Turbo PMC V3 – 1024 Bit Block Cipher for Storage Device Block Level Encryption

C. B. Roellgen

11.06.2008

Abstract

A fast and provably secure Polymorphic Block Cipher consisting of a three-round Luby Rackoff Pseudorandom Permutation Generator with a Decorrelation Stage employing a large number of interdependent pseudo-random number generators, combiner routines and permutation functions is proposed. Data-dependent selection of cryptographic primitives with a shared internal state at runtime provides a novel mode of operation; in conjunction with true polymorphism, the cipher is highly variable while operating in a trusted and provably secure configuration. The proposed encryption algorithm forms a combined secrecy system as described in "Communication theory of secrecy systems" by C.E. Shannon, 1949. The combined secrecy system contains a substantially higher number of variables than available samples that an attacker can use for cryptanalysis. Precompiled Polymorphic Pseudorandom Number Generators enable for portability to all existing 32 and 64 bit microprocessor platforms. The 1024 bit cipher outperforms AES Rijndael with its comparably small 128 bit S-box by 20% in terms of encryption/decryption speed on modern 64 bit microprocessors while preserving the capability to resists to all kinds of attack, including Power Attacks, where AES demonstrably fails. The encryption algorithm is designed for use in the TurboCrypt OTFE (disk encryption software) and similar applications tolerating long key setup time and excessive use of hardware resources.

*Key words: cipher of ciphers, additive combined secrecy system, multiplicative combined secrecy system endomorphic cipher, Luby Rackoff, Feistel, Pseudorandom Permutation Generator, block cipher, polymorphic encryption, polymorphism, compiled pseudo-random number generator stack, provable security, Feistel, power attack, DPA, differential power attack, linear cryptanalysis, differential cryptanalysis, key setup time, distinguishability, random permutation, a*lmost perfect nonlinear permutation*, one-time pad, TurboCrypt, OTFE, disk encryption, storage device,hard disk, sector.*

1. Introduction

With the transition from 32 bit to 64 bit microprocessor architectures, very fast but processor-dependent polymorphic encryption algorithms utilizing a crypto compiler are increasingly perceived as being not sufficiently flexible. Resistance to all known attacks but also the assumption that a cipher can potentially be regarded as vulnerable after some time, a block- and key size of 1024 bit and a DPA proof design appear to be mandatory in the new millennium. It should be noted that classic block ciphers like DES, AES, Twofish, etc. are easily broken with the Differential Power Attack (DPA)[11].

Why block sizes that are significantly larger than what is regarded as being safe for the next millennium?
Orr Dunkelman and Nathan Keller suggest in [10] that it's possible to measure effective linearity of block ciphers using almost perfect nonlinear permutations and to distinguish between them. Complexity O($^{2n/3}$) can potentially be sufficient to gain knowledge about a specific cipher. Dunkelman and Keller point out that effective linearity of a block cipher can be approximately computed with complexity O($2^{n/2}$). The S-box used in AES comes close to an Almost Perfect Nonlinear Permutation. This feature allows to distinguish AES from other ciphers or from a random permutations and it might even be possible to classify strong and weak keys and thus guess the key.
The potential threat of being able to classify a cipher should be minimized for new designs. In the first place, a three-round Luby Rackoff construction is favourable as effective linearity is, according to Dunkelmann and Keller [10], approximately 2 (which is the effective linearity of truly random permutations) and for 1024 bit key size there is still sufficient safety margin down to an effective key size of 512 bit if an attack with complexity O($2^{n/2}$) might be applicable.
What if an attack was found for AES that cuts attack complexity down to O($2^{n/2}$) ?
AES could then be broken instantaneously! As a matter of consequence, wouldn't it be desirable to have a comfortable security margin? If AES had 256 bit S-boxes, such an attack would still be unpractical. But with

the actual 128 bit, an attack with complexity 2^{64} could be mounted very easily – today !

Three-round Luby Rackoff is provably secure. This opens up phantastic possibilities to create highly secure Polymorphic Encryption Algorithms with 8 times the block length of AES and still to outperform AES in terms of encryption speed.

2. Design goals

Design goal	Polymorphic Cipher TPMC V3	AES Rijndael / Twofish and others
Resistance against all known attacks	**TPMC V3 is even DPA proof**, but this is not too important for very complex target machines like modern microprocessors with transistor counts exceeding 100 million transistor equivalents.	**Can be broken easily by DPA (Differential Power Attack) on small microprocessors and microcontrollers [11]**
Resistance against future attacks that can even cut effective key size by ½ or even 2/3	**Cutting of effective key size by ¾ would result in still extremely high complexity of O(2^{256}), which is regarded as totally safe for the next trillion years.**	**Cutting of effective key size by ½ results in an extremely low complexity of O(2^{64}). The cipher would be regarded as being broken. [10]**
Proven security	**Three round Luby Rackoff features proven security [4]; polymorphic encryption is increasingly popular among experts but it's probably impossible to prove security.**	**Security is not proven. Extensive peer review indicates that the cipher could be broken in the future: For 128-bit Rijndael, the problem of recovering the secret key from one single plaintext can be written as a system of 8000 quadratic equations with 1600 binary unknowns. [9]**
Platform independence	Runs on any 32 or 64 bit microprocessor or microcontroller	Runs on any 8-, 16-, 32- and 64 bit microprocessor and microcontroller
Polymorphism and data dependent selection of functions	**Both features make TPMC V3 a completely variable cipher with no static weakness.**	**Classic ciphers are static and can thus be thoroughly reverse-engineered and analyzed. Cryptanalysis of a mechanism that does always exactly the same is somewhat easier than for a mechanism that never does the same operation twice.**
Use of large amounts of resources	**TPMC V3 with 320kbit internal state requires at least approx. 1.000.000 transistor equivalents to run. This alone makes Brute Force Attack more difficult and much more expensive compared with conventional ciphers.**	Less than 50.000 transistor functions are required to build an AES block. **Approx. 1.000.000 AES blocks can run in parallel on an 8'' wafer to try and break a code with Brute Force.**
Extremely long key setup time	**> 100ms on a modern microprocessor make comparably short keys safe against Brute Force attacks conducted on a few machines. Extremely long key setup time extends energy consumption multiplied by the time needed for Brute Force by factor 2.000.000.**	**<1µs help attackers to try each and every password combination. This is highly dangerous if short passwords are being used to protect data.**
Attacks need to be expensive for an attacker	**As TPMC V3 requires a lot of resources and extremely much time for key setup, an attacker requires a "time x resources product" of approx. 2.000.000 times compared with AES Rijndael when using keys with a similar length.**	**Trying different AES keys requires 50.000 transistor equivalents and less than 1µs. This isn't really all that much. This is a REAL weakness.**
Possibility to customize the encryption algorithm so that customizations are conceptionally different	TPMC V3 can be customized in complexity, polymorphic worker functions can be replaced by conceptionally different functions and block size can be adapted.	**Not possible at all.**
High speed	Approx. 920 Mbit/s on an Intel Core Duo 6600 (2.4GHz) (64 bit C++ code)	Approx. > 730 Mbit/s on an Intel Core Duo 6600 (2.4GHz) (64 bit C++ code)

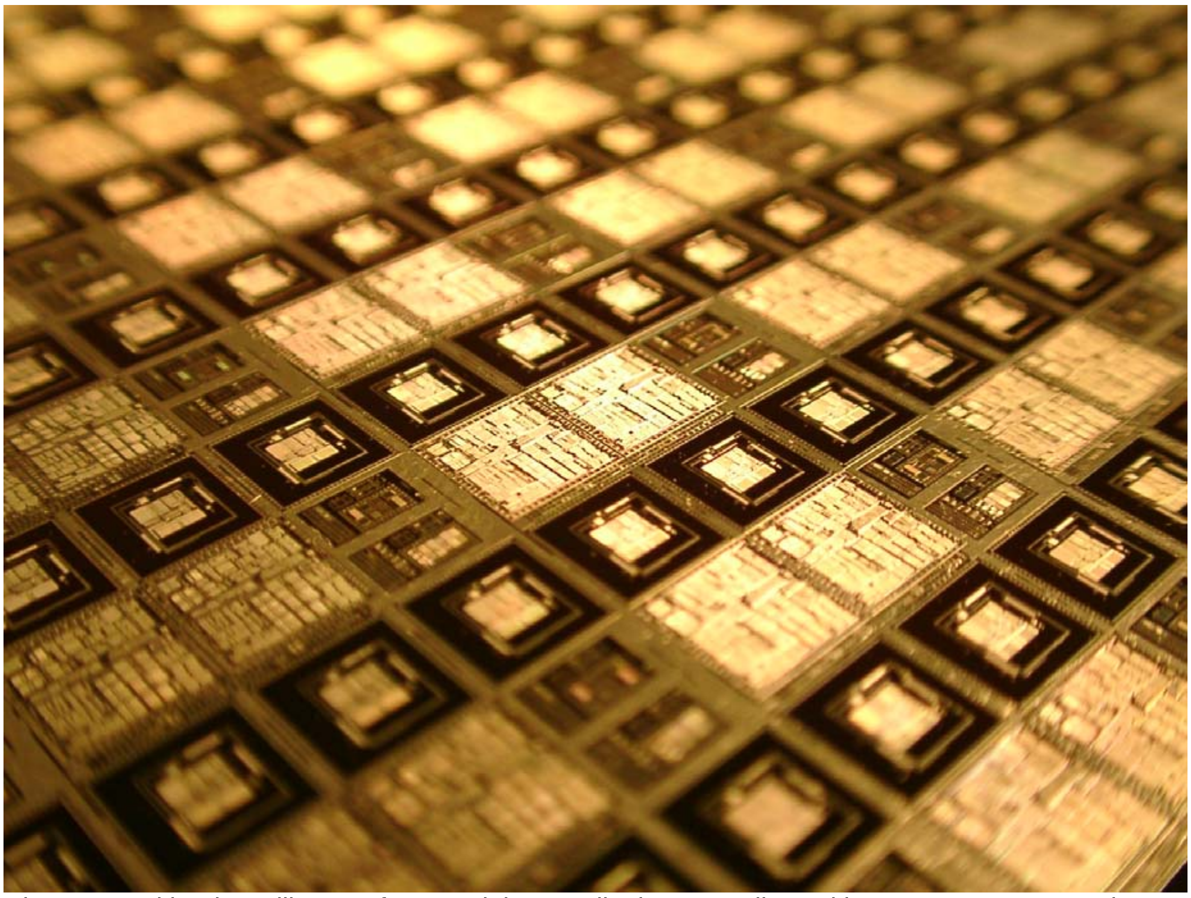
Figure 1: Multiproject silicon wafer containing small microcontrollers with approx. 20.000 transistors

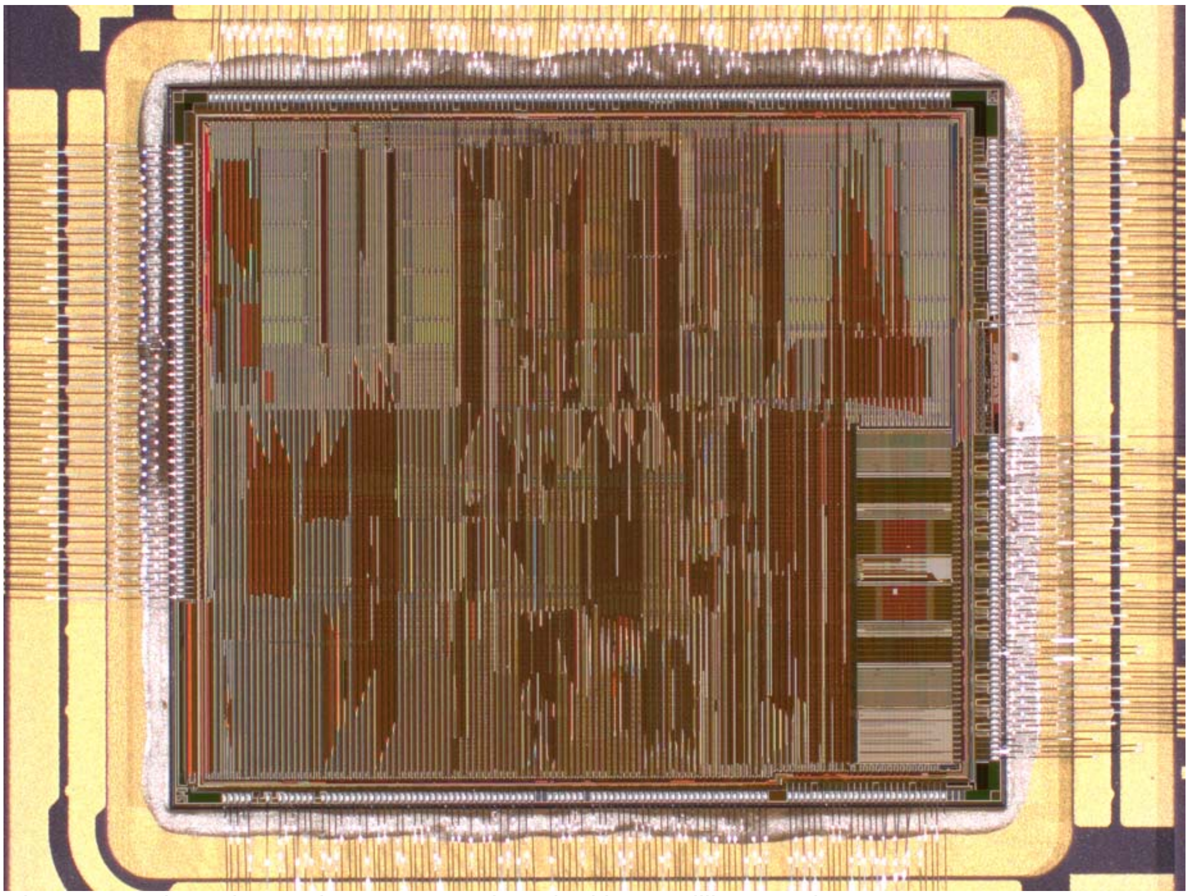
Figure 2: Intel Pentium I microprocessor (1993) with 3.1 million transistors

3. Alleged use of Polymorphic Encryption by Professionals

Polymorphic Encryption (PMC) is increasingly becoming popular among experts. The reason for this is certainly the logic that can be derived from the following example:

A very simple and basic polymorphic cipher requires a crypto engine which can choose from let's say four 128 bit encryption algorithms that are regarded as being pretty secure. The set of ciphers may e.g. consist of AES Rijndael, Twofish, RC6 and Mars.

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 a secure 130 bit cipher. Why? Simply because all four base ciphers are secure and we only select a cipher from this set. Each base cipher generates ciphertext with very good pseudo-randomness and thus cannot be identified by its output bit pattern (at least not easily, although Dunkelman and Keller [10] provide a pratical way to do exactly this as there’s no security margin for our four base ciphers!!!).

The two additional bits make the proposed "Cipher of Ciphers" stronger than each of the base ciphers with the following advantages. Why?
1. Brute Force Attack takes obviously longer on 130 bit than on 128 bit

2. The two additional bits consume only once a little CPU time. They even consume NO CPU time at all during encryption/decryption ! Cipher selection takes a few nanoseconds ONCE, but that’s it. The advantage is obvious.

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. Breaking four base ciphers might still be manageable, but Polymorphic Encryption starts to be fun when there’s a set of several thousand different base algorithms. The advantage is again obvious.

To our knowledge, research on Polymorphic Encryption is currently conducted in the following countries:

China:
Xidian University, Peoples Republic of China, is researching Polymorphic Encryption. The work is supported by the National Laboratory for Modern Communications Foundation of China under Grant No. 51436030105DZ0105 and the National Natural Science Foundation of China under Grant No. 60273084. See the publication of Yin Yi-Feng, Xinshe Li and Yupu Hu from the Key Laboratory of Computer Network and Information Security of Xidian University, Xi'an 710071, People's Republic of China [20] for details.

USA:
STU-III telephone: *“The "keying stream" is a polymorphic regenerating mathematic algorithm which takes a initialization key and mathematically morphs it into a bit stream pattern. The "keying stream" is created by the "Key Generator" and is the actual heart of the STU. A portion of the "keying stream" are then mixed back into to the original key, and process repeated. The results is a pseudo-random bit stream that if properly implemented is extremely difficult (but not impossible) to decrypt.*
Even the most sophisticated cryptographic algorithm can be easily expressed in the form of a simple equation in boolean algebra, with the "initialization keys" being used to define the initial key generator settings, and to provide morphing back to the equation.”
Source: http://www.tscm.com/stu.html

Especially for the USA it is highly unlikely that any information about state secrets leak at all. Privately operating companies will thus change the wording so heavily that it’s even funny to read. The passage above might not contain any useful information at all because every word typically undergoes very strict censorship. The word choice although still gives a clue.

The information about the nuclear superpower China is definitely authentic. It should be noted that use of encryption technology is illegal for civilians in China. Solely the Chinese State has the right to use and to research encryption technology.

4. Polymorphic Encryption with an Interpreter using Precompiled Building Blocks

Since the invention of Polymorphic Encryption in 1999 it has always been our goal to compile an encryption algorithm from the key. By doing this, as many different algorithms exist and are equally probable as there are key combinations. We've suddenly been the only people on the planet who could choose from 2^{8192} ciphers for an 8192 bit encryption algorithm that we implemented into a demo application called "BPP", which sold quite well in Germany, Austria and Switzerland.
Most of the known attacks were rendered inapplicable as the underlying algorithm was totally variable. Imagine 2^{8192} different ciphers and you select one with your password!
For 128 bit the number of possible key combinations and ciphers is still impressing: $2^{128} = 3.40282366921 * 10^{38}$ = 340282366921000000000000000000000000000. This compares compares with a 0 bit information for conventional ciphers like DES, Serpent, Blowfish, Twofish or AES.
When compiling a cipher from a passphrase at runtime, incredibly fast and very flexible code can be compiled. Portability to different microprocessor architectures naturally suffers.

The patent also contains a claim which describes that a Polymorphic Encryption Algorithm can as well be interpreted at runtime. Increasing execution time and less flexibility oppose portability and the possibility to use polymorphism in a construction with proven security. The latter is definitely an advantage. Until 1988 when Luby and Rackoff showed that security can be proved for a three-round process similar to DES [12], only the One-Time Pad featured proven security. I had taken advantage of this in the first designs of Polymorphic Ciphers by providing a superb way to create almost true randomness from a variable pseudorandom generator. Especially fast are runtime-compiled pseudorandom number generators.

Although almost every expert condems the one-time pad, the NSA used one-time tape systems called SIGTOT and 5-UCO. British counterparts were called ROCKEX and NOREEN. Moscow and Washington D.C. communicated after the 1963 Cuban missile crisis with a one-time tape system.
The Soviet KGB intelligence agency often gave its agents one-time pads printed on "flash paper" – paper that was chemically treated by nitric acid so that it would burn almost instantly without leaving too many traces.

Bruce Schneier writes about the One-time pad:
"One-time pads are the only provably secure cryptosystem. Because the key is the same size as the plaintext, every possible plaintext is equally likely. With different keys, the ciphertext DKHS could decrypt to SELL, STOP, BLUE, or WFSH. With a normal algorithm, such as DES or AES or even RSA, you can tell which key is correct because only one key can produce a reasonable plaintext. (Formally, the message size needed is called the "unicity distance." It's about 19 ASCII bytes for an English message encrypted with a cipher with a 128-bit block. With a one-time pad, the unicity distance approaches infinity and it becomes impossible to recognize plaintext. This is the security proof.) Because a one-time pad's key is the same size as the message, it's impossible to tell when you have the correct decryption.

This is the only provably secure cryptosystem we know of.

It's also pretty much useless. Because the key has to be as long as the message, it doesn't solve the security problem. One way to look at encryption is that it takes very long secrets -- the message -- and turns them into very short secrets: the key. With a one-time pad, you haven't shrunk the secret any. It's just as hard to courier the pad to the recipient as it is to courier the message itself. "

The first passage is brillant as it explains why OTP is unbreakable and why using normal algorithms is potentially dangerous: With AES and similar algorithms, only one key can be correct. This enables codebreaking machines to run autonomously !!!
The last passage although contains an important bug: One has most certainly got plenty of time to courier the pad, but if let's say a war needs to be avoided, one might not have the time to courier the message safely within a few minutes. It's pretty logical why real experts have relied on OTP and why they might still do so in very special cases.

The classic OTP algorithm (XOR data with random "noise" from a piece of paper, magnetic tape or likewise) is highly unpractical in todays world, but the underlying principle is useful if it's combined with a good source of pseudorandom noise. I'd like to stress explicitly that it shouldn't be a design goal for an encryption algorithm to enable attackers to identify the key used to encrypt some ciphertext, but a cipher using a key that is shorter than the message to be encrypted would on the other hand be pretty bad if weak keys could be easily identified. It's definitely better for a practical cipher to feature low effective linearity – as low as if real random permutations were applied, but not lower.
The cipher reacts different for each and every key combination, which makes it difficult to keep its output sequence apart from real randomness. The ultimate goal of encryption technology is to make code breaking

difficult. The intuitive notion of characteristics that change with every key combination are definitely a plus. In addition to this have almost perfect permutation functions recently become distinguishable [10]. In contrast to classic commercial ciphers have Polymorphic Ciphers always been designed to be indistinguishable from random permutations. It seems as if we've been right since my invention of Polymorphic Encryption in 1999.

Three-round Luby-Rackoff is an ideal construction to apply Interpreted Polymorphic Encryption as the mathematics behind it allow for a great amount of flexibility for the one-way functions that the block encryption algorithm is constructed from while the hunger for fresh pseudorandomness is limited. Pseudorandom function generators cannot be directly used for block encryption because they are not invertible. Luby and Rackoff were however able to show that there is a way to do so. In 1992 Maurer [4] provided a strongly simplified explanation, which is cited in the next paragraph.
Three-round Luby-Rackoff is a process that comprises a sufficiently high number of operations so that an interpreter for a Polymorphic Encryption Algorithm won't consume an excessive amount of CPU time on the interpretation of atomic tokens. Effective linearity according to Dunkelmann and Keller [10] is the same as of random permutations, which is an ideal design goal for block ciphers.

5. Luby-Rackoff Construction

Luby and Rackoff [12] showed that a provably secure block cipher can be constructed from just three good pseudorandom functions that are used as round functions in a Feistel structure reduced to only three rounds. This paragraph contains a condensed version of the mathematical proof, which largely consists of a citation of [4]. The following paragraphs will be dedicated to to the pseudorandom functions which are used as round functions.

Let $\{0,1\}^n$ denote the set of binary strings of length n, let F^n denote the set of all $(2^n)^{2^n} = 2^{n2^n}$ functions $\{0,1\}^n \to \{0,1\}^n$, and let P^n denote the subset of functions of F^n that are permutations of $\{0,1\}^n$. For $f_1 \in F^n$ and $f_2 \in F^n$, $f_1 \circ f_2$ denotes the composition of f_1 and f_2 : $f_1 \circ f_2(x) = f_2(f_1(x))$.

For two binary strings a and b, $a \bullet b$ denotes their concatenation. If a and b have the same length, $a \oplus b$ denotes their bitwise *exclusive or* combination.

Motivated by the Feistel round structure of the DES cipher, Luby and Rackoff defined a mapping H: $F^n \times F^n \times F^n \to P^{2n}$ assigning every triple of functions in F^n a permutation in P^{2n}. In other words, three functions F^n working with binary strings of length n are combined to create a set of permutation functions H of P^{2n} that map binary strings of twice the length n.

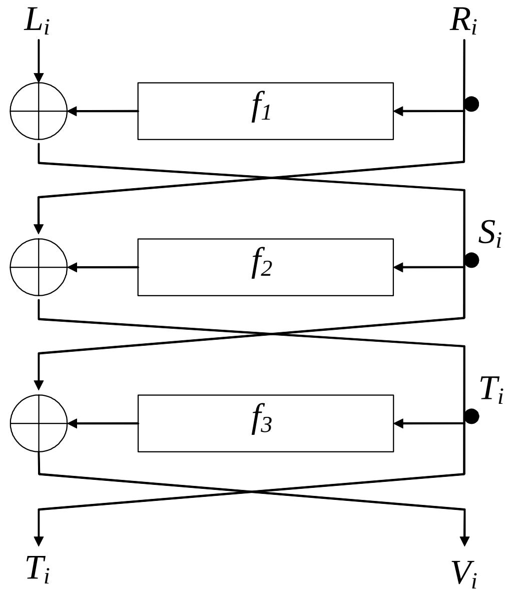

Figure 3: Three-round Luby-Rackoff construction

Mathematically, this mapping looks as follows:

Let L and R denote the left and right half of a $2n$ – bit string $L \bullet R$ and let for $f \in F^n$ the permutation $\overline{f} \in P^{2n}$ be defined as

$$\overline{f}(L \bullet R) = R \bullet [L \oplus f(R)] ,$$

i.e., the right half of the argument appears unchanged while the left half of the result equals $L \oplus f(R)$. This corresponds in principle with one round of DES (Data Encryption Standard, a design that was derived from Horst Feistel's Lucifer cipher and that became the commercial standard after IBM had been "convinced" by the NSA that a reduced key size was sufficient).
For a list of functions, $f_1,..., f_s \in F^n$, let the permutation function $\psi(f_1,..., f_s) : \{0,1\}^{2n} \to \{0,1\}^{2n}$ be defined by

$$\psi(f_1, f_2,..., f_s) = \bar{f}_1 \circ \bar{f}_2 ... \circ \bar{f}_s ,$$

i.e., $\psi(f_1,..., f_s)(L \bullet R) = \bar{f}_s\left(\bar{f}_{s-1}\left(...\bar{f}_1(L \bullet R)...\right)\right)$. The mapping H can now be exactly defined by $H(f_1, f_2, f_3)(L \bullet R) = \psi(f_1, f_2, f_3)$ (cf. Figure 3), where

$$\psi(f_1, f_2, f_3)(L \bullet R) = [R \oplus f_2(L \oplus f_1(R))] \bullet [L \oplus f_1(R) \oplus f_3(R \oplus f_2(L \oplus f_1(R)))] .$$

The decisive question is the security of this construction. Luby and Rackoff broke this problem to calculating the probabilty for being able to distinguish F^{2n} from a function randomly chosen from the much smaller set $\psi(F^n, F^n, F^n)$. An oracle circuit is supposed to do this job. An oracle circuit C_{2n} is a circuit with gates consisting of $2n$ input and $2n$ output gates where all oracle gates in a circuit evaluate the same fixed function in F^{2n}.

Let $g : (\{0,1\}^{2n})^k \to \{0,1\}$ be a function taking as input k $2n$ – bit strings. For a given set of k arguments $x_1, x_2,..., x_k$, let

$$P[g(f(x_1), f(x_2),..., f(x_k)) = 1 : f \in_R \psi(F^n, F^n, F^n)]$$

and

$$P_g \overset{\Delta}{=} [g(f(x_1), f(x_2),..., f(x_k)) = 1 : f \in_R F^{2n}]$$

be defined as the probabilities that $g(f(x_1), f(x_2),..., f(x_k)) = 1$ when f is chosen randomly from $\psi(F^n, F^n, F^n)$ and from F^{2n}, respectively.

If the two probabilities are equally likely, their difference is zero. If one of the two probabilities is less or more likely than the other, i.e. if the oracle is able to create a like between $f \in \psi(F^n, F^n, F^n)$ and $f \in F^{2n}$, the absolute value of the difference of both probabilities is high. If an upper limit for this difference of probabilities exists and this limit is very small, three-round Luby Rackoff would be proven secure.

Lemma 1. *For every function* $g : (\{0,1\}^{2n})^k \to \{0,1\}$ *and for every set of* k *arguments* $x_1,..., x_k$,

$$\left| P[g(f(x_1), f(x_2),..., f(x_k)) = 1 : f \in_R \psi(F^n, F^n, F^n)] - P_g \right| \leq \frac{k^2}{2^n} .$$

Proof of Lemma 1. Let f_1, f_2 and f_3 be functions randomly chosen from F^n, and let $f = \psi(f_1, f_2, f_3)$. Let

$x_i = L_i \bullet R_i$ for $1 \le i \le k$ be the k arguments of *f*, and define S_i, T_i and V_i for $1 \le i \le k$ as follows (cf. Figure 3):

$$S_i = L_i \oplus f_1(R_i)$$

$$T_i = f_2(S_i) \oplus R_i$$

and

$$V_i = f_3(T_i) \oplus S_i .$$

Note that when the evaluation of f for the argument x_i is viewed as a three-round process (similar to three rounds of DES), the outputs of the first, second and third round are $R_i \bullet S_i, S_i \bullet T_i$ and $T_i \bullet V_i = f(L_i \bullet R_i)$, respectively. We may for the rest of the proof assume, without loss of generality, that the x_i , $1 \le i \le k$, are distinct. Choosing identical arguments provides no new information and can thus certianly not help.

Let ε_S and ε_T denote the events that $S_1,...,S_k$ as well as $T_1,...,T_k$ are distinct. Let ε further be the event that both ε_S and ε_T occur. As a matter of consequence, $T_1 = f_2(S_1) \oplus R_1, T_2 = f_2(S_2) \oplus R_2, \ldots, T_k = f_2(S_k) \oplus R_k$ are completely random because f_2 is a random function and hence $f_2(S_1), f_2(S_2),..., f_2(S_k)$ are completely random. Similarly, if ε_T occurs, then $V_1 = S_1 \oplus f_3(T_1), \ldots, V_k = S_k \oplus f_3(T_k)$ are completely random because f_3 is a random function. Thus if both ε_S and ε_T occur, $f(x_1) = T_1 \bullet V_1,..., f(x_k) = T_k \bullet V_k$ are completely random and thus $f = \psi(f_1, f_2, f_3)$ behaves precisely like a function chosen randomly from F^{2n}.

Therefore the distinguishing probability is upper bounded by

$$\left| P\left[g\left(f(x_1), f(x_2),..., f(x_k)\right) = 1 : f \in_R \psi(F^n, F^n, F^n)\right] - P_g \right| \le 1 - P[\varepsilon] .$$

We now derive an upper bound for $1 - P[\varepsilon] = P[\bar{\varepsilon}]$, where $\bar{\varepsilon}$ denotes the complementary event of ε. ε is the union of the $\binom{k}{2}$ events $\{S_i = S_j\}$ for $1 \le i < j \le k$ and the $\binom{k}{2}$ events $\{T_i = T_j\}$ for $1 \le i < j \le k$.
The probability of the union of several events is upper bounded by the sum of the probabilities, and hence

$$1 - P[\varepsilon] = P[\bar{\varepsilon}] \le \sum_{1 \le i < j \le k} P[S_i = S_j] + \sum_{1 \le i < j \le k} P[T_i = T_j] .$$

Since f_1 is a random function, and for $i \ne j$ we have

$$P[S_i = S_j] = \begin{cases} 2^{-n} & if \quad R_i \ne R_j \\ 0 & if \quad R_i = R_j \end{cases}$$

which further simplifies to yield

$$P[S_i = S_j] \le 2^{-n}$$

for $i \ne j$ simply because $S_i \ne S_j$ when $R_i = R_j$ because f_1 is a random function.
By a similar argument we obtain

$$P[T_i = T_j] \le 2^{-n}$$

for $i \neq j$.

For the upper bound $1 - P[\varepsilon] = P[\bar{\varepsilon}]$ we finally yield

$$1 - P[\varepsilon] = P[\bar{\varepsilon}] \leq \binom{k}{2} \cdot 2^{-n} + \binom{k}{2} \cdot 2^{-n} = 2 \cdot \left(\frac{k!}{2! \cdot (k-2)} \right) \cdot 2^{-n} = 2 \cdot \frac{k(k-1)}{2 \cdot 2^n}$$

$$\Leftrightarrow \quad 1 - P[\varepsilon] \leq \frac{k(k-1)}{2^n}$$

As $k(k-1) < k^2$, Lemma 1 follows.

As long as the functions f_1, f_2 and f_3 are pseudorandom functions, the three-round Luby-Rackoff construction features proven security. The proof of Lemma 1 also works well if invertibility (permutations are invertible) is not required. Good stream ciphers, hash functions block ciphers are all suitable and have been successfully used in the past to create Luby-Rackoff block ciphers.

6. Decorrelation Modules

In order to discourage attackers to apply differential- or differential-linear cryptanalysis and try to analyze the round functions, it's potentially advisable to break up correlations of plaintext bit patterns.

In [13], Naor and Reingold propose an encryption $E = DM_2 \circ F_2 \circ F_1 \circ DM_1$ where DM_i is a decorrelation module (actually a strongly universal hash function) and F_i is one Feistel round with a keyed pseudorandom function as round function. The result is a secure block cipher that cannot be distinguished from a random permutation using chosen plaintext/ciphertext attacks.

Decorrelation can alternatively be achieved by applying another proven method [14]: the Vernam cipher (One-time pad). Intuitively, if E has a perfect 1-wise decorrelation (the key is only used once), the encryption $E(x_1)$ contains no information on the plaintext block x_1, so the cipher E is unconditionally secure if we use it only once as one-time pad. This corresponds with Shannon's perfect secrecy theory [5]. OTP is a highly unpractical method to encrypt and decorrelate data, but for a Polymorphic Cipher, it is possible to take advantage of its mathematical simplicity.

Entropy for the case that x is a plaintext block and X is a random variable such that $X \neq x$ is defined as

$$H(X) = -\sum_x P[X = x] \cdot \log_2(P[X = x])$$

Due to the fact that the definition required $X \neq x$, $P[X = x]$ must be 0 and $H(X) = 0$. With no entropy at all, it is impossible for an adversary to gain knowledge using chosen plaintext/ciphertext attacks or any other attack.

A reliable method to create a high-quality pseudorandom bitstream is consequently all that is required to create a provably secure three-round Luby-Rackoff cipher with a decorrelation module that is based on the modus operandi of the one-time pad. The creation of high-quality pseudorandom bitstreams is something for which polymorphic base functions have proved to be very useful.

7. Generation of High-Quality Pseudorandom Bitstreams

The decorrelation module as well as the three round functions in a Luby-Rackoff structure require good pseudorandom functions, preferrably ones that are more than difficult to analyse. This is where a fast interpreted polymorphic cipher becomes polymorphic.

7.1 Dynamically selected Pseudorandom Number Generators

Unknown but cryptographically weak pseudorandom generators can only be identified by looking at their output sequence. A real spectacular example is the Linear Congruential Generator $x_{i+1} \equiv (3x+5)\,\mathrm{mod}15$. For the start value $x \equiv 7$, the output sequence becomes $7,11,8,14,2,11,8,14,2,11,...$. Without knowing the recurrence relation $x_{i+1} \equiv (3x+5)\,\mathrm{mod}15$, it is possible to identify it after taking only a few samples.

Let $P_b[Y_{i+b} = y_{i+k}]$ be the probability that an oracle can guess output values Y_i after recording and analysing the last b output values; after k output values have been recorded, let the oracle be able to identify the pseudorandom number generator RNG and to predict all following output values.

For $P_b[Y_{i+b} = y_{i+k}]$ we have:

$$P_b[Y_{i+b} = y_{i+k}] = = \begin{cases} 1 & if \quad b \geq k \\ 0 & if \quad b < k \end{cases}$$

Theorem 1. *Let Z and Y denote left and right parts of a binary string and let $Z_{i+1} \bullet Y_{i+1} = PRNG_{Yi}(Z_i \bullet Y_i)$ be the recurrence relation of a set of similar but not identical pseudorandom number generators. Before executing a pseudorandom number generator, the actual function is selected by the Y part of the binary string that was output upon the last execution of a (probably different) pseudorandom number generator function. As long as a different pseudorandom number generator is selected before the oracle is able to gain sufficient knowledge about its identity, $P_b = 0$.*

Proof. $P_b[Y_{i+b} = y_{i+k}]$ is defined to be 0 if $b<k$. The oracle is not given more than k samples of the output sequence of one specific pseudorandom number generator function. Thus it is unable to make any prediction at all.

Another, much more hypothetical assumption for an oracle, may be the ability to predict on average every *n*- th output value of a pseudorandom number generator function, as well as the ability to preditct which pseudorandom number generator function is used on average for every *m*- th output value.

Again we have for the recurrence relationship

$$Z_{i+1} \bullet Y_{i+1} = PRNG_{Yi}(Z_i \bullet Y_i)$$
$$Z_{i+2} \bullet Y_{i+2} = PRNG_{Yi+1}(Z_{i+1} \bullet Y_{i+1})$$
$$\ldots$$
$$Z_{i+j+1} \bullet Y_{i+j+1} = PRNG_{Yi+j}(Z_{i+j} \bullet Y_{i+j})$$

For the probability of the oracle to be able and guess the first output value $P_b[Z_{i+1} \bullet Y_{i+1} = y_g]$, and by assuming that the value $Z_i \bullet Y$ is known, we yield

$$P_b[Z_{i+1} \bullet Y_{i+1} = y_g] = \frac{1}{n}$$

which is identical to the probability to guess each of the parts of the binary output string

$$P_b[Z_{i+1} \bullet Y_{i+1} = y_g] = P_b[Z_{i+1} = y_g] = P_b[Y_{i+1} = y_g] = \frac{1}{n}$$

Due to the fact that the oracle needs to predict the actual function as well as the output value, we yield for the prediction probability of the oracle after executing the recurrence relationship two times

$$P_b\left[Z_{i+2} \bullet Y_{i+2} = y_g\right] = \frac{1}{n} \cdot \frac{1}{m}$$

After j executions we yield

$$P_b\left[Z_{i+j+1} \bullet Y_{i+j+1} = y_g\right] = \frac{1}{n} \cdot \left(\frac{1}{n} \cdot \frac{1}{m}\right)^{j-1}$$

Even if an oracle as powerful as outlined above was available to an attacker, the sequential execution of a Polymorphic Pseudorandom Number Generator can potentially render comparably insecure base functions very powerful. It should be noted that if the very same pseudorandom number generator function $PRNG_{Yi}$ was used again and again, the probability of the oracle to be able and guess output values would remain at $1/n$.
A clever design of a pseudorandom number generator using dynamic base function selection will not only take history into account for the selection of base functions, but also keying information and, if available, other data.

7.2 Compiled Pseudorandom Number Generator stack forming a multiplicative/additive combined secrecy system

Very fast Polymorphic Cipher designs so far have always relied on the strength of compiled cryptographic base functions. A crypto compiler is used to compile an algorithm directly from a key. Each key thus generates one unique cipher or a stack consisting of different pseudorandom number generators (PRNGs). Throughout the first part of this chapter it is assumed that a crypto compiler compiles identical Linear Congruential Generator (LCG) primitives to form a PRNG stack that operates with an internal state that is shared by all compiled PRNGs. PRNGs pass 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).

$$\begin{gathered} X_{n+1} = (a \cdot X_n) \bmod M \\ X_{n+2} = (a \cdot X_{n+1}) \bmod M \\ X_{n+3} = (a \cdot X_{n+2}) \bmod M \\ \cdots \end{gathered}$$

As there are three unknowns (a, X_n and M), consequently three consecutive samples are sufficient to break this generator.

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

$$X_{n+1} = (r(n) \cdot X_n) \bmod M \;\; ;$$

with r(n) being a sequence of numbers randomly selected by the crypto compiler

yielding the congruential sequence

$$X_n = (r(0) \cdot X_0 + r(1) \cdot X_1 + \ldots + r(n) \cdot X_n) \bmod M$$

The minimum number of samples required to determine all unknowns equals $n+2$. The unknows are r(0), $r(1)$, .., $r(n)$, X_0, M. An opponent gets n samples to try and break the stack but has to deal with $n+2$ unknowns. This is impossible.

LCGs are good examples for base functions that are comparably insecure, but that can be hardened by using them in a stack of compiled base functions. Almost any function can be added to such a stack – even complete ciphers like DES, Magenta, RC6 or AES. Such base functions are very easy to parameterize: The crypto compiler simply assigns a key to such base functions.

Faster and much smaller base functions that can be stacked more often than slow and complex base functions include:

Add-with-carry generators (ACG):

$$X_n = (X_{n-s} + X_{n-r} + \mathrm{carry}) \bmod M$$

These generators have long periods, easily exceeding 10^{200}, and they are faster than LCGs.

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

$$X_n = (\mathrm{a}X_{n-1} + \mathrm{carry}) \bmod M$$

Multiplier *a* can be chosen from a large set of integers without affecting the period of around 2^{31}-1 for 32 bit implementations. MWCGs easily pass standard randomness tests.

Add-with-carry generators can feature 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 PRNGs are concatenated to form one single PRNG, security holes of each primitive PRNG are filled easily. Such 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.

A very useful application of a Compiled Pseudorandom Number Generator is the re-keying of potentially weak functions in a cipher.
P. C. Yeh and 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 limited number of PRNGs form a polymorphic pseudorandom function $f(x)$ with the following program-controlled recurrence relation:

$$X_n = \begin{cases} LCG(X_{n-1}, a) & if \quad PROG\_SEQ[n] = 0 \\ ACG(X_{n-1}, s, r) & if \quad PROG\_SEQ[n] = 1 \\ \cdots & if \quad \ldots \\ MWCG(X_{n-1}, a, carry) & if \quad PROG\_SEQ[n] = m \end{cases}$$

with a ,s ,r and carry being numbers selected pseudo-randomly by the crypto compiler and PROG_SEQ[n] being a program sequence. There exist m conceptually different PRNG base functions

The program sequence *PROG_SEQ[]* can be of almost arbitrary size. Initialization of this sequence is a keyed operation that can be performed during setup of the encryption context. The process is preferrably part of the key expansion step.

The polymorphic pseudorandom function $f(x)$ is the sum of *m* PRNGs. This system can be described as the sum of operations O with each O_i being a specific PRNG corresponding to key choice i, which has probability p_i:

$$O = p_1 O_1 + p_1 O_1 ... + p_m O_m$$

When executing the polymorphic pseudorandom function $f(x)$ repeatedly, the multiplicative system P is formed:

$$P(n) = O(n)O(n-1)...O(0)$$

It should be noted that P is not commutative. After n iterations, one out of nm different multiplicative/additive PRNG systems has been executed.

The principle can additionally be employed to create a polymorphic pseudorandom function $f(x)$ that resists power attacks (SPA, DPA). A battery of almost identical PRNGs which compile into almost identical machine code is employed. The following assumptions need to be made for almost identical machine code:

- Execution times of interchanged instructions must be identical
- Power consumption of interchanged instructions must be identical

As an example, the following two lines of C code do compile into machine instructions with absolutely identical current consumption:

```
        a=b*params[0x05];
        a=b*params[0x0a];
```

The only difference is the array index. As 0x05 equals 0101b and 0x0a equals 1010b, both bit patterns lead to exactly the same number of logical gates to change their state if these gates have previously output 1111b or 0000b and will revert back to 1111b or 0000b. This precondition needs to be checked for true power attack proofness.

For OTFE software the risk of power attacks is close to zero. Computer hardware would need to be extensively altered and modern microprocessors with more than 100 million transistors, quiescent currents of more than 10A and a total current consumption in excess of 80A virtually prevents such kind of attack even in a laboratory. There still remains a small risk to become a victim of a timing attack as this is a software-only attack (at least it could be carried out with software only). It is certainly desirable to use a set of PRNGs that feature almost the same power consumption as well as exactly the same timing.

This is made possible by calling so-called delegate functions. All delegates are compiled into almost identical machine code. By exchanging xor, subtraction or addition functions, all base PRNG functions become conceptually different although remaining similar.

Geore Marsaglia's KISS (keep it simple stupid) PRNG is well-suited to create highly similar PRNG functions. The operations of the modified 32 bit KISS PRNG highlighted in red colour can be substituted by subtraction and logical exclusive or operations:

```
        // + + + + (modified) KISS PRNG
unsigned int kiss(unsigned int *z,unsigned int *w,unsigned int *jsr,unsigned int *jcong)
{
  unsigned int result=0;

  *z=36969*(*z & 0x0000FFFF)+(*z >> 16)+((*jcong >> 15) & 0x00000FFF);

  *w=18000*(*w & 0x0000FFFF)+(*w >> 16)+((*jsr >> 3) & 0x00000FFF);
  *jcong=69069*(*jcong)+1234567;
  *jsr=*jsr ^ (*jsr << 17);
  *jsr=*jsr ^ (*jsr >> 13);
  *jsr=*jsr ^ (*jsr << 5);
  result=(((*z << 16)+(*w  & 0x0000FFFF)) ^ (*jcong))+(*jsr);

  return(result);
}
```

The compiler generates the following code for the line:

```
  *z=36969*(*z & 0x0000FFFF)+(*z >> 16)+((*jcong >> 15) & 0x00000FFF);
00551F65 8B 45 08          mov         eax,dword ptr [z]
00551F68 8B 08             mov         ecx,dword ptr [eax]
00551F6A 81 E1 FF FF 00 00 and          ecx,0FFFFh
00551F70 69 C9 69 90 00 00 imul         ecx,ecx,9069h
00551F76 8B 55 08          mov         edx,dword ptr [z]
00551F79 8B 02             mov         eax,dword ptr [edx]
00551F7B C1 E8 10          shr         eax,10h
00551F7E 03 C8             add         ecx,eax
00551F80 8B 55 14          mov         edx,dword ptr [jcong]
00551F83 8B 02             mov         eax,dword ptr [edx]
00551F85 C1 E8 0F          shr         eax,0Fh
00551F88 25 FF 0F 00 00    and         eax,0FFFh
00551F8D 03 C8             add         ecx,eax
00551F8F 8B 55 08          mov         edx,dword ptr [z]
00551F92 89 0A             mov         dword ptr [edx],ecx
```

The red lines contain the two arithmetic operations marked with red colour in the C source code above. In case of a subtraction operation, a **sub ecx,eax** command and in case of an exclusive or operation, an **xor ecx,eax** command would be compiled. All three machine instructions consume exactly the same CPU time and are prefetched in an identical way. Different KISS PRNG versions thus are indistinguishable to an attacker who cannot trace instructions.

The proposed strategy differs from the strategy suggested in [15]: J.-S. Coron and L. Goubin suggest to use a secret sharing scheme so that each intermediate that appears in the cryptographic algorithm is splitted. An attacker would have to analyze multiple point distributions, making his task grow exponentially in the number of elements in the splitting.
As splitting requires true randomness to be available on a very large scale as well as an excess of machine instructions, it is certainly better to select subfunctions with an identical power signature pseudorandomly and to execute them.

The combination of all previously described methodologies form the multiplicative/additive combined pseudorandom number generator stack which the pseudorandom functions for the Luby-Rackoff structure and the decorrelation module are derived from. It accounts for more than 60% of the Turbo PMC V3 source code.

The figure below shows the structure of the complete crypto context, which is generated during the initialization phase of the cipher (key setup/key expansion):

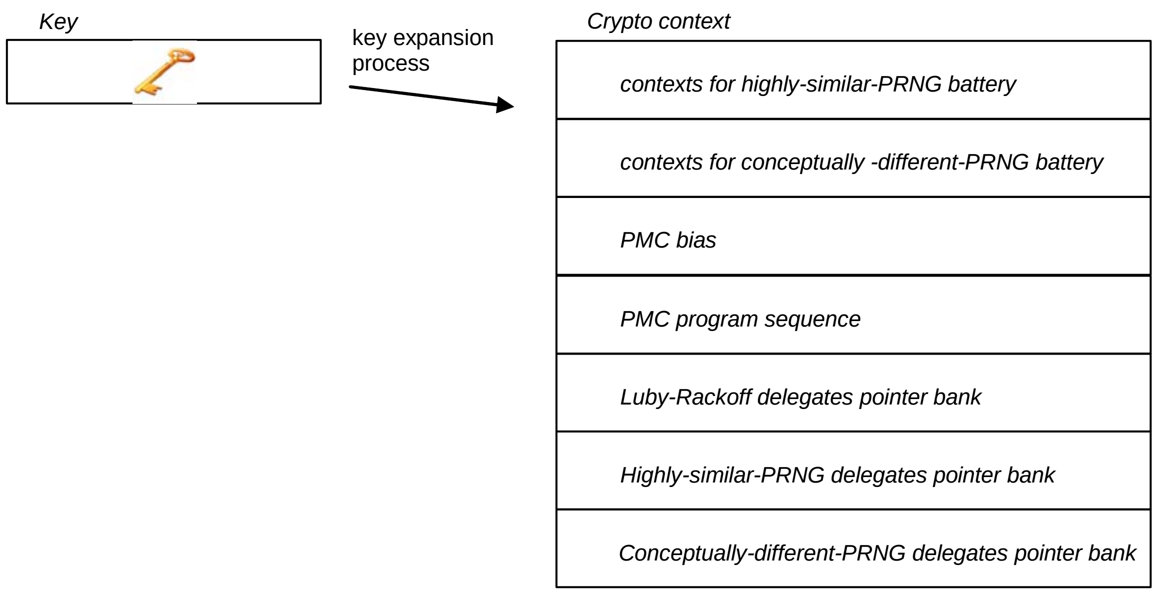
Key
key expansion process
Crypto context
contexts for highly-similar-PRNG battery
contexts for conceptually -different-PRNG battery
PMC bias
PMC program sequence
Luby-Rackoff delegates pointer bank
Highly-similar-PRNG delegates pointer bank
Conceptually-different-PRNG delegates pointer bank

Figure 4: Crypto Context of Multiplicative/Additive Combined Pseudorandom Number Generator Stack

7.3 Parameterizing and operating the multiplicative/additive combined pseudorandom number generator stack in the context of the complete cipher

Turbo PMC V3 was intentionally conceived as an algorithm that is fast, but complex, challenging to analyse and greedy for resources. The decisive design idea for the rapid interpretation of PRNGs is the ability of all PRNG base functions to self-supply with parameters. Administrative overhead is held low while frequent use of polymorphic pseudorandom functions (highly conceptually different as well as timing-attack-proof highly similar PRNGs) forces an attacker to analyse multiple paths. This makes his task grow virtually exponentially in the number of calls to polymorphic pseudorandom functions. Similar to character objects in computer games are polymorphic pseudorandom functions granted access to a large number of elements in the crypto context. These functions fetch parameters freely and they modify a smaller number of elements – the ones that are computed freshly during context reset - freely.

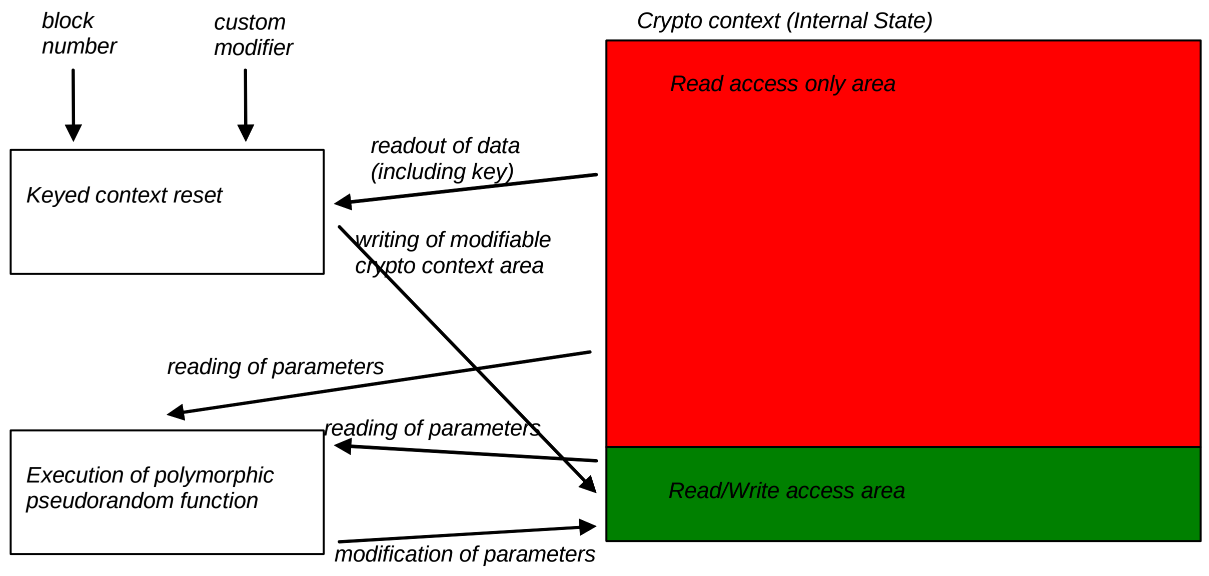
block number
custom modifier
Crypto context (Internal State)
Read access only area
Keyed context reset
readout of data (including key)
writing of modifiable crypto context area
reading of parameters
reading of parameters
Execution of polymorphic pseudorandom function
Read/Write access area
modification of parameters

Figure 5: Polymorphic pseudorandom functions parameter access scheme and context reset

8. The cipher

The cipher Turbo PMC V3 consists of

- Initialization of modifiable crypto context area
- an initial decorrelation step comprising interpreted execution of polymorphic pseudorandom functions
- three Luby-Rackoff rounds with interpreted execution of conceptually similar Luby-Rackoff delegate functions that in turn execute interpreted polymorphic pseudorandom functions as round functions

The following figure exhibits interpretation sequence and associations of function blocks with specific storages within the crypto context for data encryption:

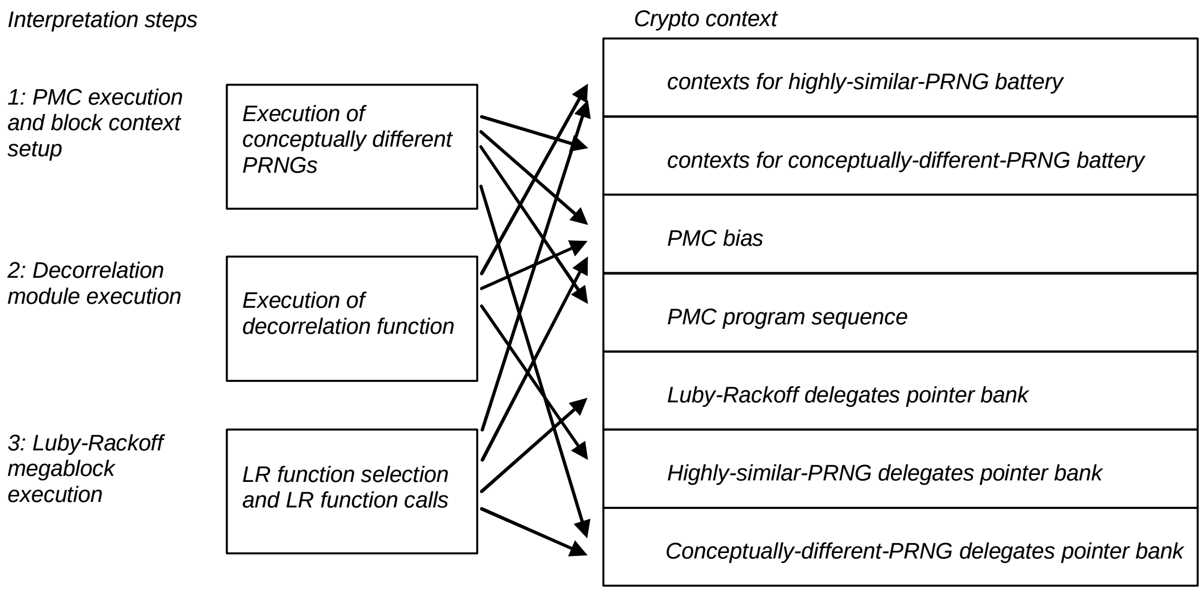
Interpretation steps
Crypto context
1: PMC execution and block context setup
Execution of conceptually different PRNGs
2: Decorrelation module execution
Execution of decorrelation function
3: Luby-Rackoff megablock execution
LR function selection and LR function calls
contexts for highly-similar-PRNG battery
contexts for conceptually-different-PRNG battery
PMC bias
PMC program sequence
Luby-Rackoff delegates pointer bank
Highly-similar-PRNG delegates pointer bank
Conceptually-different-PRNG delegates pointer bank

Figure 6: Function Blocks and Interpretation sequence

8.1 The inverse cipher

The structure of Turbo PMC V3 is such that the sequence of transformations of its inverse is equal to that of the cipher itself, with the transformations executed in reverse direction.

8.2 Target CPU platforms

Turbo PMC V3 is especially compatible with x86, x64 (AMD64) and IA64 platforms. Compatibility with any other standard 32 or 64 bit CPU platform is highly likely.

8.3 Hardware suitability

In contrast to conventional ciphers, great care was taken during the design of Turbo PMC V3 that hardware implementations require an enormous amount of chip space. 100% of the crypto context (internal state) is potentially accessed at any time. This either requires to keep the modifiable portion of the internal state in cache memory or to use fast static RAM for the complete internal state. 40kbyte in SRAM translates into 40.000 x 8 bit x 6 transistors = 1.920.000 transistors. Due to the fact that almost all functions require memory lookup, there’s not much choice but to implement a universal microprocessor unit and ROM in order to execute the algorithm. A 80386 CPU consists of only 275.000 transistors. 50kB in ROM translates into a little more than 400.000 transistors. This sums to yield a minimum transistor count of 320.000 for the internal state (if DRAM is used) plus 275.000 for the CPU and 400.000 for ROM = 995000. A faster variant with SRAM requires chip space for 2.595.000 transistors. Reasonably fast hardware implementations consume in excess of 20.000.000 transistor equivalents (e.g. using an AMD K7 microprocessor core). This compares

with 52 bytes for the AES internal state, a little more than 1kbyte AES code and approx. 10.000 transistors for a very basic CPU. This sums up to yield approx. 20.000 transistors that are required at minimum to make AES run. Reasonable AES implementations consist of approx. 50.000 transistor equivalents.
Clearly it is pretty expensive to implement Turbo PMC V3 in hardware. Finally does nobody expect an OTFE software to run on a smart card processor. Consequently it is only logical that the only application for a Turbo PMC V3 hardware implementation is a code breaking machine owned by an adversory.

8.4 Performance

64 bit CPUs will replace 32 bit CPUs in the future. Turbo PMC V3 is optimized for high performance on 64 bit CPUs.

Cipher	***Turbo PMC V3***	***Turbo PMC V3***	***AES (table-based)***	***AES (table-based)***
Type of machine code	32 bit C++ x86 code	64 bit C++ x64 code	32 bit C++ x86 code	64 bit C++ x64 code
Encryption speed on an Intel Core Duo 6600 CPU, clocked at 2.4GHz [Mbit/s]	414	920	447	730
Encryption speed on an Intel Pentium 4 CPU, clocked at 3.2GHz [Mbit/s]	290	processor cannot execute 64 bit code	378	processor cannot execute 64 bit code

Table 1: Encryption speed comparison: Turbo PMC V3 vs. AES (Compiler: Microsoft Visual C++ 2005)

8.5 Key setup

Conventional ciphers need to be compatible with very basic types of microprocessors. Well-known basic CPUs are the 8051 and 68HC05. A modern Intel Core Duo microprocessor is several thousand times faster. Due to the fact that compatibility with very basic microprocessors cannot be a design goal for OTFE software which is running on comparingly powerful computers, fast key setup cannot be desirable as well.

Key setup thus takes at least 100ms on an Intel Core Duo 6600 microprocessor. During this operation, the complete crypto context is initialized by bootstrapping function after function.

9. Strength against known attacks

Attacks are general approaches that need to be reinvented for every new type of cipher. It is generally assumed that an opponent knows the design of the cipher and that he can generate virtually any amount of plaintext and corresponding ciphertext.

9.1 Differential Cryptanalysis

Differential cryptanalysis was first described by Eli Biham and Adi Shamir [16]. DC analyses the effect of particular differences in plaintext pairs on the differences of the resultant ciphertext pairs.
A difference propagation is composed of differential trails, where its propagation ratio is the sum of the propagation ratios of all differential trails that have the specified initial and final difference patterns. To be resistant against DC, it is therefore a necessary condition that there are no differential trails with a predicted propagation ratio higher than 2^{1-n} (*n* is the block length).

The round functions of Turbo PMC V3 are polymorphic pseudorandom functions that change with each bit pattern. As the most probable key is to be located by applying DC, the precondition for success is constant behaviour of the cipher. Certain groups of keys may exhibit differential trails, although. It should be kept in mind that lowest possible effective linearity is not an advisable design goal [10].

9.2 Linear Cryptanalysis

Linear cryptanalysis was first described by Mitsuru Matsui [17]. LC tries to find a linear expression for a given cipher algorithm to determine one key bit. LC attacks are effective if there are predictable correlations between input and output over all rounds that are significantly larger than $2^{-n/2}$ (*n* is the block length).

Similar to DC, LC requires the cipher to feature constant behaviour for all key bit patterns. As the round functions of Turbo PMC V3 are polymorphic pseudorandom functions that change for every key bit combination, the likelyhood for predictable correlations will be close to $2^{-n/2}$.

9.3 Weak keys

Weak keys are keys that result in poor block cipher mapping so that weaknesses become noticeable. For both DES and IDEA there exist known weak keys. This kind of weakness tends to occur for ciphers in which the non-linear operations depend on the actual key value.
In Turbo PMC V3 do the polymorphic pseudorandom functions mainly depend on the internal state which is derived from the key in a very lengthy operation. Consequently there is no direct link between non-linear operations and the actual key bit pattern. For Turbo PMC V3 there exists no restriction on the selection of the key bit pattern.

9.4 Related-key attacks

Keys with a chosen relation are used to encrypt a chosen plaintext. Potentially present linearity and insufficient diffusion become noticeable if this attack is successful.

The amount of diffusion is extraordinary for Turbo PMC V3. Although locally linear behaviour of groups of keys can potentially occur, the high amount of diffusion and non-linearity leads to a high degree of improbability that this type of attack can be successful.

9.5 Interpolation Attack

If the ciphertext can be represented as a polynomial or rational expresson (with N coefficients) of the plaintext, then the polynomial or rational expression can be reconstructed using N plaintext/ciphertext pairs.

Similar to DC and LC, this attack requires the cipher to feature constant behaviour for all keys. Although the number of rounds is only 3, plenty of operations from different algebraic groups (XOR, additions, multiplications, etc.) are combined virtually arbitrarily. Therefore, susceptibility to interpolation attacks is highly improbable.

9.6 Dictionary Attacks

As the block size is 1024 bits (it should be noted that AES has only 128 bits – regardless of key length), a dictionary attack will require 2^{1024} different plaintexts to allow the attacker to encrypt or decrypt arbitrary messages under an unknown key. This attack applies to any deterministic block cipher with its respective block length regardless of its design.

9.6 Key-Collision Attacks

For key size *n*, key collision attacks can be used to forge messages with complexity only $2^{n/2}$ [18]. Thus, the complexity of forging messages under 1024 bit keys is 2^{512} (compared with AES using 128 bit keys: 2^{64}). This attack applies to any deterministic block cipher, and depends only on its key length, regardless of its design. It was Eli Biham who had the idea to apply the birthday paradox to block ciphers. The paradox suggests that in a class of 23 children, probability for two children is more than half for to children havin the same birthday. Theoretic strength of a block cipher is consequently bounded by the square root of the length of the key.

9.7 Timing Attacks

The number of instructions used to encrypt or decrypt of the majority of polymorphic pseudorandom functions is not key dependent as well as data dependent. Conceptually different PRNGs are present pairwise in order to give an attacker insufficient information about the internal state. Cache accesses could although potentially help an attacker. The total size of the internal state is small compared with cache size of modern microprocessors. It is thus highly improbable for timing attacks to succeed.

9.8 Power Attacks and Attacks based on Electromagnetic Leakage

Highly similar PRNGs implemented in Turbo PMC V3 leak no information as they are almost conceptually identical, but yield different results. Such attacks are definitely harder than on DES, Triple-DES, AES, Twofish, Serpent, Magenta, IDEA, etc., which are anyways known for this weakness [11]. As microprocessors with huge caches and more than 40 million transistors anyways leak almost no useful information to mount such attacks, a compromise between strengthening of the cipher against these attacks and encryption speed was chosen.

9.9 Fault Analysis

Turbo PMC V3 contains no protection against induced faults. An attacker can definitely change the cipher and and extract the key. Rather than modifying the cipher, an attacker has will try to get hold of the key or plaintext more conveniently somewhere else in a software application. The complete OTFE software needs to be protected against alterations with a suitable mechanism. The cipher is only part of the software package. It clearly needs to be protected as well.

9.10 Algebraic Attacks

In [9] Courtois and Pieprzyk point out the unexpected property of Rijndael and Serpent that they can be described as a system of overdefined and sparse quadratic equations over *GF(2)*. Security of Rijndael and Serpent probably does not grow exponentially with the number of rounds. 128 bit Rijndael (AES) can be described by a system of 8000 quadratic equations with 1600 binary unknowns. Attacking AES by an algebraic attack requires only a few known plaintexts to succeed.
This attack can only succeed on ciphers with a very regular structure (like AES Rijndeal and Serpent). Turbo PMC V3 is polymorphic and it doesn't rely on boolean functions for the S-Boxes which can be expressed as a multivariate polynomial. Chances for a cryptanalyst to come up with an overdefined system of equations that describes Turbo PMC V3 is zero.

9.11 Distinguishability between random permutations and almost perfect nonlinear permutions

In [9] Dunkelman and Keller describe methods that are able to distinguish effective linearity of a cipher which enables an attacker to distinguish one cipher from another and which might distinguish certain key classes from others.
There seems to be a direct relationship between extreme differential properties and excessive nonlinearity which makes a cipher distinguishable. Dunkelmann and Keller point out that a permutation that is very close to be an APNP (an Almost Perfect Nonlinear Permutation) is the S-box SubBytes of AES.
Data complexity of the best distinguisher is $O^{(2n/3)}$, which corresponds with 2^{43} for AES !!!

Turbo PMC V3 features changing differential as well as linear behaviour with the key bit pattern. This is due to the quasi-random choice of PRNGs. The three-round Luby Rackoff construction helps to adjust effective linearity according to Dunkelmann and Keller [10] near 2. EL=2 is the effective linearity of truly random permutations.
The huge block length is additionally helpful. Thus it is unlikely that it is possible to gather usueful information for an attack at $O^{(2n/3)} = 2^{341}$.

9.12 Cold Boot Attacks

In [19] the authors describe an attack for which software engineers have taken countermeasures since at least 10 years. DRAM memory chips used in most modern computers retain their contents for seconds to minutes after power is lost. The lower the ambient temperature, the longer data will be retained. As soon as a memory module is powered up again (e.g. by plugging it into a different motherboard), memory refresh takes place again and all memory locations can be read out conveniently.
This kind of attack is all but practical. It is more practical although for a trojan horse with ring 0 access rights to scan all RAM memory. Although the mode of operation differs, both attacks are similar.

Cipher designers regard such issues commonly as “implementation issues”, which is definitely correct. Unlike conventional ciphers, Turbo PMC V3 overwrites the complete internal state as soon as this is possible. As part of the mechanism which is required to harden a cryptographic application is already built into the cipher, software engineers will certainly find it easier to create secure applications.

9.13 Virus Attacks

A trojan horse with ring 0 access rights can potentially be built into the Operating System of a computer. Crypto contexts do not change during operation of disk encryption software like TrueCrypt and they can easily be found by a memory scanner in the non-paged portion of system RAM due to the inevitably high amount of entropy of the data. For AES, the crypto context is 52 bytes long, which is a sufficiently small amount of information to be hidden on a harddisk for months or years and that can be read out during an inspection at the customs when entering a country.
In contrast, Turbo PMC V3 alters the crypto context for each harddisk sector in the TurboCrypt on-the-fly-encryption software. Several kilobytes of information change in different memory regions when the TurboCrypt encryption driver reads or writes files on a Turbo PMC V3-encrypted virtual volume.

Another, even simpler attack can be mounted on disk encryption software: As an example, TrueCrypt uses IO control code 466944 to signal a mount request to the encryption driver. This request is passed through the driver stack. Together with this mount request, the software passes the password used to open a specific encrypted volume in the clear through the stack. A trojan horse has not much more to do than to filter IRPs (IO Request Packets) for known IO control codes. It is obvious that this kind of weakness is disastrous, but it has nothing to do with the cipher itself.
When applying Turbo PMC V3, it is highly recommended to perform an RSA or DH key exchange for key encryption in order not to expose any password information.

10. Design rationale

Unlike conventional designs that are all based on a fixed algorithm, Turbo PMC V3 is based on interpreted crypto code with the result that different keys yield conceptually different encryption algorithms. This design principle makes Turbo PMC V3 an outstanding deterministic 1024 bit block cipher.
In short, the following design criteria were taken into account:

- Complex design
- Resistance against all known attacks
- Huge block size (1024 bit) in order to keep safety margin of factor 4 to ensure resistance against future attacks
- Balancing performance, use of CPU and memory resources, as well as number of operations needed to initialize the crypto context for only one application: OTFE software
- Huge crypto context: approx. 40kbyte (compares with 52 byte for AES)
- Extremely lengthy key setup/key expansion process
- Huge code size: >50kbyte (of 32 bit version; compares with 1135 bytes for AES on a 68HC05 smart card)
- Use of provable concepts (Luby-Rackoff in conjunction with Polymorphic Pseudorandom Functions)
- Cipher differs conceptually with each and every key combination
- High encryption/decryption speed
- Forcing attacker to analyse multiple paths and thus make his task grow exponentially in the number of calls to polymorphic pseudorandom functions

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] U. M. Maurer. A simplified and Generalized Treatment of Luby-Rackoff Pseudorandom Permutation Generators. EuroCrypt '92, Springer LNCS v.658, pp.239-255, 1992

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

[6] R. Anderson, B. Shamir, Two Practical and Provably Secure Block Ciphers: BEAR and LION. http://citeseer.ist.psu.edu/anderson96two.html,1996

[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

[10] Orr Dunkelman, Nathan Keller. A New Criterion for Nonlinearity of Block Ciphers. http://vipe.technion.ac.il/~orrd/crypt/apnp.pdf, 2006

[11] 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

[12] M. Luby, C. Rackoff. How to construct pseudorandom permutations from pseudorandom functions. SIAM Journal on Computing, Vol. 17, No. 2, pp 373-376, 1988

[13] M. Naor, O. Reingold, On the construction of pseudo-random permutations: Luby-Rackoff revisited, Journal of Cryptology, Vol. 12, pp. 29-66, 1999

[14] S. Vaudenay, Provable security for block Ciphers by decorrelation, Lectures Notes in Computer Science 1373, pp. 249--275, SpringerVerlag, 1998. http://citeseer.ist.psu.edu/vaudenay98provable.html

[15] J.-S. Coron, L. Goubin, On Boolean and Arithmetic Masking against Differential Power Analysis, Provable security for block Ciphers by decorrelation, Lectures Notes in Computer Science 1965, pp. 231-237, SpringerVerlag, 2000. http://www.gemplus.com/smart/rd/publications/pdf/CG00mask.pdf

[16] E. Biham and A. Shamir, Differential cryptanalysis of DES-like cryptosystems, Journal of Cryptology, Vol. 4, No. 1, 1991, pp. 3-72.

[17] M. Matsui, Linear cryptanalysis method for DES cipher, Advances in Cryptology, Proceedings Eurocrypt'93, LNCS 765, T. Helleseth, Ed., Springer-Verlag, 1994, pp. 386-397.

[18] E Biham, How to Forge DES-Encrypted Messages in 2^{28} Steps, Technical Report CS884, Technion, August 1996

[19] J. A. Haldermany, S. D. Schoenz, N. Heningery, W. Clarksony, W. Paulx, J. A. Calandrinoy, A. J. Feldmany, J. Appelbaum, E. W. Felten, Lest We Remember: Cold Boot Attacks on Encryption Keys, 2008, http://citp.princeton.edu.nyud.net/pub/coldboot.pdf

[20] Yin Y., Li X., Hu Y., Fast S-box security mechanism research based on the polymorphic cipher, Information Sciences: an International Journal 178(6): 1603-1610, Elsevier Science Inc., 2008, http://portal.acm.org/citation.cfm?id=1332140.1332370&coll=GUIDE&dl=GUIDE

For more information: http://www.pmc-ciphers.com