Sentinel Security Corporation

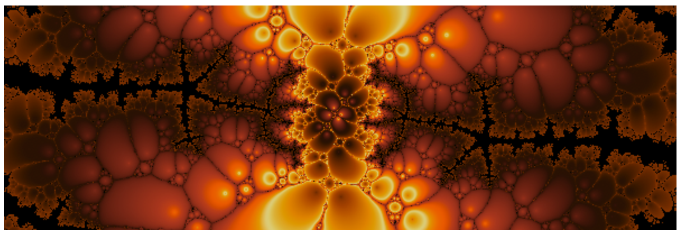
Applied biomorphics in HYDRA

HYDRA is an innovative network security platform from Sentinel Security Corporation. HYDRA uses a patented system of biomorphic mathematics to provides a level of security which no traditional system can match. This paper will demystify the math behind this powerful technology, as well as describe HYDRA's implementation and applications of the algorithms, such as session ID protection, software diversity, Web content checking, and more.

Eric Ridvan Üner

History of Chaotic Dynamics

Biomorphics is a subset of a larger field of study called “Chaotic Dynamics.” Credit for beginning the study of chaotic dynamics is often given to French physicist Henri Poincaré, who published the first glimpses into the science in the late 1800’s. A surge of discoveries by physicists around the world in the early 1970’s has fueled the widespread interest in fractals, strange attractors, and chaos theory in general. At least 2,000 books are currently published on the subject, and applications of the math have propagated from the Graphic Arts (where the fractal has become a design cliche) to Meteorology, and even many military applications too sensitive to discuss in this document.



The most popular of the applications, Fractal Geometry, is now familiar to most people. The images, such as the Maldelbrot set shown above, seem to resemble the growth or movement of a living creature, or some other natural phenomenon related to life. This resemblance led scientists to invent the description “biomorphic mathematics,” which expands on the word “biomorphics,” which describes the art and science of making artificial creations resemble natural ones.

Chaos In A Nutshell

Aside from the aesthetic aspects, the jury is still out on a concrete definition for chaos or what it means for a system to be chaotic. I myself have been involved in more than one very fueled debate on the subject. Having said that, the most popular definition is one given by the afore mentioned Henri Poincaré:

“It may happen that small differences in the initial conditions produce very great ones in the final phenomena. A small error in the former will produce an enormous error in the latter. ***Prediction becomes impossible****.”*

This is to say that systems, equations, or algorithms which produce large changes in output from small changes in input are, by definition, chaotic. By now, you are considering all the elements of your life that are chaotic - traffic to work, random computer crashes, mood swings (someone else’s of course). True chaos, however, requires a mathematical proof to determine its status as a chaotic system.

All About Entropy

Part of that mathematical proof involves a concept known as “entropy.” Entropy is the measure of the disorder or randomness in a system. The more entropy a system has, the more difficult it is to predict, or more to the point, to backward calculate the initial input.

By way of example, consider the difference in entropy between your PIN for your bank card and your password on your computer. The automated bank machines typically only allow a four digit number. A four digit PIN has 10,000 possible values, while a seven character password can have nearly 27 million possible values. The password, however, may not actually have any more entropy. It’s not the complexity of the result we are interested in when talking about entropy, but the randomness that went into generating the result.

In choosing a password, many users will choose a dictionary word, or some modified version of an aspect of their personal life. For example, I might chose a simple combination of my favorite number, and the name of my amazingly beautiful and fabulous wife. The result could be something like:

$$f_T(\text{"Linda"}, \text{"5"}) = \text{"L1nd405"}$$

Given that I can only easily remember a few hundred personal facts and figures without some kind of reminder to form such a password, a randomly assigned four digit PIN with 10,000 options may actually have more entropy than this weak password. That would depend greatly, however, on the algorithm used to select my PIN.

Pseudo-Random Numbers

To get a random number out of a computer, you need to simulate a random process. This is because computers are deterministic machines. Given the same inputs under the same conditions, they will always produce the same associated outputs. The most popular way of simulating randomness is to use an algorithm that is complex enough to resist basic analysis. However, because these algorithms are still deterministic, they only appear to be random, and so the results are called “pseudo-random.”

One common source of pseudo- randomness is a numerical approximation of the value of π. Because one digit has little discernible relationship to the next digit, the sequence appears random. There is no shortage of techniques to get the values. One popular method is to use a formula from John Machin, a professor of Astronomy at Gresham College, London back in the early 1700's:

$$\frac{\pi}{4} = 4\arctan\frac{1}{5} - \arctan\frac{1}{239}$$

To approximate the arctangent, we can use the well-known Taylor series:

$$\arctan(x) = (x - \frac{x^3}{3} + \frac{x^5}{5} + \frac{x^7}{7} + \frac{x^9}{9}\ldots) \text{ where } (-1 < x < 1)$$

Obviously this is a hefty task for a even a powerful computer. Even using more efficient approximations, it can take an average PC several minutes to calculate 250,000 PINs. That doesn't sound too bad for a bank, but keep in mind that pseudo-random numbers form the basis for cryptographic seeds used for algorithms that provide security in the form of ciphers (e.g. those used by your Web browser for SSL), digital signatures, and more. So for a system like a Web server that might need thousands of such numbers a second, these calculations become a processing bottleneck.

To make it worse, certain systems may have requirements for even more complex algorithms. Some U.S. Federal government systems, for example, must adhere to the Federal Information Processing Standard (FIPS). FIPS requires algorithms so computationally expensive that an entire market of specialized co-processors sprung up to offload the calculations from a system's main processor.

True Randomness Can Be False Security

Given the disadvantages intense computing requirements and predictability of pseudo-random number generation, many cryptographic applications have turned to sources of true random data, such as radioactive noise from space or small temperature variations inside microchips. This solution, however, is plagued by two shortcomings:

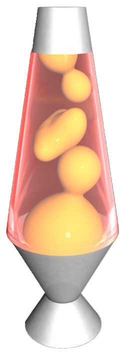
1. Since the data is non-deterministic (meaning you can not repeat a test and get the same results twice) you can not prove the system's entropy mathematically. One run may be very random, and the next very predictable.

2. The source of entropy can often be manipulated.

Manipulation of the data or the source of entropy is often the killing blow. Implementers have always looked, without success, for a source of random data that could not be manipulated - with some creative attempts.

An Interesting Anecdote: Lava Lamp vs. The World

In 1996 a rather creative individual discovered that by coupling a digital imaging system to a lava lamp, he could generate true random numbers. I took the concept to the next level, introducing a system later that year that used satellite imaging of the clouds over the Western Hemisphere, producing even more random data, albeit not on clear days. My system never caught on, probably because it's not nearly as funny.



Neither system provided a true solution. My satellite data, for example, can be forged. Along a similar vein, one distribution of Linux used the time between the arrival of network packets as a source of randomness. This solution was short-lived, as hackers discovered they could manipulate the timing by flooding the server with network traffic. Cyber-thieves calculated values of the resulting "random" data, potentially allowing them to steal network sessions and decipher encrypted messages with ease.

Biomorphic Sets

We have seen that hardware pseudo-random number generators are often impractical or inadequate, and software based solutions are often ineffective or computationally expensive; so is there a solution? Enter biomorphic mathematics.

In his book "Fractal Geometry of Nature" published in 1982, Benoit B. Mandelbrot suggests that the answer to nature may lie in mathematics. Mandelbrot introduced a simple equation:

$$z = z^2 + c$$

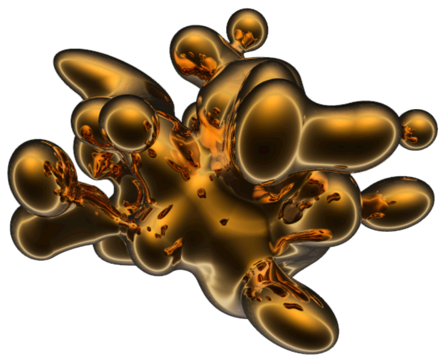
The equation, when combined with an iterative algorithm (meaning an equation that depends on feedback into itself), produces the familiar fractal images now known as the Mandlebrot set. Shortly before Mandlebrot's discoveries, a Meteorologist by the name of Edward Lorenz began using computers to predict the weather. Lorenz found that very small truncation or rounding errors in his algorithms produced large changes in the resulting predictions. This led to the study of "strange" or chaotic attractors. A Lorenz system can be described as:

$$\begin{aligned} x_1 &= \delta(x_2 - x_1) \\ x_2 &= rx_1 - x_2 - x_1x_3 \\ x_3 &= x_1x_2 - bx_3 \end{aligned}$$

Where δ is a constant. The system exhibits chaotic behavior under the following conditions:

$$r \quad \leq \quad \frac{\delta(\delta+b+3)}{\delta-b-1}$$

Sentinel Security has now discovered that using these techniques, combined with a simple algorithm, it is possible to model a biomorphic set that exhibits chaotic behavior similar to biological growth models. In fact, as shown below, when modeled in a three dimensional rendering software package, the data resembles a blob of amorphous matter reminiscent of movement or growth models associated with cellular automata.



Modeling Randomness

The exact parameters for modeling the specific biomorphic set used by HYDRA are a trade secret (not for reasons of security, but rather to protect the intellectual property). The modeling process, however, is fairly straight forward.

The first step is to create a set using a simple geometric progression of the form:

$$x_n y_n z_n = \delta(x_{n-1} y_{n-1} z_{n-1} + i)$$

Where δ is a constant derived from a hash against system parameters such as hardware and software serial numbers. The algorithm to draw this set is similar to that used in creating fractal images, where individual values are fed back into the equation, and used in determining subsequent values.

Once the set is defined, the parameters are now used to create a series of orbits around and within the set. To generate the random numbers, called "bodacions," the algorithm simply tracks the orbits, delivering a representation of the x, y, and z coordinates in the form of a string of characters. Much of the computation is front-loaded, and subsequent requests for bodacions require very little computation.

Bodacions have three very unique properties:

1. Bodacions are nearly impossible to guess or calculate without the seed, orbit, and reference to position along the orbit. That reference positions itself changes chaotically. The smallest bodacion consists of 320 bits of entropy, with larger hashes or combinations of bodacions producing even more entropy.
2. Bodacions have almost no correlation to each other. As shown below by the test results from various tests against random or pseudo-random numbers, bodacions exhibit near true-randomness with deterministic, provable results.

Test	HW generated	Bodacions	Optimal Result
Entropy	0.999318	1.000000	1.000000
X^2	0.01	50.00	25.00-75.00
Arithmetic Mean	0.4846	0.5002	0.5000
Monte Carlo Error	10.80%	0.13%	0.00%
Correlation Coefficient	0.149488	0.000372	0.000000

3. Bodacions never repeat until all possible values are delivered. In a typical set size of 2^{256}, the system could generate thirty-two bodacion per second and never repeat a value for close to one thousand years.

   An astute reader would point out that this characteristic may reduce the security of bodacions. Think of a group of people guessing a number between one and ten. The first guess has a 1/10 chance of being correct. However, if you were privy to the first guess, and you are the second guesser, your odds improve to 1/9. The odds improve in that way until after ten guesses, you have a 1/1 chance (i.e. you know the answer without guessing).

   Since bodacions are shared among disparate subsystems and connections, however, all parties are not privy to all previous bodacions, reducing the odds of a correct guess. Furthermore, a well-designed hash on a bodacion can turn it into a random number with a varying probability of repeating a value, while still maintaining the chaotic characteristics.

Full details are available in the rather lengthy patent number 20020064279, which may be found at www.uspto.gov.

HYDRAs Applications

Since bodacions fulfill the entropy and computational needs of a system like HYDRA, the HYDRA server makes extensive use of the numbers throughout every subsystem.

Diversity

The primary use for bodacions in HYDRA is to create diversity at the process, subsystem, and device level. By making each element different from another, HYDRA enhances its security posture. HYDRA achieves the diversity by maximizing the divergences while making the path of divergences difficult to predict using chaotic algorithms.

To understand why this is so powerful, we again turn to nature. In a behavior known as Evolutionary Psychology, living organisms prefer to reproduce under the maximum heterozygosity, or put another way, under situations where two sets of compatible genes have the most differences. Opposites really do attract.

A famous case where this went horribly wrong is the cheetah. Some time ago, all species of cheetah expect the one we know today became extinct. The remaining species was forced to inbreed due to their small numbers, and this in turn has produced a situation where any one cheetah is nearly 99% the same as all other cheetah, compared with less than 80% similarity among different humans.

The difficulty for cheetahs is that in the event of a deadly virus targeting cheetahs, it is likely that the entire species would be wiped out. Bear with me, but this is exactly the problem behind the reason Internet worms and viruses spread so quickly. With millions of computers running identical software all connected to each other, a hacker need only create one virus and it can spread to all similar systems.

If those systems had diversity, the virus would be limited to only those systems sharing common memory layouts and configurations. HYDRA uses its ability to create diversity among its subsystems and sets of devices to prevent this very situation. If a hacker with the resources to crack a password or exploit any kind of vulnerability in their HYDRA tries to use the same attack on another HYDRA, it is extremely unlikely the attack would succeed.

TCP Initial Sequence Numbers

Another way HYDRA uses biomorphics is in the creation of TCP Initial Sequence Numbers (ISNs). The ability to predict TCP sequence numbers can allow a malicious user to gain access to a system by masquerading as a legitimate host. This untrusted system can receive all of the system privileges that were afforded to the spoofed system. A malicious user could gain access to files, databases, and system information. The security of the system and other remote systems could be placed in jeopardy.

To thwart this attack, HYDRA uses bodacions to create TCP initial sequence numbers. HYDRA must create these sequence numbers upon each new network connection, which speaks to the low computational power required to create a bodacion.

Web Content Verification

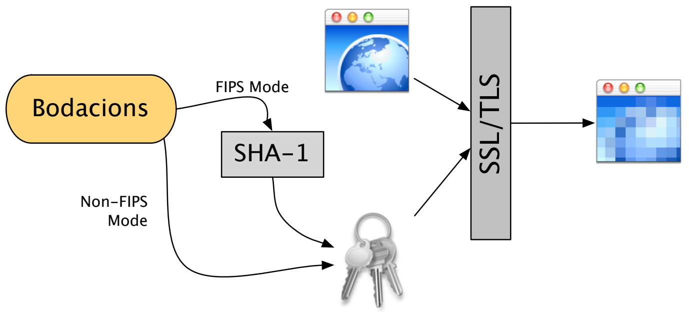
Malicious modification of Web content has been known to result in tremendous damages. Hackers have successfully launched so-called “subversion of information” attacks to modify financial records, change the cost of retails items, and even to post false news stories from reputable sources regarding important political officers as high as the President of the United States.

To prevent these attacks, HYDRA creates a signature on each file, be it XML, SOAP, HTML or any other content delivered via HTTP. The signature is based on a simple chaotic function and tainted, or combined, with a bodacion. HYDRA’s Web server then checks the resulting signature against a recalculation before delivering any Web content.

SSL/TLS

HYDRA also uses bodacions to secure Web content delivered over SSL/TLS. Web browsers (e.g. Safari, Netscape, Internet Explorer) and Web servers (e.g. HYDRA, Apache, IIS) use the Secure Sockets Layer (or SSL) protocol to encrypt information in transit between the browser and server. TLS makes some subtle, but important changes from SSL, and has a different name as proof that eight people in one room have eight opinions on what anything should be named.

Bodacions provide the source for cryptographic key material for use in standard ciphers like Triple DES, AES, and RC4, as illustrated below.



Note that HYDRA uses FIPS approved implementations of Triple DES and SHA-1, so when operating in FIPS mode, bodacions are first hashed to create a FIPS compliant PRNG. Mathematical analysis shows the incorporation of SHA-1 to actually increase correlation (lower entropy), however the impact on the resulting key material is negligible.

HYDRA employs techniques similar to those used to create key material and verifying content in dozens of additional subsystems and functions, including:

- General Random Number generation
- SSL Session ID's
- TCP Initial Sequence Numbers
- One-way Password Hashing
- Disk Cache Verification
- Buffer Overrun Protection (Canary Values)
- IP ID's
- X.509 Certificate Ciphering
- In-RAM Protection by Scramble Ciphers
- HTTP Virtual Server Lookups
- User and Group ID assignment
- Enabler Codes and Serial Number Generation

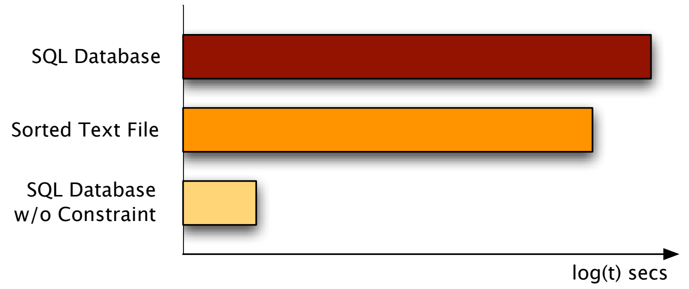
External Availability

Outside of HYDRA's internal systems, remote applications may obtain bodacions via a simple protocol. Given the unique properties of bodacions, applications may use them for session ID's, transaction numbers, and for countless other values that benefit from unique, difficult to guess numbers or characters.

Database Performance Boost

Many database applications require unique values to allow for searching of a row set of data by a single column value. For example, banks typically give each financial transaction a unique ID. These ID's should be difficult to guess, so that unauthorized users can not easily locate transactions.

In this case, the system must query the database for each transaction ID before inserting that ID to avoid duplicates. Database implementations almost always include a feature called "unique constraint" that can perform this operation for the application, albeit at significant cost. Using bodacions (which are always unique) as ID's allows the developer to remove the unique constraint on the column ID, which (as shown below) can dramatically improve database performance.



This graph shows the result of inserting ten million bodacions into first a Sybase database running on an IBM UNIX server, next into a sorted flat file, and finally Sybase again with the unique constraint turned off. The uniqueness of bodacions, in this example, accelerates database performance one hundred fold.

URL Protection

Information on the Internet is addressed primarily with a Uniform Resource Locator, or URL, such as the one below.

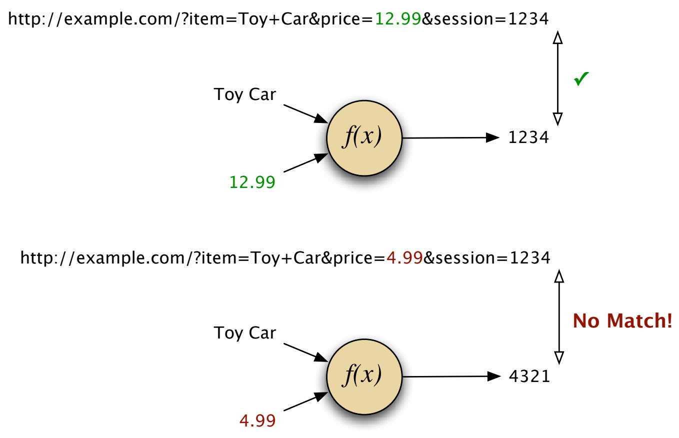
```
http://example.com/?item=Toy+Car&price=12.99
```

A URL consists of an access protocol (`http`), a domain name (`example.com`) and many other optional components, including parameters passed to application in the form `name=value`.

Applications can use keyed bodacions (a bodacion made from a string of data rather than the default set) to insure a user has not modified privileged components of a URL. For example, consider the above example URL that includes an item name (Toy Car) and a price ($12.99).

If the user modified this URL by changing the price, the server would have no way of knowing this without looking up the price again. To save that step, the system can create a identifier made from the item number and price.

Each time the user returns to the site, the server could recalculate the ID or look it up in a table. If the ID matches what the one in the user's URL, the elements involved in making the ID have not been modified. In this use, bodacions resemble an efficient hash or digital signature, depending on the details of their use.


http://example.com/?item=Toy+Car&price=12.99&session=1234
Toy Car
f(x)
12.99
1234
✓
http://example.com/?item=Toy+Car&price=4.99&session=1234
Toy Car
f(x)
4.99
4321
No Match!

Session Protection

You might have noticed that the ID in the example URL above was named "`session`". This leads us to yet another important application of bodacions: session ID's.

The Internet operates primarily on a connectionless protocol. That is to say that, unlike a telephone conversation where two parties are directly connected, downloading a typical Web page requires multiple connections, and subsequent pages require new connections. Web servers have no way to associate multiple requests with a single user, and so server applications often employ a unique session ID which the two parties in the conversation must exchange upon each discrete connection.

Systems that need to create session ID's face numerous, significant challenges. These challenges can differ from system to system, and vary from CPU usage requirements in busy or slow systems to session locking to security issues. The majority of systems, however, share two common issues: guaranteeing that session ID's are unique, and that session ID's are secure from hackers guessing the ID's of other sessions.

Difficulty In Guaranteeing Uniqueness
For a session ID to be effective in linking discrete sections of a dialog in a connectionless system, the session ID for each dialog must be unique among all dialogs. For example, say a particular server assigns session ID's based on the time in seconds since the system began operation. If two users accessed the system within the same second, the server would think that these two users are actually the same user.

Similarly, say a system uses the user's last name and first three digits of their social security number to assign a customer ID. To this system, John Smith, born in Chicago, IL and Jenny Smith, born in Schaumburg, IL, may both receive the same customer ID of Smith321. The system now has no way to distinguish Jenny from John.

Depending on the complexity of the system and its user base, it can be difficult, if not impossible, to make a unique key from user data elements alone.

The guessing of session ID's by computer bandits is one of the most prolific methods used to break into systems. Session ID's created by even complex algorithms are often simple to guess, and once a hacker has a session ID, they can assume any identity. Hackers frequently look for typical session generation techniques such as:

- Sequential Session Keys
  Some sites use a sequence to create ID's. For example, a system may create sessions for different users of the form A1, B2, C3...etc. All a hacker has to do is get one valid session, say E5, and then wait to assume the identity of the next person who comes to the site, who's session will be F6.

- Random Session Numbers
  Many systems use a pseudo-random number generator, such as those built in to the microprocessor, to produce session ID's. As explained previously, these techniques often provide very little randomness, and are easily exploited.

Bodacions, in contrast, are both difficult to guess or back-calculate, and provide hackers near zero correlation (recall the numerical results given above).

Results Analysis

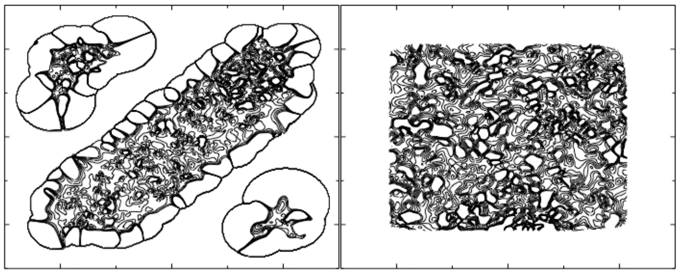
The most dramatic proof of bodacions as session ID's or random numbers versus other methods comes from diagrams produced by phase space analysis. Sorry, but it's time for just a little more math.

Since session ID's (and indeed nearly all random data used in HYDRA) are single dimensional in nature, an algorithm to map these linear data into three dimensional space is employed to create a more visually concise representation. For the purposes of plotting bodacions, a technique called the method of delays is used to reconstruct a phase space. The method simply maps each successive value of a series into x, y, and z coordinates.

$$\begin{aligned} x &= f(g(t_{n-1}), g(t_n)) \\ y &= f(g(t_{n-2}), g(t_{n-1})) \\ z &= f(g(t_{n-3}), g(t_{n-2})) \end{aligned}$$

The following figure shows plots rendered using data generated with the method of delays. The shape on the left represents session ID's generated from a Java application that uses the default ID's from the application server connector. The shape on the right represents session ID's from the same application using bodacions.

The reader can clearly see that the so-called "random" session ID's from the Java application server (left) produce a clear pattern, while the bodacions (right) do not.



HW-based RNG Bodacions

In Closing

Bodacions made from HYDRA's implementation of chaotic, biomorphics sets have powerful applications in internal security, cryptography, and Web systems from the application layer and down.

For more information about HYDRA, please visit Sentinel Security Corporation online at `http://www.sentinelsecurity.us`. You can also send an email directly to me via `http://tinyurl.com/k4288` with comments, suggestions, and questions, though please keep in mind I may not be able to respond directly to every message.

Further Reading

For further reading, I recommends the following:

- Mandlebrot, Benoit B. (1982). Fractal Geometry of Nature. WH Freeman & Co. ISBN 0716711869
- Schroeder , Manfred (1995) Fractals, Chaos, Power Laws: Minutes from an Infinite Paradise. W H Freeman & Co. ISBN: 0716723573
- Sparrow, C.T. (1983) Lorenz Equations: Bifurcations, Chaos, and Strange Attractors. Springer Verlag. ASIN: 0387907750
- Scambray, Joel and McClure, Stuart (2002) Hacking Exposed Web Applications. McGraw-Hill Osborne Media. ISBN: 007222438X
- Uner, Eric. On Password Branching. http://www.uner.com/passwordBranching.html
- Uner, Eric. Generating random numbers. http://www.embedded.com/showArticle.jhtml;jsessionid=?articleID=20900500

My thanks to J. McKenzie Alexander, who created the freeware LaTeX editor I used in creating the equations in this and so many other papers, and to Mitchell Weiss at `http://members.optusnet.com.au/~mitchweiss/` for the use of his lava lamp image.