ELectronics EXpress

LETTER

IEICE Electronics Express, Vol.13, No.21, 1–6

Hardware accelerated search for resource-efficient and secure permutation matrices

Tolga Yalçın[a)]

Computer Engineering Dept, Food and Agriculture University, Konya, Turkey

a) *tolga.yalcin@gidatarim.edu.tr*

Abstract: Permutation layer is a core component of substitution-permutation network block ciphers. Its design directly affects security and resource usage of the block cipher. It is a challenging problem to find permutation matrices with respect to predefined trade-off targets. In our work, we developed a hardware search engine on Xilinx Virtex-6 FPGA in order to accelerate the search of resource-efficient and secure (maximal branch number) 16×16 permutation matrices. Our engine completed the full spectrum search in 129 hours 48 minutes and found non-involutory and involutory permutation matrices with maximal branch number of 5 and minimum Hamming weight (HW) of 74 and 80, respectively. To the best of our knowledge, this is the *first* time that such a hardware accelerated custom search engine has been built and full spectrum permutation matrix search has been performed.

Keywords: hardware acceleration, FPGA, symmetric cryptography, block cipher, permutation layer

Classification: Electron devices, circuits and modules

References

[1] M. Albrecht, *et al.*: CRYPTO (2014) 57 (DOI: 10.1007/978-3-662-44371-2_4).
[2] M. Albrecht, *et al.*: EUROCRYPT (2015) 430 (DOI: 10.1007/978-3-662-46800-5_17).
[3] A. Bogdanov, *et al.*: CHES (2007) 450 (DOI: 10.1007/978-3-540-74735-2_31).
[4] D. Hong, *et al.*: CHES (2006) 46 (DOI: 10.1007/11894063_4).
[5] C. De Cannière, *et al.*: CHES (2009) 272 (DOI: 10.1007/978-3-642-04138-9_20).
[6] Z. Gong, *et al.*: RFIDSec (2011) 1.
[7] C. Lim and T. Korkishko: LNCS **3786** (2006) 243 (DOI: 10.1007/11604938_19).
[8] J. Guo, *et al.*: CHES (2011) 326 (DOI: 10.1007/978-3-642-23951-9_22).
[9] K. Shibutani, *et al.*: CHES (2011) 342.
[10] J. Daemen and V. Rijmen: IMA (2001) 222.
[11] S. Wu, *et al.*: SAC (2012) 355.
[12] B. Aslan and T. Sakallı: Security Commun. Networks **7** (2014) 53 (DOI: 10.1002/sec.556).

DOI: 10.1587/elex.13.20160352
Received April 8, 2016
Accepted April 22, 2016
Publicized May 13, 2016
Copyedited November 10, 2016

1 Introduction

Hardware acceleration replaces computationally intense operations in an algorithm with a dedicated hardware engine. Since Application Specific Integrated Circuits (ASICs) offer very limited or even no reconfigurability, Field Programmable Gate Arrays (FPGAs) are the logical choices as the hardware accelerator platforms. They offer affordable prices even for low-budget researchers and their ease-of-use can dramatically affect design and development process. In our work, we make utilize FPGAs for hardware acceleration in solving a specific symmetric cryptography problem, which computationally requires very high-performance.

Substitution-permutation network type of block ciphers have a non-linear substitution layer and a linear permutation layer to provide confusion and diffusion, respectively. Permutation operation in a block cipher can be defined as a multiplication of the state vector with a binary (permutation) matrix. Design of permutation layer directly affects block cipher performance both in terms of security and resource usage [1, 2]. Number of non-zero entries of the permutation matrix determines cost of multiplication. *One* entries of a binary matrix constitute to XOR gates in hardware. Low number of *ones* is desirable especially for lightweight block ciphers [3, 4, 5, 6, 7, 8, 9].

Branch number is an integral tool for the security of a cipher. It corresponds to the minimal number of active S-boxes for two consecutive rounds of a cipher. For better security, we need binary matrices with higher branch number, which generally have a large HW. In our study, we focus on the search for 16×16 permutation matrices with the lowest possible HW and maximal linear/differential branch number.

For this, we developed a simple and yet efficient search engine architecture on hardware and found maximal branch number non-involutory and involutory matrices with lowest HW. This is the first time that such an architecture has been realized and a comprehensive search has been performed. We applied several search optimizations to reduce the computational complexity from 2^{256} down to $2^{47.4}$ and utilized high performance of our architecture to complete our search in less than 6 days, which would otherwise take years. We also investigated the effect of HW on actual hardware realization of the permutation layer. It should be noted that the proposed search engine targets maximal branch number, therefore simple permutation layers with lower costs (as in PRESENT or PRINCE) are outside the scope of this study.

The background information and the details of our architecture are presented in the following sections.

2 Background

We denote the field with two elements by $\mathbb{F}_2$ and the n-dimensional vector-space over $\mathbb{F}_2$ by $\mathbb{F}_2^n$, and we split the vector-space $\mathbb{F}_2^{16}$ into the nested vector-space $(\mathbb{F}_2^4)^4$ (corresponding to the application of 4 parallel S-boxes of 4 bits each) and consider linear mappings $L : (\mathbb{F}_2^4)^4 \rightarrow (\mathbb{F}_2^4)^4$.

Branch number corresponds to the minimal number of active S-boxes in any two consecutive rounds. Differential branch number and linear branch number are

DOI: 10.1587/elex.13.20160352
Received April 8, 2016
Accepted April 22, 2016
Publicized May 13, 2016
Copyedited November 10, 2016

defined as $\mathcal{B}_d(L) := \min\{wt_4(x) + wt_4(L(x)) \mid x \in (\mathbb{F}_2^4)^4, x \neq 0\}$ and $\mathcal{B}_l(L) := \min\{wt_4(x) + wt_4(L^*(x)) \mid x \in (\mathbb{F}_2^4)^4, x \neq 0\}$, respectively. $wt_4(x)$ is the weight of a given vector $x = (x_1, \ldots, x_4) \in (\mathbb{F}_2^4)^4$, $x_i \in \mathbb{F}_2^4$, and is defined as $wt_4(x) = |\{1 \leq i \leq 4 \mid x_i \neq 0\}|$. L^* is the inverse of the adjoint linear mapping, and it corresponds to inverse of the transposed matrix of L.

We target linear mappings with maximal branch number, which in our setting is 5. As shown in [10], they correspond to MDS codes of length 8 and dimension 4 over $\mathbb{F}_2^4$. In this case, $\mathcal{B}_d(L) = \mathcal{B}_l(L)$. Thus a maximal differential branch number automatically ensures a maximal linear branch number. Let the matrix representation of L be given as

$$L = \begin{pmatrix} A_0 & A_1 & A_2 & A_3 \\ A_4 & A_5 & A_6 & A_7 \\ A_8 & A_9 & A_{10} & A_{11} \\ A_{12} & A_{13} & A_{14} & A_{15} \end{pmatrix}$$

where A_i are each 4×4 matrices. Theorem 4 in [11] shows that L has branch number 5 if and only if any square block-sub-matrix of L has full rank (i.e., determinants of all smaller square matrices in L are non-singular).

The easiest and rather powerful way of construction mappings with branch number 5 is to use the finite field $\mathbb{F}_{2^4}$ as the basis alphabet instead of the vector-space $\mathbb{F}_2^4$ only. A limited search (focusing only on cyclic matrices of this form) results already in rather competitive non-involutory matrices with HW of only 76, whereas the best reported involutory matrices have HW of 112 [12]. We therefore focus our search on HW below 76 and 112 for non-involutory and involutory matrices, respectively.

3 Proposed search engine architecture

In our search engine architecture, we check determinants of all smaller square matrices in L. If they are all non-singular, then the linear and differential branch number of L is 5. Smaller square matrices within L can be listed as: 4×4 matrices (from A_0 to A_{15}), 8×8 and 12×12 square matrix combinations, and finally the overall 16×16 matrix L. 8×8 and 12×12 square matrices can be formed as

$$\begin{pmatrix} A_0 & A_1 \\ A_4 & A_5 \end{pmatrix}, \ldots, \begin{pmatrix} A_{10} & A_{11} \\ A_{14} & A_{15} \end{pmatrix}, \begin{pmatrix} A_0 & A_1 & A_2 \\ A_4 & A_5 & A_6 \\ A_8 & A_9 & A_{10} \end{pmatrix}, \ldots, \begin{pmatrix} A_5 & A_6 & A_7 \\ A_9 & A_{10} & A_{11} \\ A_{13} & A_{14} & A_{15} \end{pmatrix}.$$

The most straightforward approach to perform the binary matrix search mentioned in Section 1 is to go through all possible binary matrices, which results in an extremely high search of complexity 2^{256}. We therefore combined nested vector spaces explained in Section 2 and tree-search together with many optimizations on a hardware-based search engine.

At first, the number of 4×4 matrices is minimized by taking only matrices with a single one in each row and column (such as the identity matrix I_4). There are only 24 of such matrices. An L formed using only these 4×4 matrices can have 64 *ones* in total. However, some of the 8×8 and 12×12 matrices of such an L would be singular. There has to be at least one *non-4* 4×4 matrix in each sub-matrix, which corresponds to at least 7 out of 16 4×4 matrices with more than 4 *ones*. Even if the minimum number of 5 *ones* are used for these 7 matrices, the least number of *ones* in L will be 71. A sample configuration is

DOI: 10.1587/elex.13.20160352
Received April 8, 2016
Accepted April 22, 2016
Publicized May 13, 2016
Copyedited November 10, 2016

$$\begin{pmatrix} 5 & 4 & 4 & 4 \\ 4 & 5 & 4 & 5 \\ 4 & 5 & 5 & 4 \\ 4 & 4 & 5 & 5 \end{pmatrix}.$$

Note that this is not the only configuration for the 7 *fives* and 9 *fours* combination[1]. There are 24 such combinations, out of which only 2 are relevant as all others can be derived by row and/or column permutations of these 2 combinations. For 4×4 matrices with 5 *ones* (1 *one* in 3 rows/columns, and 2 *ones* in 1 row/column), there are 288 different matrices. The complexities are still extremely high ($\approx 2^{98}$).

Table I. Possible placement of 1's within L (combinations)

Number of 1's	$A_0A_1A_2A_3 \ldots A_{12}A_{13}A_{14}A_{15}$
71	5555555444444444
72	6555555444444444 5555555544444444
73	7555555444444444 6655555444444444 6555555544444444 5555555554444444

In the next step, all possible combinations for L made up of smaller sub-matrices are listed (see Table I) up to 73 *ones* (higher *ones* combinations are not shown, as they are much more than these listed). Number of *ones* in each sub-matrix ranges from 4 to 7. Since row and/or column permutations do not affect the branch number, diagonal sub-matrices can be fixed to specific ones in order to reduce the complexity. Placing sub-matrices with higher number of *ones* in the diagonal further reduces the complexity. Now, L looks like

$$\begin{pmatrix} I_5 & 4 & 4 & 4 \\ 4 & I_5 & 4 & 5 \\ 4 & 5 & I_5 & 4 \\ 4 & 4 & 5 & I_5 \end{pmatrix}.$$

Following the process of listing all possible combinations and configurations to be checked, the search engine to check determinants was designed and implemented on the target FPGA platform. It mimics the nested for-loops of a software program. In the first for-loop, non-singularity of the first 8×8 matrix is checked. If it passes, the next 8×8 and/or 12×12 matrix is checked, and so on until all square matrices are checked. This corresponds to a *tree-search*. In order to check non-singularity of each matrix combination, determinant of each square sub-matrix is computed in parallel. This flow is implemented as shown in Fig. 1.

Here, darker squares show the fixed matrices and circled ones represent the counters of that stage. In each stage, all possible values are checked for non-zero determinant, singular possibilities are canceled out, and the next stage proceeds with non-singular ones. In the end, if a matrix solution is reached where all block sub-matrices are invertible, then a binary matrix with branch number 5 is found.

[1]We refer to different matrix types (*fours*, *fives*, … etc.) to form a certain number of *ones* as combinations, and different permutations of these combinations as configurations.

DOI: 10.1587/elex.13.20160352
Received April 8, 2016
Accepted April 22, 2016
Publicized May 13, 2016
Copyedited November 10, 2016

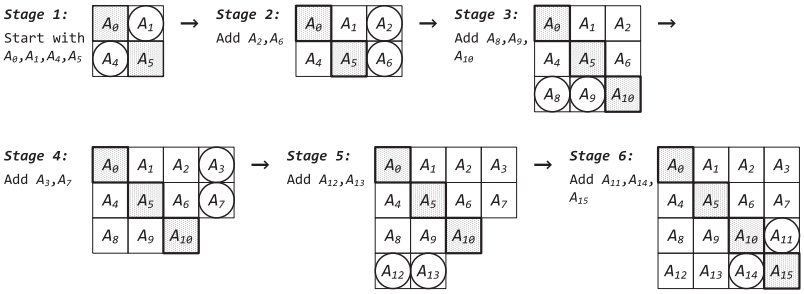

Fig. 1. Determinant check flow (stage by stage) for L

In order to exploit parallelism and further accelerate the matrix search, two search engines running in parallel are implemented. A simple circular scheduler assigns jobs to each engine. Our search engine architecture is shown in Fig. 2. At each stage of the check engine, a dual counter is kept in order to index all possible sub-matrices of that stage. The determinant check unit computes determinant of the indexed sub-matrix. Fixing the diagonal sub-matrices reduces the number of possibilities for the dual counters.

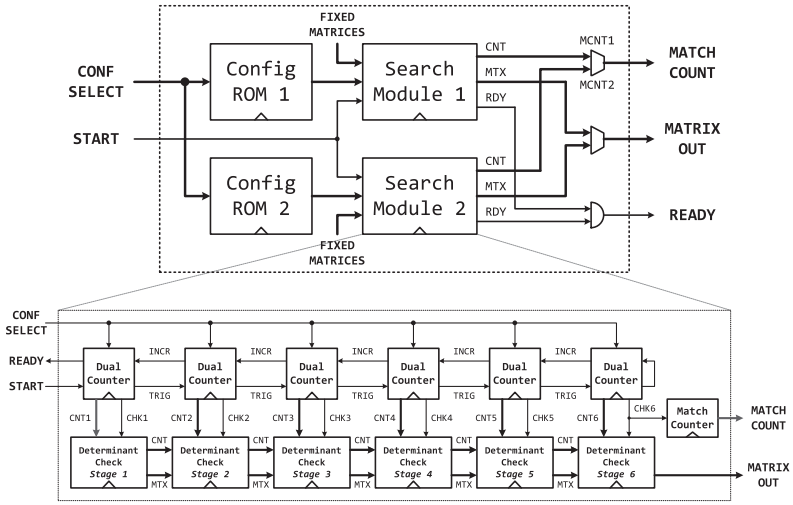

Fig. 2. Matrix search engine architecture

The proposed search engine and hence the method is fully scalable. It is possible to add more stages to the search chain in order to increase the matrix dimensions – 10 stages for a 20×20 matrix search, 15 stages for a 24×24 matrix search, and so on. Search steps up to 32×32 matrices are illustrated in Fig. 3. It is also possible to run as many search engines as allowed by the resources on the target platform in parallel. Scalable structure of the circular scheduler allows this. In our specific case, the FPGA platform allowed up to two search engines without any place-route and timing problems.

Furthermore, configuration ROMs and determinant check modules can be modified in order to implement different searches. For example, we modified them in order to search for involutory matrices only. It is also possible to replace determinant check with sparsity check in order to search for specific sparse matrices. Similarly, configuration ROMs can be filled with logic equations/tables, and determinant check can be replaced with algebraic normal form computation to search for specific S-boxes. Additionally, search speed can be adjusted by replacing dual-counters with triple or more counters.

DOI: 10.1587/elex.13.20160352
Received April 8, 2016
Accepted April 22, 2016
Publicized May 13, 2016
Copyedited November 10, 2016

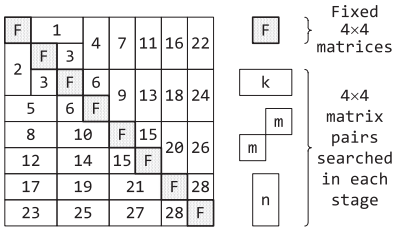

Fig. 3. Search stages up to 32×32 matrices

4 Results and conclusion

We performed our search using a Xilinx Virtex-6 ML605 kit, which provided us reconfigurability we needed in order to optimize our search. We used Xilinx ISE design tools and applied several optimizations to reduce complexity of the search from 2^{256} down to $2^{47.4}$. With two search engines running at 200 MHz and using %57 of resources on Virtex-6 XC6VLX240T, we were able to cover the whole search spectrum in less than one week. This is a speed-up factor of over $\times 100$ compared to a 256-node Opteron 6276 cluster. It is also a huge improvement considering US$ 2,000 cost of an FPGA board with respect to US$ 40,000 cost of the cluster.

This work, to the best of our knowledge, is the very first use of hardware acceleration for designing block ciphers rather than attacking them. Using our hardware architecture, we were able to find the most optimal 16×16 non-involutory and involutory matrices with HW of "74" and "80", respectively. Results of our search are listed in Table II.

In our results, we also include XOR and gate equivalent (GE) counts obtained from Synopsys DC syntheses using a 90 nm standard cell library. As can be seen, minimum Hamming weight also corresponds to minimum area. This also applies to MDS type matrices where permutation is performed using finite field arithmetics instead of direct matrix multiplication [8, 9]. It should be noted that XOR count is different from gate count, where synthesis tool further combines several XOR gates using other combinational logic gates.

Table II. Search results

Work	Hamming W.	XOR count	Area (GE)
LED [8] permutation matrix M (= A^4)	–	65	136.25
Piccolo [9] permutation matrix M	–	58	119.75
Best *non-involutory* matrix (existing)	76	60	104.75
Best *non-involutory* matrix **(this work)**	**74**	**56**	**99**
Best *involutory* matrix (existing [12])	112	66	138.25
Best *involutory* matrix **(this work)**	**80**	**53**	**113**

Acknowledgments

The author would like to thank Prof. Dr. Gregor Leander and Dr. Elif Bilge Kavun for their valuable insights and contributions.

DOI: 10.1587/elex.13.20160352
Received April 8, 2016
Accepted April 22, 2016
Publicized May 13, 2016
Copyedited November 10, 2016