# Efficient DPA Countermeasures for the TEA Block-Ciphers Family

(Anonymous submission to AFRICACRYPT 2012)

**Abstract.** The Tiny Encryption Algorithm (TEA) is a light-weight block-cipher introduced by Needham and Wheeler in the mid-nineties. This cipher was subsequently extended to XTEA and XXTEA by their designers in order to correct some weaknesses. The TEA block-ciphers are widely used in applications where efficient (memory-wise and speed-wise) encryption and decryption are required. They are for example implemented in the Linux kernel.
This paper studies the security of the TEA block-ciphers regarding differential power analysis (DPA) and proposes novel efficient countermeasures. In contrast to generic countermeasures, the techniques introduced in this paper are tailored so as to retain the efficiency of the unprotected implementations. Figures detailing the code size, the RAM memory requirements and the numbers of cycles for various implementations are provided.

**Keywords:** DPA, countermeasures, efficient implementation, XTEA, XXTEA.

## 1 Introduction

*TEA family* The TEA block-ciphers [21, 17, 22] are ciphers designed by Needham and Wheeler, featuring a 128-bit key-size and running over (at least) 64 rounds. They are based on a Feistel structure without use of any S-box, nor any key expansion routines. The ciphers make alternate use of XOR, Shift and modular addition, resulting in simple, efficient, and easy to implement algorithms.

Various attacks have been reported on TEA. These include related-key attacks in [12] and differential cryptanalysis in [16, 10]. The *XTEA cipher* [17] was later proposed as an improvement to TEA to counter the attacks of [12]. The *XXTEA cipher* [22], also known as Corrected Block TEA, was proposed as an improvement to Block TEA cipher that was originally published in the XTEA paper ([17]). A recent attack against a reduced round version of XXTEA was reported in [23]; the 64-bit or 128-bit block versions, which use 32 and 19 rounds respectively, are considered secure. Finally, there is the *XETA cipher*, which is due to a bug in the C implementation of XTEA, where the precedence was given to the XOR instead of the addition in the round function.

*Differential power analysis (DPA)* DPA and related attacks, introduced by Kocher *et al.* in [14], exploit side-channel leakage to uncover secret information. During the execution of a cryptographic algorithm, the secret key or some related information may be revealed by monitoring the power consumption of the electronic device executing the cryptographic algorithm. DPA-type attacks potentially apply to all cryptosystems, including popular block-ciphers like DES or AES [20, 18]. Protection against DPA is achieved thanks to randomization techniques. The commonly suggested way to thwart DPA-type attacks for implementations of block-ciphers is random masking [4, 9, 6]. The idea is to blind sensitive data with a random mask at the beginning of the algorithm. The algorithm is then executed as usual. Of course, at some step within a round the value of the mask (or a value derived thereof) must be known in order to correct the corresponding output value. This general technique is referred to as the *duplication method* or the *splitting method*. The *transformed masking method* [1] is a specialized technique wherein the *same* mask is used throughout the computation. More specifically, all intermediate values are XORed with a random mask and the inner operations

are modified such that the output of a round is masked by the same mask as that of the input. This was for example applied to DES by modifying its non-linear components (namely, the original S-boxes were replaced with modified S-boxes so as to output the correct masked values), which resulted in an implementation shown to be secure against (first-order) DPA attacks.

For block-ciphers involving different types of operations, two masking techniques must usually be used: a Boolean masking (generally by applying an XOR) and an arithmetic masking. Both techniques were for instance used for protecting the AES finalists against DPA [15]. Further, as exemplified in [15], it is useful to have efficient and secure methods for switching from Boolean masking to arithmetic masking, and conversely. The algorithm suggested in [15] was however shown to be vulnerable to a 2-bit DPA attack in [6]. A more secure algorithm was later proposed by Goubin in [8]. The algorithm works in both directions. The arithmetic-to-Boolean conversion is however less efficient and depends on the length of the values to be masked; a more efficient conversion can be found in [7].

*Our contribution* The TEA family of block-ciphers enjoys several salient features making them attractive for light-weight applications: simplicity, minimal key-setup, no look-up tables, and small footprint. In this paper, we introduce countermeasures against DPA-type attacks which, advantageously, retain the efficiency of the unprotected implementations in terms of memory requirements and speed.

Our starting point is the transformed masking strategy in tandem with a *Boolean* mask. Shifting (right or left) with constants and XOR operations straightforwardly work with Boolean masking. The addition operations need more work as they are not compatible with a Boolean mask. They so have to be modified in a way to avoid unmasking the sensitive operands. This is usually achieved by switching back and forth from arithmetic masking to Boolean masking. We also make use of such secure conversions but, unlike previous works, in *one* direction only: from Boolean to arithmetic. Our unidirectional conversion equally applies to TEA, XTEA, XXTEA, and XETA. The difference resides in the number of conversions per round that are required in each of the algorithms. The cost of one conversion is about 7 elementary operations.

Fully specified algorithms for XTEA and XXTEA when used in decryption mode are provided. We also benchmark the resulting implementations with their unprotected counterparts and with implementations of AES.

*Notation* The following symbols are used throughout the paper.

| Symbol | Meaning |
|---|---|
| $\oplus$ | bit-wise exclusive OR |
| $\&$ | bit-wise AND |
| $\boxplus$ | addition modulo $2^{32}$ |
| $\boxminus$ | subtraction modulo $2^{32}$ |
| $\gg$ | right shift |
| $\ll$ | left shift |
| $\|$ | concatenation |
| $\bar{a}$ | one's complement of $a$ |
| $\boldsymbol{a}$ (bold-face) | masked value of $a$ |

*Outline of the paper* The rest of this paper is organized as follows. The next section details the specifications of the XTEA block-cipher. Section 3 reviews the principles behind DPA-type attacks and presents our approach to thwart them with an illustration to XTEA. Subsequently, Section 4 briefly describes the XXTEA cipher and extends our masking techniques. A performance analysis is provided in Section 5. Finally, the paper concludes in Section 6.

## 2 XTEA

XTEA is a 64-bit block cipher that has 64 rounds and operates with a key size of 128 bits. It uses similar routines for encryption and decryption module. The next figure (Fig. 1) illustrates two rounds of XTEA. Actually, XTEA can be seen as the repetition of 32 times these 2 rounds (called a *cycle*).

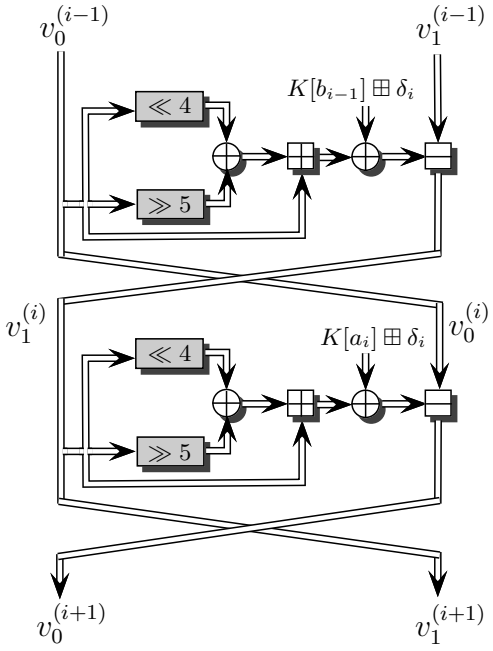


**Fig. 1.** Two rounds of XTEA (decryption)

### 2.1 Encryption

Let $\mathsf{rk}[2i]$ and $\mathsf{rk}[2i+1]$ be the round keys for cycle $i$ (for $0 \leqslant i \leqslant 31$). Let also $v_0$ and $v_1$ denote the two 32-bit inputs. The outputs are then updated as

$$\begin{cases} v_0 \leftarrow v_0 \boxplus \big(F(v_1) \oplus \mathsf{rk}[2i]\big) \\ v_1 \leftarrow v_1 \boxplus \big(F(v_0) \oplus \mathsf{rk}[2i+1]\big) \end{cases} \qquad \text{where } F(v) = [(v \ll 4) \oplus (v \gg 5)] \boxplus v \ .$$

The pair $(v_0, v_1)$ is initialized with 64-bit plaintext $m$, $m = v_0 \| v_1$. The ciphertext is the output of cycle 31 (*i.e.*, the above procedure is run for $i = 0, \ldots, 31$).

The master secret key is a 128-bit value $K = (K[0], K[1], K[2], K[3])$ where $K[j]$'s are 32-bit values ($0 \leqslant j \leqslant 3$). Regarding the key schedule, the round keys are defined as

$$\begin{cases} \mathsf{rk}[2i] = K[a_i] \boxplus \delta_i \ , \ a_i = \delta_i \ \& \ 3 \\ \mathsf{rk}[2i+1] = K[b_i] \boxplus \delta_{i+1} \ , \ b_i = (\delta_{i+1} \gg 11) \ \& \ 3 \end{cases} \qquad \text{where } \delta_i = i \cdot \delta \pmod{2^{32}}$$

with $\delta =$ `0x9E3779B9`.

### 2.2 Decryption

The decryption operates in the reverse order. Given a 64-bit ciphertext $c$, write $c = v_0 \| v_1$ where $v_0$ and $v_1$ are 32-bit values. They are updated as

$$\begin{cases} v_1 \leftarrow v_1 \boxminus \big(F(v_0) \oplus \mathsf{rk}[2i+1]\big) \\ v_0 \leftarrow v_0 \boxminus \big(F(v_1) \oplus \mathsf{rk}[2i]\big) \end{cases} \qquad \text{for } i = 0, \ldots, 31 \ .$$

The decryption module uses the same key schedule process. The variable $\delta_i$ is computed in this case as $\delta_i = (32-i)\cdot\delta \pmod{2^{32}}$. The plaintext is then given as the output the last cycle, $m = v_0 \| v_1$.

## 3 An Efficient Masking Method

DPA-type attacks are generic attacks aimed at retrieving the secret key used in a cryptographic algorithm. Suppose that at some point of the algorithm, an intermediate value, say $I(x, s)$, only depends on a *known* data $x$ and a small portion $s$ of the secret key (*i.e.*, small enough so that all possible values can be exhausted). The attack then proceeds as follows. For a possible value $\hat{s}$ for $s$, the attacker prepares two sets, $\mathcal{S}_0(\hat{s})$ and $\mathcal{S}_1(\hat{s})$, given by $\mathcal{S}_b(\hat{s}) = \{x \mid g(I(x, \hat{s})) = b\}$ ($b \in \{0, 1\}$) where $g$ is an appropriate Boolean selection function (for example, the value of a certain bit in $I(x, \hat{s})$). Next, for each set, the corresponding power-consumption traces are averaged and compared. Provided that there are sufficiently measurements so that the distribution of $I(x, \hat{s})$ is close to uniform, a DPA peak appears when the guessed value $\hat{s}$ matches the value of $s$ (otherwise the difference between the two average curves appears as a flat curve). Many first-order variations of the above DPA attack have been proposed since the publication of [14] by defining a multi-bit selection function $g$ or by using more direct approaches (*e.g.*, [3]). The way to prevent the attacks is however the same: it basically consists in preventing the attacker to predict the values of $I(x, \hat{s})$. This can be done by applying a random mask to the computation so that the attacker will no longer be able to evaluate $g(I(x, \hat{s}))$. Of course, this must hold true throughout the entire algorithm. Second-order DPA-type attacks, or more generally higher-order attacks wherein the attacker considers multiple samples within the *same* power-consumption trace, can be mitigated analogously by making use of multiple randomizing masks.

### 3.1 Preventing first-order DPA

For most cryptographic applications, decryption is performed at the user's side. In many cases, the decryption has to be implemented in a way to protect against DPA-type attacks. The TEA block-ciphers involve subtraction and addition in their round function when used for decryption (or encryption). The other basic operations are Boolean operations (XORs and Shifts). The Boolean operations are easily masked through an XOR; only masking the subtraction operations is problematic. All operations work on 4 bytes (32 bits) at a time. To increase the performance, the same mask is used for input and output values, as in the transformed masking method ([1]). We assume that a fresh 32-bit mask $w$ is uniformly picked at random for each decryption process. This mask is applied to the input ciphertext (viewed as a vector of 32-bit values). The same mask is then maintained across all rounds. At the end of the algorithm the same mask is applied to the output data (also viewed as a vector of 32-bit values) to recover the matching, unmasked plaintext.

In order to enable the masking of the subtraction operations with an XOR, we will rewrite a subtraction (modulo $2^{32}$) using a series of Boolean operations. For this purpose, we rely on a theorem stated by Goubin in [8].

**Theorem 1 (Goubin, 2001).** *Let $I = \{0, \ldots, 2^k - 1\}$. Define $\Phi_{x'} : I \to I, r \mapsto \Phi_{x'}(r) = (x' \oplus r) \boxminus r$ and $\Psi_{x'} : I \to I, r \mapsto \Psi_{x'}(r) = \Phi_{x'}(r) \oplus \Phi_{x'}(0)$. Then $\Psi_{x'}$ is linear over* GF(2).

Hence, viewing $I$ as a vector space of dimension $k$ over GF(2), one obtains that, for any $\gamma \in I$,

$$\Psi_{x'}(r) = \Psi_{x'}(\gamma \oplus (\gamma \oplus r)) = \Psi_{x'}(\gamma) \oplus \Psi_{x'}(\gamma \oplus r)\,,$$

which, in turn, yields

$$\Phi_{x'}(r) = \Phi_{x'}(0) \oplus \Phi_{x'}(\gamma) \oplus \Phi_{x'}(\gamma \oplus r)$$

or, equivalently,

$$(x' \oplus r) \boxminus r = x' \oplus \big[(x' \oplus \gamma) \boxminus \gamma\big] \oplus \big[(x' \oplus (\gamma \oplus r)) \boxminus (\gamma \oplus r)\big] \ . \tag{1}$$

Equation (1) is in essence Goubin's Boolean-to-arithmetic conversion. We derive a slightly simplified form thereof.

**Corollary 1.** *For any $x, r \in I$ and letting $r = r_1 \oplus r_2$, the following relation holds true*

$$x \boxminus r = (x \oplus r) \oplus (x \oplus r_1 \boxminus r_2) \oplus (x \oplus r_2 \boxminus r_1) \ . \tag{2}$$

*Proof.* This follows from Eq. (1) with $x' = x \oplus r$ and $\gamma = r_2$ (and thus $r \oplus \gamma = r_1$). □

Back to our problem, the objective is to evaluate an operation of the form

$$u \boxminus G(\cdot, k) \oplus w$$

on input masked values for $u$ and $v$, and their corresponding masks — without manipulating $u$, $v$ nor $G(\cdot, k)$ since their values only depend on the input message and the key! Remember that we assume that the same input mask is used for $u$ and $v$ and the output mask remains unchanged. We let $\boldsymbol{u} = u \oplus w$ and $\boldsymbol{v} = v \oplus w$ denote the masked values for $u$ and $v$ with Boolean mask $w$, respectively. Now, if $G(\cdot, k)$ is split in two shares $X_1$ and $X_2$ (*i.e.*, $G(\cdot, k) = X_1 \oplus X_2$), an application of Corollary 1 yields

$$\begin{aligned} u \boxminus G(\cdot, k) \oplus w &= \big[(u \oplus (X_1 \oplus X_2)) \oplus (u \oplus X_1 \boxminus X_2) \oplus (u \oplus X_2 \boxminus X_1)\big] \oplus w \\ &= (\boldsymbol{u} \oplus X_1 \oplus X_2) \oplus (\boldsymbol{u} \oplus w \oplus X_1 \boxminus X_2) \oplus (\boldsymbol{u} \oplus w \oplus X_2 \boxminus X_1) \\ &= \boldsymbol{u} \oplus X_1 \oplus (\boldsymbol{u} \oplus X_1 \oplus w \boxminus X_2) \oplus (\boldsymbol{u} \oplus X_2 \oplus w \boxminus X_1) \oplus X_2 \ . \end{aligned} \tag{3}$$

(Note that the order the operations are evaluated is important.)

It remains to show how to split $G(\cdot, k)$ in two shares, given on input the masked values $\boldsymbol{u} = u \oplus w$, $\boldsymbol{v} = v \oplus w$ and mask $w$. This can be achieved by a second application of Corollary 1. The transformation specifically depends on the round function $F$. We present an implementation for XTEA in the next section. The case of XXTEA is dealt with in § 4.2.

### 3.2 Application to XTEA

As described in § 2.2, a round of XTEA can be written as

$$v_b \leftarrow v_b \boxminus \big(F(v_{\bar{b}}) \oplus \mathsf{rk}[2i + b]\big) \quad \text{where } F(v_{\bar{b}}) = [(v_{\bar{b}} \lll 4) \oplus (v_{\bar{b}} \ggg 5)] \boxplus v_{\bar{b}}$$

for $b \in \{0, 1\}$. Given that the operations within the two Feistel rounds are symmetric with respect to $v_0$ or $v_1$, we only focus on one round (*i.e.*, $b = 0$). For the sake of clarity, we also avoid the use of subscripts, $v_0$ and $v_1$ are denoted by $u$ and $v$, and the round key is simply denoted by $\mathsf{rk}$. We let $G(\cdot, k) = F(v) \oplus \mathsf{rk}$ and use the previous masking technique. As the function $F$ uses an addition operation, we do not split $G$ here but $F$ instead. Once $F$ splitted using $X_1$ and $X_2$, $G$ can be represented as $G(\cdot, k) = X_1 \oplus X_2 \oplus \mathsf{rk}$. A round of XTEA can thus be expressed in two steps as

$$\begin{cases} X_1 \leftarrow X_1 \oplus \mathsf{rk} \\ \boldsymbol{u} \leftarrow \boldsymbol{u} \oplus X_1 \oplus (\boldsymbol{u} \oplus X_1 \oplus w \boxminus X_2) \oplus (\boldsymbol{u} \oplus X_2 \oplus w \boxminus X_1) \oplus X_2 \end{cases} \tag{4}$$

with $X_1 \oplus X_2 = F(v) \oplus \mathsf{rk}$.

Letting $f(v) = (v \ll 4) \oplus (v \gg 5)$, we can write $F(v) = f(v) \boxplus v$. In order to apply Corollary 1, we need first to express $F(v) = f(v) \boxplus v$ using a subtraction. We have:

$$F(v) = (\overline{f(v)} \boxminus v) \oplus (\boxminus 1) \tag{5}$$

since $\bar{v} = v \oplus (\boxminus 1) = \boxminus v \boxminus 1$ (*e.g.*, see [8, Lemma 3]). Hence, noting that $v = \boldsymbol{v} \oplus w$, Corollary 1 immediately yields

$$\overline{f(v)} \boxminus v = (\overline{f(v)} \oplus \boldsymbol{v} \oplus w) \oplus (\overline{f(v)} \oplus \boldsymbol{v} \boxminus w) \oplus (\overline{f(v)} \oplus w \boxminus \boldsymbol{v}) \ . \tag{6}$$

Moreover, since the Shift commutes with the XOR, we also have

$$\begin{aligned} \overline{f(v)} &= (v \ll 4) \oplus (v \gg 5) \oplus (\boxminus 1) = (\boldsymbol{v} \ll 4) \oplus (\boldsymbol{v} \gg 5) \oplus (w \ll 4) \oplus (w \gg 5) \oplus (\boxminus 1) \\ &= (\boldsymbol{v} \ll 4) \oplus (\boldsymbol{v} \gg 5) \oplus \overline{f(w)} \ . \end{aligned} \tag{7}$$

Putting together Eqs (5), (6) and (7), we so obtain the following algorithm for computing $(X_1, X_2)$. We suppose that $S := \overline{f(w)} = (w \ll 4) \oplus (w \gg 5)$ is computed once for all at the beginning of the decryption algorithm.

1. Compute $\mu \leftarrow (\boldsymbol{v} \ll 4) \oplus (\boldsymbol{v} \gg 5)$;
2. Compute $X_1 \leftarrow (((\mu \oplus \boldsymbol{v}) \oplus S) \boxminus w) \oplus \boldsymbol{v}$;
3. Compute $\mu \leftarrow \mu \oplus (w \oplus S)$;
4. Compute $X_2 \leftarrow (\mu \boxminus \boldsymbol{v}) \oplus \mu \oplus (\boxminus 1)$.

*Remark 1.* For better performance, the value $w \oplus S$ can be pre-computed (at the expense of an extra memory-word for its storage).

In the above description, the brackets indicate the order in which the operations must be performed. If there are no brackets, the operations can be done in any order. A careful inspection shows that the evaluation of $X_1$ and $X_2$ never manipulates a value depending on both the input ciphertext and the key. Moreover, as $w$ is chosen uniformly from $\{0,1\}^{32}$, it can be verified that all intermediate values are uniformly distributed over $\{0,1\}^{32}$. As a consequence, since the values of $X_1$ and $X_2$ cannot be predicted, a DPA-type cannot be successfully mounted during the course of their evaluation. Further, this shows that the evaluation of a round as per Eq. (4) is also protected against DPA and so the entire decryption algorithm is. Note that there is no need to mask the round keys as their derivation solely depends on the master key (*i.e.*, their computation is independent of the input ciphertext).

A detailed implementation of a DPA-protected version of XTEA can be found in Appendix A.

## 4 XXTEA

XXTEA uses a cryptographic key of 128 bits to decrypt data. While XTEA has a 64-bit block size and a constant number of rounds (like TEA), XXTEA accommodates variable block-sizes and has a number of round iterations (ranging from 6 to 32) that depends on the block size.

### 4.1 Decryption

XXTEA can be implemented with a variable block length (yet multiple of 32 bits). But the basic operation handles only 64-bit at a time. Each 64-bit input is split into two halves, $y$ and $z$, which then go through a Feistel routine for $N$ rounds where $N$ is given by $N = 6 + 52/b$ and $b$ is the

number of 32-bit blocks on input. If $\mathsf{rk}_i[j]$ denotes the $j$-th round-key for cycle $i$, then the new value of $y$ and $z$ are obtained using the following function $F$,

$$F(y, z, \delta_i, \mathsf{rk}_i[j]) = \left([(z \gg 5) \oplus (y \ll 2)] \boxplus [(y \gg 3) \oplus (z \ll 4)]\right) \oplus \left((\delta_i \oplus y) \boxplus (\mathsf{rk}_i[j] \oplus z)\right), \quad (8)$$

which takes 4 parameters on input: $y$, $z$, $\delta_i$, and $\mathsf{rk}_i[j]$.

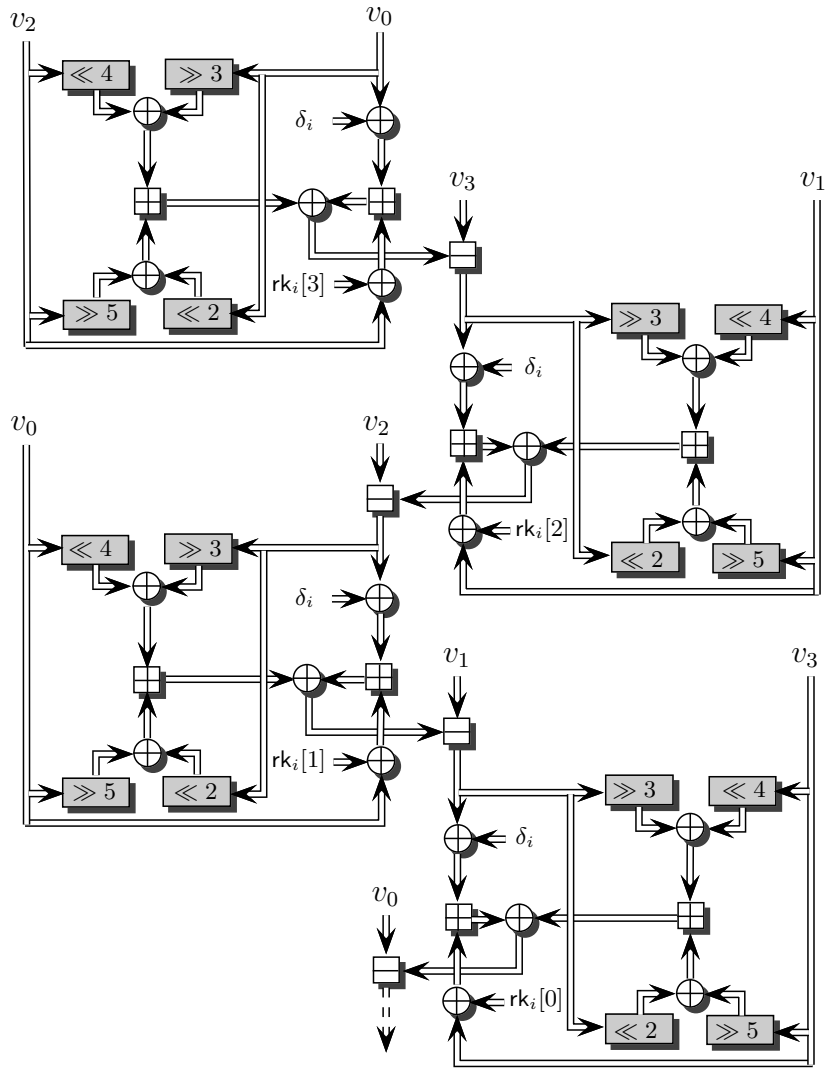


**Fig. 2.** Four rounds of XXTEA-128 (decryption)

For concreteness, we focus on the 128-bit block version of the algorithm (*i.e.*, $b = 4$). A Feistel cycle for XXTEA-128 comprises 4 rounds and the total number of cycles is given by $N = 6 + 52/4 =$

19. A cycle $i$ is then defined as

$$\begin{cases} v_3 \leftarrow v_3 \boxminus F(v_0, v_2, \delta_i, \mathsf{rk}_i[3]) \\ v_2 \leftarrow v_2 \boxminus F(v_3, v_1, \delta_i, \mathsf{rk}_i[2]) \\ v_1 \leftarrow v_1 \boxminus F(v_2, v_0, \delta_i, \mathsf{rk}_i[1]) \\ v_0 \leftarrow v_0 \boxminus F(v_1, v_3, \delta_i, \mathsf{rk}_i[0]) \end{cases} .$$

Variables $(v_0, v_1, v_2, v_3)$ are initialized with 128-bit ciphertext $c$, $c = v_0 \| v_1 \| v_2 \| v_3$. The calculations are operated using the two registers $y$ and $z$ as follows:

$$y \leftarrow v_0 \quad \text{and} \quad \text{for } j \in \{3, \ldots, 0\}\text{:} \begin{cases} z \leftarrow v_{(j-1) \bmod 4} \\ y \leftarrow v_j \boxminus F(y, z, \delta_i, \mathsf{rk}_i[j]) \\ v_j \leftarrow y \end{cases} .$$

The plaintext is the output of cycle 18. For the key schedule, the round keys are derived from the master 128-bit key value $K = (K[0], K[1], K[2], K[3])$ as

$$\mathsf{rk}_i[j] = K[\big(j \,\&\, 3\big) \oplus a_i] \quad \text{for } 0 \leqslant j \leqslant 3 \text{ and } 0 \leqslant i \leqslant 18\,,$$

where $a_i = (\delta_i \gg 2) \,\&\, 3$ and $\delta_i$ is computed as for XTEA. This function selects one block out of the four 32-bit blocks of the master key, depending on both the index $j$ and the two least significant bits of $\delta_i$ when shifted by 2 bits to the right.

### 4.2 Masking XXTEA

We apply the masking technique described in Section 3.1. If we let $u$ represent the value of $v_j$ $(3 \geqslant j \geqslant 0)$ and drop the subscripts, a round of XXTEA becomes

$$u \leftarrow u \boxminus F(y, z, \delta, \mathsf{rk})$$

with $\delta := \delta_i$ and $\mathsf{rk} := \mathsf{rk}_i[j]$. Therefore, using Eq. (3), if $G(\cdot, k) = F(y, z, \delta, \mathsf{rk})$ and $\boldsymbol{u} = u \oplus w$ denotes the masked value of $u$ with Boolean mask $w$, a round of XXTEA can be expressed as

$$\boldsymbol{u} \leftarrow \boldsymbol{u} \oplus X_1 \oplus (\boldsymbol{u} \oplus X_1 \oplus w \boxminus X_2) \oplus (\boldsymbol{u} \oplus X_2 \oplus w \boxminus X_1) \oplus X_2 \tag{9}$$

with $X_1 \oplus X_2 = F(y, z, \delta, \mathsf{rk})$.

From the definition of function $F$ (Eq. (8)), we see that it can be decomposed in two sub-functions. We write $F(y, z, \delta, \mathsf{rk}) = f(x, y) \oplus g(y, z, \delta, \mathsf{rk})$ where

$$f(y, z) = [(z \gg 5) \oplus (y \ll 2)] \boxplus [(y \gg 3) \oplus (z \ll 4)]$$

and

$$g(y, z, \delta, \mathsf{rk}) = (\delta \oplus y) \boxplus (\mathsf{rk} \oplus z) \ .$$

Now, to split $F(y, z, \delta, \mathsf{rk})$, we proceed in three steps:

1. First, we express $f(\cdot)$ as the one's complement of the XOR of two shares $A$ and $B$; *i.e.*, $f(y, z) = \overline{A \oplus B}$;
2. We do the same for $g(\cdot)$; *i.e.*, $g(y, z, \delta, \mathsf{rk}) = \overline{C \oplus D}$;
3. Finally, $X_1$ and $X_2$ are obtained by crossing the previous shares and XORing them together; *i.e.*, $X_1 = A \oplus C$ and $X_2 = B \oplus D$.

Both functions $f(\cdot)$ and $g(\cdot)$ include an addition operation. We first convert the addition into a subtraction using Eq. (5). Next, obtaining the shares $A$ and $B$ (Step 1) is carried out through Corollary 1, and similarly for $C$ and $D$ (Step 2). By evaluating $\overline{A \oplus B}$ and $\overline{C \oplus D}$ instead of $A \oplus B$ and $C \oplus D$ we save two complement operations per round. The correctness of the last step follows by observing that $X_1 \oplus X_2 = A \oplus B \oplus C \oplus D = \overline{(A \oplus B)} \oplus \overline{(C \oplus D)} = f(y,z) \oplus g(y,z,\delta,\mathsf{rk}) = F(y,z,\delta,\mathsf{rk})$, as desired.

Defining $f_1(y,z) = (y \gg 3) \oplus (z \ll 4)$ and $f_2(y,z) = (z \gg 5) \oplus (y \ll 2)$, we have

$$\begin{aligned} A \oplus B = \overline{f(y,z)} &= \big(f_1(y,z) \boxplus f_2(y,z)\big) \oplus (\boxminus 1) = \overline{f_1(y,z)} \boxminus f_2(y,z) \\ &= \overline{f_1(y,z)} \boxminus \big((z \gg 5) \oplus (y \ll 2)\big) \ . \end{aligned}$$

Hence, letting $\boldsymbol{y} = y \oplus w$ and $\boldsymbol{z} = z \oplus w$, an application of Corollary 1 together with the obvious relation that $\overline{f_1(y,z)} = f_1(\boldsymbol{y},\boldsymbol{z}) \oplus \overline{f_1(w,w)}$ leads to the following algorithm for computing $(A,B)$. This algorithm makes use of some values precomputed at initialization (*i.e.*, at the beginning of the decryption algorithm); namely

$$w_2 = w \ll 2,\ w_5 = w \gg 5,\ S_2 = \overline{f_1(w,w)} \oplus w_2, \ \text{ and } \ S_5 = \overline{f_1(w,w)} \oplus w_5 \ .$$

1. Compute $X \leftarrow (\boldsymbol{y} \ll 3) \oplus (\boldsymbol{z} \gg 4)$;
2. Compute $\mu_1 \leftarrow \boldsymbol{y} \gg 5$;
3. Compute $A \leftarrow (\mu_1 \oplus X) \oplus w_5$;
4. Compute $\mu_2 \leftarrow \boldsymbol{y} \ll 2$;
5. Compute $B \leftarrow \mu_2 \oplus S_2$;
6. Compute $A \leftarrow A \oplus (A \boxminus B)$;
7. Compute $B \leftarrow B \oplus (((X \oplus \mu_2) \oplus w_2) \boxminus (\mu_1 \oplus S_5))$.

The values of $C$ and $D$ are obtained analogously by noting that

$$\begin{aligned} C \oplus D = \overline{g(y,z,\delta,\mathsf{rk})} &= \big((\mathsf{rk} \oplus z) \boxplus (\delta \oplus y)\big) \oplus (\boxminus 1) = \overline{(\mathsf{rk} \oplus z)} \boxminus (\delta \oplus y) \\ &= (\overline{\mathsf{rk}} \oplus \boldsymbol{z} \oplus w) \boxminus (\delta \oplus \boldsymbol{y} \oplus w) \ . \end{aligned}$$

However, special cautions are appropriate as the application of Corollary 1 leads to the evaluation of $\boldsymbol{y} \oplus \boldsymbol{z} \oplus w$. Whatever the order this evaluation is performed, this computation is not secured against DPA. We so resort on an extra mask $t$, randomly chosen in $\{0,1\}^{32}$ at initialization. The pair $(C,D)$ is then computed as follows:

1. Compute $\mu_1 \leftarrow \boldsymbol{z} \oplus \overline{\mathsf{rk}_i}$;
2. Compute $\mu_2 \leftarrow \mu_1 \oplus \delta_i$;
3. Compute $C \leftarrow (\mu_2 \boxminus \boldsymbol{y}) \oplus \mu_2$;
4. Compute $\mu_2 \leftarrow (\boldsymbol{y} \oplus t) \oplus w$;
5. Compute $D \leftarrow \boldsymbol{y} \oplus (((t \oplus \mu_1) \oplus \mu_2) \boxminus (w \oplus \delta_i))$.

One may wonder whether $X_1 = A \oplus C$ and $X_2 = B \oplus D$ are a secure split of $F$ as the input $y$ and $z$ are masked with the same value $w$. A careful inspection of $A \oplus C$ and $B \oplus D$ shows they do not result in a value that is dependent on some known data and a portion of the secret key. This comes from the fact that the masking of $f(\cdot)$ involves only shifted values $\boldsymbol{y}$, $\boldsymbol{z}$ and shifted mask whereas the masking of $g(\cdot)$ involves non-shifted masked values and mask.

A detailed implementation is provided in Appendix A.

## 5 Performance Analysis

The proposed unidirectional, fixed mask method has advantage in terms of memory and processing time.

XTEA requires only one mask storage whereas XXTEA needs two mask values. Remember that XXTEA uses an additional mask for the evaluation of the pair $(C, D)$; see §,4.2.

The cost for an unprotected XTEA round is 8 elementary operations. The DPA protection method for XTEA costs 23 operations in total for one round (*i.e.*, the overhead is 15 operations). In addition, 9 extra operations are required in the pre- and post-computation steps. The DPA protection for XXTEA costs 43 operations for one round — this has to be compared to the 16 operations for the unprotected version, plus $10 + 2 \cdot b$ operations for the pre- and post-computation steps. As a result, we see that XTEA is more advantageous than XXTEA in terms of number of operations for 64-bit block encryption/decryption.

We implemented unmasked versions and DPA-protected versions for both XTEA and XXTEA. A 32-bit Intel-based processor was used for evaluating the implementations. The code was written in C and is given in Appendix A (see Algorithms 1 and 2). The details such as the code size, RAM requirements and cycle count are compiled in the next table. For XXTEA, the results for the 64-bit and the 128-bit versions are reported.

**Table 1.** Implementation details for various XTEA and XXTEA versions

| Algorithms | ROM [bytes] | RAM [bytes] | Cycle count |
|---|---|---|---|
| XTEA | 112 | 16 | 590 |
| masked XTEA (Alg. 1) | **272** | **40** | **1,240** |
| XXTEA-64 | 160 | 16 | 681 |
| masked XXTEA-64 (Alg. 2) | **448** | **60** | **1,664** |
| XXTEA-128 | 336 | 16 | 740 |
| masked XXTEA-128 (Alg. 2) | **864** | **60** | **1,941** |
| AES-128 ([15]) | 1756 | 52 | 7,086 |
| masked AES-128 ([15]) | **2393** | **326** | **13,867** |

As expected, Table 1 shows that the DPA countermeasure for XTEA is more efficient than the protected XXTEA for a 64-bit block-size. We also see that the protected versions of XXTEA are faster than that of XTEA as soon as the block-size becomes larger or equal to 128 bits.

Interestingly, the overall impact on speed is limited by a factor of 2 for XTEA and of 3 for XXTEA. For the sake of comparison, Table 1 also reports the AES benchmarks provided by Messerges in [15]. It clearly appears that XTEA and XXTEA are much more efficient; in particular in terms of memory requirements.

## 6 Conclusion

This paper presented DPA-protected yet efficient implementations for the TEA block-ciphers family. The proposed techniques are based on the transformed masking method using a single mask, work at the word level (32-bit), and do not require precomputed tables. Remarkably, they solely make use of arithmetic-to-Boolean conversion (backward converting is not needed), which makes them faster, memory-efficient, and easy to implement. All these features make the resulting implementations well suited to light-weight applications or devices, especially when compared to DPA-protected AES implementations.

## References


1. Mehdi-Laurent Akkar and Christophe Giraud. An implementation of DES and AES, secure against some attacks. In Ç. K. Koç, D. Naccache, and C. Paar, editors, *Cryptographic Hardware and Embedded Systems — CHES 2001*, volume 2162 of *Lecture Notes in Computer Science*, pages 309–318. Springer-Verlag, 2001.
2. Mehdi-Laurent Akkar and Louis Goubin. A generic protection against high-order differential power analysis. In T. Johansson, editor, *Fast Software Encryption (FSE 2003)*, volume 2887 of *Lecture Notes in Computer Science*, pages 192–205. Springer-Verlag, 2003.
3. Eric Brier, Christophe Clavier, and Francis Olivier. Correlation power analysis with a leakage model. In M. Joye and J.-J. Quisquater, editors, *Cryptographic Hardware and Embedded Systems — CHES 2004*, volume 3156 of *Lecture Notes in Computer Science*, pages 16–24. Springer-Verlag, 2004.
4. Suresh Chari, Charanjit S. Jutla, Josyula R. Rao, and Pankaj Rohatgi. Towards sound approaches to counteract power-analysis attacks. In M. Wiener, editor, *Advances in Cryptology — CRYPTO '99*, volume 1666 of *Lecture Notes in Computer Science*, pages 398–412. Springer-Verlag, 1999.
5. Jiazhe Chen, Meiqin Wang, and Bart Preneel. Impossible differential cryptanalysis of the lightweight block ciphers TEA, XTEA and HIGHT. Cryptology ePrint Archive, Report 2011/616, 2011. Available at URL `http://eprint.iacr.org/2011/616.pdf`.
6. Jean-Sébastien Coron and Louis Goubin. On Boolean and arithmetic masking against differential power analysis. In Ç. K. Koç and C. Paar, editors, *Cryptographic Hardware and Embedded Systems — CHES 2000*, volume 1965 of *Lecture Notes in Computer Science*, pages 231–237. Springer-Verlag, 2000.
7. Jean-Sébastien Coron and Alexei Tchulkine. A new algorithm for switching from arithmetic to Boolean masking. In C. D. Walter, Ç. K. Koç, and C. Paar, editors, *Cryptographic Hardware and Embedded Systems — CHES 2003*, volume 2779 of *Lecture Notes in Computer Science*, pages 89–97. Springer-Verlag, 2003.
8. Louis Goubin. A sound method for switching between Boolean and arithmetic masking. In Ç. K. Koç, D. Naccache, and C. Paar, editors, *Cryptographic Hardware and Embedded Systems — CHES 2001*, volume 2162 of *Lecture Notes in Computer Science*, pages 3–15. Springer-Verlag, 2001.
9. Louis Goubin and Jacques Patarin. DES and differential power analysis (the "duplication" method). In Ç. K. Koç and C. Paar, editors, *Cryptographic Hardware and Embedded Systems (CHES '99)*, volume 1717 of *Lecture Notes in Computer Science*, pages 158–172. Springer-Verlag, 1999.
10. Seokhie Hong, Deukjo Hong, Youngdai Ko, Donghoon Chang, Wonil Lee, and Sangjin Lee. Differential cryptanalysis of TEA and XTEA. In J. I. Lim and D. H. Lee, editors, *Information Security and Cryptology — ICISC 2003*, volume 2971 of *Lecture Notes in Computer Science*, pages 402–417. Springer-Verlag, 2004.
11. Jens-Peter Kaps. Chai-tea, cryptographic hardware implementations of xTEA. In D. R. Chowdhury, V. Rijmen, and A. Das, editors, *Progress in Cryptology — INDOCRYPT 2008*, volume 5365 of *Lecture Notes in Computer Science*, pages 363–375. Springer-Verlag, 2008.
12. John Kelsey, Bruce Schneier, and David Wagner. Related-key cryptanalysis of 3-WAY, Biham-DES, CAST, DES-X, NewDES, RC2, and TEA. In Y. Han, T. Okamoto, and S. Qing, editors, *Information and Communication Security (ICICS '97)*, volume 1334 of *Lecture Notes in Computer Science*, pages 233–246. Springer-Verlag, 1997.
13. Youngdai Ko, Seokhie Hong, Wonil Lee, Sangjin Lee, and Ju-Sung Kang. Related key differential attacks on 27 rounds of XTEA and full-round GOST. In B. K. Roy and W. Meier, editors, *Fast Software Encryption (FSE 2004)*, volume 3017 of *Lecture Notes in Computer Science*, pages 299–316. Springer-Verlag, 2004.
14. Paul Kocher, Joshua Jaffe, and Benjamin Jun. Differential power analysis. In M. Wiener, editor, *Advances in Cryptology — CRYPTO '99*, volume 1666 of *Lecture Notes in Computer Science*, pages 388–397. Springer-Verlag, 1999.
15. Thomas S. Messerges. Securing the AES finalists against power analysis attacks. In B. Schneier, editor, *Fast Software Encryption (FSE 2000)*, volume 1978 of *Lecture Notes in Computer Science*, pages 150–164. Springer-Verlag, 2000.
16. Dukjae Moon, Kyungdeok Hwang, Wonil Lee, Sangjin Lee, and Jongin Lim. Impossible differential cryptanalysis of reduced round XTEA and TEA. In J. Daemen and V. Rijmen, editors, *Fast Software Encryption (FSE 2002)*, volume 2365 of *Lecture Notes in Computer Science*, pages 49–60. Springer-Verlag, 2002.

17. Roger M. Needham and David J. Wheeler. TEA extensions. Technical report, Computer Laboratory, University of Cambridge, October 1997. Available at URL `http://www.cl.cam.ac.uk/ftp/users/djw3/xtea.ps`.
18. Siddika Berna Örs, Frank K. Gürkaynak, Elisabeth Oswald, and Bart Preneel. Power-analysis attack on an ASIC AES implementation. In *International Conference on Information Technology: Coding and Computing (ITCC '04), Volume 2*, pages 546–552. IEEE Computer Society, 2004.
19. Gautham Sekar, Nicky Mouha, Vesselin Velichkov, and Bart Preneel. Meet-in-the-middle attacks on reduced-round XTEA. In A. Kiayias, editor, *Topics in Cryptology – CT-RSA 2011*, volume 6558 of *Lecture Notes in Computer Science*, pages 250–267. Springer-Verlag, 2011.
20. François-Xavier Standaert, Siddika Berna Örs, Jean-Jacques Quisquater, and Bart Preneel. Power analysis attacks against FPGA implementations of the DES. In J. Becker, M. Platzner, and S. Vernalde, editors, *Field Programmable Logic and Application (FPL 2004)*, volume 3203 of *Lecture Notes in Computer Science*, pages 84–94. Springer-Verlag, 2004.
21. David J. Wheeler and Roger M. Needham. TEA, a tiny encryption algorithm. In B. Preneel, editor, *Fast Software Encryption (FSE '94)*, volume 1008 of *Lecture Notes in Computer Science*, pages 363–366. Springer-Verlag, 1995.
22. David J. Wheeler and Roger M. Needham. Corrections to XTEA. Technical report, Computer Laboratory, University of Cambridge, October 1998. Available at URL `http://www.movable-type.co.uk/scripts/xxtea.pdf`.
23. Elias Yarrkov. Cryptanalysis of XXTEA. Cryptology ePrint Archive, Report 2010/254, 2010. Available at URL `http://eprint.iacr.org/2010/254.pdf`.

# A Masked Versions of XTEA/XXTEA

**Algorithm 1** DPA-protected XTEA decryption algorithm

**Input:** A 64-bit ciphertext $v[0]$, $v[1] \in \{0, \ldots, 2^{32}\}$, and a 128-bit key $K[0..3]$
**Output:** The corresponding 64-bit plaintext $w[0]$, $w[1]$

```
/* Initialization */
   unsigned long sum = 0xC6EF3720, delta=0x9E3779B9;
   unsigned long mu, X1, X2, X3, m, wp;

/* Pre-calculus */
   m = rand();
   wp = ~(m << 4)^(m >> 5);
   v0 = v[0]^m, v1=v[1]^m;

/* Main loop */
   while(sum){
  /* F = (((v0 << 4) ^ (v0 >> 5)) + v0)^(sum + key[(sum>>11) & 3]) */
      mu =  (v0 << 4) ^ (v0 >> 5);
      X1 = ((wp ^ (mu ^ v0)) - m) ^ v0;
      mu = (m ^ wp) ^ mu;
      X3 = ~(sum + K[(sum >> 11) & 3]);
      X2 = (mu - v0) ^ mu ^ X3;
  /* v1 -= F */
      X3 = (((v1 ^ X2) ^ m) - X1) ^ v1 ^ X2;
      v1 = (((v1 ^ X1) ^ m) - X2) ^ X1 ^ X3;

      sum -= delta;
  /* F = (((v1 << 4) ^ (v1 >> 5)) + v1)^(sum + key[sum & 3]) */
      mu =  (v1 << 4) ^ (v1 >> 5);
      X1 = ((wp ^ (mu ^ v1)) - m) ^ v1;
      mu = (m ^ wp) ^ mu;
      X3 = ~(sum + K[sum & 3]);
      X2 = (mu - v1) ^ mu ^ X3;
  /* v0 -= F */
      X3 = (((v0 ^ X2) ^ m) - X1) ^ v0 ^ X2;
      v0 = (((v0 ^ X1) ^ m) - X2) ^ X1 ^ X3;
   }

/* Post-calculus */
   w[0] = v0^m; w[1] = v1^m;
```

**Algorithm 2** DPA-protected XXTEA decryption algorithm

**Input:** The number of 32-bit blocks $b$, a ciphertext $v[0], \cdots v[b-1] \in \{0, \ldots, 2^{32}\}$ and a 128-bit key $K[0..3]$
**Output:** The corresponding $b \times 32$-bit plaintext

```
/* Initialization */
   unsigned long y, z, p, q, sum, delta=0x9E3779B9;
   unsigned long mu, X1, X2, X3, X4, m, t, wp2, wp5, w2 , w5;
/* Pre-calculus */
   q = 6+52/b; sum=delta*q; m = rand(); t = rand(); w2 = m << 2;  w5 = m >> 5;
   mu = ~(m >> 3)^(m << 4); wp2 = mu^w2; wp5 = mu^w5;
   for (p=b; p>0; p--)
    v[p-1] ^= m;
   y = v[0];
/* Main Loop */
do {
for (p=b-1; p>0; p--) {
         z = v[p-1];
  /* ((z>>5^y<<2) + (y>>3^z<<4)) */
         X4  = (y >> 3) ^ (z << 4); mu =  z >> 5;
         X1 = (mu ^ X4 ) ^ w5;
         X3 =  y << 2;
         X2 = X3 ^ wp2;
         X1 = X1 ^ (X1 - X2);
         X2 = X2 ^ (((X4 ^ X3) ^ w2) - (mu ^ wp5));
  /* ((sum^y) + (K[(p&3)^j] ^ z))) */
         X4  = ~key[(p&3) ^ ((sum >> 2) & 3)]; mu = z ^ X4;
         X4 = mu ^ sum;
         X3 =  X4 ^ (X4 - y);
         X4 = (y ^ t) ^ m;
         X4 =  y ^ (((mu ^ t)^ X4) - (sum ^ m));
  /* v-= F */
         X1 = X1 ^ X3; X2 = X2 ^ X4;
         X4  = (((v[p] ^ X2) ^ m) - X1) ^ v[p] ^ X2;
         y = v[p] = (((v[p] ^ X1) ^ m) - X2) ^ X1 ^ X4;
      }
      z = v[b-1];
  /* ((z>>5^y<<2) + (y>>3^z<<4)) */
      X4  = (y >> 3) ^ (z << 4); mu =  z >> 5;
      X1 = (mu ^ X4 ) ^ w5;
      X3 =  y << 2;
      X2 = X3 ^ wp2;
      X1 = X1 ^ (X1 - X2);
      X2 = X2 ^ (((X4 ^ X3) ^ w2) - (mu ^ wp5));
  /* ((sum^y) + (K[(p&3)^j] ^ z))) */
      X4  = ~key[(p&3) ^ ((sum >> 2) & 3)]; mu = z ^ X4;
      X4 = mu ^ sum;
      X3 =  X4 ^ (X4 - y);
      X4 = (y ^ t) ^ m;
      X4 =  y ^ (((mu ^ t) ^ X4) - (sum ^ m));
  /* v-= F */
      X1 = X1 ^ X3; X2 = X2 ^ X4;
      X4 = (((v[0] ^ X2) ^ m) - X1) ^ v[0] ^ X2;
      y = v[0] = (((v[0] ^ X1) ^ m) - X2) ^ X1 ^ X4;
   } while ((sum -= delta) != 0);
/* Post-calculus */
for (p=b; p>0; p--)
   v[p-1] ^= m;
```