FAST DDP-BASED CIPHERS: FROM HARDWARE TO SOFTWARE

Nikolay Moldovyan[1], Nikolay Goots[1], Peter Moldovyanu[1], and Doug Summerville[2],
[1] Specialized Center of Program Systems "SPECTR", Kantemirovskaya, 10, St.Petersburg 197342, Russia; nmold@cobra.ru
[2] Binghamton University, PO Box 6000, Binghamton NY, USA; dsummer@binghamton.edu

Abstract - *Data-dependent (DD) permutations (DDP) that are very suitable to cheap hardware implementation have been introduced as a cryptographic primitive for the design of fast firmware and software encryption systems. DDP can be performed with so called controlled permutation boxes (CPB) which are fast while implemented in cheap hardware. The latter defines the efficiency of the embedding of CPB in microcontrollers and microprocessors when adding a new fast instruction that allows one to perform DDP. Software and firmware encryption algorithms combining DDP with fast arithmetic operations are described.*

Acknowledgement: This research was supported by EOARD/AFRL grant #1994p.

I. INTRODUCTION

Data-dependent (DD) permutations (DDP) suites well to the design of fast and secure block ciphers [1]-[3]. The DDP can be performed with so called controlled permutation (CP) boxes (CPB) having layered topology [4]-[6], which are fast while implemented in cheap hardware. The CPB can be easily embedded in microcontrollers and general purpose CPUs and used while designing fast firmware and software encryption systems.

In present paper we propose a variant of the fast CPB instruction and consider the design and security of the block DDP-based ciphers oriented to firmware and software implementation.

The paper is organized in the following way: In the second section we characterize DDP performed with CPB presenting detailed structure of the symmetric CPB used in the designed cryptalgorithms and in a new instruction for embedding in microcontrollers and CPUs. In section 3 we present two 64-bit block ciphers appropriate for firmware implementation and a software-oriented 128-bit cipher. Section 4 presents discussion of the results.

II. DESIGN OF THE DDP-BOXES

Different types of the layered CPBs [5] can be constructed using elementary switching elements $\mathbf{P}_{2/1}$ as elementary building blocks performing controlled transposition of two input bits x_1 and x_2. In the general case each $\mathbf{P}_{2/1}$-box is controlled with one bit v and forms two-bit output (y_1, y_2), where $y_1 = y_{1+v}$ and $y_2 = x_{2-v}$. In this paper a layered CPB with n-bit input and m-bit control input is denoted as $\mathbf{P}_{n/m}$. The dotted lines corresponding to CP boxes indicate the controlling bits.

A $\mathbf{P}_{n/m}$-box can be represented as a superposition

$$\mathbf{P}_{n/m} = \mathbf{L}^{(V_1)} \circ \pi_1 \circ \mathbf{L}^{(V_2)} \circ \pi_2 \circ \ldots \circ \pi_{s-1} \circ \mathbf{L}^{(V_s)},$$

where $\mathbf{L}$ is an active layer composed of $n/2$ switching elements, $V_1, V_2, \ldots, V_s$ are controlling vectors of the active layers from 1 to s, and $\pi_1, \pi_2, \ldots, \pi_{s-1}$ are fixed permutations.

The inverse CPB has the following structure

$$\mathbf{P}^{-1}_{n/m} = \mathbf{L}^{(V_s)} \circ \pi^{-1}_{s-1} \circ \mathbf{L}^{(V_{s-1})} \circ \pi^{-1}_{s-2} \circ \ldots \circ \pi^{-1}_{1} \circ \mathbf{L}^{(V_1)}.$$

The components $V_1, V_2, \ldots, V_s$ compose the controlling vector of the $\mathbf{P}_{n/m}$-box: $V = (V_1, V_2, \ldots, V_s)$. The topology of the CPBs $\mathbf{P}_{8/12}$ and $\mathbf{P}^{-1}_{8/12}$ is presented in Fig. 1.

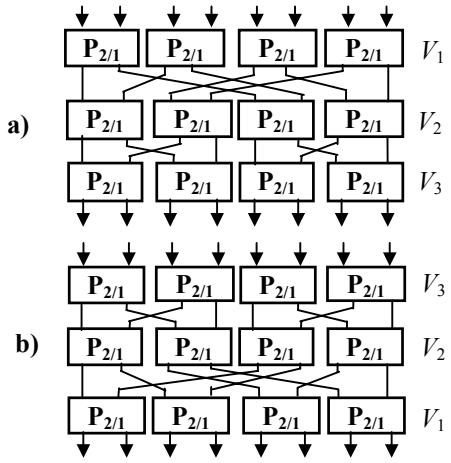

Figure 1
The boxes $\mathbf{P}_{8/12}$ (a) and $\mathbf{P}^{-1}_{8/12}$ (b).

Suppose for arbitrary $h \le n$ input bits $x_{\alpha_1}, x_{\alpha_2}, \ldots, x_{\alpha_h}$ and arbitrary h output bits $y_{\beta_1}, y_{\beta_2}, \ldots, y_{\beta_h}$ there is at least one value of the controlling vector V which specifies a CP-box permutation moving x_{α_i} to y_{β_i} for all $i = 1, 2, \ldots, h$. Such a $\mathrm{P}_{n/m}$-box is called a CP-box of order h [1]. Is is easy to see that the boxes $\mathbf{P}_{8/12}$ and $\mathbf{P}^{-1}_{8/12}$ have the first order.

Figure 2 shows structure of the second-order boxes $\mathbf{P}_{32/96}$ and $\mathbf{P}^{-1}_{32/96}$. Due to symmetric structure the mutual inverses $\mathbf{P}_{32/96}$ and $\mathbf{P}^{-1}_{32/96}$ differ only with the distribution of controlling bits over the boxes $\mathbf{P}_{2/1}$ in the same topology. When performing DDP operations with CPB $\mathbf{P}_{32/96}$ we form 96-bit controlling vector depending on some 32-bit data subblock. Let L be a controlling data subblock. Thus, bits of $L = (l_1, \ldots, l_{32})$ are used on the average three times while

defining the controlling vector. When designing respective extension box it is reasonable to use the following criteria:

Criterion 1. Let $X = (x_1, ..., x_{32})$ is the input vector of the $\mathbf{P}^{(V)}_{32/96}$-box. Then for all L and i the bit x_i should be permuted depending on six different bits of L.

Criterion 2. For all i the bit l_i should define exactly three bits of V.

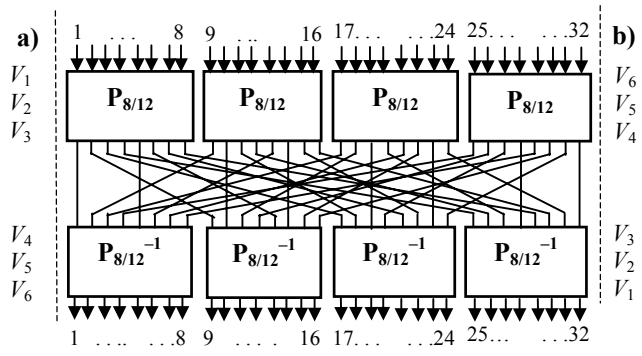

Figure 2
The boxes $\mathbf{P}_{32/96}$ (a) and $\mathbf{P}^{-1}_{32/96}$ (b).

Below we use the estension box **E** providing the following relation between V and L:

$$V_1 = L_l; \quad V_2 = L_l^{>>>6}; \quad V_3 = L_l^{>>>12};$$
$$V_4 = L_r \quad V_5 = L_r^{>>>6}; \quad V_6 = L_r^{>>>12},$$

where $L_l = (l_1, ..., l_{16})$, $L_r = (l_{17}, ..., l_{32})$, and $Y = X^{>>>k}$ denotes rotation of the n-bit word X by k bits, where we have $y_i = x_{i+k}$ for $1 \le i \le n-k$ and $y_i = x_{i+k-n}$ for $n-k+1 \le i \le n$. Due to symmetric structure of $\mathbf{P}_{32/96}$ its modifications $\mathbf{P}^{(V)}_{32/96}$, where $V = (V_1, V_2, \ldots, V_6)$, and $\mathbf{P}^{(V')}_{32/96}$, where $V' = (V_6, V_5, \ldots, V_1)$ are mutually inverse.

This property of the symmetric CPB can be used in order to construct switchable CP boxes. This idea can be realized using very simple transposition box $\mathbf{P}^{(e)}_{96/1}$ implemented as some single layer CPB consisting of three parallel single-layer boxes $\mathbf{P}^{(e)}_{2\times16/1}$ (Fig. 3a). Input of each $\mathbf{P}^{(e)}_{2\times16/1}$-box is divided into 16-bit left and 16-bit right inputs. The box $\mathbf{P}^{(e)}_{2\times16/1}$ contains 16 parallel $\mathbf{P}^{(e)}_{2/1}$-boxes controlled with the same bit e. For example, $\mathbf{P}^{(0)}_{2\times16/1}(U) = U$ and $\mathbf{P}^{(0)}_{2\times16/1}(U) = U' = (U_r, U_l)$, where $U = (U_l, U_r) \in \{0, 1\}^{32}$. The left (right) inputs of the $\mathbf{P}^{(e)}_{2/1}$-boxes correspond to the left (right) 16-bit input of the box $\mathbf{P}^{(e)}_{2\times16/1}$. If the input vector of the box $\mathbf{P}^{(e)}_{96/1}$ is $(V_1, V_2, \ldots, V_6)$, then at the output of $\mathbf{P}^{(e)}_{96/1}$ we have $V' = (V_1, V_2, \ldots, V_6)$ (if $e = 0$) or $V' = (V_6, V_5, \ldots, V_1)$ (if $e = 1$). Structure of the switchable CPB $\mathbf{P}^{(L,e)}_{32/32}$ is shown in Fig. 3b. In hardware the box $\mathbf{P}^{(e)}_{2/1}$ can be implemented using 6 nand gates transistors. The operational box $\mathbf{P}^{(L,e)}_{32/32}$ can be implemented with 864 nand gates. The time delay of some CP box is defined by the number of active layers. Time delay of one layer is approximately equal to that of the XOR operation $t_\oplus$. Time delay of the $\mathbf{P}^{(L,e)}_{32/32}$-box operation ($6t_\oplus$) is less than that of the addition modulo 2^{32} with high-speed carry. Straightforward estimates show that the $\mathbf{P}^{(L,e)}_{32/32}$-instruction can be added in microprocessor within less than 432 sqmil (for the 0.33 μm ASIC technology). Thus, the CPB $\mathbf{P}^{(L,e)}_{32/32}$ can be easily implemented as a new fast instruction on some 32-bit processors and microcontrollers.

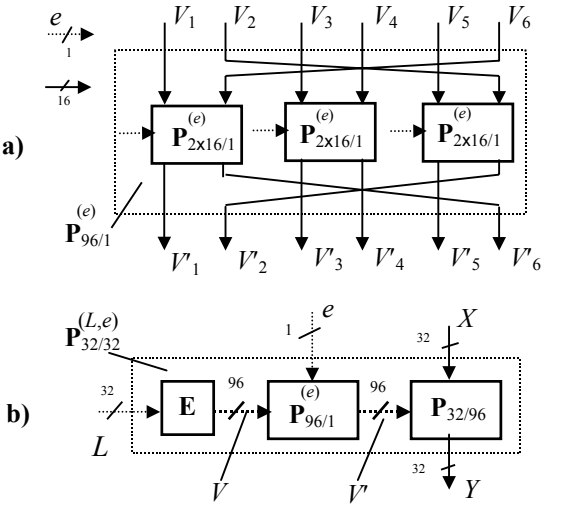

Figure 3
Switchable CPB $\mathbf{P}^{(e)}_{96/1}$ (a) and $\mathbf{P}^{(e)}_{32/32}$ (b).

Another interesting variant is embedding the nine-layer CPB $\mathbf{P}^{(V)}_{32/144}$ of the maximal order ($h = 32$) [5]. The operation $\mathbf{P}^{(V)}_{32/144}(X)$ can perform arbitrary given bit permutation on 32-bit words. The hardware implementation cost of this instruction is about the same as that of the switchable CPB $\mathbf{P}^{(L,e)}_{32/32}$. Performing the operation $\mathbf{P}^{(V)}_{32/144}$ takes 1-2 cycles (depending on the architecture of the hypothetical microcontroller or CPU). Operation $\mathbf{P}^{(V)}_{32/144}$ can be used for cryptographic purposes (construction of fast ciphers and hash functions) and for some other special purposes. For example, the instruction $\mathbf{P}^{(V)}_{32/144}$ allows to perform on a 32-bit word $X = (X_1, X_2, X_3, X_4)$ different variants of rotation operation:

$$Y = X^{>>>g}, \text{ where } 0 \le g \le 31,$$
$$Y = (X_1^{>>>g_1}, X_2^{>>>g_2}, X_3^{>>>g_3}, X_4^{>>>g_4}), \quad 0 \le g_1, \ldots, g_4 \le 7,$$
$$Y = ((X_1, X_2)^{>>>g_5}, (X_3, X_4)^{>>>g_6}), \quad 0 \le g_5, g_6 \le 15,$$
$$Y = (X_1^{>>>g_7}, (X_2, X_3, X_4)^{>>>g_8}), \quad 0 \le g_7 \le 7, \ 0 \le g_8 \le 23,$$

In addition to being well-suited towards cryptographic purposes, $\mathbf{P}^{(V)}_{32/144}$ can be used for fast and efficient implementations of a number of common software functions. A prominent example is the bit-reversal permutation, which is used in a number of Fast Fourier Transform (FFT) algorithms. A large number of multimedia applications apply the Discrete Cosine Transform (DCT) or Discrete Fourier Transform (DFT) as steps in the processing of multimedia data. Many implementations rely on the FFT to perform these transforms. On a general-purpose uniprocessor, a bit-reversal operation can require 50 or more cycles to execute. The instruction $\mathbf{P}^{(V)}_{32/144}$ could perform a

single bit-reversal in as little as one cycle. Other permutations could be used to dramatically increase the performance of higher-radix FFTs.

Thus, the CP-box instruction $\mathbf{P}^{(V)}{}_{32/144}$ can replace the already embedded rotation operation, economizing hardware resources and reducing to a minimum the hardware cost of the implementation of the CP-box instruction. If the CPU makers support encryption method based on DDP, then cryptographers will have the possibility to develop different variants of the software-oriented ciphers and hash functions based on DDP providing performance 400 - 1000 Mbit/s and more. In present paper we consider the instruction $\mathbf{P}^{(L,\,e)}{}_{32/32}$ which is oriented to cryptographic use.

III. Firmware and Software Oriented DDP-Based Ciphers

We propose two 64-bit firmware-suitable ciphers Cobra-F64a and Cobra-F64b and a 128-bit software-oriented cipher Cobra-S128. All ciphers use 128-bit key $K = (K_1, K_2, K_3, K_4)$, where $\forall\, i\ K_i \in \{0, 1\}^{32}$. No secret key preprocessing is used. While performing j round transformation subkeys are used directly as 32-bit round subkeys $Q_j^{(1,\,e)}$, $Q_j^{(2,\,e)}$, where $j = 1, ..., R + 1$ and $e = 0$ ($e = 1$) denotes encrypton (decryption). The number of rounds is $R = 16$ for Cobra-F64a, $R = 20$ for Cobra-F64b and $R = 12$ (8) for Cobra-S128. Correspondence between secret key and round subkeys is defined by Table 1 and the following formulas

for Cobra-F64a and Cobra-F64b:

$$(Q_1^{(1,\,1)}, Q_{R+1}^{(1,\,1)}) = (Q_{R+1}^{(1,\,0)}, Q_1^{(1,\,0)}),$$
$$(Q_1^{(2,\,1)}, Q_{R+1}^{(2,\,1)}) = (Q_{R+1}^{(2,\,0)}, Q_1^{(2,\,0)}),$$
$$(Q_j^{(1,\,1)}, Q_j^{(2,\,1)}) = (Q_{R-j+2}^{(2,\,0)}, Q_{R-j+2}^{(1,\,0)}),\quad \forall\, j = 2, ..., R,$$

and for Cobra-S128:

$$(Q_j^{(1,\,1)}, Q_j^{(2,\,1)}) = (Q_{R-j+1}^{(2,\,0)}, Q_{R-j+1}^{(1,\,0)}),\quad \forall\, j = 1, ..., R.$$

A. Firmware-suitable ciphers

Input 64-bit data block X is divided into two 32-bit subblocks A and B. Encryption and decryption described by the general formula $Y = \mathbf{F}^{(e)}(X, K)$ are performed in two stages: i) R rounds with e-dependent procedure $\mathbf{Crypt}^{(e)}$ and ii) final transformation. Due to peculiarities of the structure of the round transformation of Cobra-F64a and Cobra-F64b initial transformation is not used. For both ciphers the data ciphering algorithm can be represented as follows:

1. For $j = 1$ to $R - 1$ do:
$\{(A, B) := \mathbf{Crypt}^{(e)}(A, B, Q_j^{(1,\,e)}, Q_j^{(2,\,e)});\ (A, B) := (B, A)\}$.
2. For $j = R$ do: $\{(A, B) := \mathbf{Crypt}^{(e)}(A, B, Q_j^{(1,\,e)}, Q_j^{(2,\,e)})\}$.
3. Perform final transformation:
$\{Y = (Y_l, Y_h) := (A \oplus Q_{R+1}^{(1,\,e)}, B \oplus Q_{R+1}^{(2,\,e)})$, where Y is the 64-bit output data block, for Cobra-F64b or $Y = (Y_l, Y_h) := (A -_{32} Q_{R+1}^{(1,\,e)},\quad B +_{32} Q_{R+1}^{(2,\,e)})$, where "$+_{32}$" ("$-_{32}$") denotes modulo 2^{32} addition (subtraction), for Cobra-F64a$\}$.

The procedure $\mathbf{Crypt}^{(e)}$ is described in Fig. 4a (for Cobra-F64a) and in Fig. 4b (for Cobra-F64b). Both procedures $\mathbf{Crypt}^{(e)}$ are based on the instruction $\mathbf{P}^{(L,\,e)}{}_{32/32}$ in which the controlling vector is specified with the left data subblock. In a cheap firmware implementation these ciphers provide performance about 20 Mbit/s for some microcontroller working at 30 MHz.

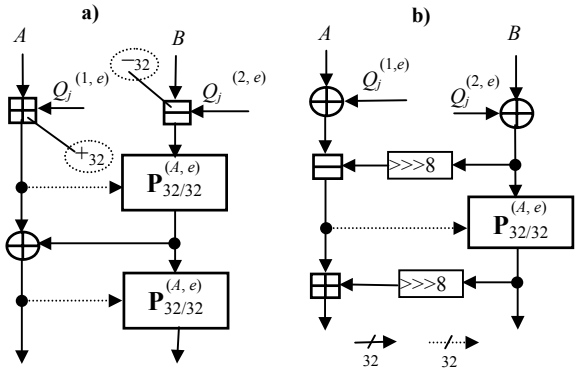

Figure 4
Procedure $\mathbf{Crypt}^{(e)}$ in Cobra-F64a (a) and Cobra-F64b (b).

Table I
Key scheduling in Cobra-F64a, Cobra-F64b, and Cobra-S128

j	$Q_j^{(1,0)}$	$Q_j^{(2,0)}$	j	$Q_j^{(1,0)}$	$Q_j^{(2,0)}$
1	K_1	K_4	12	K_3	K_2
2	K_2	K_3	13	K_1	K_3
3	K_3	K_1	14	K_4	K_1
4	K_4	K_2	15	K_2	K_3
5	K_2	K_3	16	K_3	K_4
6	K_1	K_2	17	K_1	K_2
7	K_4	K_1	18	K_4	K_1
8	K_3	K_4	19	K_3	K_4
9	K_1	K_2	20	K_1	K_2
10	K_2	K_3	21	K_2	K_3
11	K_4	K_1	-	-	-

B. Software-encryption system Cobra-S128

Input 128-bit data block X is divided into four 32-bit subblocks A, B, C, D and data ciphering procedure $Y = \mathbf{F}^{(e)}(X, K)$ is performed as follows:

1. Perform initial transformation:
$\{(A,B,C,D) := (A \oplus Q_1^{(1,\,e)}, B \oplus Q_2^{(1,\,e)}, C \oplus Q_3^{(1,\,e)}, D \oplus Q_4^{(1,\,e)})\}$.
2. Using procedure $\mathbf{Crypt}^{(e)}$ (Fig. 5) for $j = 1$ to R–1 do:
$\{(A,B,C,D) := \mathbf{Crypt}^{(e)}(A,B,C,D,Q_j^{(1,\,e)},Q_j^{(2,\,e)});\ (A,B,C,D) := (B,A,D,C)\}$.
3. Do: $\{(A,B,C,D) := \mathbf{Crypt}^{(e)}(A,B,C,D, Q_R^{(1,\,e)}, Q_R^{(2,\,e)})\}$.
4. Perform final transformation:

$$\{Y = (A,B,C,D) :=$$
$$:= (A \oplus Q_R^{(2,e)}, B \oplus Q_{R-1}^{(2,e)}, C \oplus Q_{R-2}^{(2,e)}, D \oplus Q_{R-3}^{(2,e)})\}.$$

We estimate that the twelve-round (eight-round) Cobra-S128 can provide very high performance, about 400 (600)

Mbit/s, for some hypothetical Pentium-like processor with the embedded CP-box instruction $\mathbf{P}^{(L,e)}{}_{32/32}$.

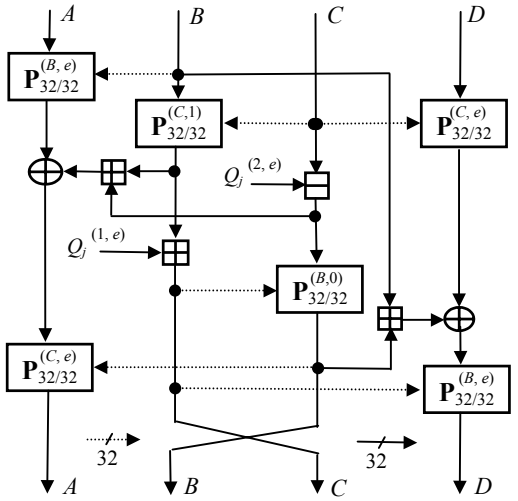

Figure 4
Procedure **Crypt**$^{(e)}$ in Cobra-S128.

C. *Secyrity estimation results*

Trying different attack against the DDP-based Cobra ciphers we have found the differential analysis (DA) is the most efficient one. Let Δ_h^X denote the X value difference with h active (non-zero) bits and $p(r|\Delta)$ denote the probability that the Δ difference passes r rounds without change. Our best DA uses the differences $\Delta^{(1)} = (\Delta_1^A, \Delta_0^B)$, $\Delta^{(2)} = (\Delta_0^A, \Delta_1^B)$, and $\Delta^{(3)} = (\Delta_1^A, \Delta_0^B, \Delta_0^C, \Delta_1^D)$ for which we have get the following probabilities $p(3|\Delta^{(1)}) = 2^{-21}$, $p(2|\Delta^{(2)}) = 2^{-12}$, and $p(2|\Delta^{(3)}) = 2^{-32}$ for the Cobra ciphers F64a, F64b, and S128, correspondingly. From the last values we derive that the ciphers are indistinguishable from a random transformation for $R \geq 9$ (for Cobra-F64a), $R \geq 10$ (for Cobra-F64b), and $R \geq 8$ (for Cobra-S128).

The used key scheduling is secure against basic related-key attacks. In spite of the simplicity of the key schedule the "symmetric" keys $K' = (X, Y, X, Y)$ or $K'' = (X, X, X, X)$ are not weak, since encryption and decryption require change of the parameter e. Indeed, from Fig. 4 and 5 it is easy to see that for all considered ciphers we have $\mathbf{F}^{(e=0)}(C, K'') \neq \mathrm{M}$, where $C = \mathbf{F}^{(e=0)}(M, K'')$. For comparison we can note that for all X the key $K'' = (X, X, X, X)$ is weak for SPECTR-H64 [5] that does not use switchable CPB operations. It seems to be difficult to calculate a semi-weak key-pair for the Cobra ciphers, if it is possible at all.

Thus, using the switchable operations one can prevent weak and semi-weak keys in the case of simple key scheduling. Some other items of the use of switchable controlled operations are considered in [6]. In the case when keys are not changed often one can use one of the known key scheduling procedures providing pseudorandom generation of the round keys.

IV. DISSCUSSION

The DDP earlier used in several hardware-oriented 64-bit ciphers can be also effectively used for designing fast software-suitable cryptosystems. We have proposed to embed some CP-box instruction in general purpose processors and in different types of microcontrollers. A simple variant of the fast switchable $\mathbf{P}^{(L,e)}{}_{32/32}$-box instruction has been designed and used in one 128-bit software and two firmware-oriented 64-block ciphers illustrating efficiency of the cryptographic use of this instruction. More advanced $\mathbf{P}^{(V)}{}_{32/144}$-box instruction can perform all possible bit-permutation operations on 32-bit words. Each of such operations can be specified by the controlling vector V and it is not difficult to find its value for all possible permutational operations [4] including special ones.

This spreads significance of the advanced CP-box instruction far beyond cryptographic applications and can attract serious attention of the CPU manufactures, since a cheap embedding of the $\mathbf{P}^{(V)}{}_{32/144}$-box instruction imparts attractive properties to the general purpose processors. One of the lasts is the potential possibility to get more than 500 Mbit/s encryption speed in software. The ability to perform special permutations, such as bit-reversal, can significantly improve the performance of multimedia applications which rely on efficient DCT and DFT algorithms. We hope this work will attract more attention of cryptographic community to DDP in respect of the cryptanalysis and design of the DDP-based block ciphers, hash functions, and key expansion algorithms.

[1] Moldovyan A.A. and Moldovyan N.A., "A cipher based on data-dependent permutations", *Journal of Cryptology,* 2002, vol. 15, no. 1, pp.61-72.

[2] Sklavos N. and Koufopavlou O., "Architectures and FPGA Implementations of the SCO (-1,-2,-3) Ciphers Family", proceedings of the 12th International Conference on Very Large Scale Integration, (IFIP VLSI SOC '03), Darmstadt, Germany, December 1-3, 2003.

[3] Sklavos N. and Koufopavlou O., "Data Dependent Rotations, a Trustworthy Approach for Future Encryption Systems/Ciphers: Low Cost and High Performance", *Computers and Security, Elsevier Science Journal*, vol 22, no 7, 2003.

[4] .Waksman A.A, "Permutation Network", *Journal of the ACM*, vol. 15, no 1 (1968), pp. 159-163.

[5] Goots N.D., Izotov B.V., Moldovyan A.A., and Moldovyan N.A., *Modern cryptography: Protect Your Data with Fast Block Ciphers,* A-LIST Publishing, Wayne, 2003.- 400 p. (www.alistpublishing.com).

[6] Moldovyan N.A., "On Cipher Design Based on Switchable Controlled operations", proceedings of MMM-ANCS'03, *Lecture Notes in Computer Science*, Springer verlag, vol. 2776 (2003), pp. 316-327.