A Generalized Model for Polymorphic Ciphers

C. B. Roellgen

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

31.12.2002
(first published on 27.07.2002)

Abstract

A generalized model for self-compiling crypto code as implemented in recent designs of Polymorphic Ciphers is proposed. Polymorphic Ciphers use a pseudo-random number generator which consists of a number of primitive pseudo-random number generators for the generation of a confusion sequence which is directly used to XOR plaintext bits. Self-compiling ciphers are basically stream ciphers while actual implementations are a mixture of block- and stream ciphers. A completely new cipher feedback mode is possible for the class of self-compiling encryption algorithms. It is shown that Polymorphic Encryption algorithms are immune to stream cipher attacks as well as block cipher attacks. In contrast to most or all commonly known symmetric encryption algorithm designs (including the AES candidates such as Rijndael and Twofish), polymorphic ciphers can be made immune to Differential Power Attack.

Key words: stream cipher, block cipher, polymorphic algorithm, pseudo-random number generator, Compiled PRNG, crypto compiler, self-compiling crypto code, one-time pad

1. Introduction

The Polymorphic Algorithm, which is discussed in this paper, belongs to the class of symmetric ciphers. It's roots are found in stream ciphers.

Common ciphers are basically fixed pieces of software which treat plaintext with a key during the process of encryption. When ciphertext is subsequently decrypted, the inverse function is applied to the ciphertext as had originally been applied to the plaintext. If the same key as during the encryption process is being used for this operation, the original plaintext is yielded.

2. Classic symmetric ciphers

In 1949 Shannon [1] describes the principles of confusion and diffusion.
He describes confusion as being "the use of enciphering transformations that complicate the determination of how the statistics of the ciphertext depend on the statistics of the plaintext".
Diffusion simply means spreading the influence of individual pieces of plaintext data over the full ciphertext or at least over big areas of ciphertext. By doing this, the statistics which might be inherent in the plaintext can be hidden.

Diffusion is not used in stream cipher designs, although. It is not possible to apply this principle in such designs, which does not necessarily mean that this class of ciphers is inherently weak.

Especially after 1974, when the Feistel function [2] had become popular, most cipher designs used iterated design approaches. If weak functions like the transposition of bytes or substitution are applied repeatedly, the weakness of the function which is repeatedly used cannot be attacked any more successfully if a certain minimum number of rounds are being executed.

The Feistel design can be briefly described as follows:
A plaintext block m of length n is encrypted as two blocks m_0 and m_1 of size $n/2$. Thus, $m = m_0 m_1$. Key k shall be defined as a set of subkeys $k_1 \dots k_r$, one for each of r rounds. Each subkey k_i acts as input to a transformation $f(k_i, \ )$ on the set of blocks of size $n/2$.

The blocks $m_2, \dots , m_{r+1}$ are defined as below:

$$m_i = m_{i-2} \text{ XOR } f(k_{i-1} , m_{i-1}).$$

Usually this recurrence does nothing more than performing the transformation $f(k_i, \ )$ r times, followed by a swap of the two halves of the data.

The Feistel cipher is extremely advantageous because the decryption process implies just the use of the subkeys $k_r \dots k_1$ in reverse order on the ciphertext $m_{r+1}m_r$. This property holds for any transformation function $f(k_i, \ )$.

3. Possible cryptanalytic attacks on symmetric ciphers

According to *Kerckhoffs' assumption*, cryptanalysts have full knowledge of the cipher and have access to plaintext as well as to ciphertext. An enciphering system consequently must solely rely on the key.

Ciphertext-only attacks only rely on encrypted data and usually fail for good ciphers.
Known plaintext attacks are generally more promising, because it might be possible that a simple dictionary of ciphertexts and the corresponding plaintexts could be set up.

Chosen plaintext attacks are the most promising ones: A server which uses a fixed key is extremely vulnerable, because the transmitted header of data packets alone could reveal the transported data. By knowing what frequently used plaintext-ciphertext pairs actually mean, missing information might be possible to reconstruct.

Chosen ciphertext attacks are rarely applicable. It might be useful for cracking systems which sometimes change codes and which disclose their behaviour by not properly disguising their operation mode. This attack is applicable on smart cards with a limited keyset stored on the smart card.

Differential cryptanalysis is a kind of attack which can only be applied to block ciphers. It exploits known properties of particular rounds in an iterated block cipher. With certain plaintext pairs that are encrypted, the amount of change which the data experiences in the final round can be predicted. The difference in the pair entering the final round and the difference in the pair leaving the final round reveals in some cipher designs the subkey used in the final round.

Key Bit Bias (or Linear cryptanalysis) is a kind of attack which uses extensive ciphering of fixed plaintext data under a variety of different keys. It is sometimes possible to associate key bits with the statistical value of some ciphertext bits.

4. Classic stream ciphers

Stream ciphers basically perform a time-varying transformation on individual plaintext digits. They operate like the famous one-time pad (or Vernam cipher) by generating a long string of keystream which consists of pure random data bits. Each bit of the keystream is combined bit by bit with the plaintext. Keystream and plaintext have the same length. As the keystream can only be used once (that's why this cipher is called one-time pad), quite a lot of keystream bits may be required.

The operation can be written as follows:
We let the plaintext message m be a sequence of bits $m = m_i$ XOR k_i for $0 =< i =< n-1$. The ciphertext $c = c_0c_1 \dots c_{n-1}$ is defined by $c_i = m_i$ XOR k_i for $0 =< i =< n-1$.

Shannon [1] proved in 1949 that the one-time pad is unbreakable. The one-time pad is the most perfect cipher because ciphertext and plaintext are statistically independent.

As it is unpractical to carry massive keystream data around, stream ciphers use a short key to generate the keystream sequence. The keystream data is consequently only pseudo-random.
Stream ciphers become extremely secure if the next state of the cryptosystem is self-synchronizing. A self-synchronizing stream cipher is influenced by some of the previously encrypted ciphertext. This mode is

called cipher-feedback (CFB).

The main problem when designing a stream cipher is how to design a keystream generator which generates near-random data bits. Every pseudorandom generator has a certain period after which the sequence of bits repeats. The finite state machine internal to the keystream generator thus needs a sufficient amount of internal state.
If the period of the keystream is too short, different parts of the plaintext will be encrypted in an identical way. A cryptanalyst who finds out about this can consequently decode parts of the ciphertext.

Keystream generators must meet the three basic rules provided by Golomb [3]:

G1. The number of 1's in every period must differ from the number of 0's by no more than one.

G2. In every period, half the runs must have length one, one quarter must have length two, one eighth must have length three etc. as long as the number of runs so indicated exceeds one. Moreover, for each of these lengths, there must be equally many runs of 1's and of 0's.

G3. Suppose we have two copies of the same sequence of period p which are off-set by some amount d. Then for each d, $0 =< d =< p-1$ we can count the number of agreements between the two sequences, A_d, and the number of disagreements, D_d. The auto-correlation coefficient for each d is defined by $(A_d - D_d)/p$ and the auto-correlation function takes on several values as d ranges through all permissible values.

G3 is basically some measure of the ability to distinguish between a sequence and its copy that has been started somewhere in the period.

5. Possible cryptanalytic attacks on stream ciphers

As for any other cipher, *Kerckhoffs' assumption* according to which cryptanalysts have full knowledge of the cipher and have access to plaintext, as well as to ciphertext, remains intact for stream ciphers.

Correlation attack: This attack tries to find a correlation between the output sequence and one or a subset of the internal sequences. If the generator uses two or more independently operating sequence generators and simply combines the results, a weak point of the sequence generated by one generator can reveal the key used to bias another sequence generator. In the next step, the other generators are analyzed in the same way. This divide-and conquer technique is very effective on some types of sequence generators.

Linear consistency attack: The idea behind this attack is to find a solution for $A(K_i)x = b$ with $A(K_i)$ being a matrix filled with a subkey K_i of the complete key K and b being a piece of output sequence. If a solution is found, the correct subkey K_i can be identified. This is pretty well possible if b is long. This attack successively reveals the complete key K.

Linear syndrome method: A successful attack if it is possible to write parts of the output sequence b as $b = a + x$ with a being a piece of a known sequence from a previous period and x being a sequence of only 0's or only 1's which occur sparsely. For sequence generators having a short period length this attack could be successful.

Linear Attack: Some linear combination of the input and output bits of the non-linear function executed by the keystream generator is more often than 50% zero than one (or vice versa). By masking one or more bits of the input bitstream it might be possible to find some subsequences for which the keystream generator is not generating a purely random bitstream. By finding a large number of such subsequences, the keystream generator output can be predicted to be random or not random for a number of bits to follow.

6. Disadvantages of common designs

The majority of symmetric ciphers use an iterated structure. Data is being encrypted by setting up data blocks, resetting the encryption engine and executing the round function depending on the key. Each bit in the key affects each bit in the data block. That consumes processor time. A linear increase in key length l_k results in a quadratic increase in processing time. The more key bits there are, the more subkeys $k_1 \ldots k_r$ must be processed. As the transformation $f(k_i, \ )$ itself depends on the key k and it's size n, the processing time is proportional to the square of key length n^2 or written in short $O(n^2)$. $O(n^2)$ is also the speed limit for stream ciphers. The necessity to use each key bit to encrypt a certain number of plaintext bits dictates at least a linear increase in the number of interactions with each single key bit for lowering the risk of bit bias. Consequently this takes n bits multiplied by n interactions or $O(n^2)$.

As the actual implementations of standard algorithms are well known, there's a latent risk that one day somebody finds a way to predict certain bits in the key and thus reduces the time to break the cipher.
For DES, which was certified in 1977, it took only 25 years until it was possible to crack any DES key. The increase in computer power and advances in cryptanalysis of the algorithm finally helped to break this still widely spread cipher.

There is no guarantee for any conventional block cipher that encrypted data remains private for a long time.

7. Polymorphic Encryption PMC

7.1 Design goals

The idea behind PMC is taking a set of measures to increase the security of known techniques which are used to encrypt data by combining them in the most effective way.
These measures are:

1. Combine a stream cipher design with post-processing methods used in block cipher designs.
2. Create a keystream generator which is nearly as powerful as the one time pad
3. Stacking of randomly selected primitive keystream generators to form one single source for a pseudo-random bitstream
4. Randomize the encryption algorithm itself
5. Increase the available history for all primitive random-number generators to a practical maximum
6. Maximize the processing speed

7.2 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 a few X_i.

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. When used with a shuffle box, they can be regarded as pretty good.

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.

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.
If the history available to the RNG primitives is increased beyond the limits which are normally granted to such generators, even the LCGRNG becomes a winner design. As an example, if s and r are large in an Add-with-carry generator

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

then the period is extremely long. For all existing PMC designs, the history is greater or equal 256 bit.

The presence of a relatively big array of bits which represent the history makes it possible to implement permutation functions for part of the history or even for all of it. This is the point where block cipher design begins to play a role.
An easy-to-implement permutation function uses word-wide Shift-Through-Carry, which is implemented in any modern microprocessor in a barrel shifter topology. It's an extremely fast operation which guarantees quite some shuffling of the history.

The complete structure of a basic, but working PMC model is shown in Fig. 1:

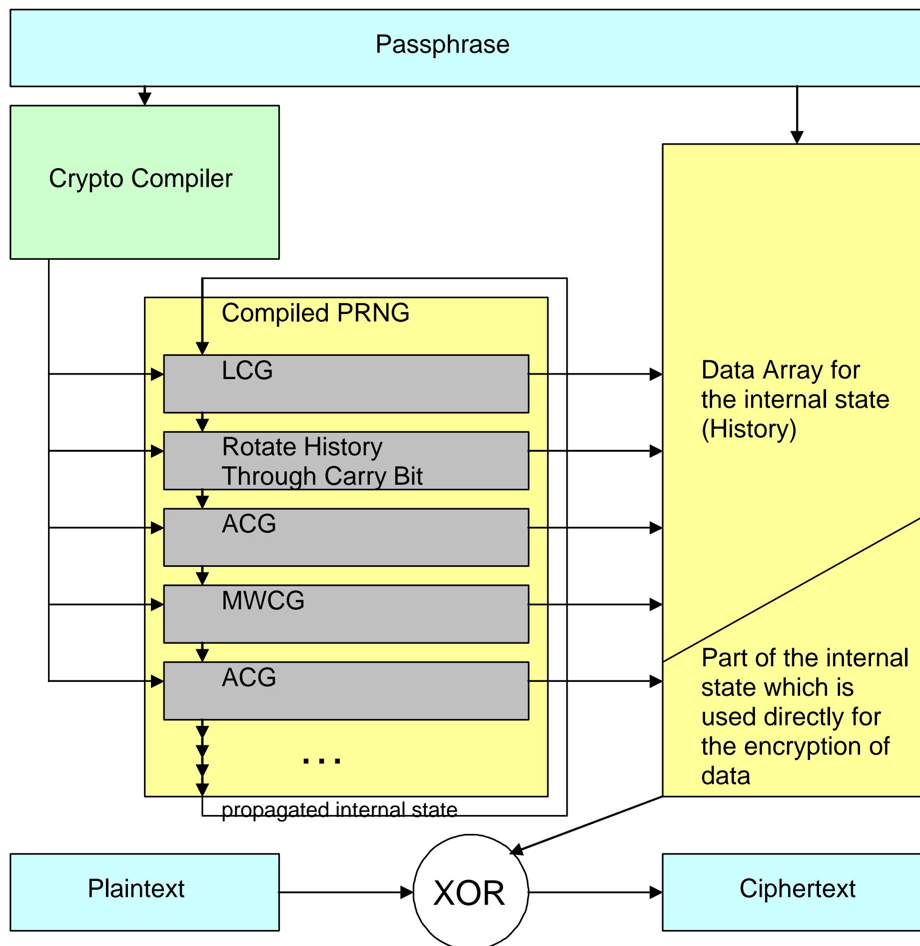

Fig. 1: Structure of a PMC implementation without Cipher Feedback (CFB) and without Cipher Block Chaining (CBC)

The passphrase is compiled into machine code. The compiler simply assembles standardized pseudo-random number generators, the "building blocks", 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.
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.

So far there exists no accurate mathematical model of the above outlined crypto engine as there is no static function.
It must be clearly noted that the size of the array which stores the internal state must be at least as big as the bit pattern of the passphrase in order not to downgrade the cipher. As an example, a 256 bit password being represented as a 128 bit history data array yields a cipher which has no better performance than a 128 bit cipher. In case of a repeated use of the Compiled PRNG, e.g. for the encryption of a large amount of plaintext and even with the data array sized smaller than the length of the passphrase, a higher effective security is yielded. This can only be predicted for a Compiled PRNG consisting of building blocks which pass a number of bits of some propagated internal state to the history data array. As the actual building blocks represent internal state by themselves, this hypothesis is obviously valid.

After the machine code has been executed, 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. By doing this, the complexity of the total crypto system increases and it becomes much more difficult to analyze the internal state of the key data array, although the information it contains in simple one-stage implementations allows no conclusion for the key.

As shown in Fig. 1, part of the internal state of the Compiled PRNG can be used directly for the bitwise XOR operation with the plaintext. For a system which encrypts sectors on a hard disk, the full internal state can be taken for this XOR operation if the sector number is used as seed for the Compiled PRNG. For a 32 bit seed there must be no more than 2^{32} sectors addressable, of course. Otherwise the keystream which is generated for sector 2^{32} is the same as for sector 0. The propagated internal state can e.g. hold the seed in the beginning.

7.3 Design of the primitive pseudo-random number generators (building blocks)

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 a variety of variables: *a*, *b* and *c* can both be chosen from a set of possible values, with c being preferably 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 (*x* and *y*).

```
push ebp;                    // ebp MUST never be destroyed
mov  eax,[ebp+x];            // load history[ebp+x] in AL and history[ebp+x+1] in the next upper
                             // byte of eax and so on up to history[ebp+x+3]
xor  eax,ebx;                // add the propagated history passed on in register ebx from the
                             // preceding primitive PRNG
imul eax,$00000203;          // a = 515, b=5
add  eax,$00000005;
mov  ecx,$000f4241;          // m = 1000001d (m should be a prime number!)
cdq;
idiv ecx;
mov  ebx,edx;                // save modulo part of the result in ebx
xor  [ebp+y],edx;            // change 32 bits in the history data array
pop  ebp;
```

With five different variables which can be chosen freely by the compiler, it is difficult to write this function

down in a “useful” mathematical model.

All other primitive pseudo-random number generators use the same framework as the LCRNG shown above. These 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. The process of choosing primitive PRNGs and defining input and output solely depends on the bit pattern of the passphrase.

The history data array can as well serve as substitution box for primitive PRNG building blocks which perform bit or word permutations. PMC designs are not limited to just one class of primitive RNG functions.

8. Security of PMC

The absence of a detailed mathematical model for the presented cipher makes cryptanalysis a difficult task. As the cipher has as many facets as the number of different password combinations and as the set of primitive PRNGs depends on the actual implementation of the crypto compiler, a more general point of view is appropriate.

8.1 Security of the keystream generator

The fact that this cipher has its roots clearly in stream cipher design makes classic cryptanalysis generally possible.
As the one-time-pad is proven to be unbreakable, only the proof for quasi ideal randomness has to be conducted for the Compiled PRNG.

Any PRNG has the tendency to increase the entropy by definition. The entropy of the input bit pattern is greater than the entropy of the output bit pattern.
The 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 $P_i[F_X(0,i)] > 0$

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 $\{0, 1\} \Rightarrow 0 =< H(i) =< \log_2(2) \Leftrightarrow 0 =< H(i) =< 1$)

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

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

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

8.2 Immunity against different kinds of attacks

Let's first check the three basic rules provided by Golomb [3].
G1 and G2 are likely to be met by any PMC implementation which uses more than 1 primitive PRNG for compiling the complete keystream generator. Any stacked structure consisting of more than a handful of such primitive PRNGs will meet G1 and G2. As each password combination leads to a different Compiled PRNG, a possibly present weakness of a single keystream generator is compensated by different kinds of weaknesses of keystream generators which are compiled for different passphrases.
As G3 is only applicable to static keystream generator implementations, it is easily met by PMC.

Correlation attack: This attack is not applicable on PMC because there is no static combination of independently operating sequence generators.

Linear consistency attack: The Goal is to find a solution for $A(K_i)x = b$ with $A(K_i)$ being a matrix filled with a subkey K_i of the complete key K and b being a piece of output sequence. This is yet another method which relies on the keystream generator to be static. Because of this, a polymorphic keystream generator, which has a highly dynamic structre, reveals nothing about the complete key K or any subkey K_i.

Linear syndrome method: It is unlikely that it might be possible to write parts of the output sequence b as $b = a + x$ with a being a piece of a known sequence from a previous period and x being a sequence of only 0's or only 1's which occur sparsely. With at least 256 bits of history data, short periods are extremely unlikely to occur.

Linear Attack: A polymorphic keystream generator which consists of a number of primitive PRNGs might rarely feature susceptibility to a linear combination of input and output bits. As the key generation function is polymorphic, the linear combination function required to conduct this kind of attack must be polymorphic, too. Consequently this attack is simply not applicable to PMC.

Differential Power Analysis (DPA): DPA is a new kind of attack on Smart Card implementations of cryptographic algorithms. In fact are all implementations on small microprocessors like e.g. 8051, 6505, ST16 in danger. Chari, Jutla, Rao and Rohatgi [4] have proven that all AES candidates, including the AES contest winner Rijndael, can be broken when running on unprotected hardware.

Most microprocessors use CMOS technology. Such chips only consume significant power when a change in the state of the logic occurs. Especially the fetching of an instruction involves many transistors to change their state, but also updating of internal registers as well as reading/writing to RAM makes the processor draw more current than usual.
Each clock edge triggers a sequence of events like charging long internal lines and transistor switching. The microcode, which controls each machine instruction, determines the sequence of these events. For every microprocessor without pipelining, caching and without parallel structures, it is possible to determine the sequence of events. Chari, Jutla, Rao and Rohatgi [4] use the term *relevant state* for that. Asynchronous events resulting from state changes in counters or DMA operations are rare in low end smart cards.
Consequently there's only little noise which could disturb DPA on this kind of device and it is possible to distinguish different states of transistors. Groups of similar operations are clearly distinguishable from other instruction sequences. They are used to trigger the recording of subsequent power states. As the sequence of instructions is known to the cryptanalyst for common ciphers like AES (Rijndael), the co-variance between the bit and power states can be calculated. For the Twofish algorithm it is possible to extract the full key with only 2000 random encryptions!

The only good defence against DPA is the modification of fixed code in order to make it impossible for an adversary to identify the state of any relevant bit within the computation.

For PMC, both the Crypto Compiler as well as the Compiled PRNG are vulnerable to this kind of attack. During the compilation process the instruction code is used as argument for a *switch* statement:

```
switch(instruction_code) {
        case 0:
        …;
        break;

        …

        case n:
```

```
        …;
        break;
    }
```

The *switch* statement sequentially compares the instruction code against constant values. After learning the timing, a cryptanalyst can easily determine the type of primitive PRNG which has been selected by the compiler during the complete compilation process. For applications which require the Crypto Compiler to be DPA-proof, equidistant memory locations for the code assembly subroutines (a simple jump table) do very well. The following pseudo-source code fragment explains this technique:

```
mov  accumulator,instruction_code;  // load data bits which actually select the primitive PRNG
shl  accumulator, 2;                // multiply by 4 to be able to jump into a 32 bit table
and  accumulator, 15;               // set high-order bits to zero
add  accumulator,base_address_of_table;
push accumulator;                   // save the determined address on the stack
ret                                 // jump into the selected code by executing a return
                                    // from subroutine instruction
nop;                                // this could be the instruction for returning safely from a
                                    // primitive PRNG routine
...
```

As always the same instructions are being executed, DPA is much harder to apply on this code fragment. If the high-order bits of the instruction code contain random data, e.g. from a counter, only the *add* instruction can be a source of information for DPA.

The primitive PRNGs themselves must be protected from DPA for critical applications. This is possible by standardising these building blocks. A DPA-proof PMC relies on a set of similar primitive PRNGs which contain a number of dummy instructions. These dummy instructions make all building blocks look identical to a cryptanalyst. By doing this, there's no way to calculate the co-variance between the bit and power states.

DPA is probably the most dangerous kind of attack which is known. As most or probably all existing ciphers can be cracked with DPA, a special PMC design could turn out to be the best cipher for smart card applications or other cryptographic applications running on low-end microprocessors.

9. Postprocessing by employing block cipher design methods

Permutation of plaintext bits is a frequently used technique in common block cipher designs.
E: $\{0,1\}^\ell$? $\{0,1\}^\ell$ denotes an encryption permutation on ℓ bits and E^{-1} is its inverse permutation. There are $\ell!$ different possibilities to permutate ℓ bits.
DES performs a fixed initial permutation. A primitive form of this operation can be executed on a microprocessor in only one machine instruction.
On the other hand a design which relies on permutation alone will be broken very quickly.

A PMC algorithm which uses large amounts of bits from the history data array for XORing the plaintext should at least make it more difficult for a cryptanalyst to look at the internal state of the Compiled PRNG. As the machine instructions *ROR EAX,Y* and *ROL EAX,Y* are quickly executed, the benefit from using them is obvious. The range of permissible values for *Y* is 0 .. 31. *ROR* and *ROL* commands rotate the bit pattern in the 32 bit processor register EAX by 0 .. 31 bit locations. *ROR* and *ROL* mutually reverse their effect on the bit pattern in register EAX if executed with the same argument *Y*. For the shift operation performed by the *ROR* and *ROL* commands, the number of possible permutations is only ℓ.

If the initial permutation depends on some internal state of the Compiled PRNG, the susceptibility to bit bias in the plaintext decreases significantly. The statistics which might be inherent in the plaintext are thus hidden and also the second of Shannon's principles [1], the principle of diffusion, is satisfied by such a PMC cipher.

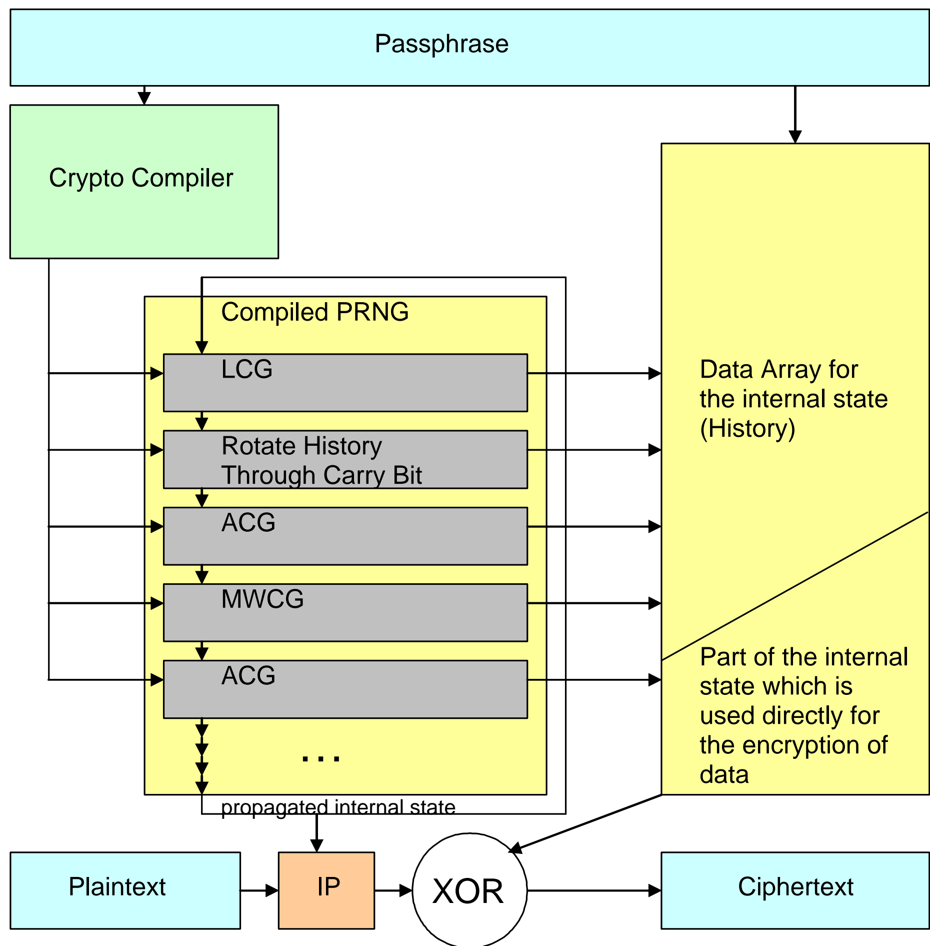

Fig. 2: Structure of a PMC implementation without Cipher Feedback (CFB), without Cipher Block Chaining (CBC), but with variable initial permutation for every 32 bit block in the plaintext

In order to implement initial permutation, a number of 32 bit words may be combined to form a permutation box (or substitution box, shortly S-box). By exchanging 32 bit words, the amount of diffusion can be further increased without excessively slowing down the encryption process.

Another fast operation which is applicable to block ciphers and which increases confusion is a simple arithmetic *add* instruction. Both *add* and *sub* commands are executed in just one clock cycle on modern microprocessors and even on low-end processors 8 or 16 bits of plaintext are processed within 2 .. 4 clock cycles.

If data which is larger than one block is encrypted, a symmetric cipher can be used in a different operating mode than just encrypting one block after the other. Electronic Code Book or shortly ECB is the name for the basic operating mode. In contrast, the different operating mode is called Cipher Block Chaining (CBC). Before a plaintext block m_i is encrypted, it is combined using XOR, with the previously computed ciphertext block. By doing this we yield: $c_i = E_k(m_i \text{ XOR } c_{i-1})$. The ciphertext previous to the very first plaintext block is called the *initialization value* and denoted IV. The IV must not be secret, but if it would be, even a poor cipher would quickly become unbreakable.

The ciphertext can as well be used to change some of the internal state of the keystream generator. By doing this, the keystream which is used to encrypt the next plaintext block is affected. This technique is named Cipher Feedback (CFB). Stream data encryption algorithms using CFB are called self-synchronizing stream ciphers.

For self-compiling crypto code there is a totally new kind of cipher feedback mode available: Cipher Recompilation Mode or Cipher Restructuring Mode (CRM). The author requests comments on the name and on the new postprocessing mode, as well as on this paper: It is possible to recompile the keystream generator after the encryption of each block or after several blocks have been processed. By XORing a hash of the ciphertext or even the full ciphertext with the history (the internal state), the subsequently encrypted plaintext is encrypted with the Best Possible Privacy.

10. Immunity against block cipher attacks

For Polymorphic Ciphers which represent a mixture of stream- and block ciphers, typical block cipher attacks could be tried by cryptanalysts.

Ciphertext-only attacks only rely on encrypted data and thus are not applicable to self-compiling crypto code because the cipher itself changes significantly with different keys.

Known plaintext attacks are a threat to ciphers which are used for encrypting data stored on hard disks. These ciphers can only operate in ECB mode as there is no possibility to feed back ciphertext in any way. Self-compiling crypto code has the advantage that at least the sector number can be used to recompile the complete key generator. Within the boundaries of one sector of a hard disk, a cryptanalyst will have good chances to guess which blocks contain identical data as they might be encrypted in an identical way. Beyond the boundaries of a sector or a small number of adjacent sectors, a polymorphic algorithm gives cryptanalysts no chance to find out anything about the encrypted plaintext or the key. Even if the keystream generator is never recompiled and only the sector number is combined with the initial internal state, there is still the problem for a cryptanalyst that known plaintext attacks generally require the actual encryption algorithm to be constant. Because there is no constant algorithm present, this kind of attack is not applicable to a basic Polymorphic Cipher. The same applies to *Chosen plaintext* and *Chosen ciphertext* attacks.

Differential cryptanalysis is not applicable as well to Polymorphic Ciphers because this class of ciphers doesn't iterate plaintext data through a number of rounds by using subkeys. Consequently only the complete key would have to be attacked making this method as useful is exhaustive search.

Key Bit Bias (or Linear cryptanalysis): This attack uses extensive ciphering of fixed plaintext data under a variety of different keys. As the keystream generator itself looks very different for each key, there is no basis for finding a dependence between key bits and a statistical value of ciphertext bits.

11. Encryption speed of PMC

In contrast to common ciphers, which all come with the inherent speed limit $O(n^2)$ with n being the size of key k, the use of a crypto compiler has a positive effect on processing speed: There is only a linear relationship $O(n)$ for the keysize n and the processing time. The reason for this is pretty simple:
The compiling process of the keystream generator can be generalized as block assembly with a constant number of key bits selecting the next block to be concatenated to the preceding ones. The processing time for that is $O(n)$. The execution time for n primitive PRNGs is $O(n)$, as well as XORing m plaintext bits with some bits from the history data array consumes only the processing time $O(m)$. Consequently the execution time of a Polymorphic Cipher is $O(n)+ O(n)+ O(m)$. As n and m are in the same range this term can be simplified to yield $O(n)$. The higher encryption speed is probably the most decisive advantage of self-compiling crypto code (Polymorphic Encryption Algorithms).

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