Asian Journal of Scientific Research

ISSN 1992-1454

science alert
http://www.scialert.net

ANSInet
an open access publisher
http://ansinet.com

Asian Journal of Scientific Research 8 (3): 278-290, 2015
ISSN 1992-1454 / DOI: 10.3923/ajsr.2015.278.290
© 2015 Asian Network for Scientific Information

Light Weight Steganography on RISC Platform-Implementation and Analysis

Siva Janakiraman, K. Thenmozhi, John Bosco Balaguru Rayappan and Rengarajan Amirtharajan
School of Electrical and Electronics Engineering, SASTRA University, Thanjavur, 613 401, India

Corresponding Author: Siva Janakiraman, School of Electrical and Electronics Engineering, SASTRA University, Thanjavur, 613 401, India

ABSTRACT

The emergence of modern digitization has resulted in versatility to eradicate the divergence among the forms of information travel flanked by the users. This paper presents a pliable approach for the erratic block size selection in an impulsive mode to boost the level of sophistication in stego algorithm. An ingrained formula for key exchange, suggested in the algorithm combines the benefit of cryptography adjoining with steganography. In contrast to the usual implementations using generic software and personal computers, the suggested software development has been intense on an embedded device LPC 2378 with the RISC architecture that includes extensive support for networking through on-chip modules supporting ethernet and CAN protocols. The focal plan of this work includes elimination of key exchange for data encryption and improving the security to a massive level without compromising the image quality and embedding capacity. This endeavor shows the aptness of embedded hardware for stego implementations using an image carrier that makes soaring demand on memory; the extremely inhibited resource of embedded devices. The efficiency of the algorithm in maintaining image quality has been measured using the metrics MSE and PSNR. The enhancement in performance of embedded software, in terms of speed and code size have been analyzed under sophisticated compiler tools from KEIL MDK and IAREW.

Key words: Light weight steganography, hardware security, embedded security, ARM, compiler optimization

INTRODUCTION

Private information sharing among the people in the current digital world through an assorted number of electronic gadgets is rising every day. This includes text, audio, still images, video, etc. Perceptible data voyage through any medium, although in illegible form may bid the eavesdroppers and makes our data defenceless. Finding the very existence of data, when hidden behind a cover becomes difficult and reduces the likelihood of vulnerability (Zaidan *et al*., 2010). The significance of any sort of information, when represented in digital form diminishes dreadfully, as we progress towards lower bit positions. Therefore, changes in the bit values present in LSB positions may not upshot conspicuous differences, when converted back to its original form.

Bearing this fact in mind, embedding techniques using LSB substitution methods has become accepted in steganography (Amirtharajan and Rayappan, 2013; Chan and Cheng, 2004; Cheddad *et al*., 2010). Variable number of data bits embedding on the LSB positions of a cover pixel as dictated by the bits in the MSB positions of the same pixels are also in practice. The other

form of LSB technique called LSB matching finds the best matching portion of the cover to embed the data. Although, the matching technique helps in maintaining excellent image quality by minimizing the distortions, it slows down the pace of the algorithm on producing the stego image. Inclusion of randomization during the selection of cover pixel sequence for data embedding has been a means to improve security.

Few of these methods oblige sending of sequence numbers that serves as a key to the receiver end, prior to the transfer of storage cover. Efforts have been taken to make a trade-off with embedding capacity that makes room in the stego cover to clutch the needed key information. In addition, procedures for dividing the data into blocks of equal sizes and encrypting the data ahead of embedding were as well exercised for improved protection of data. Scrambling the data prior to embedding on a cover provides a two fold security level (Amirtharajan *et al*., 2013). The use of encryption can be made at the augmented cost of key exchange algorithms apart from data exchange (Abomhara *et al*., 2010; Muda *et al*., 2010; Salem *et al*., 2011). The majority of these implementations were made by means of software platforms (Salem *et al*., 2011; Praveenkumar *et al*., 2014a-d, 2015).

Embedded processors and reconfigurable devices are the prime choices for the implementation of security on hardware platforms (Wang *et al*., 2007; Yalla and Kaps, 2009; Rajagopalan *et al*., 2014a-c). Limitations on memory in embedded processors invite the special kind on crypto algorithm called Light Weight Cryptography (LWC) (Eisenbarth *et al*., 2007, 2012). These algorithms curtail the demand on memory to be the most apposite option for low end embedded devices such as 8-bit and 16-bit embedded processors (Hong *et al*., 2006; Janakiraman *et al*., 2014a, b). On the other hand, security implementations using steganography with widely used covers like still images needs large store area. This requirement makes the reconfigurable devices like FPGAs to win the battle and forces the embedded processors to quit.

The development of advanced embedded processors, such as devices with ARM core satisfies the memory requisite for multimedia based security algorithms bringing them back in to the battle (Janakiraman *et al*., 2014d, e). Initially using the benefit of interfacing external memory devices, the concept of image steganography on RGB images was proved with ARM processor by Stanescu *et al*. (2009). As an alternative, the effectual use of on-chip RAM and images with grey scale pixel values were recommended to realize block based random image steganography with LPC2136 a 32-bit RISC device (Rajagopalan *et al*., 2012).

In this study a novel scheme is presented that embeds the data in encrypted form to perk up the security. The algorithm provides flexibility to the user to divide the image in to blocks of diverse measuring and to make use of dissimilar keys that encrypts the data to be embedded on each block. Every block gets 10-bit header information embedded in it that comprises of block size, encryption key and a seed for LFSR thereby eliminating the supplementary algorithm for key exchange in contrast to customary approaches. In order to satisfy the memory requirement, the device LPC2378 a quad byte RISC architecture with synthesizable ARM7TDMI core is used. The device supports 512 KB of non-volatile flash ROM with In-System Programming (ISP) and total of 58 KB Static RAM for In-Application Programming (IAP). The use of unique key that encrypts the data embedded in each block has been considered as a means to improve the security. To keep the image quality in a level comparable with LSB matching, The error reduction has been done with OPAP technique that greatly improves the error metrics MSE and PSNR and maintains the imperceptibility of the stego images.

MATERIALS AND METHODS

This work suggests a fiction block based on fixed embedding procedure on grey scale images, where, the secret data will be encrypted before the embedding process. In this sequential embedding approach, the use of different keys for the encryption of secret bytes to be embedded in each block makes the same secret to appear differently at every point of embedding, even, when same data gets embedded in multiple places. The user can decide on the number of chunks to be made in every cover image. Also the number of Cover Bytes (CB) in each block for embedding is selectable by the user. In the beginning, the user has to select the Block Size (BS), a 10 bit integer number as given in Eq. 1.

$$[(BS-5) \bmod 4] = 0 \quad (1)$$

The lower 8 bits are extracted from the selected BS value of the ith block and taken as the Base Key ($BK_i = BS_i$ and 0×FF) for encrypting the data for the ith block. Since, we go for fixed 'K' bit LSB embedding on each pixel, the number of secret data bytes to be encrypted using the selected key can be given as j = (BS-HB)/4, when, K = 2 and Header Bytes, HB = 5. A 3-bit LFSR circuit shown in Fig. 1, is used to transposition each bit of BK_i to obtain a Bit Shuffled Key (BSK_i). The 3-bit LFSR with an EX-NOR gate in its feedback path generates pseudo random numbers in the range of 0 to 6 as given in Table 1. As the LFSR circuit with a feedback path through EX-OR gate by no means output all zeros, a feedback set by an EX-NOR gate on no account produce all ones. Therefore, prior to the start of embedding the data in every block, the LFSR is made to run 7 times and the value 7 is inserted at the end to complete a cycle.

Table 1: Output of 3-bit LFSR with EX-NOR feedback

LFSR stage			
D0 = (D2 v D1)	D1 = D0	D2 = D1	Value f (x) = x^3+x+1)
0	0	1	1 (seed)
0	0	0	0
1	0	0	4
0	1	0	2
1	0	1	5
1	1	0	6
0	1	1	3

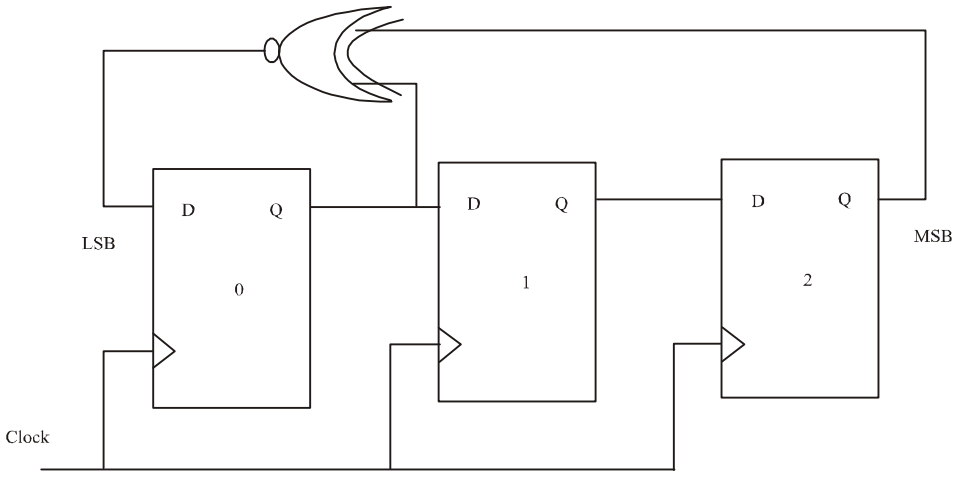

Fig. 1: LFSR (3-bit) with EX-NOR feed back

The pseudo random output of the LFSR in one cycle, PS_i for the respective seed value Sd_i dictates the sequence by which the bit positions of the 8-bit key, are to be rearranged to get a Bit Shuffled Key (BSK_i) that scrambles the data to be embedded in the ith cover block. The LFSR seed value for the ith block is agreed as $Sd_i = Bk_i \bmod 8$. The simple symmetric encryption for each secret byte comprises of a transposition mechanism followed by an Ex-or operation given by the Eq. 2 and 3, respectively.

$$[DB_j >> (BSK_i \bmod 8)] \mid [DB_j << (8-(BSK_i \bmod 8))] \quad (2)$$

$$DB_j \wedge BSK_i \text{ where, } -1<j>(BS-HB)/4 \quad (3)$$

In this approach, the selected BS value for each block is embedded in the five foregoing cover bytes of the respective block, which are called Header Bytes (HB) and the remaining cover bytes of the relevant block will get embedded with encrypted secret data block, to obtain the stego block, SB (BS_i). Block size of any selected block is given by Eq. 4.

$$BS_i \leq [(RSB \times 4 + HB)] \quad (4)$$

where, RSB is the Remaining Secret Bytes to be embedded.

In practical cases, the final block that gets the secret data embedded is called the residual block. It may not follow the size given by Eq. 1 due to the fact that the size of the cover image will usually be more than the size of secret data. Once the left over secret data bytes got embedded, the next cover block is embedded with 10 zeros in the place of header bytes called as Concluding Bits (CCB = 00 00 00 00 00) indicating the closing stage of secret data embedding.

Pseudo code for embedding:

Embedding bits per pixel, K = 2;
Header Bytes per block, HB = 5;
Cover image = ($C_m \times C_n$);
Secret data image = ($D_x \times D_y$)
Select, Number of Blocks 'r';
Select, Block Sizes = $\left(C_m \times C_n\right) = \sum_{i=0}^{r} BS_i$;
for (i = 0; i< r ; i++)
{
Block Size, BS_i: [(BS-HB) mod 4] = 0
Base Key, $BK_i = BS_i/256$;
LFSR seed value, $SD_i = BK_i \bmod 8$;
LFSR output for i^{th} block, PS_i = LFSR(SD_i);
Bit Shuffled Key, BSK_i = Bit shuffling (BK_i, PS_i);
Size of Data block to Encrypt, j = (BS-HB)/4;
Encrypted Data block, EDB_j
Encrypt (DB_j, BSK_i)
{ While (j != 0)
{
EDB_j = [DBj>>(BSK_i mod 8)] | [DB<<(8 - (BSK_i mod 8))] ^BSK_i;
}
}
Where, $\left(ED_m \times ED_n\right) = \sum_{j=0}^{r} EDB_j$;
Stego Block, Sbi = Embed(CB[BS_i,], EDB_j);
Where, $\left(S_m \times S_n\right) = \sum_{i=0}^{r} SB_i$;
}
for (i = 0; i< HB ; i++)
{
Embed (CB_{i+1}, CCB) ;

}
Display_cover ($C_m \times C_n$);
Display_secret ($D_x \times D_y$);
Display_encrypted_secret ($ED_m \times ED_n$);
Display_ stego ($S_m \times S_n$);

The retrieval process starts with the extraction of block size from the Header bytes in the first block of the stego image. The extracted BS value is used to reconstruct the Base Key and 3-bit LFSR seed, SD_i used at the encryption side. The output sequence of LFSR and Bit Shuffled Key, BSK_i to decrypt the secret data bytes in the encrypted form from every ith stego block is conceded in a way similar to the encryption process. This extraction and decryption are sustained until we attain the concluding bytes in the header. Finally, the retrieved secret image, ($RD_x \times RD_y$) is reconstructed using the decrypted data. The overview of the entire process is depicted in Fig. 2.

Pseudo code for retrieval:
Embedding bits per pixel, K = 2
Header Bytes per block, HB = 5
do
{
for (i = 0; i< HB; i++)
{
Block Size, BS_i = Extraction (SB_i)-HB;
}
Base Key, $BK_i = BS_i/256$;
LFSR seed value, SDi = BK_i mod 8;
LFSR output for i^{th} block, PS_i = LFSR(SD_i);
Bit Shuffled Key, BSK_i = Bit shuffling (BK_i, PS_i);
Size of Data block to Decrypt, j = (BS-HB)/4
Extracted Data block in Encrypted form, EDB_j = Extraction (SB[BS_i]);
Where, $\left(ED_m \times ED_n\right) = \sum_{j=0}^{r} EDB_j$;
Decrypted Data block,
Decrypt (DB_j, BSK_i)
{ While(j ! = 0)
{
DB_j = [EDB_j>>((BSK_i mod 8)] | [EDB<<(8 - (BSK_i mod 8))] ^BSK_i;
}
}
Where, Retrieved secret image $\left(RD_x \times RD_y\right) = \sum_{j=0}^{r} DB_i$ and
'r' - Number of Blocks.
}
Display_Retrieved_secret ($RD_x \times RD_y$);
Where, ($RD_x \times RD_y$) = ($D_x \times D_y$)

Hardware implementation: The algorithm focuses on grey scale images that can be fit in to the existing on-chip SRAM of 32 KB in the embedded processor LPC 2378 chosen for hardware

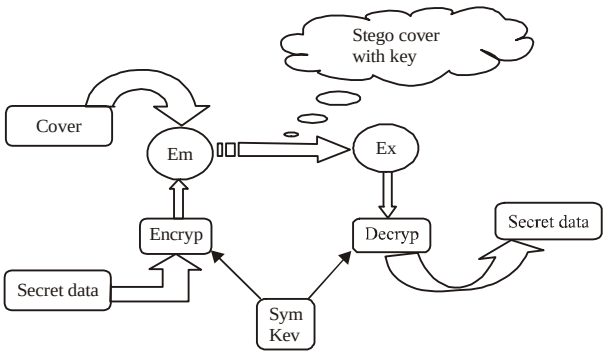

Fig. 2: Block diagram of stego system

implementation. The real cover image, stego cover, secret image in its actual plus encrypted form and the retrieved secret image as well are stored in local on-chip SRAM of 32KB. The algorithm was developed as an embedded software using the Embedded-C language. The embedded code was compiled with ARMCC compiler of KEIL MDK version 4.7 and IAR C compiler of IAR Embedded Workbench version 6.5 (IAREW). The impact of both compilers in the reduction of code size and the time taken to execute the algorithm are compared.

RESULTS AND DISCUSSION

The grey scale images with size $(C_m \times C_n) = 100 \times 100$ and $(D_x \times D_y) = 50 \times 50$ are taken as cover and secret image, respectively so as to analyze the quality of stego image at full embedding capacity. In order to store all the images in its various forms with the above said sample image sizes in addition to the memory required for any temporary data storage, the algorithm demands around 85% of the available 32 KB on-chip local SRAM. The embedding scheme seems virtually fixed, because of the fixed number of bits embedded in each pixel of the cover image (K = 2). In reality, the total data embedding capacity for the taken image size is always depends on the number of header bytes to be sacrificed. Table 2 shows the sample values for multiple number of blocks with diverse sizes, number of header bytes to be spared and the respective embedding capacity.

The use of different keys to encrypt various blocks of secret data gives a linear rise to the number of blocks in the cover image thereby demanding more cover bytes to embed header information. Trade-off can be made between the rise in security level and reduction in data embedding capacity up on dividing the cover and data in to more number of segments. The various error metrics and performance metrics of the proposed method are compared in Table 3. The error metrics MSE and PSNR were calculated as said in the Eq. 5 and 6.

$$\text{Mean Square Error, MSE} = \frac{1}{MN}\sum_{i=1}^{M}\sum_{j=1}^{N}(X_{i,j} - Y_{i,j})2 \quad (5)$$

Table 2: Sample block numbers, sizes, cover bytes and embedding capacity

No. of blocks	Total CB for header (5 per block)+ concluding bytes (5 per image)	(BS-HB)×No. of blocks = CB for embedding secret data (Cover image = 100×100)	Embedding capacity [Bits/Pixel] (when K = 2)
12	60+5	1000×9+600×1+300×1+35×1 = 9935	1.987
20	100+5	1000×5+500×7+250×4+100×3+95×1 = 9895	1.979
100	500+5	1000×1+500×9+100×20+50×20+20×49+15×1 = 9495	1.899
200	1000+5	200×10+100×20+50×20+30×100+20×49+15×1 = 8995	1.799

Data embedding capacity is given by $\{[(C_m \times C_n)-(r \times HB)-CCB] \times K\}$ bits, where, number of blocks and HB = CCB = 5

Table 3: Error metrics of proposed method

		MSE		PSNR (dB)		
No. of blocks (r)	Cover image (100×100)	Before OPAP	After OPAP	Before OPAP	After OPAP	Cover pixels altered (%)
12	Cameraman	2.4556	1.4932	44.23	46.39	75.19
	Boat	2.4499	1.5083	44.24	46.34	74.81
	House	2.5205	1.4757	44.11	46.44	74.40
20	Cameraman	2.5991	1.5111	43.98	46.33	75.03
	Boat	2.6050	1.5194	43.97	46.31	74.93
	House	2.7060	1.5092	43.80	46.34	74.66
100	Cameraman	2.9941	1.5061	43.36	46.35	74.68
	Boat	3.0150	1.5222	43.33	46.30	75.06
	House	3.0836	1.4924	43.24	46.39	74.72
200	Cameraman	3.2854	1.4822	42.96	46.42	74.39
	Boat	3.2651	1.4955	42.99	46.38	74.85
	House	3.3541	1.4973	42.87	46.37	75.09

where, $X_{i,j}$-Stego pixel value $Y_{i,j}$-Cover pixel value.

$$\text{Peak signal to noise ratio, PSNR} = 10 \log_{10} \{255^2/\text{MSE}\} \text{ dB} \quad (6)$$

As per the literature, the worst case PSNR value for K = 2 is noted as 38.59dB (Chan and Cheng, 2004). Improvements in image quality can be achieved by introducing the technique Optimum Pixel Adjustment Process (OPAP) on LSB substituted image, when K≥2. The algorithm was tested on 3 different standard cover images Cameraman, boat and house with same data image at full embedding capacity. Table 3 shows the error metrics MSE and PSNR values along with percentage of cover pixels altered in cover image for 4 different sample block sizes as given in Table 2.

The results obtained after the OPAP shows the apparent enhancement in error metrics. This is obtained at the cost of few bytes in code memory and a slight hike in cycles taken for execution. On an average 75% of the cover pixel values are getting altered after embedding. It is also clear from the results that the percentage of pixels altered in the cover does not contribute directly in picture quality rather, it depends only on the maximum possible number of pixels that can be adjusted to bring down the difference between the cover and stego pixels.

The performance of embedded code on any target device is analyzed with 2 major factors the memory footprint and speed. The prime constraint in the selection of target device is that the existing memory size should be large enough to fit in the application code and data as well. The data memory requirement is always application dependent and no programming technique can serve better in reducing this demand. The use of embedded processors for image staganography is typically restrained by its claim on data memory (RAM) the storage place for the cover and stego cover images. Some algorithms that wants to keep the cover image as static data may use a portion of program memory as the storage area for cover image. On the other hand storing the cover image in RAM helps the user for the dynamic selection of cover images before data embedding. The required amount of RAM memory for this algorithm is calculated as:

Cover image	:	100×100	:	10,000 bytes
Data image	:	50×50	:	2,500 bytes
Encrypted data	:	50×50	:	2,500 bytes
Stego cover	:	100×100	:	10,000 bytes
Total RAM memory required			:	25,000 bytes (<25KB)
Available on-chip SRAM of LPC 2378			:	32KB

This calculation ensures the suitability of LPC2378 in satisfying the RAM requirement for the image steganography implementation with the above said image sizes.

The program memory size required for the application code is based on the compiler efficiency in generating the optimal output. The compilers of IDEs for embedded systems are tailored with the option for the selection of various optimization levels that can produce results favouring speed or space. Compiler based optimization techniques were previously analyzed for LWC by Janakiraman *et al*. (2014a, b). Here two well known IDEs KEIL MDK 4.7 and IAREW 6.5 are used to compile the code with 4 different levels of optimization. The snap shots of IDEs KEIL MDK and IAREW are shown in Fig. 3 and 4, respectively. The various optimization levels in KEIL and IAR compilers used are described in Table 4.

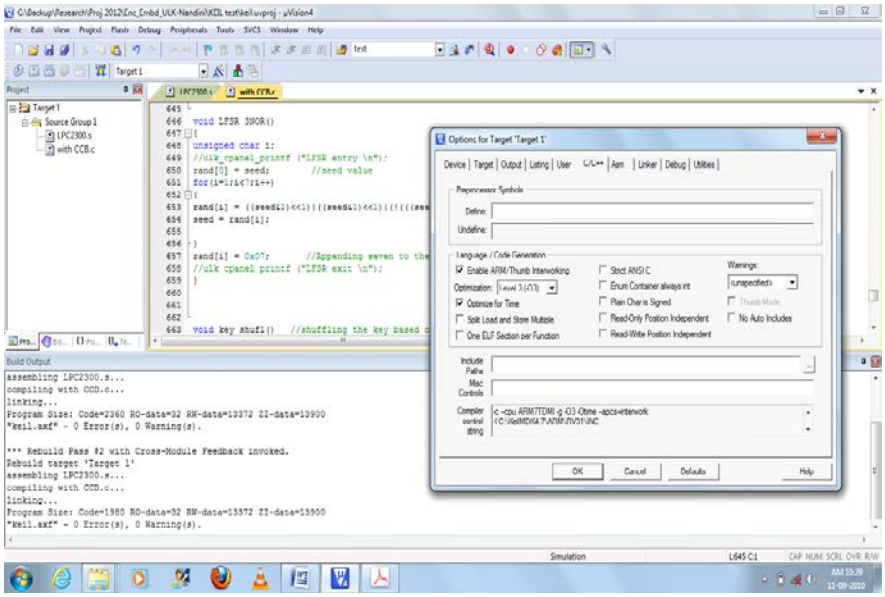

Fig. 3: Snapshot of IDE-KEIL MDK 4.7

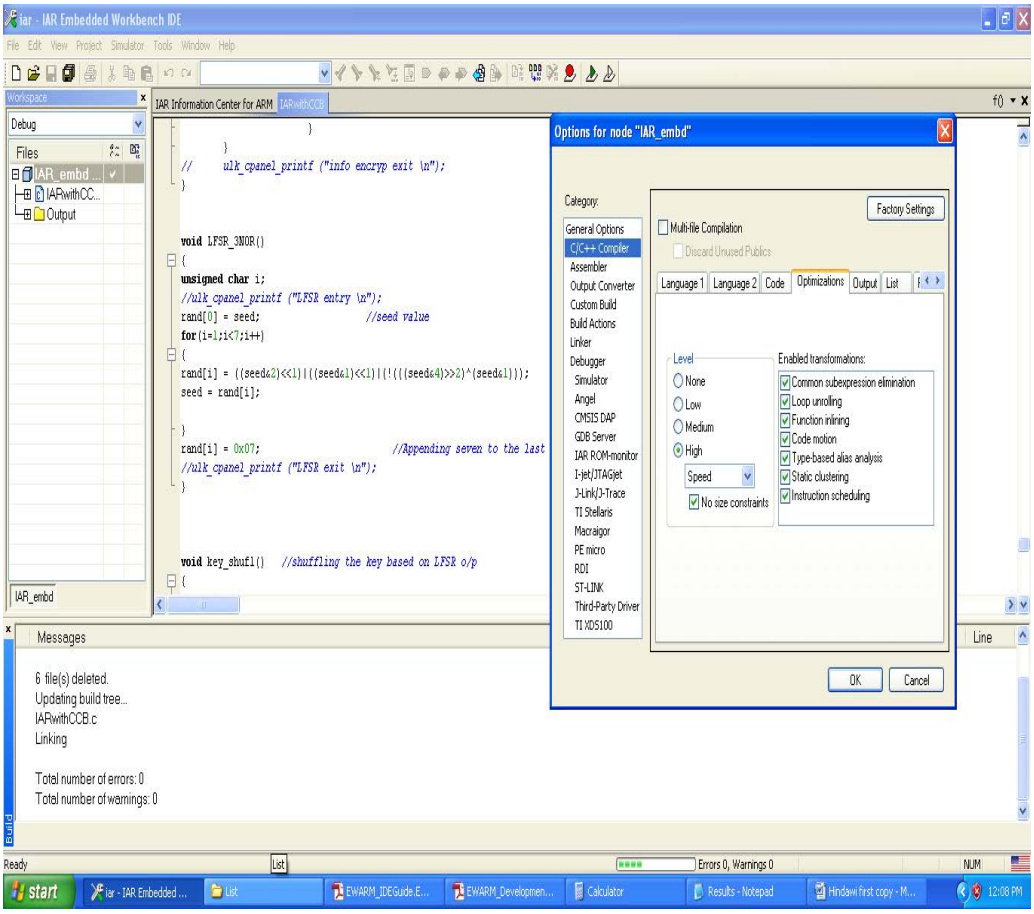

Fig. 4: Snapshot of IDE-IA REW 6.5

Table 4: Optimization levels of compilers

Optimization level	KEIL MDK4.7 ARMCC compiler	IAREW6.5 IAR C compiler
Level 0	O0-minimum optimization for space	Low-minimum balanced speed and space optimization
Level 1	O1-more optimization for z space	Medium-more balanced speed and space optimization
Level 2	O2-maximum optimization for space	High-maximum optimized for space
Level 3	O3-maximum optimization for speed	High-maximum optimized for speed (with no space constraint)

Table 5: Compiler based optimization result analysis

		Optimization levels							
		Execution time (cycle count)				Code size (bytes)			
Cover image (100×100)									
IDE/compiler	No. of blocks (r)	0	1	2	3	0	1	2	3
Keil MDK 4.7	12	1776358	900806	900609	493705	2456	1748	1720	1984
ARMCC	100	1842772	937478	935873	521891				
	200	1913944	977200	973995	552991				
IAREW 6.5 for ARM	12	908815	428392	401510	390823	680	908	844	924
IAR C	20	911538	430328	402198	392595				
	100	935624	449994	410537	409312				
	200	970924	473524	432192	417394				

Table 5 gives a detailed values of code sizes in terms of bytes and execution time in terms of cycle count obtained with four different optimization levels on both compilers.

The embedded code was written in such a way that, the code size remains constant irrespective of the image used as cover and when they are divided in to different number of blocks with various sizes. The cycle count that decides the execution time being unvarying for all the 3 cover images used in the testing. Change in number of segments (blocks) in cover images varies the execution cycle count. The inference from the results Table 5 shows that in KEIL MDK, LEVEL-2 produces lesser code size than any other levels and LEVEL-3 takes fewer cycles for execution at the expenditure of few bytes in code size than in LEVEL-2. On the other side, IAREW results in more rate values of code size and execution cycles with LEVEL-0. Using LEVEL-2 in IAREW brings more than 50% reduction in execution cycles with a slight hike in code size than in LEVEL-0. As in the case for KEIL, LEVEL-3 of IAREW becomes the most optimal choice to cut back the execution cycles.

Analyzing the competence between the compilers of KEIL MDK and IAREW on footprint reduction, KEIL takes around 5.5-7.5% whereas; IAREW outperforms KEIL by taking only 2-3% of available on-chip FLASH memory of LPC2378. As given in the Fig. 5 using IAR C compiler helps to achieve around 50% space reduction with all optimization levels.

Also in terms of execution cycles shown in Fig. 6, IAR takes 50% lesser number of cycles than KEIL in all optimization levels except in LEVEL-3 where it atleast strives to get close with IAR.

All the results in Table 5 on execution time and code size are specified excluding the OPAP function, time load input images in RAM memory (cover and data) and time to display the images on GLCD. As the footprint of code without OPAP did not even require 10% of the existing FLASH memory, we can ignore the memory overhead produced by the inclusion of OPAP technique. In order to evaluate the timing overhead generated by the OPAP function we use the debugging aid, the Performance Analyzer provided by the KEIL MDK tool.

On comparing the results obtained from performance analyzer, the data embedding function consumes more than 80% of the total execution time with all the four different block numbers. The secret image encryption process takes level of considerable time around 10% where, the remaining modules altogether needs only less than 10%. The time that needs to complete the embed module and info_encryp module are based on the size of cover and secret images respectively which are considered as constant sizes in our implementation.

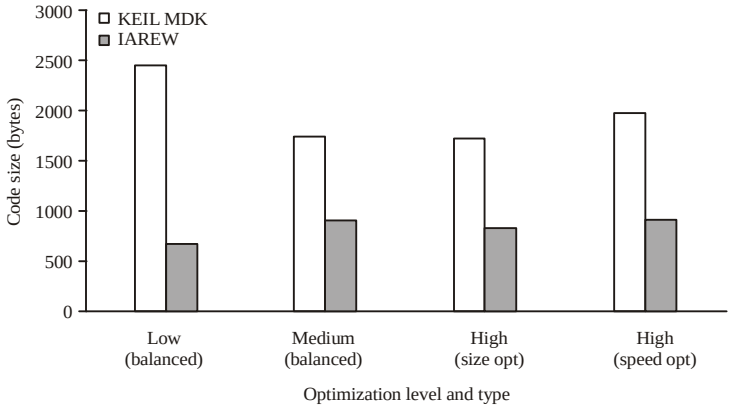

Fig. 5: Code size KEIL vs IAR

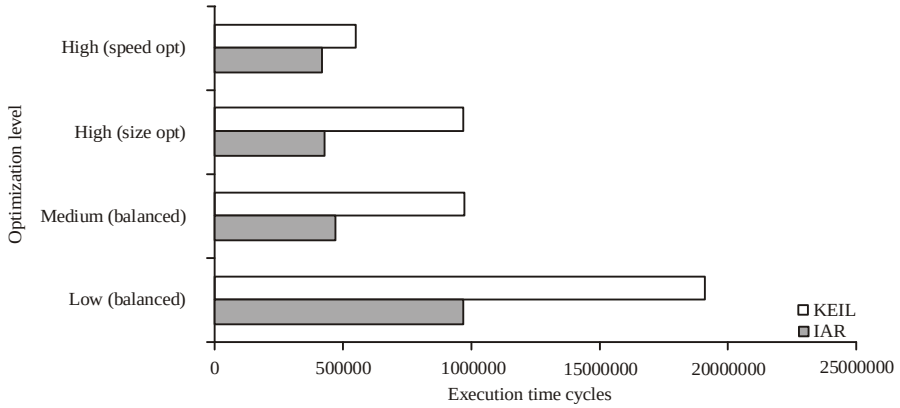

Fig. 6: Execution time KEIL vs IAR

As a result of this, even when the cover image is segmented in to more number of blocks, there is only a negligible amount of raise in execution time due to the function call overhead. The algorithm expects an output from the modules LFSR_3NOR and key_shufl for every segment of the cover image therefore, the execution time for these modules are decided only by the number of cover blocks and not by the size of cover image. This aspect makes the execution time for these module to raise linearly with the raise in number of cover blocks. On measuring the impact of OPAP, it is found around 30% raise in the execution time with improvement in stego image quality with 2 to 3 dB hike in value of PSNR. Changing the cover image when using the OPAP technique may lead to a slight difference in the total time taken by the code to complete the embedding process. This is solely depends on the number of stego pixels that needs correction from OPAP for error reduction.

Janakiraman *et al*. (2014c) proposed a byte wise XOR based encrypted embedding of secret image on gray scale cover image in method 2. The method 2 named as "Key Embedding and Encrypted Hiding" embeds the four trailing image pixels of cover image with the 8-bit key used for encrypting the secret image. In contrast, this method proposed in this study embeds the data, key and block size as well in to the cover image. On comparing the results from Table 6, the better imperceptibility on stego image achieved by the proposed method is represented with grater PSNR

Table 6: Image quality vs embedding capacity (proposed method Vs Janakiraman *et al*., 2014c-Method 2)

		Without OPAP		With OPAP		
No. of blocks (r)	Cover image	MSE	PSNR (dB)	MSE	PSNR (dB)	Embedding capacity (%)
Janakiraman *et al*. (2014c) (Method 2)	Cameraman (128×128)	17.516	35.696	NA	NA	25.0
Proposed (r = 12)	Cameraman (100×100)	2.4556	44.23	1.4932	46.39	24.8
Proposed (r = 20)	Cameraman (100×100)	2.5991	43.98	1.5111	46.33	24.7
Proposed (r = 100)	Cameraman (100×100)	3.2854	42.96	1.4822	46.42	23.7
Proposed (r = 200)	Cameraman (100×100)	2.9941	43.36	1.5061	46.35	22.5

Fig. 7: Hardware implementation with image results displayed on GLCD, Top row: Boat (100×100) Left: Original cover, Right: Stego cover, Bottom row: Peppers (50×50), Original data Centre: Encrypted data and Right: Retrived data

values with and without the use of OPAP technique. On the other side, the flexibility of the proposed algorithm which was not present in the Quad block embedding algorithm (Rajagopalan *et al*., 2012) in making tradeoff between the embedding capacity and algorithm complexity (using more number of blocks) has been also marked.

The code was actually targeted for smaller ARM devices with small RAM area in disparity to the work done by Stanescu *et al*. (2009) in a similar ARM device with external RAM interface. By utilizing the code compatibility among the various ARM versions, the code was also implemented on a board with CORTEX-A8 ARM processor. The processor is wired with a high resolution Colour Graphical GLCD, that provides 2, 30 and 400 dots with an active color matrix comprising 320 RGB columns and 240 rows. All the grey scale images were displayed on the 3.5 inch diagonally wide TFT LCD panel (GLCD) to test the intensity of scrawling on the encrypted secret image and visual imperceptibility on stego image. This was carried out only for the purpose of instant verification. In contrast, the GLCD was used as core module to provide security through user authentication by Janakiraman *et al*. (2014d). The snapshot of the embedded target board together with projected view of GLCD showing original and stego cover images of size 100×100 on the top row, along with plain, encrypted and retrieved secret data images of size 50×50 in bottom row is provided in Fig. 7.

CONCLUSION

The results obtained for the implementations shows the possibility of embedding encrypted data on grey scale images without the exchange of a separate key prior to the exchange of stego cover.

The image quality on stego image can be boosted to a considerable extent using OPAP method when K>1. Dividing the cover image into any number of blocks does not have any impact on the footprint or code size of the program. The implementation concentrated only on the use of on-chip memory resources for the purpose of implementing steganogarphy on grey scale images. This will result in the significant reduction of the memory accessing time when compared to external memory accesses and also helps in minimizing the cost and space of embedded hardware. On comparing the results, this study recommands the use of IAR C compiler (IAREW version 6.5) for better optimization in terms of both space and time. On the other hand, KEIL MDK version 4.7 provides more debugging mechanisms like Execution profiler, Performance analyzer, etc. that facilitates better result analysis. This paper shows a practical method to provide a low cost dual level of security through a combination of crypto and stego algorithm on data communication network using ethernet or CAN module built in the LPC2378 embedded device with ARM7 core, without any demand on additional resources. Based on the demand on speed by the application, the LPC2378 can run upto the maximum frequency of 72 MHz. Finally, the error metrics to compare the quality of stego image with cover image and performance metrics with respect to execution time are found to be in pahse with the ranges suggested in literatures.

ACKNOWLEDGMENTS

Authors wish to thank SASTRA University for providing infra structural support through Research and Modernization fund Reference No. R and M/0026/SEEE-010/2012-13.

REFERENCES

Abomhara, M., O.O. Khalifa, O. Zakaria, A.A. Zaidan, B.B. Zaidan and H.O. Alanazi, 2010. Suitability of using symmetric key to secure multimedia data: An overview. J. Applied Sci., 10: 1656-1661.

Amirtharajan, R. and J.B.B. Rayappan, 2013. Steganography-time to time: A review. Res. J. Inform. Technol., 5: 53-66.

Amirtharajan, R., P. Archana and J.B.B. Rayappan, 2013. Why image encryption for better steganography. Res. J. Inform. Technol., 5: 341-351.

Chan, C.K. and L.M. Cheng, 2004. Hiding data in images by simple LSB substitution. Pattern Recognit., 37: 469-474.

Cheddad, A., J. Condell, K. Curran and P. Mc Kevitt, 2010. Digital image steganography: Survey and analysis of current methods. Signal Process., 90: 727-752.

Eisenbarth, T., S. Kumar, C. Paar, A. Poschmann and L. Uhsadel, 2007. A survey of lightweight-cryptography implementations. IEEE Design Test Comput., 24: 522-533.

Eisenbarth, T., Z. Gong, T. Guneysu, S. Heyse and S. Indesteege *et al*., 2012. Compact implementation and performance evaluation of block ciphers in ATtiny devices. Proceedings of the 5th International Conference on Cryptology, July 10-12, 2012, Africa, Ifrance, Morocco, pp: 172-187.

Hong, D., J. Sung, S. Hong, J. Lim and S. Lee *et al*., 2006. HIGHT: A new block cipher suitable for low-resource device. Proceedings of the 8th International Workshop on Cryptographic Hardware and Embedded Systems, October 10-13, 2006, Yokohama, Japan, pp: 46-59.

Janakiraman, S., J. Chakravarthy, B. Radhakrishnan, K. Thenmozhi, J.B.B. Rayappan and R. Amirtharajan, 2014a. Cover as key and key as data: An inborn stego. Inform. Technol. J., 13: 1969-1976.

Janakiraman, S., K. Thenmozhi, J.B.B. Rayappan and R. Amirtharajan, 2014b. Audio fingerprint indicator in embedded platform: A way for hardware steganography. J. Artif. Intell., 7: 82-93.

Janakiraman, S., K. Thenmozhi, J.B.B. Rayappan and R. Amirtharajan, 2014c. Compiler optimization and plain text pre-processing to hoist the height of HIGHT in AVR platform. Res. J. Inform. Technol., 6: 356-367.

Janakiraman, S., K. Thenmozhi, J.B.B. Rayappan and R. Amirtharajan, 2014d. Graphical password authentication scheme for embedded platform. J. Artif. Intell., 7: 161-171.

Janakiraman, S., K.V.S.K. Kumar, R.R.K. Reddy, A. Srinivasulu, R. Amirtharajan, K. Thenmozhi and J.B.B. Rayappan, 2014e. Humming bird with coloured wings: A feedback security approach. Inform. Technol. J., 13: 2022-2026.

Muda, Z., R. Mahmod and M.R. Sulong, 2010. Key transformation approach for rijndael security. Inform. Technol. J., 9: 290-297.

Praveenkumar, P., G. Ashwin, S.P.K. Agarwal, S.N. Bharathi, V.S. Venkatachalam, K. Thenmozhi and R. Amirtharajan, 2014a. Rubik's cube blend with logistic map on RGB: A way for image encryption. Res. J. Inform. Technol., 6: 207-215.

Praveenkumar, P., K. Thenmozhi, J.B.B. Rayappan and R. Amirtharajan, 2014b. Why information security demands transform domain, compression and encryption? J. Artif. Intell., 7: 136-144.

Praveenkumar, P., R. Amirtharajan, K. Thenmozhi and J.B.B. Rayappan, 2014c. Double layer encoded encrypted data on multicarrier channel. J. Applied Sci., 14: 1689-1700.

Praveenkumar, P., R. Hemalatha, R. Uma, K. Madhunisha, K. Thenmozhi and R. Amirtharajan, 2014d. Image *Zoning→* encryption. Res. J. Inform. Technol., 6: 368-378.

Praveenkumar, P., R. Amirtharajan, K. Thenmozhi and J.B.B. Rayappan, 2015. Pixel scattering matrix formalism for image encryption-A key scheduled substitution and diffusion approach. AEU-Int. J. Electron. Commun., 69: 562-572.

Rajagopalan, S., S. Janakiraman, H.N. Upadhyay and K. Thenmozhi, 2012. Hide and seek in silicon: Performance analysis of Quad block Equisum Hardware Steganographic systems. Procedia Eng., 30: 806-813.

Rajagopalan, S., H.N. Upadhyay, J.B.B. Rayappan and R. Amirtharajan, 2014a. Dual cellular automata on FPGA: An image encryptors chip. Res. J. Inform. Technol., 6: 223-236.

Rajagopalan, S., H.N. Upadhyay, J.B.B. Rayappan and R. Amirtharajan, 2014b. Galois field proficient product for secure image encryption on FPGA. Res. J. Inform. Technol., 6: 308-324.

Rajagopalan, S., H.N. Upadhyay, J.B.B. Rayappan and R. Amirtharajan, 2014c. Logic elements consumption analysis of cellular automata based image encryption on FPGA. Res. J. Inform. Technol., 6: 291-307.

Salem, Y., M. Abomhara, O.O. Khalifa, A.A. Zaidan and B.B. Zaidan, 2011. A review on multimedia communications cryptography. Res. J. Inform. Technol., 3: 146-152.

Stanescu, D., V. Stangaciu, I. Ghergulescu and M. Stratulat, 2009. Steganography on embedded devices. Proceedings of the 5th International Symposium on Applied Computational Intelligence and Informatics, May 28-29, 2009, Timisoara, pp: 313-318.

Wang, L., H. Zhao and G. Bai, 2007. A cost-efficient implementation of public-key cryptography on embedded systems. Proceedings of the 2007 International Workshop on Electron Devices and Semiconductor Technology, June 3-4, 2007, Tsinghua University, pp: 194-197.

Yalla, P. and J. Kaps, 2009. Lightweight cryptography for FPGAs. Proceedings of the International Conference on ReConFigurable Computing and FPGAs, December 9-11, 2009, Quintana Roo, pp: 225-230.

Zaidan, B.B., A.A. Zaidan, A.K. Al-Frajat and H.A. Jalab, 2010. On the differences between hiding information and cryptography techniques: An overview. J. Applied Sci., 10: 1650-1655.