Attraction Fields of One-Dimensional Cellular Automata, Santa Fe Institute Studies in the Sciences of Complexity, Addison-Wesley, Reading, MA, 1992.
6. Wuensche, A., Complexity in 1D cellular automata; Gliders, basins of attraction and the Z parameter, Santa Fe Institute Working Paper 94-04-025, 1994.
7. Wuensche, A., Classifying cellular automata automatically; finding gliders, filtering, and relating space-time patterns, attractor basins, and the Z parameter, Complexity, Vol.4/no.3, 47–66, 1999.
8. Wuensche, A., Encryption using Cellular Automata Chain-rules, Examples implemented in DDLab, unpublished, 2000.
9. Wuensche, A., The DDLab Manual (for ddlabx24), section 16.7 Setting a chain rule, www.ddlab.org, 2001.
10. Wuensche, A., Encryption using Cellular Automata Chain-rules, unpublished, 2004.
11. Wuensche, A.,Discrete Dynamics Lab: Tools for investigating cellular automata and discrete dynamical networks, updated for multi-value, Section 23 Chain rules and encryption, in Artificial Life Models in Software, eds. A.Adamatzky and M.Komosinski, chapter 11, 263-297, Springer. 2005
12. Wuensche, A., Discrete Dynamics Lab (DDLab), software for investigating discrete dynamical networks, www.ddlab.org (1994-2006)