On the Security of Polymorphic Encryption

C. B. Roellgen

PMC Ciphers, Inc, 92 Derrick Road, Bradford, PA 16701
{ broellgen@pmc-ciphers.com }

15.03.2004

Abstract

*A basic Polymorphic Cipher consists of a number of stacked primitive pseudo-random number generators that generate a confusion sequence which is directly XORed to plaintext bits. It forms a combined secrecy system as described in "*Communication theory of secrecy systems" by *C.E. Shannon,* 1949*. The Polymorphic Cipher provides as many different ciphers as different keys are available in a given keyspace. It can be shown that for the range of available primitive pseudo-random number generators containing cryptographically weak functions, the combined secrecy system contains a substantially higher number of variables than available samples that an attacker can use for analysis.*

Key words: cipher of ciphers, combined secrecy system, endomorphic ciphers, stream cipher, block cipher, polymorphic encryption, pseudo-random number generator, compiled PRNG, crypto compiler, self-compiling crypto code, one-time pad

1. Introduction

The idea of polymorphic encryption is based on a "Cipher of Ciphers" concept which is described in [1] as "combined secrecy system". A combined secrecy system can be the sum of a number of systems. As an example, a crypto engine which can choose from four secure 128 bit encryption algorithms, e.g. AES Rijndael, Twofish, RC6 and Mars is practical and easy to implement. Each of the base ciphers is regarded as unbreakable. A 130 bit key is used to encrypt messages - 2 bit select one of the four available base ciphers and the remaining 128 bit represent the key for the chosen base cipher. The result is an unbreakable 130 bit cipher as each of the 128 bit base ciphers is unbreakable and each base cipher generates ciphertext with good pseudo-randomness and thus cannot be identified by its output bit pattern. The two additional bits make the proposed "Cipher of Ciphers" stronger than each of the base ciphers with the following advantages:

1. Brute Force Attack takes longer on 130bit than on 128 bit. Although 128 bit are practically unbreakable, additional key bits are generally welcome.

2. Two additional bits consume only once little CPU time while they don't require any time when encrypting the next 100 terabytes e.g. when used in server-to-server communication

3. An attacker must try to crack four base ciphers instead of one in order to be able to read all encrypted messages which are encrypted using this polymorphic cipher. It might be possible to reduce the number of rounds for Twofish or for some other algorithm, but it's unlikely that the whole base cipher set can be cracked with time. A more powerful polymorphic cipher could feature a set of 128 base ciphers. Cracking one of them would expose less than 1% of all encrypted messages. Cracking one of those base ciphers might be as difficult as cracking Rijndael. If Rijndael was cracked, it could be a disaster for the industry, but if Rijndael was only a base cipher of a polymorphic cipher of ciphers, the news that this specific cipher was broken, would be rather insignificant.

4. Terry Ritter writes in his e-mail archive [6] that "a 3-cipher stack gives us exponentially many (n^3, or perhaps just n^2 for weenies) different ciphering stack possibilities. The intent here is *not* to add keyspace, since reasonable ciphers already have enough. Instead, the point is the easy construction of exponentially many conceptually different overall ciphering functions which the opponents must engage."

5. Ritter also writes in his e-mail archive [6] that “a cipher change terminates any existing break. Since we will not know when a break exists (and that our information is being exposed), this provides an alternative to just using the same cipher forever and hoping for the best.”

2. Theoretical basis for the basic idea behind the Polymorphic Cipher

The advantages of the proposed polymorphic cipher are a noticeable increase in security and additional key bits which do not consume processing time after key setup.

A combined secrecy system can be either the sum of a number of systems or the product thereof. This is seen in any practical cipher today on the function level. On the system level, the sum is nothing else than a keyed selection between two ciphers which are applied alternatively, while the product is the keyed selection between ciphers and the successive application on the ciphertext output by the previously selected cipher.

C. E. Shannon [1] writes:

THE ALGEBRA OF SECRECY SYSTEMS

If we have two secrecy systems T and R we can often combine them in various ways to form a new secrecy system S. If T and R have the same domain (message space) we may form a kind of ``weighted sum,''

$$S = pT + qR$$

Where $p+q=1$. This operation consists of first making a preliminary choice with probabilities p and q determining which of T and R is used. This choice is part of the key of S. After this is determined T or R is used as originally defined. The total key of S must specify which of T and R is used and which key of T (or R) is used.

If T consists of the transformations $T_1, ..., T_m$ with probabilities $p_1, \ldots, p_m$ and R consists of $R_1, \ldots R_k$ with probabilities $q_1, \ldots, q_k$ then $S = pT + qR$ consists of the transformations $T_1, ..., T_m, R_1, \ldots R_k$ with probabilities $pp_1, pp_2, \ldots, pp_m, qq_1, qq_2, \ldots qq_k$ respectively.

More generally we can form the sum of a number of systems.

$$S = p_1T + p_2R + \ldots + p_mU \qquad \sum p_i = 1$$

We note that any system T can be written as a sum of fixed operations

$$T = p_1T_1 + p_2T_2 + \ldots + p_mT_m$$

T_i being a definite enciphering operation of T corresponding to key choice i, which has probability p_i.

A second way of combining two secrecy systems is by taking the ``product,'' shown schematically in Fig. 1. Suppose T and R are two systems and the domain (language space) of R can be identified with the range (cryptogram space) of T. Then we can apply first T to our language and then R

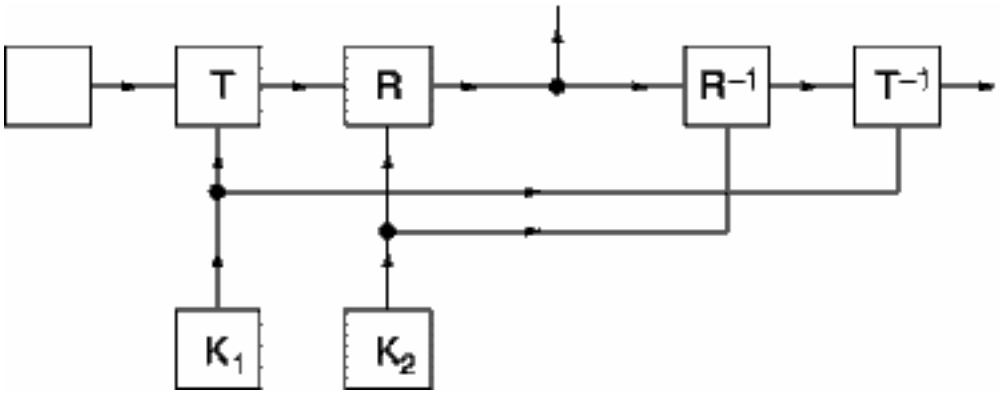

Figure 1: Product of two systems $S = RT$

to the result of this enciphering process. This gives a resultant operation S which we write as a product

$$S = RT$$

The key for S consists of both keys of T and R which are assumed chosen according to their original probabilities and independently. Thus, if the m keys of T are chosen with probabilities

$$p_1, p_2 \dots p_m$$

and the n keys of R have probabilities

$$p'_1, p'_2 \dots p'_n$$

$$p_i\, p'_j$$

then S has at most mn keys with probabilities $p_i\, p'_j$. In many cases some of the product transformations R_iT_i will be the same and can be grouped together, adding their probabilities.
Product encipherment is often used; for example, one follows a substitution by a transposition or a transposition by a Vigenère, or applies a code to the text and enciphers the result by substitution, transposition, fractionation, etc.
It may be noted that multiplication is not in general commutative (we do not always have $RS = SR$), although in special cases, such as substitution and transposition, it is. Since it represents an operation it is definitionally associative. That is, $R(ST) = (RS)T = RST$. Furthermore, we have the laws

$$p(p'T + q'R) + qS = pp'T + pq'R + qS$$

(weighted associative law for addition)

$$T(pR + qS) = pTR + qTS$$

$$(pR + qS)T = pRT + qST$$

(right and left hand distributive laws)
and

$$p_1T + p_2T + p_3R = (p_1 + p_2)T + p_3R$$

It should be emphasized that these combining operations of addition and multiplication apply to secrecy systems as a whole. The product of two systems TR should not be confused with the product of the transformations in the systems $T_i\, R_j$, which also appears often in this work. The former TR is a secrecy system, i.e., a set of transformations with associated probabilities; the latter is a particular transformation. Further the sum of two systems $pR + qT$ is a system--the sum of two transformations is not defined. The

systems T and R may commute without the individual T_i and R_j commuting, e.g., if R is a Beaufort system of a given period, all keys equally likely.

$$R_iR_j \neq R_jR_i$$

in general, but of course RR does not depend on its order; actually

$$RR = V$$

the Vigenère of the same period with random key. On the other hand, if the individual T_i and R_j of two systems T and R commute, then the systems commute.
A system whose M and E spaces can be identified, a very common case as when letter sequences are transformed into letter sequences, may be termed *endomorphic*. An endomorphic system T may be raised to a power T^n.
A secrecy system T whose product with itself is equal to T, i.e., for which

$$TT = T$$

will be called idempotent. For example, simple substitution, transposition of period p, Vigenère of period p (all with each key equally likely) are idempotent.
The set of all endomorphic secrecy systems defined in a fixed message space constitutes an ``algebraic variety,'' that is, a kind of algebra, using the operations of addition and multiplication. In fact, the properties of addition and multiplication which we have discussed may be summarized as follows:

The set of endomorphic ciphers with the same message space and the two combining operations of weighted addition and multiplication form a linear associative algebra with a unit element, apart from the fact that the coefficients in a weighted addition must be non-negative and sum to unity.

The combining operations give us ways of constructing many new types of secrecy systems from certain ones, such as the examples given. We may also use them to describe the situation facing a cryptanalyst when attempting to solve a cryptogram of unknown type. He is, in fact, solving a secrecy system of the type

$$T = p_1A + p_2B + \ldots + p_rS + p'S \qquad \sum p_i = 1$$

where the $A, B, \ldots, S$ are known types of ciphers, with the p_i their *a priori* probabilities in this situation, and $+ p'X$ corresponds to the possibility of a completely new unknown type of cipher.

3. The Polymorphic Cipher as very large combined secrecy system

The Polymorphic Cipher is a combined secrecy system that provides as many different ciphers as different keys are available in a certain keyspace. This constructs exponentially many conceptually different ciphering functions which opponents must engage in order to successfully crack the Polymorphic Cipher.
Being able to choose from 2^{128} ciphers for a 128 bit encryption algorithm has the advantage that it renders known attacks requiring a static system inapplicable.

In order to provide a significant number of different encryption algorithms, a highly flexible base design relying on a set of freely stackable pseudorandom number generators in a basic stream cipher configuration can be chosen. This configuration is described subsequently.

3.1 Operating Principle of PMC

In order to randomize the encryption algorithm, a simple and interestingly novel approach has been chosen: The compilation of machine code from the passphrase (or key). In order to generate a cryptographically

useful sequence of machine instructions which are later executed by a microprocessor on the target machine, a set of primitive pseudo-random number generators is defined which the compiler can assemble in a totally random sequence.
The set of primitive pseudo-random number generators consists of functions which are cryptographically insecure when used alone. As an example, the Linear Congruential random number generator (LCGRNG) which uses the recurrence

$$X_{i+1} = aX_i + b \bmod m$$

to generate an output sequence $\{X_0, X_1, \ldots\}$ from secret parameters a, b and m, and a starting point X_0, reveals all of it's secrets with just knowing three to four X_i only!

Add-with-carry generators (ACG) form another simple and even better source for pseudo-random bits. Their function can be written as

$$X_n = X_{n-s} + X_{n-r} + carry \bmod m$$

These generators have long periods, easily exceeding 10^{200}, and they are almost as fast as LCGs.

Multiply-with-carry generators (MWCG) use this simple function:

$$X_n = aX_{n-1} + \text{carry} \bmod m$$

Multiplier a can be chosen from a large set of integers without affecting the period of around 2^{31}-1 for 32 bit implementations. There is probably no test it will not pass.

Add-with-carry generators can have a very long period if s and r are large:

$$X_n = X_{n-s} + X_{n-r} + carry \bmod m$$

If a sufficiently large number of such primitive RNGs are concatenated to form one single RNG, security holes of each primitive RNGs are filled easily.

The concept is not limited to classic random number generator primitives. As an example, a permutation function using word-wide shift-through-carry operations, which execute in any modern microprocessor fast because these devices generally come with a barrel shifter topology, breaks up possible linear dependencies between other primitives that are similar. Linear dependencies are likely to occur for keys that lead to the selection of, as an example, only LCG- type primitives. Short cycles can potentially occur and consequently making known plaintext attacks extremely likely to succeed.

The complete structure of a basic, but working PMC model is shown in figure 2:

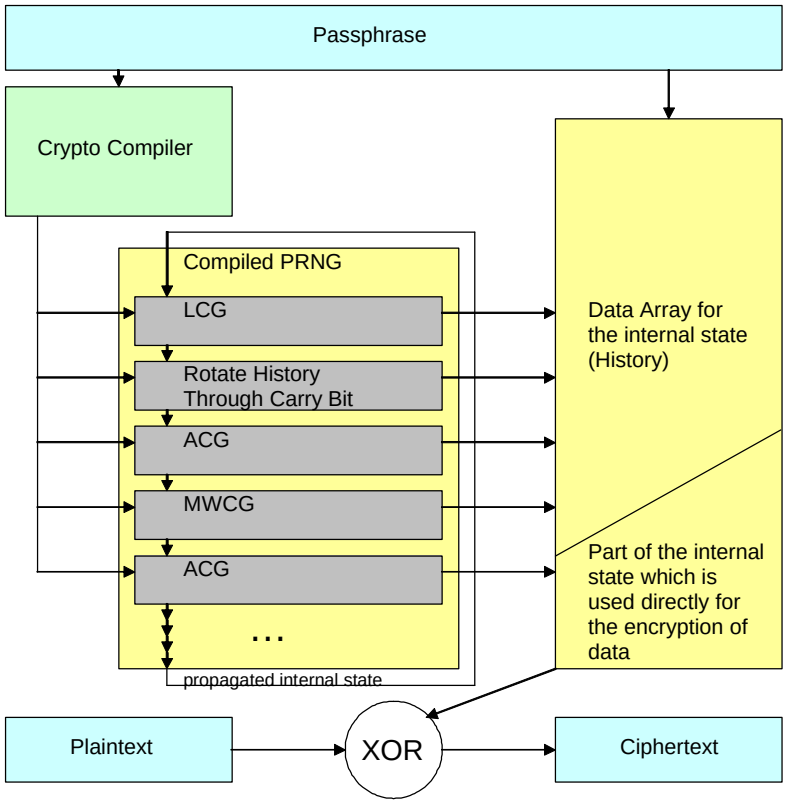

Figure 2: Structure of a PMC implementation without Cipher Feedback (CFB) and without Cipher Block Chaining (CBC)

The passphrase is compiled into machine code. The compiler simply assembles standardized pseudo-random number generators, the “building blocks”, sets variables in the building blocks using keyed operations, adjusts addresses as well as entry- and exit points to generate a piece of machine code which acts like a huge pseudo-random number generator that is working on the history data array and that passes data from one primitive to the next.
After initializing the history data array with the passphrase or a binary representation or a hash thereof, the instruction pointer of the microprocessor on the target machine is set to the start of the Compiled PRNG. After finishing the execution of the Compiled PRNG, the bit pattern stored in the history data array consists of near-random data.

After the machine code has been executed, part of the content of the key data array can be used to encrypt plaintext through the application of the XOR function. The content of the history data array can and should alternatively be used for biasing an underlying cryptographic algorithm which is simple and fast. This doesn’t expose the internal state.

3.2 Actual i386 compiler output of an LCG building block

The following example shows a simple LCGRNG which computes $X_{i+1} = aX_i + b \bmod m$. a = 515, b = 5 and m = 1000001d.

The propagated history is passed from one building block to the next in the 32 bit register ebx internal to the microprocessor.
The processor register ebp is used as pointer to 32 bit words in the history data array. In order not to destroy its base, the content of ebp is pushed onto the stack at the beginning of each building block. At the end of each building block, ebp is fully restored by popping its original content from the stack.
The crypto compiler can choose from a set of values for a variety of variables: a, c and m can both be chosen from a set of possible values, with m being preferrably prime. The compiler can further predetermine which parts of the history data array are used by the LCGRNG and which data bits are later changed.

```
push ebp;                   // ebp MUST never be destroyed
and  eax,60d;               // use intermediate result from preceding primitive PRNG to select 32
                            // bit  from internal state
```

```
add  ebp,eax;
mov  eax,[ebp+0];           // load history[ebp+eax] in AL and history[ebp+x+1] in the next upper
                            // byte of eax and so on up to history[ebp+x+3]
mov  edx,eax;               // init edx with data from internal state (not every primitive PRNG
                            // needs that)
xor  eax,ebx;               // add the propagated history passed on in register ebx from the
                            // preceding primitive PRNG
imul eax,0x000014ad;        // a = 5293, c=0
mov  ecx,0x01a74ded;        // m = 27741677 (m is prime and a is roughly the square root of m)
cdq;
idiv ecx;
xor  eax,edx;               // xor modulo with quotient
xor[ebp+0],eax;             // modify history
add  ebx,eax;               // modify propagated history (ebx)
pop  ebp;
push ebp;
xor  [ebp+5],eax;           // change 32 bits in the history data array
xor  [ebp+21],eax;          // change 32 bits in the history data array
xor  [ebp+37],eax;          // change 32 bits in the history data array
xor  [ebp+52],eax;          // change 32 bits in the history data array
pop  ebp;
```

All primitive pseudo-random number generators use a framework similar to the modified LCGRNG shown above. This example is cryptographically weak because short cycles are nearly inevitable, bit bias can be observed, and identical instances are linear dependent.

Primitive PRNGs can be positioned anywhere inside the code of the Compiled PRNG. Any primitive PRNG can follow any other primitive PRNG.

The crypto compiler predetermines the data words which serve as input to a primitive PRNG and where the output bits will be stored. Each primitive fetches and stores data also at dynamically selected locations in the internal state. The process of choosing primitive PRNGs and defining input and output solely depends on the bit pattern of the passphrase.

The history data array also serves as substitution box for primitive PRNG building blocks which perform bit or word permutations.

4. Security increase of a PRNG stack forming a multiplicative combined secrecy system

Throughout this chapter it is assumed that the crypto compiler compiles identical LCG primitives to form a PRNG stack that works on an internal state that is common to all compiled PRNGs and that passes information from one primitive to the next in the stack.

The linear congruential sequence of a pure multiplicative LCG is determined by (a, X_n and M).

$$X_{n+1} = aX_n \ (mod\ M)$$
$$X_{n+2} = a^2X_n \ (mod\ M)$$
$$...$$

As there are three unknowns (a, X_n and M), consequently three consecutive samples are enough to crack the generator.

If used with a randomiser, the task to crack an LCG primitive in a compiled PRNG can be described as

$$X_{n+1} = aX_n + c \ (mod\ M) \qquad with\ c = R(i);\ R(i)\ is\ assumed\ to\ be\ truly\ random$$

yielding the congruential sequence

$$X_n = a^nX_0 + a^{(n-1)}R(1) + a^{(n-2)}R(2) + ... \ (mod\ M)$$

To get the minimum number of samples to determine the unknowns, the number of equations must at least equal the number of unknowns ($R(1), R(2), .., R(n), a, X_0, M$). An opponent gets n samples to crack but has to cope with $n+3$ variables.

Randomness of the bit stream $R(i)$ is not necessarily improved (true randomness can only be worsened) by feeding it through the LCG, but if pseudorandom numbers are fed through the LCG, the entropy of the random number stream is improved and there are three more variables to determine: (a, X_0, M). P. C. Yeh, R. M. Smith show in [8] that it is good design practice to use a PRNG that is continuously seeded with a number of truly random bits to increase entropy:
"The 64-bit real-time counter T is incremented continuously at the fastest rate possible for the machine … The use of T ensures that the output of the PRNG does not have a short-term cycle of repetition. (On IBM's G5 processor, the time for T to wrap around is several hundred years.)"

A Polymorphic Cipher generally uses a large internal state that is shared by the primitive PRNGs. To test absolute worst-case conditions, a crypto compiler is forced to generate identical primitive RNGs with a potentially high linear dependency. The crypto compiler further assigns only a small portion of the internal state to all primitive PRNGs. By doing this, parts of the internal state remain static.

Input to the next primitive in a row is a variable from the shared internal state that depends on the result from the previous primitive *(internal_state[eax & 3Ch])*. This data is subsequently changed by the primitive. Input is further one variable that is passed from one primitive to the next *(ebx)* and the constants $(A \text{ and } M_1)$, which the compiler would usually select from a list of 32 possible values each. Input to the complete compiled PRNG is a seed for *eax* and *ebx* with *ebx == eax*. Like in actual implementations, the seed is a pseudorandom number.

$$
\begin{aligned}
&ebp == eax \;\&\; 3Ch;\\
&X_n == internal\_state[ebp];\\
&eax == int((5293 \cdot (X_n \,(+)\, ebx)/27741677)) \,(+)\, (5293 \cdot (X_n \,(+)\, ebx) \bmod 27741677));\\
&internal\_state[ebp] == internal\_state[ebp] \,(+)\, eax;\\
&ebx == ebx \,(+)\, eax;\\
&internal\_state[5] == internal\_state[5] \,(+)\, eax;\\
&internal\_state[21] == internal\_state[21] \,(+)\, eax;\\
&internal\_state[37] == internal\_state[37] \,(+)\, eax;\\
&internal\_state[52] == internal\_state[52] \,(+)\, eax;
\end{aligned}
$$

As *ebp* depends on the result of an earlier executed operation, all of the internal state influences the computed values with time. A total of 64 identical primitives are concatenated.

The experiment proves the obvious: the internal state contains bits that are less often changed than other bits. Although the randomness in the internal state is far from being perfect, the compiled PRNG is seeded sufficiently for not to get caught in cycles. The experiment further yields for the propagated internal state perfect randomness which is pretty astonishing for a primitive function that has massive bit bias. Current implementations reseed the compiled RNG on every cycle and only few bits are taken and used in the underlying bit combination functionality in order to tolerate absolute worst case keys.

A combined secrecy system has the unique feature to exhibit no static weakness and it overwhelms an opponent with a large number of variables. The number of variables is at any time greater than the number of knowns. Assuming that each available primitive PRNG has at least 3 variables, there are a total of at least four additional variables per compiled primitive PRNG in a compiled PRNG: 3 variables and the fourth variable being the type of primitive PRNG itself. A compiled PRNG consisting of 32 primitive PRNGs has at least 128 variables, but generates only 32 intermediate results and one result used to bias an underlying cipher.

The variables-to-knows-ratio can be written as:

$$VTN = [\, v(P_1) + v(P_2) + \ldots + v(P_n)]/\, k$$ *with VTN = variables-to-knowns-ratio, $v(P_i)$ = number of variables of primitive i and k = number of knowns of the combined secrecy system*

The security increase of a polymorphic cipher, which is just a combined secrecy system, comes primarily from a high variables-to-knowns ratio even allowing for weaknesses in individual cases as these remain isolated cases.

It must be noted that there is a decisive difference between a polymorphic cipher and a standard block cipher like AES:
In [9], Chapter 4.2 it was shown by Courtois and Pieprzyk that "..the problem of recovering the key of the 128-bit Rijndael, will be written as a system of 8000 quadratic equations with 1600 variables".
This shows that for AES the number of variables could be more than sufficient for breaking codes.

In contrast can a polymorphic cipher always be characterized by a substantially higher variables-to-knowns ratio than 1.

5. Entropy gain through the multiplicative combination of primitive PRNGs

Describing variable systems can be very challenging. Earlier, a convenient and simple approach was taken.
Any PRNG has the tendency to increase entropy by definition. The entropy of the input bit pattern is greater than the entropy of the output bit pattern.
Entropy of the output bit pattern of a set of primitive PRNGs forming the Compiled PRNG is greater with every additional primitive PRNG.

Here's some mathematics which proves this hypothesis:

Let $F_X(r)$ be a PRNG function out of the set $X=\{x_1,x_2, .., x_n\}$ of available PRNG functions which are available to the compiler and let r be the position of the primitive PRNG in the complete Compiled PRNG with $0 <= r <= R-1$ with R representing the total number of primitive PRNGs which are concatenated.

$P_i[F_X(r,i)]$ is the probability P of the i-th bit in the history data array or internal state of being biased towards the bit value 0 or 1. We assume that $F_X(r,i)$ is a primitive PRNG with a poor performance. Consequently $P_i[F_X(r,i)] > 0$

The uncertainty or entropy of the i-th bit is defined by

$$H(i) == -\sum P_i[F_X(r,i)] \log_2(P_i[F_X(r,i)])$$

which satisfies

$0 <= H(i) <= \log_2(2)$ (bit i has 2 possible states {0, 1} => $0 <= H(i) <= \log_2(2) \Leftrightarrow 0 <= H(i) <= 1$)

After executing all R primitive PRNGs, the probability $P_i[F_X(R,i)]$ of bit i to be biased yields

$$P_i[F_X(R,i)] = P_i[F_X(0,i)] \cdot P_i[F_X(1,i)] \cdot \ldots \cdot P_i[F_X(R\text{-}10,i)] = \prod P_i[F_X(r,i)]$$

$\prod P_i[F_X(r,i)]$ approaches 0 for a sufficiently big number of primitive PRNGs R which are concatenated to form a single Compiled PRNG. Consequently $H(i) \approx \log_2(2) = 1$.

6. Conclusion

When regarding a Polymorphic Cipher as a multiplicative secrecy system constructing exponentially many conceptually different ciphering functions which opponents must engage in order to successfully break the Polymorphic Cipher, it is obvious that known attacks requiring a static system are applicable. The number of variables is at any time greater than the number of knowns thus further limiting the number of available

cryptanalytic tools.

It could be shown through experimenting with a Polymorphic Cipher consisting solely of identical and potentially highly linear dependent Primitive RNGs that even under worst case conditions entropy of the propagated internal state could be increased.
This supports results from the earlier adducted entropy theory which in short says that the entropy of the output bit pattern of a set of Primitive RNGs forming a Polymorphic Cipher is greater with every additional Primitive RNG.

References:

[1] C.E. Shannon. Communication theory of secrecy systems. Bell System Technical Journal, 1949

[2] H. Feistel. Block cipher cryptographic system. U.S. Patent No. 3,798,359, 1974

[3] S.W. Golomb. Shift Register Sequences. Holden-Day, San Francisco, 1967.

[4] S. Chari, C. Jutla, J.R. Rao, P. Rohatgi. A cautionary Note Regarding Evaluation of AES Candidates on Smart-Cards. http://citeseer.nj.nec.com/chari99cautionary.html, 1999

[5] C.E. Shannon. A mathematical theory of communication. Bell System Technical Journal, 1948

[6] Terry Ritter. Multiciphering and Random Cipher Selection in Shannon. http://www.ciphersbyritter.com/NEWS6/MULTSHAN.HTM, 2001

[7] Amy Glen. On the Period Length of Pseudorandom Number Sequences, http://www.maths.adelaide.edu.au/people/aglen/thesis2002_pdf.pdf, 2002

[8] P. C. Yeh, R. M. Smith, Sr. S/390 CMOS Cryptographic Coprocessor Architecture: Overview and design considerations http://www.research.ibm.com/journal/rd/435/yeh.pdf

[9] Nicolas T. Courtois, Josef Pieprzyk. Cryptanalysis of Block Ciphers with Overdefined Systems of Equations http://eprint.iacr.org/2002/044.pdf, 2002