Chapter 02

Cryptography driven IP steganography for DSP Hardware Accelerators

Anirban Sengupta
Computer Science and Engineering
Indian Institute of Technology Indore

The chapter describes a cryptography driven IP steganography process for securing hardware accelerators. The chapter focusses on hardware accelerators that are used popularly in DSP applications for modern electronics systems/products. A detailed elaboration on the salient features of cryptography driven IP steganography process, its differences from DSP watermarking approaches, other hardware steganography approaches, details of secret steganography constraint generation process, embedding process, detection process and details on case studies have been provided. The chapter is organized as follows: Section 2.1 discusses the background of this topic; Section 2.2 presents the contemporary approaches for securing hardware accelerators. Section 2.3 presents the crypto-based steganography process for securing hardware accelerators; Section 2.4 introduces a new crypto-stego tool for securing hardware accelerators; Section 2.5 presents the case studies on DSP hardware accelerators; Section 2.6 concludes the chapter; Section 2.7 provides some exercise for the readers.

2.1. Introduction

Hardware acceleration is a mechanism of realizing computationally intensive tasks using hardware, in order to boost-up system performance and throughput. In other words, general-purpose processors such as central processing units (CPUs), and custom hardware work together to enhance overall performance and throughput of an electronic system. Some popular custom hardware used for hardware acceleration are field programmable gate array (FPGAs), application specific integrated circuits (ASICs) and graphics processing units (GPUs). The goal of making hardware and software work together is to simultaneously leverage the advantages of both. Software part of the system leads to advantages such as (i) faster system development i.e. lesser time to market (ii) less complications in updating features (iii) easiness in locating and patching bugs (iv) reduced non-recurring engineering (NRE) costs. However the software part pays off in terms of poor performance when it comes to performing highly data-intensive tasks. This performance lagging can be managed with the aid of hardware accelerators which lead to the following advantages: (i) high speed computation (ii) high parallelism (iii) less power consumption. Following are some applications and the corresponding hardware accelerators employed to enhance performance: (i) digital signal processing (DSP) applications using DSP hardware accelerators (ii) artificial intelligence (AI) applications using AI accelerators (iii) computer networking

applications using network processor and network interface controller (iv) computer graphics using GPU hardware accelerator (v) sound processing using sound card (vi) cryptography applications using crypto-processor or cryptographic accelerator and so forth. The focus of this chapter is mainly on DSP hardware accelerators.

Hardware accelerators have paramount importance for DSP or image processing applications because of following reasons: (i) computational intensiveness (i.e. a large number of operations are required to be computed in a pre-defined time constraint) (ii) vast utilization of DSP applications in low power portable devices such as mobile phones, digital camera, laptop, tablet etc. Considering the aforementioned facts, it is highly efficient to realize DSP algorithms using hardware accelerators for achieving high performance at low-power (Schneiderman, 2010; Mahdiany *et al*., 2001).

Apart from design objectives such as low latency, low power and low area, one more objective is grasping the attention of hardware accelerator integrated circuit (IC) developers. Here, we are highlighting about the objective of *'security' or 'protection' of the hardware accelerators* against hardware counterfeiting, cloning, false claim of intellectual property (IP) ownership and Trojan insertion threats. The security standpoint is highly relevant today because the entire process of IC design/development involves various multi-vendor third parties. The entire design process mainly involves following entities: IP vendors, system-on-chip (SoC) integrators, IC fabrication unit (foundry). For a system-on-chip integrator, IP vendors and fabrication unit are considered third parties. For a product integrator/designer/manufacturer, a SoC integrator is considered a third party. In conclusion, IP vendors, SoC integrators and foundries all play a role of third party somewhere in the entire process of designing of an end electronic system product. These third parties are situated globally and may have their own personal or national interests. This leads to malfunctioning or infringement of hardware designs at different phases of IC development. Therefore, third parties involved in the hardware accelerator IC development process are considered to be unreliable. Thereby untrustworthiness of offshore third parties poses security concerns for hardware accelerator designs and hence they are required to be secured, during their design process, against ownership abuse, counterfeiting, cloning and hardware Trojan threats (Sengupta, 2017; Sengupta, 2016; Castillo *et al.*, 2007; Plaza and Markov, 2015; Roy and Sengupta, 2019). If we discuss especially about DSP hardware accelerators, there design process is initiated with the high level synthesis (HLS) phase (McFarland *et al*., 1988) of IC design. This is due to the fact that the DSP algorithms are highly complex and large in size; hence it is challenging to be implemented directly at lower abstraction levels such as register transfer level (RTL) or gate level. In addition, embedding security during higher abstraction level is relatively easier and also enables exploration of low-cost security solution using design space exploration (DSE) process (Sengupta *et al*., 2010; Mishra and Sengupta, 2014).

The security of DSP hardware accelerators (Pilato *et al*., 2018) against counterfeiting, cloning and false claim of IP ownership threats can be enabled using detective control mechanisms such as hardware watermarking and hardware steganography.

Hardware watermarking (Sengupta and Bhadauria, 2016; Sengupta and Mohanty, 2019a; Sengupta and Mohanty, 2019b; Ziener and Teich, 2008; Le Gal and Bossuet, 2012; Koushanfar et al. 2005; Sengupta *et al*., 2019) inserts vendor's secret signature into the design to make the hardware accelerators authorized and authenticated. By detecting the vendor's secret signature into the design, fake hardware accelerators can be separated from authenticated ones. Let's come to the hardware steganography (Sengupta and Rathor, 2019a; Sengupta and Rathor, 2019b) which is a newer approach of securing DSP hardware accelerators than watermarking approach. This is a signature free approach of securing designs against aforementioned threats. The steganography approach uses stego-encoder mechanism to produce stego-constraints to be embedded into the design (Sengupta and Rathor, 2019a). **Hardware steganography differs from watermarking in terms of the following, in the context of HLS for DSP hardware accelerators** (Sengupta and Rathor, 2019a; Sengupta and Rathor, 2019b):

(i) *Kind of secret constraints to be embedded*: Watermarking embeds vendor's signature comprising of two or multiple variables, where each variable is encoded as security constraint to be embedded. However, hardware steganography embeds secret information in the form of stego-constraints which are mapped to the hardware security constraints using designer's specified mapping rules.

(ii) *Secret (or security) constraints generation process*: In watermarking approaches, first a desired signature is chosen which is then converted to security constraints using designer's encoding rules. In hardware steganography, stego-constraints are generated by a stego-encoder process which employs a more scientific or mathematical algorithm driven through a controlling parameter. This controlling parameter is referred as stego-key or hardware entropy depending on the type of hardware steganography to be employed.

(iii) *Controllability of amount of security employed*: In watermarking, designer has less control over the amount of secret constraints embedded because one cannot pre-determine a particular size and combination of a signature that would correspond to maximum security constraints. Sometimes a larger size signature can result into lesser security constraints embedded. This is because some security constraints corresponding to a large size signature may not be implanted because of their default existence in the design. Moreover, the vendor's signature is vulnerable to theft by an adversary. Once the vendor's signature is compromised, s/he fails to prove it and the goal of watermarking is defeated. However, steganography approach offers flexibility of controlling the amount of security employed using a controlling parameter (entropy threshold or stego-key). By increasing the value of entropy threshold, more security constraints can be embedded which leads to higher security. Similarly by increasing the stego-key size, more security against the theft of security constraints can be achieved. Even if the attacker somehow gets the access of security constraints, s/he cannot prove them without the knowledge of secret entropy threshold or secret stego-key

value. In conclusion, a steganography approaches offers more controllability over the security employed than watermarking approaches as well as stronger digital evidence.

(iv) *Controllability of design overhead incurred due to embedding secret constraints*: In watermarking approaches, it is challenging to estimate in advance the impact of vendor's signature on design overhead. Various signature combinations may pose different impact on design overhead. However for entropy based steganography approach, design cost overhead is controllable by threshold entropy value. Design overhead may increase with the increase in entropy threshold value. Further in case of stego-key based steganography, the design cost overhead can be controlled using designer's chosen size of stego-constraints. Design overhead may increase with the increase in stego-constraints size.

The above discussion also highlights the advantages of hardware steganography over watermarking approaches.

The focus of this chapter is on the discussion of a cryptography driven hardware stenography approach (Sengupta and Rathor, 2019b) to secure DSP hardware accelerators against piracy and false claim of ownership threats. This approach is capable to provide higher security than other steganography and watermarking approaches. The robustness of the approach lies in the fact that the stego-constraints generation process is highly intricate to be back engineered by an adversary. This is because various complex crypto-mechanisms are incorporated in the stego-constraints generation process. In addition, a very large size key drives the stego-constraints generation process. Therefore, it is infeasible for an adversary to regenerate/extract and prove stego-constraints during forensic detection (Sengupta and Rathor, 2019b). A discussion on contemporary steganography and watermarking approaches of securing DSP hardware accelerators is briefed in the next section.

2.2. Contemporary Approaches for Securing Hardware Accelerators

Some major contemporary approaches of securing hardware accelerators using hardware steganography and hardware watermarking are discussed in the subsection below.

1. Entropy Threshold based Hardware Steganography (Sengupta and Rathor, 2019a)

Sengupta and Rathor, 2019a proposed first hardware steganography for securing DSP hardware accelerators against counterfeiting and cloning threats. This hardware steganography enables counterfeiting and cloning detection by embedding stego-constraints in the register allocation phase of high level synthesis process. Control data flow graph (CDFG) representation (a high level representation) of DSP hardware accelerator application is fed as input to this approach. Upon embedding security constraints during HLS, a stego-embedded DSP hardware accelerator is generated as output. The main steps of the threshold entropy based steganography approach are highlighted in Fig. 1. Stego-constraints generation process leverages a colored interval

graph (CIG) representation of register allocation (to the storage variables of the design) to list out all the possible constraints to be embedded. Further, the set of final constraints to be embedded are shortlisted using a controlling parameter called entropy threshold. The secret constraints are embedded in the form of additional artificial edges in the CIG. Added edge constraints in the CIG are reflected in the scheduled and hardware allocated design in the form of enforced register allocation to the storage variables of the design. Amount of security constraints to be embedded can be controlled by the designer by appropriately choosing the desired entropy threshold value. This approach being signature free eliminates the possibility of leaking of signature to an adversary, unlike watermarking approaches. Hence this steganography approach emerged as more secure solution against the targeted hardware threats, in contrast to DSP watermarking approaches. However, this approach has some limitations such as non-involvement of stego-key in the stego-constraints generation process and embedding stego-constraints only in the single phase (i.e. register allocation phase) of HLS process. This weakens the secrecy of stego-constraints and renders the regeneration/extraction easier for an attacker.

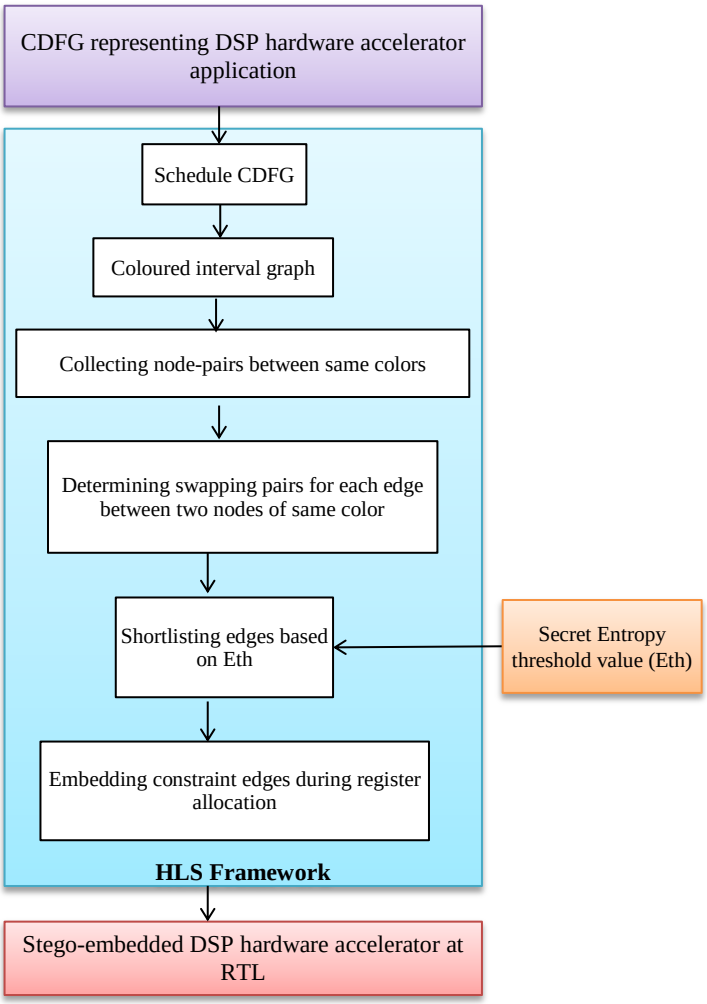

Fig. 1. *Hardware steganography approach based on entropy threshold (Sengupta and Rathor, 2019a)*

2. Cryptography driven Hardware Steganography Approach (Sengupta and Rathor, 2019b)

Accounting the limitations of entropy based hardware steganography and watermarking approaches, Sengupta and Rathor, 2019b proposed a highly robust steganography mechanism which is driven through a very large size stego-key. Moreover, stego-constraints are embedded during two different phases of HLS viz. register allocation phase and functional unit (FU) vendor

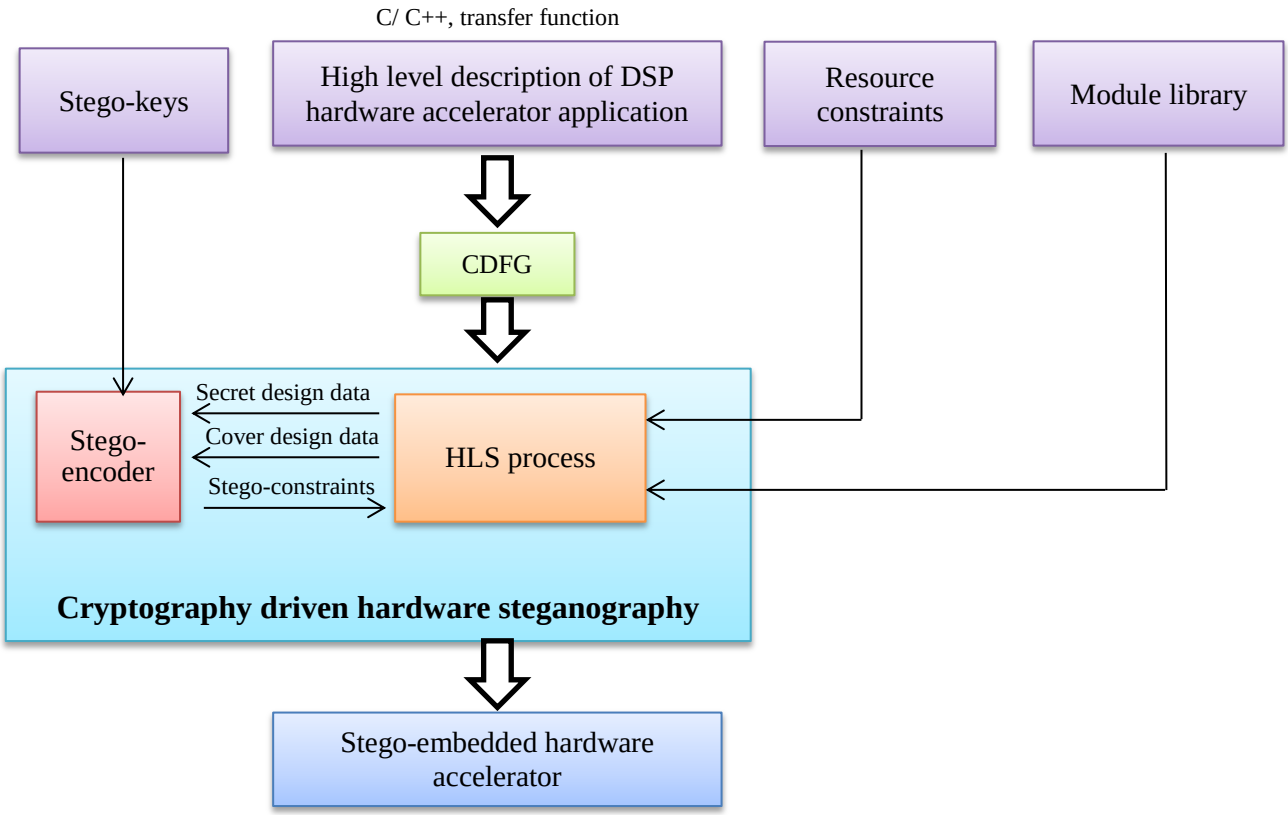

Fig. 2. *High level view of cryptography driven hardware steganography approach (Sengupta and Rathor, 2019b)*

allocation phase. This leads to embedding of higher and distributed digital evidence into the design as well as deeper embedding of stego-constraints (because of level of embedding constraints is enhanced to two phases). In addition, the stego-encoder employed to generate stego-constraints is highly complex to be back engineered or cracked by an adversary. This is because the stego-constraints generation process employs a number of cryptographic mechanisms such as **byte-substitution using S-Box**, **row diffusion, column diffusion, Trifid cipher** etc. The overview diagram capturing inputs, outputs and basic process of the cryptography driven steganography approach for hardware accelerators is shown in Fig. 2. As shown in the figure, primary inputs required to be fed to the cryptography driven hardware steganography approach are as follows:

(i) High level description of DSP hardware accelerator application. The high level description can be in the form of C/C++ code, transfer function or control data flow graph (intermediate high level representation).

(ii) Resource/ hardware constraints

(iii) Module library

(iv) Stego-keys

The stego-encoder accepts secret design data, cover design data and stego-keys as inputs to generate stego-constraints. The secret design data and cover design data are generated from an intermediate step of HLS process (the details of formation of secret design data and cover design data and generation of stego-constraints is discussed in subsequent sections of this chapter). The stego-encoder process comprises of several key-based cryptographic processes that execute in a sequence to generate stego-constraints. Thus obtained stego-constraints are in the form of a bit-stream which is truncated to the designer chosen secret size. Further, stego-constraints are embedded into the design during register allocation and FU vendor allocation phase of HLS process. Post embedding stego-constraints, the output is a stego-embedded DSP hardware accelerator design.

3. Watermarking Approaches (Sengupta and Bhadauria, 2016; Sengupta and Roy, 2017; Sengupta and Roy, 2018; Sengupta et al., 2018)

Some watermarking approaches for securing DSP hardware accelerators against piracy and false claim of ownership have been employed during higher phase of design process i.e. architectural or behavioural level. This subsection discusses two different kind of watermarking approaches employed during high level synthesis viz. single phase watermarking (Sengupta and Bhadauria, 2016; Sengupta and Roy, 2017) and multi-phase watermarking (Sengupta and Roy, 2018; Sengupta et al., 2018). Single phase watermarking approach exploits only single phase i.e. register allocation phase of HLS to embed secret watermarking constraints. In the single phase watermarking approach, vendor's signature is a combination of four variables where each variable can be utilized multiple times in order to upsurge the magnitude of digits in the signature. The vendor or designer associates an encoding rule with each signature variable in order to covert the signature into respective security constraints. To embed the signature digits as security constraints, a colored interval graph is constructed which represents allocation of storage variables of the design to the minimum possible register. In the single phase watermarking approach, (Sengupta and Bhadauria, 2016) encoded each signature variable in such a way that each digit of the signature is embedded as an extra artificial edge in the colored interval graph. However, some edge constraints corresponding to signature digits may exist by default in the CIG. This causes diminution in number of effective constraints, corresponding to a vendor's signature, embedded into the design.

Multi-phase watermarking approach exploits three divergent phases viz. scheduling phase, FU vendor allocation phase and register allocation phase of HLS to embed secret watermarking constraints. Here, vendor's signature is a combination of seven variables where one variable is encoded to embed watermarking constraints in the scheduling phase, two variables are encoded to embed watermarking constraints in the FU (e.g. adders, multipliers etc.) vendor allocation phase and remaining four variables are

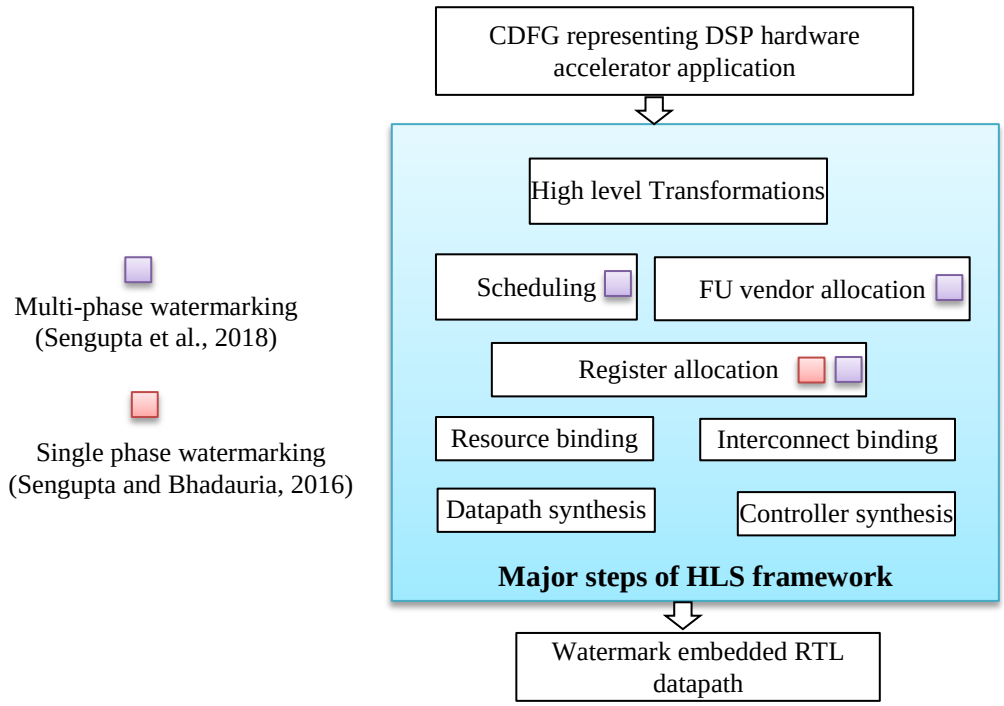

Fig. 3. *Basic difference between single phase and multi-phase watermarking during HLS*

encoded to embed constraints in the register allocation phase. Embedding watermarking constraints at multiple phases of HLS process enables embedding of deeper and higher amount digital evidence into the design. This leads to stronger proof of authorship to nullify a false claim of authorship threat by an adversary or providing detective control of piracy. Fig. 3 captures a very basic difference of multi-phase and single phase watermarking, highlighting the broader coverage of HLS design phases for embedding constraints using multi-phase watermarking.

2.3. Crypto-based Steganography for Securing Hardware Accelerators (Sengupta and Rathor, 2019b)

Sengupta and Rathor, 2019b proposed a crypto-based hardware steganography approach that also leverage HLS framework to embed secret stego-constraints like related steganography approach (Sengupta and Rathor, 2019a). However, unlike related steganography approach, the generation process of stego-constraints is stego-key driven and exploits a number of security properties/mechanisms in sequence to generate secret stego-constraints. The exploited security properties/mechanisms are of two types viz. cryptographic and non-cryptographic. Following are the cryptographic and non-cryptographic security properties incorporated in the stego-constraints generation process: (a) bit-manipulation or byte substitution using cryptographic S-box (b) cryptographic row diffusion (c) multi-layered Trifid cipher based encryption (d) alphabet substitution (e) matrix transposition (f) cryptographic mix-column diffusion (g) byte concatenation (h) bit-stream truncation and (i) bit-mapping. The stego-encoder system of crypto-based hardware steganography approach performs aforementioned security algorithms to generate stego-

constraints. It takes secret design data and stego-keys as inputs to generate stego-constraints. The stego-key is a combination of five sub-keys where each sub-key controls an intermediate step of stego-constraints generation process.

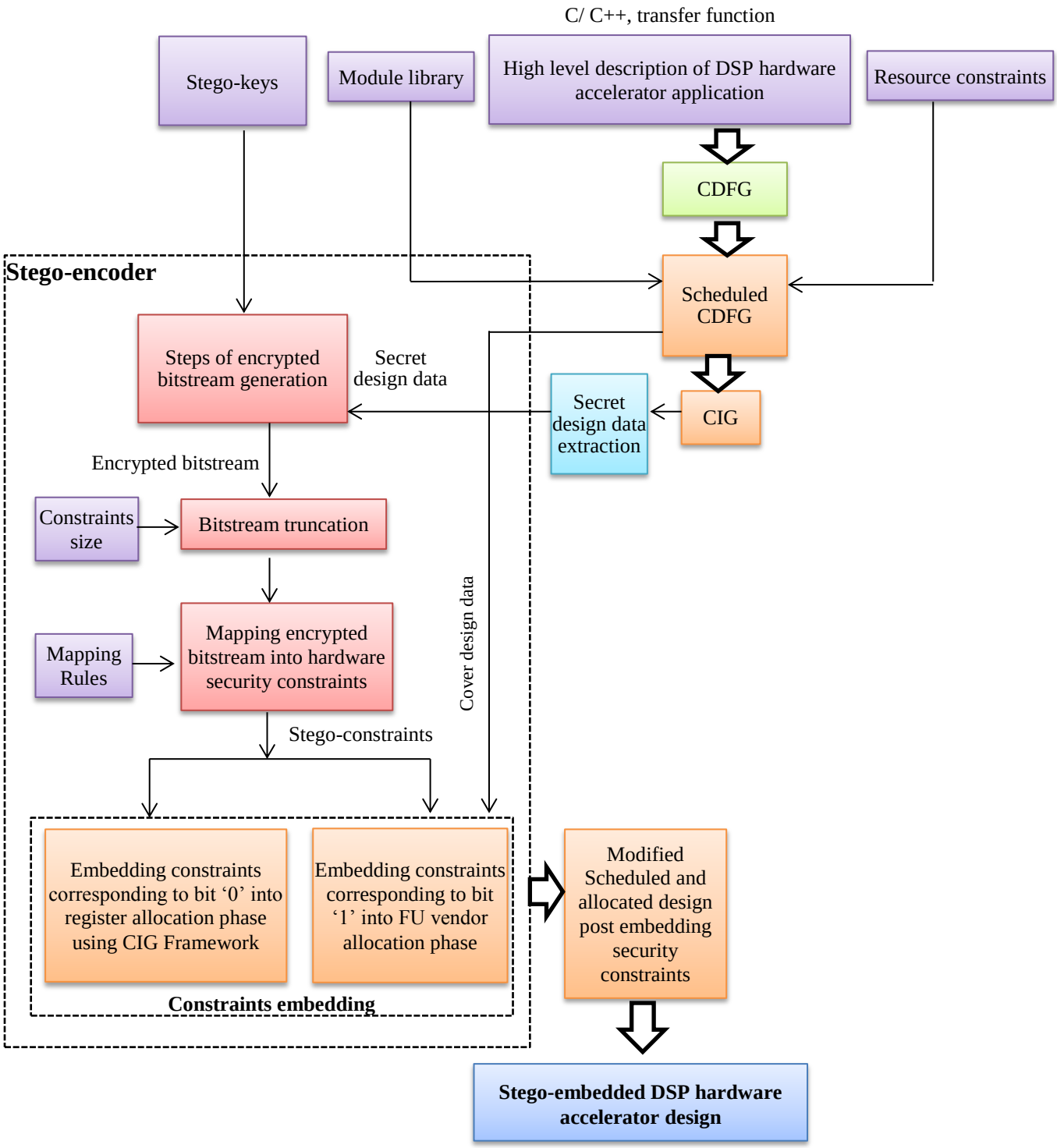

Fig. 4. *Flow of cryptography driven hardware steganography approach (Sengupta and Rathor, 2019b)*

The detailed flow of the crypto-based hardware steganography approach is shown in Fig. 4. As shown in the figure, a high level description of DSP application is first converted into an intermediate representation in the form of control data flow graph (CDFG). Further, designer's specified resource constraints and module library are used to generate a scheduled CDFG. Thereafter, a coloured interval graph is created using the information of allocation of storage variables of the design to the registers. Nodes in the CIG represent storage variables (S) of the design and the colour of a node represents its assignment to a register. Therefore, the total number of distinct colours used in the CIG is equal to the minimum number of register required to accommodate all storage variables of the design. Further, overlapping of lifetime of storage variables is indicated by drawing edges between nodes in the CIG. Thus obtained CIG is leveraged to generate secret design data which is fed to stego-encoder system. In addition, cover design data and stego-keys are also fed to the stego-encoder. The stego-encoder system uses secret design data and secret stego-keys to generate stego-constraints through following steps: (i) state matrix formation, (ii) bit manipulation, (iii) row diffusion, (iv) Trifid cipher based encryption, (v) alphabet substitution, (vi) matrix transposition, (vii) mix mix-column diffusion, (viii) byte concatenation, (ix) bit-stream truncation, (v) bit-mapping (these steps are discussed in details in the later part of this section). Here, step (i)-(viii) generate an encrypted bitstream. This encrypted bitstream is truncated based on designer's chosen secret constraints size as shown in Fig. 4. Further, each bit in the bitstream is converted to the hardware security constraints (or stego-constraints) by using designer's specified mapping rules for bit '0' and bit '1'. The mapping rules (Sengupta and Rathor, 2019b) are shown in Fig. 5. Thus generated secret stego-constraints are finally embedded into the cover design data during HLS process. This modifies the scheduled and hardware allocated design. Thus incurred modifications reflect the stego-constraints embedded into the design. Thus a steganography embedded hardware accelerator design is generated.

The brief description of the three inputs to the stego-encoder is as follows (Sengupta and Rathor, 2019b):

(a) *Secret design data*: the secret design data is obtained from CIG and used to generate stego-constraints. To obtain secret design data from the CIG, all possible pairs of the nodes of same colours are extracted. The set or collection of indices (i,j) of all node-pair (S_i,S_j) of same colours in the CIG represents the secret design data. The secret design data depends on the DSP application and register allocation scheme.

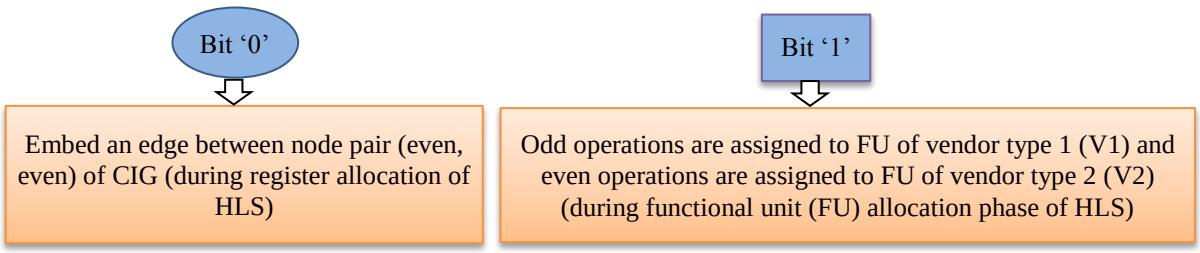

Fig. 5. *Mapping of bits of encrypted bitstream into stego-constraints (Sengupta and Rathor, 2019b)*

(b) *Cover design data*: the generated stego-constraints are embedded into the cover design data. The scheduled and allocated CDFG is exploited as cover design data to embed stego-constraints. Stego-constraints are embedded into the scheduled and allocated CDFG by performing register re-allocation and FU vendor re-allocation in HLS. Thereby, two different phases of HLS are utilized to embed stego-constraints into the cover data. Hence, this approach (Sengupta and Rathor, 2019b) is referred as **dual-phase crypto-based steganography**.

(c) *Stego-key*: The stego-key used in cryptography based steganography is a combination of five sub-keys viz. stego-key1,

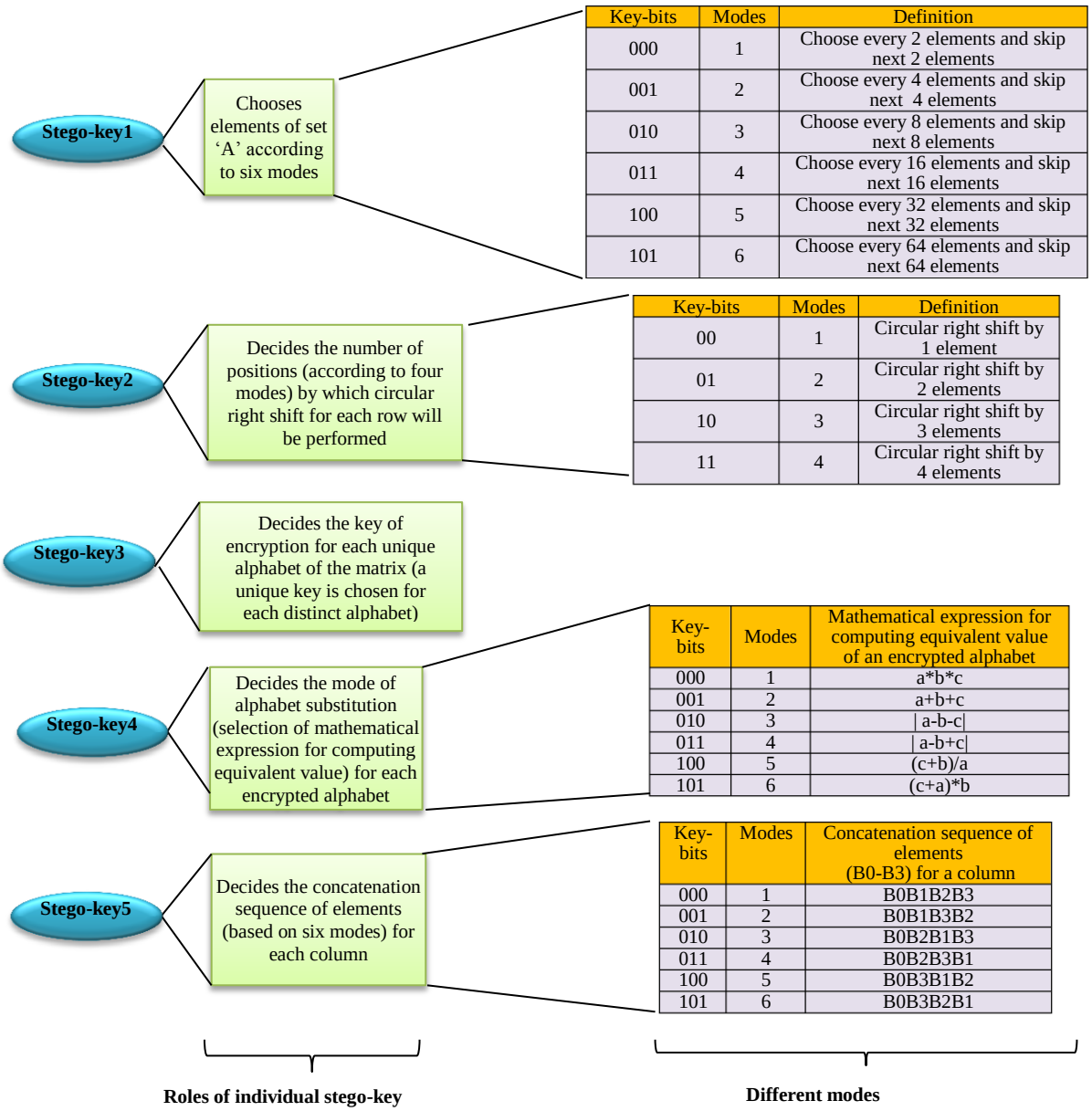

Fig. 6. *Roles of five stego-keys and definition of their different modes*

stego-key2, stego-key3, stego-key4 and stego-key5. The stego-key1 to stego-key5 control the following steps of stego-constraints generation process respectively: **state matrix formation, row diffusion, Trifid cipher based encryption,**

alphabet substitution and byte concatenation. The role of each sub-key and its different mode of usage are highlighted in Fig. 6. The stego-key regulates the amount of stego-information embedded into the design and the security employed. Larger the size of stego-key, higher (stronger) is the security of generated stego-constraints. This is because as the key

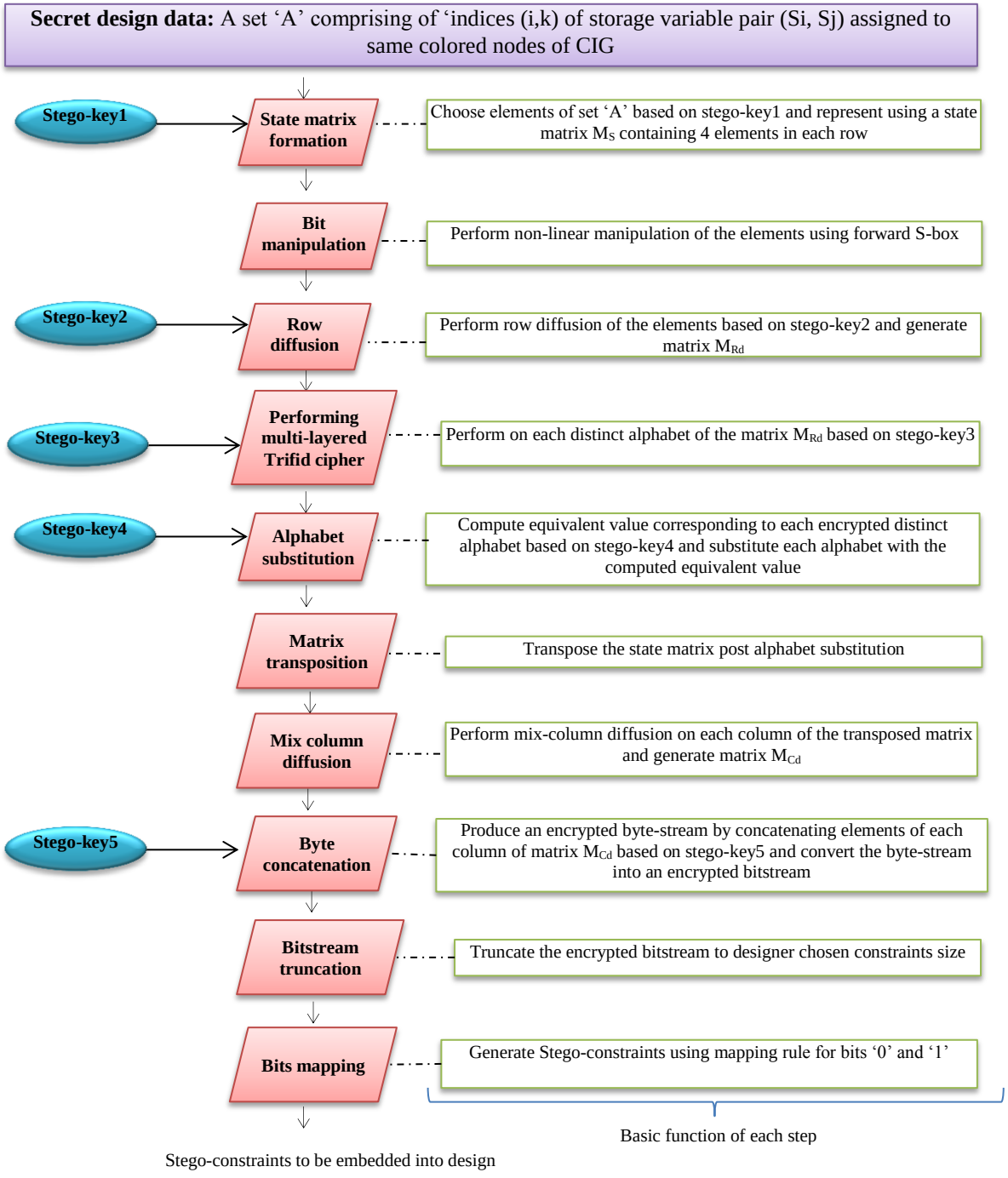

Fig. 7. *Steps of generating stego-constraints through stego-encoder in the crypto-based dual-phase steganography*

size increases, the difficulty level for an attacker in extracting/ regenerating secret stego-constraints escalates.

1. Process of Designing Stego-Embedded Hardware Accelerator for DCT Core

So far, we have discussed the basic flow of crypto-based steganography approach, where we noted that the stego-encoder generates stego-constraints using secret design data and stego-keys. The generated stego-constraints are embedded into the scheduled and allocated CDFG which acts as cover design data. Further, we have discussed the basic definition of secret design data, cover design data and roles/different modes of stego-keys. However, the different steps used in stego-encoder to generate the secret stego-constraints are yet to be discussed in details. All the steps of secret stego-constraints generation and the basic function of each step are highlighted in Fig. 7. Their details have been discussed in this sub-section.

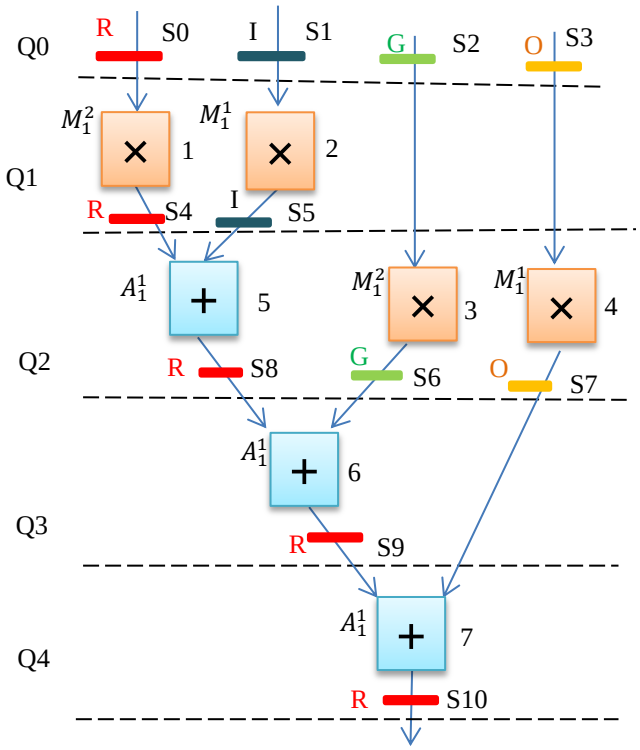

Fig. 8. *Scheduled and hardware allocated 4point DCT using resource constraints of 1 (+) and 2(*) (Sengupta and Rathor, 2019b)*

Let's discuss each step of crypto-based steganography approach in more details with the aid of demonstration on 4-point discrete cosine transform (DCT) core. A DCT core is a DSP hardware accelerator used to accelerate the process of image compression. Therefore, it finds wide utility in such consumer electronics systems where image compression is required. Since the crypto-based steganography approach requires HLS framework to embed steganography information, therefore a high level description of 4-point DCT core is fed as input to the HLS process. The high level description of the 4-point DCT core is first converted into a DFG representation. Further, the DFG is scheduled using LIST scheduling based on designer's chosen resource constraints of 2

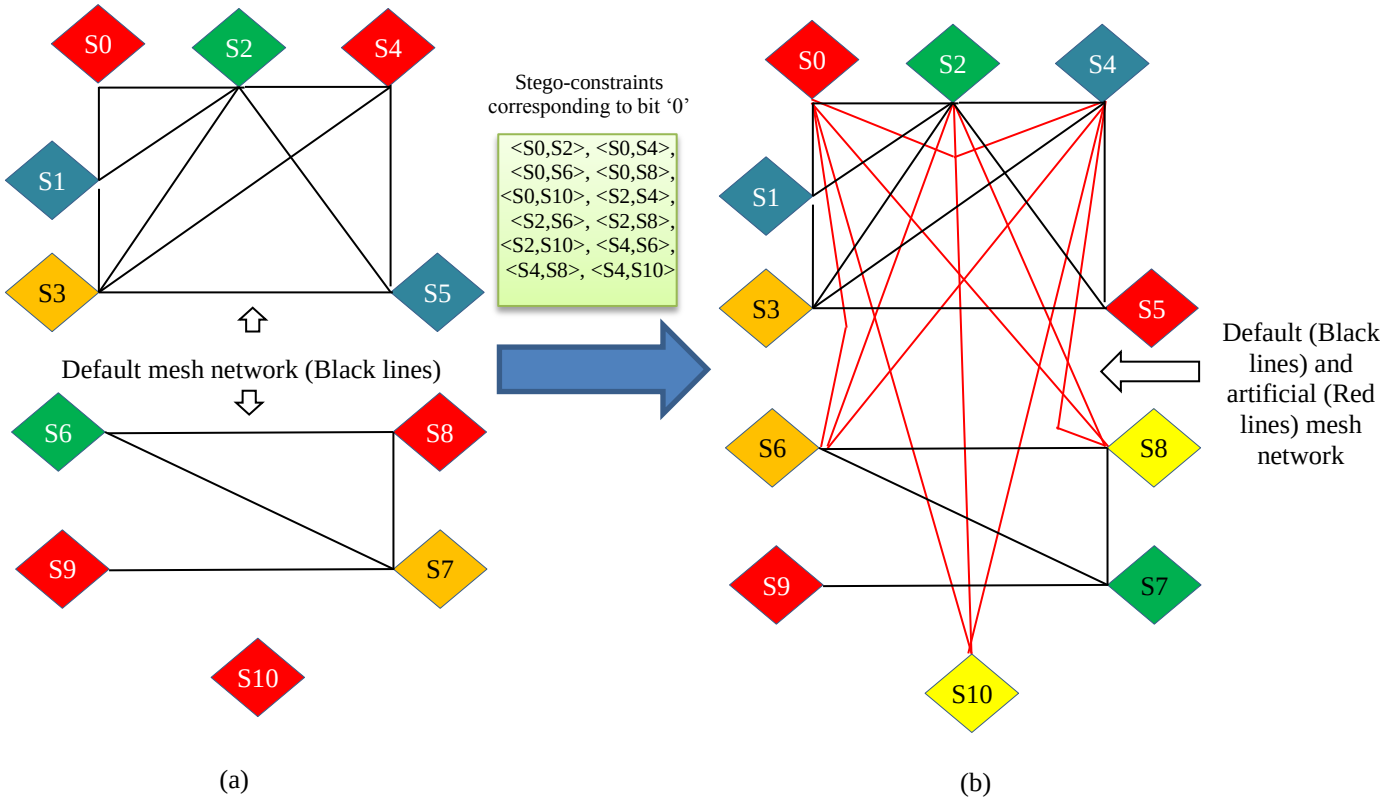

Fig. 9. *(a) CIG pre embedding stego-constraints (b) CIG post embedding stego-constraints (Sengupta and Rathor, 2019b)*

multipliers (M) and 1 adder (A). This results into a scheduled DFG as shown in Fig. 8. There are total seven operations (four multiplications and three additions) which are executing in four control steps Q1 to Q4. Two-vendor allocation scheme has been used to allocate hardware resources (functional units) to the operations. In two-vendor allocation scheme, two or more operations of same type in a same control step are assigned to the respective functional unit (FU) from two different vendors (V1 and V2). FU allocations to the operations using two vendors are shown in Fig.8. For a functional unit A_j^i or M_j^i, superscript 'i' indicates vendor type and subscript 'j' indicates instance number. Further in the scheduled DFG, primary/internal inputs and outputs of the

Table 1. *Register allocation of 4-point DCT before implanting stego-constraints corresponding to bit 0*

Control Steps	R	I	G	O
Q0	S0	S1	S2	S3
Q1	S4	S5	S2	S3
Q2	S8	--	S6	S7
Q3	S9	--	--	S7
Q4	S10	--	--	--

design have been assigned to eleven storage variables S0 to S10. These eleven storage variables are executing through four distinct registers which have been represented by four distinct colours viz. Red, Indigo, Green and Orange. Two or more storage variables executing in same control steps are essentially assigned to distinct registers/colours in order to avoid overlapping of lifetime of storage variables. Further, the information of register assignment of storage variables and their lifetime is extracted from scheduled DFG to create a CIG. The CIG of 4-point DCT is shown in Fig 9 (a) and the corresponding register allocation is shown in Table 1. Nodes in the CIG represent storage variables (S) of the design and the colour of a node represents its assignment to a register. And, overlapping of lifetime of storage variables is indicated by drawing edges between nodes in the CIG. Further, secret design data is extracted from the CIG. This secret design data is fed as input to stego-encoder. For the 4-point DCT core, the secret design data is represented in terms of a set 'A' as follows:

A={(0,4), (0,8), (0,9), (0,10), (4,8), (4,9), (4,10), (8,9), (8,10), (9,10), (1,5), (2,6), (3,7)}

Where, each element of the set indicates the indices of node pairs of same colours in the CIG. If any digit 'I' in the set is greater than 15, then it is reduced using following expression: **I'= I mod 15**. Thereafter, the set A is updated after applying modulo-15 reduction. This reduction is performed because each digit in the set are further represented in hexadecimal notation. Hence, the updated secret design data, post-conversion into hexadecimal notation, is given as follows:

A={(0,4), (0,8), (0,9), (0,A), (4,8), (4,9), (4,A), (8,9), (8,A), (9,A), (1,5), (2,6), (3,7)}

Once the secret design data is obtained, it is exploited by the stego-encoder system to generate stego-constraints using a number of cryptographic and non-cryptographic steps executing in sequence, as shown in Fig.7. The demonstrations of the different steps for generating stego-constraints are elaborated as follows:

(i) State-matrix formation

In this step, a state matrix is formed using secret design data in the set A. The formation of state matrix is driven through secret stego-key1. The stego-key1 decides the different mode of choosing elements for state-matrix formation. There are total six designer's specified modes of state matrix formation as shown in Fig. 6, therefore the size of stego-key1 is of three ($\lceil(\log_2(6)\rceil$) bits. To form the state matrix, a particular mode of state-matrix formation is chosen based on the stego-key value. Let's assume the chosen stego-key1 value is "001", hence mode 2 is selected for state-matrix formation. The mode 2 states that every 4 elements in the set should be chosen and next 4 elements should be skipped to form the entire state matrix (based on the mode definition given in Fig. 6). Therefore, there will be four elements in each row of the state matrix. The state matrix M_S based on chosen value of stego-key1 (i.e. "001") is given below:

$$M_S = \begin{bmatrix} 04 & 08 & 09 & 0A \\ 8A & 9A & 15 & 26 \end{bmatrix} \quad (1)$$

As shown in the state matrix M_S, first four elements of set A are chosen to form the first row. Next four elements of set A are skipped. Then subsequent four elements are chosen to form the second row. In case during the formation of last row, if a complete quartet is unavailable (remaining elements are less than four) then the row is not formed.

(ii) Bit manipulation or byte substitution using S-box

Once the state matrix is obtained, non-linear bit manipulation is performed in the matrix elements. To do so, each byte is substituted using forward S-box of AES. The matrix after bit manipulation (M_B) is given below:

$$M_B = \begin{bmatrix} F2 & 30 & 01 & 67 \\ 7E & B8 & 59 & F7 \end{bmatrix} \quad (2)$$

Security property: The objective of performing bit manipulation is to employ nonlinearity in the data using Shannon's property of confusion. This security property incorporates obscurity in the relationship between the input and the final output (i.e. stego constraints generated).

(iii) Row diffusion

This step incorporates row diffusion in the matrix using secret stego-key2. The value of stego-key2 controls the amount of diffusion. This is because the key value decides the mode of row diffusion to be applied on each row. A mode decides the number of positions by which circular right shift in each row will be executed. The stego-key2 is a multiple of 2 wherein each pair of two bits decides the mode of row diffusion, for each row of the matrix. Therefore the size of stego-key2 is equal to 2*(number of rows in the matrix) bits. For each pair of consecutive bits, there are four modes of row diffusion as shown in Fig. 6. Based on the designer's secret key value, the corresponding mode is applied to each row to perform row diffusion in the matrix.

In this demonstration, the matrix has two rows therefore the size of stego-key2 is of 2*2=4 bits. Let's assume the chosen stego-key2 value is "01-00", hence mode 2 is selected for first row and mode 1 for second row (first two bits decides the mode for first row and next two bits decides the mode for the second row). Based on the definition of the modes given in Fig. 6, row diffusion is performed in the matrix. The matrix post row diffusion (M_{Rd}) is given below:

$$M_{Rd} = \begin{bmatrix} 01 & 67 & F2 & 30 \\ F7 & 7E & B8 & 59 \end{bmatrix} \quad (3)$$

Security property: The aim of performing row diffusion in the matrix is to incorporate obscurity in the relationship between input secret design data and the final output (i.e. stego constraints generated). This security property is also known as Shannon's property of diffusion.

(iv) Multi-layer Trifid cipher based encryption

The Trifid cipher based encryption is performed on each distinct alphabet of the matrix, hence referred to as 'multi-layer Trifid cipher'. This step is driven through the stego-key3. Each distinct alphabet of the matrix is encrypted by an encryption key of 27 characters long. Since 27 characters can have total 27! permutations, therefore the size of encryption key required to encipher each distinct alphabet is $\lceil(\log_2(27!)\rceil$ bits. Since a unique key is used for each distinct alphabet, therefore the total size of stego-key3 to encrypt all distinct alphabets is equal to (# of distinct alphabets in the matrix)* $\lceil(\log_2(27!)\rceil$.

Let's apply Trifid cipher based encryption on the matrix shown in eq. (3). There are total three distinct alphabets in the matrix viz. B, E and F. Each distinct alphabet has to be encrypted using an encryption key of 27 characters long. A distinct key is chosen for encrypting B, E and F respectively. To do the encryption of an alphabet, 27 characters of the chosen key are arranged in three 3x3 matrices. The alphabet to be encrypted belongs to one of the square matrix. The encrypted output for the alphabet is a three digit value "abc" where, 'a' indicates the row number, 'b' indicates the column number and 'c' indicates the square matrix number. Let's process the Trifid cipher based encryption for each alphabet one by one:

(a) *Trifid cipher based encryption on alphabet 'B'*: First of all, a 27 characters long encryption key is chosen to encrypt alphabet 'B'. Let's assume the encryption key is "Q A W S E D R F T G Y H U J I K # O L P Z M X N C B V".

The key is arranged in three 3x3 matrices as follows (where, SQ indicates the square matrix):

$$SQ1 = \begin{bmatrix} Q & A & W \\ S & E & D \\ R & F & T \end{bmatrix} \quad SQ2 = \begin{bmatrix} G & Y & H \\ U & J & I \\ K & \# & O \end{bmatrix} \quad SQ3 = \begin{bmatrix} L & P & Z \\ M & X & N \\ C & \mathbf{B} & V \end{bmatrix}$$

As shown above, the alphabet B to be encrypted (highlighted in red) belongs to 3rd row and 2nd column of the 3rd square matrix (SQ3). Therefore the encrypted value "abc" for 'B' is "323".

(b) *Trifid cipher based encryption on alphabet 'E'*: Let's assume the 27 characters long encryption key to encrypt alphabet 'E' is "F T G Y H U J I K O L P Z M X N C B V # Q A W S E D R".

The key is arranged in three 3x3 matrices as follows:

$$SQ1 = \begin{bmatrix} F & T & G \\ Y & H & U \\ J & I & K \end{bmatrix} \quad SQ2 = \begin{bmatrix} O & L & P \\ Z & M & X \\ N & C & B \end{bmatrix} \quad SQ3 = \begin{bmatrix} V & \# & Q \\ A & W & S \\ \mathbf{E} & D & R \end{bmatrix}$$

As shown above, the alphabet E to be encrypted (highlighted in red) belongs to 3rd row and 1st column of the 3rd square matrix (SQ3). Therefore the encrypted value for E is "313".

(c) *Trifid cipher based encryption on alphabet 'F'*: Let's assume the 27 characters long encryption key to encrypt alphabet 'F' is "L P Z M X N C B V Q A W S E D R F T G Y H U J I K # O".

The key is arranged in three 3x3 matrices as follows:

$$SQ1 = \begin{bmatrix} L & P & Z \\ M & X & N \\ C & B & V \end{bmatrix} \quad SQ2 = \begin{bmatrix} Q & A & W \\ S & E & D \\ R & \boldsymbol{F} & T \end{bmatrix} \quad SQ3 = \begin{bmatrix} G & Y & H \\ U & J & I \\ K & \# & O \end{bmatrix}$$

As shown above, the alphabet F belongs to 3rd row and 2nd column of the 2nd square matrix (SQ2). Therefore the encrypted value for F is "322".

Security property: The aim of using Trifid cipher based encryption is to incorporate following security properties: (i) confusion to obscure the relationship of the stego keys with generated stego constraints (ii) diffusion to obscure the relationship of the input secret design data with the stego constraints. The Trifid cipher offers these security properties by combining following techniques: fractionation, transposition and substitution.

(v) Alphabet substitution

Once all distinct alphabets are encrypted, their encrypted 3 digit values are converted into an equivalent value based on a mathematical expression. The mathematical expression to be used is decided by secret stego-key4. Post evaluating mathematical expression, the corresponding alphabet is substituted with the output of the expression. A number of mathematical expressions can be possible, where each distinct mathematical expression defines a mode for alphabet substitution. The mode of alphabet substitution (i.e. the mathematical expression to be used to generate the substituting value) is determined by the stego-key4. Total six kind of mathematical expressions are defined as shown in Fig. 6, therefore total $\lceil(\log_2(6)\rceil$ = 3bits are required to determine a mode for each encrypted alphabet. Thus, the total size of stego-key4 is equal to (# of distinct alphabets encrypted using Trifid cipher)* $\lceil(\log_2$(# of modes for alphabet substitution)$\rceil$.

Since total three distinct alphabets (B, E and F) have been encrypted in the previous step, therefore total size of stego-key4 is $3*\lceil(\log_2(6)\rceil$=9 bits. Let's assume the stego-key4 is "001-000-010". Each group of three bits from left to right is used to decide the mode for an alphabet in alphabetic order. Therefore, "001", "000" and "010" decide the mode for B, E and F respectively and corresponding mathematical expression (shown in Fig. 6) to be evaluated are selected. Table 2 shows the corresponding mathematical expression for each alphabet to be substituted and the corresponding output value of mathematical expression. Hence alphabets B, E and F are substituted in the matrix with 8, 9 and 1 respectively. The matrix

Table 2. *Details of obtaining equivalent value for alphabet substitution*

Alphabets	Encrypted value "abc" (output of Trifid cipher)	Corresponding key bits in stegokey4	Corresponding mathematical expression	Output of mathematical expression
B	323	001	a+b+c	8
E	313	000	a*b*c	9
F	322	010	|a-b-c|	1

post alphabet substitution (M_{AS}) is given below:

$$M_{AS} = \begin{bmatrix} 01 & 67 & 12 & 30 \\ 17 & 79 & 88 & 59 \end{bmatrix} \quad (4)$$

(vi) Matrix Transposition

The matrix obtained in previous step is transposed. The transposed matrix (M_T) is given below:

$$M_T = \begin{bmatrix} 01 & 17 \\ 67 & 79 \\ 12 & 88 \\ 30 & 59 \end{bmatrix} \quad (5)$$

(vii) Mix column diffusion

In this step, each column of the transposed matrix (M_T) is subjected to mix column diffusion by exploiting a circulant MDS (Maximum Distance Separable) matrix. Note: this matrix is also used in AES encryption for column diffusion.

(a) Mix column diffusion using MDS matrix for the 1st column:

$$\begin{bmatrix} B_0^1 \\ B_1^1 \\ B_2^1 \\ B_3^1 \end{bmatrix} = \begin{bmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{bmatrix} * \begin{bmatrix} 01 \\ 67 \\ 12 \\ 30 \end{bmatrix} = \begin{bmatrix} 89 \\ C9 \\ 12 \\ 16 \end{bmatrix} \quad (6)$$

Where, B_j^i indicates the j[th] byte of i[th] column after performing mix column operation. Equations for computing each new value of the 1st column using mix column diffusion are as follows:

$$B_0^1 = (02*01) \oplus (03*67) \oplus (01*12) \oplus (01*30) = 89$$
$$B_1^1 = (01*01) \oplus (02*67) \oplus (03*12) \oplus (01*30) = C9$$
$$B_2^1 = (01*01) \oplus (01*67) \oplus (02*12) \oplus (03*30) = 12$$
$$B_3^1 = (03*01) \oplus (01*67) \oplus (01*12) \oplus (02*30) = 16$$

(b) Mix column diffusion using MDS matrix for the 2nd column:

$$\begin{bmatrix} B_0^2 \\ B_1^2 \\ B_2^2 \\ B_3^2 \end{bmatrix} = \begin{bmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{bmatrix} * \begin{bmatrix} 17 \\ 79 \\ 88 \\ 59 \end{bmatrix} = \begin{bmatrix} 74 \\ 3F \\ 8E \\ 7A \end{bmatrix} \quad (7)$$

Equations for computing each new value of the 2nd column using mix column diffusion are as follows:

$$B_0^2 = (02*17) \oplus (03*79) \oplus (01*88) \oplus (01*59) = 74$$
$$B_1^2 = (01*17) \oplus (02*79) \oplus (03*88) \oplus (01*59) = 3F$$
$$B_2^2 = (01*17) \oplus (01*79) \oplus (02*88) \oplus (03*59) = 8E$$
$$B_3^2 = (03*17) \oplus (01*79) \oplus (01*88) \oplus (02*59) = 7A$$

The matrix after performing mix column diffusion (M_{Cd}) is given below:

$$M_{Cd} = \begin{bmatrix} 89 & 74 \\ C9 & 3F \\ 12 & 8E \\ 16 & 7A \end{bmatrix} \quad (8)$$

Security property: The mix-column step enhances the Shannon's property of diffusion by further obscuring the relationship between input secret design data and generated stego-constraints.

(viii) Byte concatenation

All elements in each column of the matrix M_{Cd} are concatenated to form a sequence of bytes. The byte concatenation step is driven through secret stego-key5. There are a number of possible ways of concatenating all byte of each column. The mode of concatenation is determined by the stego-key5. Six modes of byte concatenation for a column in a matrix are given in Fig.6. Since byte concatenation is performed for each column separately based on the selected mode, therefore the total size of stego-key5 is = (# of columns in the matrix M_{Cd}) * $\lceil \log_2$(# of modes of bytes concatenation)$\rceil$.

In this demonstration, since there are two column in the matrix M_{Cd}, therefore the size of stego-key5 is 2*$\lceil \log_2 6 \rceil$= 6 bits. Let's assume the stego-key5 is "001-000". First combination of three bits ("001") decides the mode of byte concatenation for 1st column and second combination of three bits ("000") decides the mode of byte concatenation for 2nd column. For column 1 and column 2, the selected modes (from Fig. 6) are B0B1B3B2 and B0B1B2B3 respectively. Hence, the final sequence of bytes post concatenation is as follows: $B_0^1 B_1^1 B_3^1 B_2^1 B_0^2 B_1^2 B_2^2 B_3^2$ = "89C91612743F8E7A".

Thus obtained sequence of bytes is an encrypted byte-stream which is converted into an encrypted bitstream given below:

"10001001110010010001011000010010011101000011111110001110011111010"

(ix) Bitstream truncation

The truncation of encrypted bitstream is performed based on the designer's secret size. For example, following is the truncated bitstream for the chosen stego-constraints size=20:

The truncated bitstream="10001001110010010001".

The truncated bitstream contains twelve 0s and eight 1s.

(x) Bits mapping to the stego-constraints

In order to embed the encrypted bitstream, each bit is mapped to a corresponding stego-constraint. Thus obtained stego-constraints represent hardware security constraints to be embedded into the cover design data. Mapping rules shown in Fig. 5 are used to covert bitstream into corresponding stego-constraints. Based on the mapping rules, the mapping of twelve 0s of the truncated bitstream to the stego-constraints is as follows:

<S0,S2>, <S0,S4>, <S0,S6>, <S0,S8>, <S0,S10>, <S2,S4>, <S2,S6>, <S2,S8>, <S2,S10>, <S4,S6>, <S4,S8>, <S4,S10>

Where, each stego-constraint corresponding to bit 0 has been represented in terms of a constraint (secret) edge to be added additionally into the CIG.

Further, in the mapping of 1s to the stego-constraints, each mapping corresponds to allocation of an operation to a specific FU vendor type. Therefore the maximum numbers of 1s that can be embedded are equal to the number of operations available in the DSP application. The mapping of eight 1s of the truncated bitstream to the stego-constraints is shown in Table 3. Since

Table 3. *Possible allocation of FU vendors to the operations based on mapping of 1s to the stego-constraints*

Operation number	1	2	3	4	5	6	7
Vendor type of FU	V1	V2	V1	V2	V1	V2	V1

total available operations in the 4-point DCT core are seven, therefore maximum seven 1s (out of eight) can be mapped.

So far, we have discussed different steps of stego-constraints generation through stego-encoder system. The size of different stego-keys used in the stego-constraints generation process is highlighted in Table 4. The total size of stego-key is given as follows:

The total stego-key size=[3 bits]+ [(number of row in state matrix M_S)*2]+ [(# of unique alphabets)* $\lceil(\log_2(27!)\rceil$]+ [(# of distinct alphabets encrypted using Trifid cipher)* $\lceil(\log_2$(#of modes for alphabet substitution)$\rceil$]+ [(# of columns in the transposed matrix M_{Cd})* $\lceil(\log_2$(# of modes of byte concatenation)$\rceil$] (9)

Embedding of Stego-constraints into cover design data (scheduled DFG):

Stego-constraints corresponding to bit 0 and bit 1 are embedded into scheduled DFG of DSP application during the register allocation and FU vendor allocation phase respectively. All stego-constraints corresponding to bit '0' are embedded as additional

Table 4. *Size of different stego-keys*

Stego-keys	**Size (in bits)**
Stego-key1	$\lceil\log_2$ (total modes of state matrix M_S formation)$\rceil$
Stego-key2	(Number of rows in matrix M_S)* $\lceil\log_2$ (total modes of row diffusion)$\rceil$
Stego-key3	(Number of distinct alphabets)* $\lceil\log_2$ (27!)$\rceil$
Stego-key4	(Number of distinct alphabets encrypted using Trifid cipher) *$\lceil\log_2$ (total modes of alphabet substitution)$\rceil$
Stego-key5	(Number of columns in matrix M_{Cd})*$\lceil\log_2$(total modes of byte concatenation)$\rceil$

edges into the CIG. Post embedding stego-constraints, the modified mesh network of the CIG contains both default and artificial edges. Thus modified CIG is shown in Fig. 9(b). As shown in the figure, the edges constraints <S0, S2> and <S2, S4> exist by

default in the CIG. And edge constraints <S0,S6>, <S2,S8>, <S2,S10>, <S4,S6> can be added without any conflict because of different colours of respective nodes in a node-pair. However, edges <S0,S4>, <S0,S8>, <S0,S10>, <S2,S6>, <S4,S8> and <S4,S10> cannot be directly added because of same colour of both nodes in these node-pairs. In order to add edge constraint <S0, S4> into the CIG, colour of one of the node is required to be swapped with the colour of another node in the same control step. This is because an edge cannot be added between two nodes of same colour. Therefore, colour/register of node/storage variable S4 (Red) has been swapped with the colour/register of node/storage variable S5 (Indigo). Hence, the colour of node S4 changes from red to Indigo. Now both nodes S0 and S4 have different colours, therefore an artificial edge can be added between them. Similarly edge constraint <S2,S6> is added by swapping the colour of node S6 with the node S7 in the same control step. However, edge constraints <S0,S8> and <S4,S8> cannot be added by swapping of node colour with another node in same control step. Therefore, an extra colour (register) yellow is used to accommodate storage variable S8. Now edge constraints <S0,S8> and <S4,S8> can be added into the CIG without any conflict. Further, allocation of storage variable S10 also to yellow register facilitates adding edge constraints <S0,S10> and <S4,S10> into the CIG. This discussion highlights that in some cases, extra register may be required to satisfy all edge constraints. However, this is not always true because large size designs have higher number of registers, hence may not require an extra register to accommodate all edge constraints. The register allocation post embedding stego-constraints is shown in Table 5. The storages variables subjected to re-allocation have been marked shaded in the table. Thus stego-constraints corresponding to 0s are embedded during register allocation phase of HLS.

Further, stego-constraints corresponding to bit 1 (shown in Table 3) are embedded by performing FU vendor re-allocation to the operations of the design. The FU vendor re-allocation based on mapping rule of bit 1 is shown in Table 3. As shown in the table, operation 1, 3, 5 and 7, being odd operations, should be allocated to the vendor V1 and operations 2, 4, and 6, being even operations, should be allocated to vendor V2. This re-allocation is followed for operation# 1, 2, 3, 4, 5 and 7. However, the

Table 5. *Register allocation of 4-point DCT post implanting stego-constraints corresponding to bit 0*

Control Steps	R	I	G	O	Y
Q0	S0	S1	S2	S3	--
Q1	S5	S4	S2	S3	--
Q2	--	--	S7	S6	S8
Q3	S9	--	S7	--	--
Q4	--	--	--	--	S10

operation # 6 is still allocated to vendor V1 (instead of V2). This is because constraint for adder is chosen to be 1 (as mentioned earlier). Since the operation # 6 is an addition operation, therefore it is essentially allocated to the only available adder of vendor V1 (i.e. A_1^1). Further it is worth noting that there are total eight 1s to be embedded during FU vendor re-allocation, however only seven are embedded as total available operations are seven. Post embedding stego-constraints corresponding to bit 1, the final

Table 6. *Final allocation of FU vendors to the operations post embedding 1s as the stego-constraints*

Operation number	1	2	3	4	5	6	7
Vendor type of FU	V1	V2	V1	V2	V1	V1	V1

allocation of FU vendors to the operations is shown in Table 6.

Thus stego-constraints corresponding to 0s and 1s are embedded into the scheduled DFG during two different phases of HLS. The modified scheduled and allocated DFG of DCT core is shown in Fig. 10.

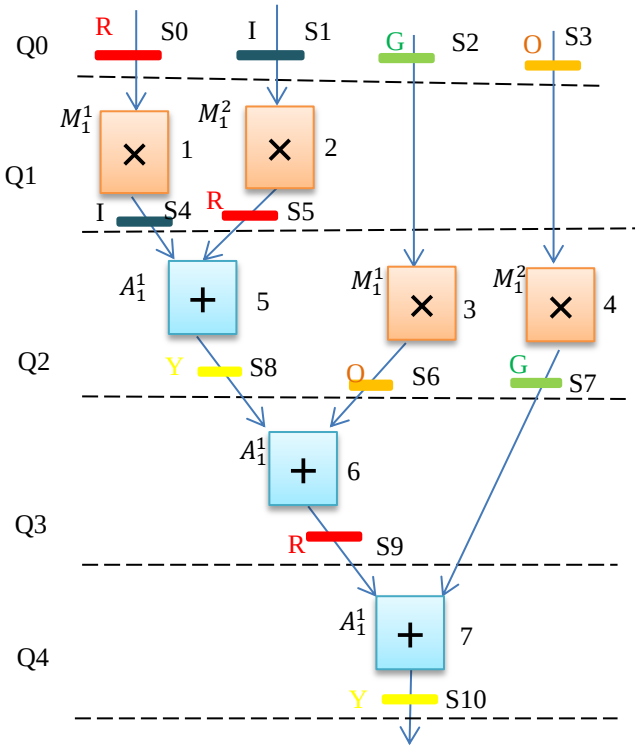

Fig. 10. *Scheduled and hardware allocated 4point DCT using resource constraints of 1 (+) and 2(*) (Sengupta and Rathor, 2019b)*

2. Detection of Steganography

Detection of embedded steganography information is very crucial in order to validate the authenticity of hardware accelerator designs. Detection of steganography disables the wrong intents of an adversay of claiming authorship fradulently or counterfeiting

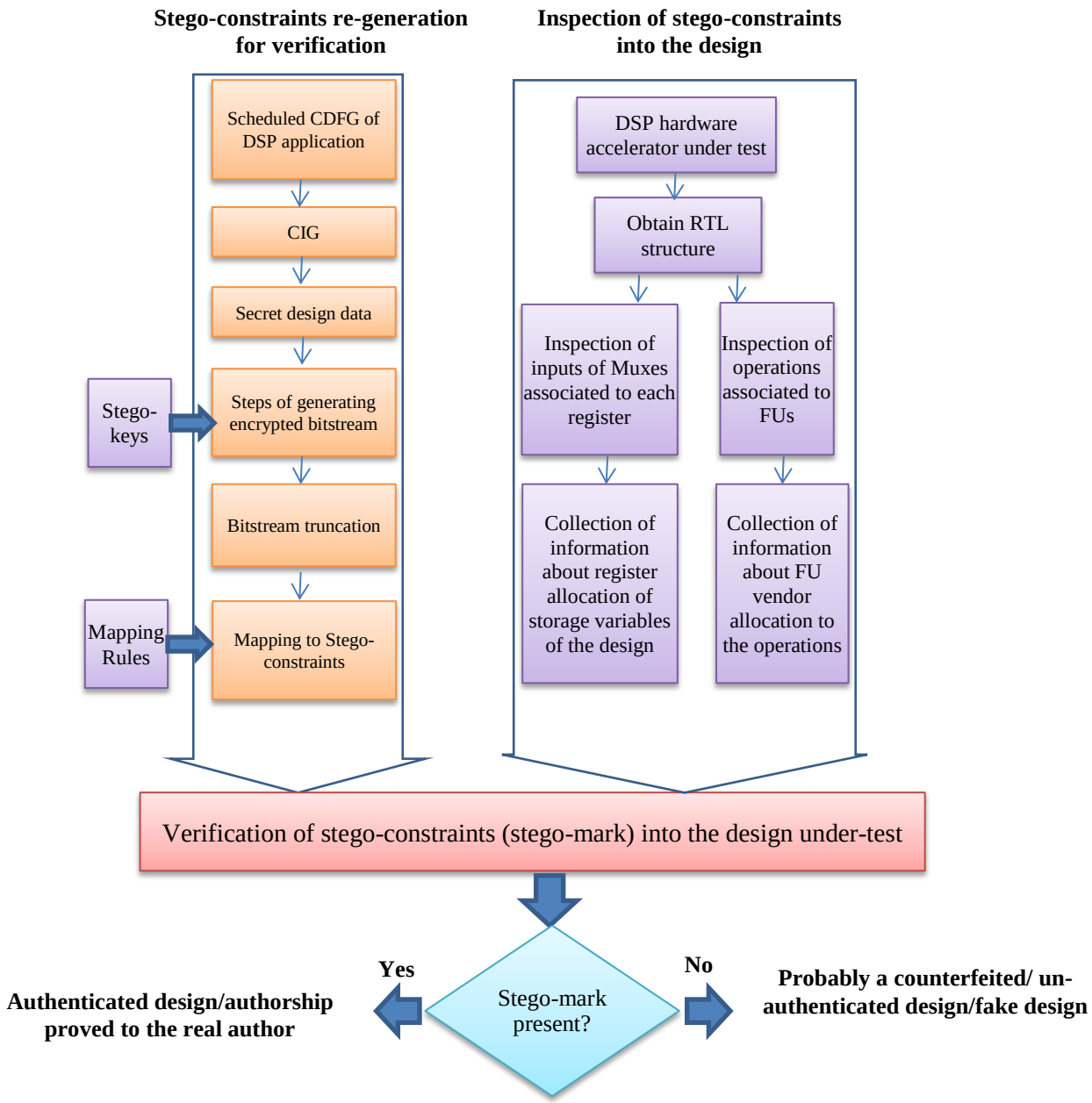

Fig. 11. *Hardware steganography detection in crypto-based steganography approach (Sengupta and Rathor, 2019b)*

designs to earn illegal income. The detection process of embedded stego-constraints which are generated using crypto-based steganography approach is shown in Fig. 11. The detection process is perfomed in three major steps as follows:

(a) **Stego-constraints re-generation for verification**: In order to verify the presence of stego-constraints (or stego-mark) embedded into the design, they are required to be regenerated. Regeneration of stego-constraints by the IP/IC owner is required so that s/he can prove his rights over the constraints scientifically. This also disables an adversary to claim stego-constraints as his/her own after pirating/copying them. Only the genuine designer/owner is capable to regenerate

the stego-constraints through a scientific algorithm and a secret key. An adversary cannot inadvertently regenerate stego-constraints without the knowledge of the stego-constraints generation algorithm and the secret keys employed.

(b) **Inspection of stego-constraints into the design**: the design under-test is subjected to inspection of embedded stego-constraints. To do so, its RTL structure is analysed. To get information about the stego-constraints corresponding to bit 0, the inputs of Muxes associated to each registers are analysed. This helps in finding the association of storage variables of the design to the registers. Further to get information about the stego-constraints corresponding to bit 1, all operations of the design associated to FUs are analysed. And, information about allocation of type of FU vendors to the operations is collected.

(c) **Verification of the stego-constraints into the design**: the presence of regenerated stego-constraints is verified with the information of register allocation and FU vendor allocation extracted from the design under-test. If stego-constraints corresponding to both 0s and 1s are present in to the design, then the design contains the author's stego-mark. Presence of author's stego-mark into the design ascertains the authenticity of the hardware accelerator. Moreover, the presence of author's stego-mark proves the authorship of the author over the design and nullifies the false claim of authorship by an adversary. If the hardware accelerator design does not contain a stego-mark (an authentic mark), then it can probably a counterfeited, hence can be separated out from the authentic ones.

2.4. Crypto-Stego Tool for Securing Hardware Accelerators

The author and his team have developed a ***Crypto-Stego tool*** to simulate and analyse the functionality of crypto-based steganography approach for DSP hardware accelerators. This tool provides a friendly graphical interface to users. A snapshot of the graphical user interface (GUI) of the tool is shown in Fig. 12. The left portion of the tool shows the panel for providing required inputs to the tool, right portion shows the panel with output buttons to see the intermediate and final outputs of the crypto-based steganography approach. The panel in the middle shows the status of the key-driven steps (i.e. state matrix formation, row diffusion, Trifid cipher, alphabet substitution and byte concatenation) of the crypto-based steganography approach. Initially, these status bars remain **Red**. Upon applying the stego-key, the respective status bar turns **Blue**. The *Crypto-Stego* tool accepts the DSP application input in the form CDFG along with module library and resource constraints. The tool shows all the intermediate steps of crypto-based steganography and the finally generated stego-constraints at the output. Further, it also shows scheduling and registers allocation pre and post embedding steganography constraints, onto the output window.

Let's generate all the intermediate and final output of crypto-based steganography approach for 4-point DCT core using the *crypto-stego* tool. We will provide the same inputs and stego-keys used during the demonstration of 4-point DCT core discussed

Fig. 12. *A snapshot of the GUI of crypto-stego tool for DSP hardware accelerators*

in section 2.3. Here, we can match the output generated with the tool and with that obtained in the demonstration. First of all, input DFG of 4-point DCT core, resource constraints of 1 adder and 2 multipliers and module library are fed to the tool as shown in Fig. 13. On clicking on the button "Reg. Allocation" on output panel, the register allocation table (pre embedding stego-constraints) becomes available on to the output window. Here, values under the column heading 1, 2, 3 and 4 show the storage variable (S) number and the heading of the column show the register number, where Red, Indigo, Green and Orange register have

Fig. 13. *Snapshot of the tool after feeding DFG of 4-point DCT , resource constraints and stego-key1; the output window is shown in the lower portion*

been denoted by the number 1, 2, 3 and 4 respectively. The rows heading (0, 1, 2, 3 and 4) show the control step number. This register allocation matches with that demonstrated in section 3. Further, upon clicking on the output button “secret design data”, the secret design data (same as obtained in demonstration) becomes available onto the output window as shown in Fig. 13. As again shown in Fig. 13, the stego-key1=”001” (same as used in demonstration) has been fed. Upon feeding stego-key1, the respective status bar turns blue because stego-key1 is used for state matrix formation. Whereas, the remaining status bar are still

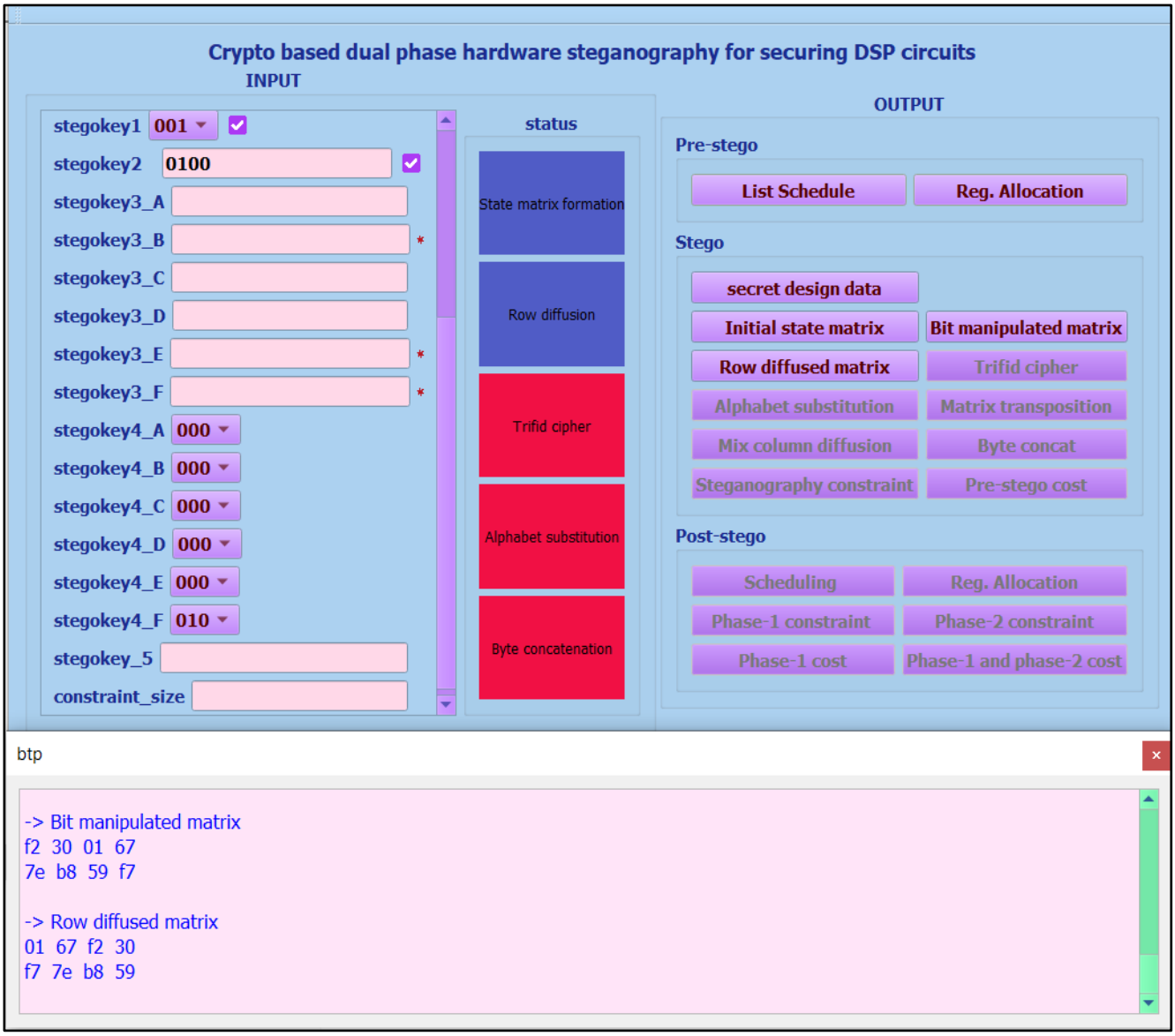

Fig. 14. *Snapshot of the tool post feeding stego-key1 and stego-key2*

Red as shown in Fig. 13. The output of state matrix formation is shown on to the output window upon clicking on the output button “initial state matrix”. Further, Fig. 14 shows the output of bit-manipulation and row diffusion steps after feeding the stego-key2. Fig. 15 shows that encryption key (stego-key3) for only alphabets B, E and F have been fed as they are the only available alphabets in the matrix post row diffusion. And the stego-key4 is fed for alphabet substitution. Therefore the corresponding status bars (Trifid cipher and alphabet substitution) turn blue. The corresponding outputs are shown on to the output window. The Fig. 15 also shows the transposed matrix. Further, Fig. 16 shows that the stego-key5 has been fed to the tool for byte concatenation and the concatenated byte-stream is shown onto the output window. Output of the step before byte concatenation (i.e. mix column

Fig. 15. *Snapshot of the tool post feeding stego-key1, stego-key2, stego-key3 and stego-key4*

diffusion) is also shown in Fig. 16. Further, Fig. 17 shows that the constraints size=20 has been fed as input and the final truncated bitstream is made available on to the output window by clicking on the button "steganography constraint". As shown in all the snapshots of the tool, same inputs and stego-keys that are used in the demonstration on 4-point DCT in section 2.3 have been fed here. The tool produces the desired outputs that match with the demonstration on 4-point DCT in section 2.3. Further, Fig. 17 also shows the register allocation and FU vendor allocation in the scheduling table post-embedding stego-constraints. One difference in the register allocation of the demonstration (in section 2.3) and that generated using tool is to be noted; which is the re-allocation of storage variable S10 to a different register. There are two possible ways to re-allocate storage variable S10 in

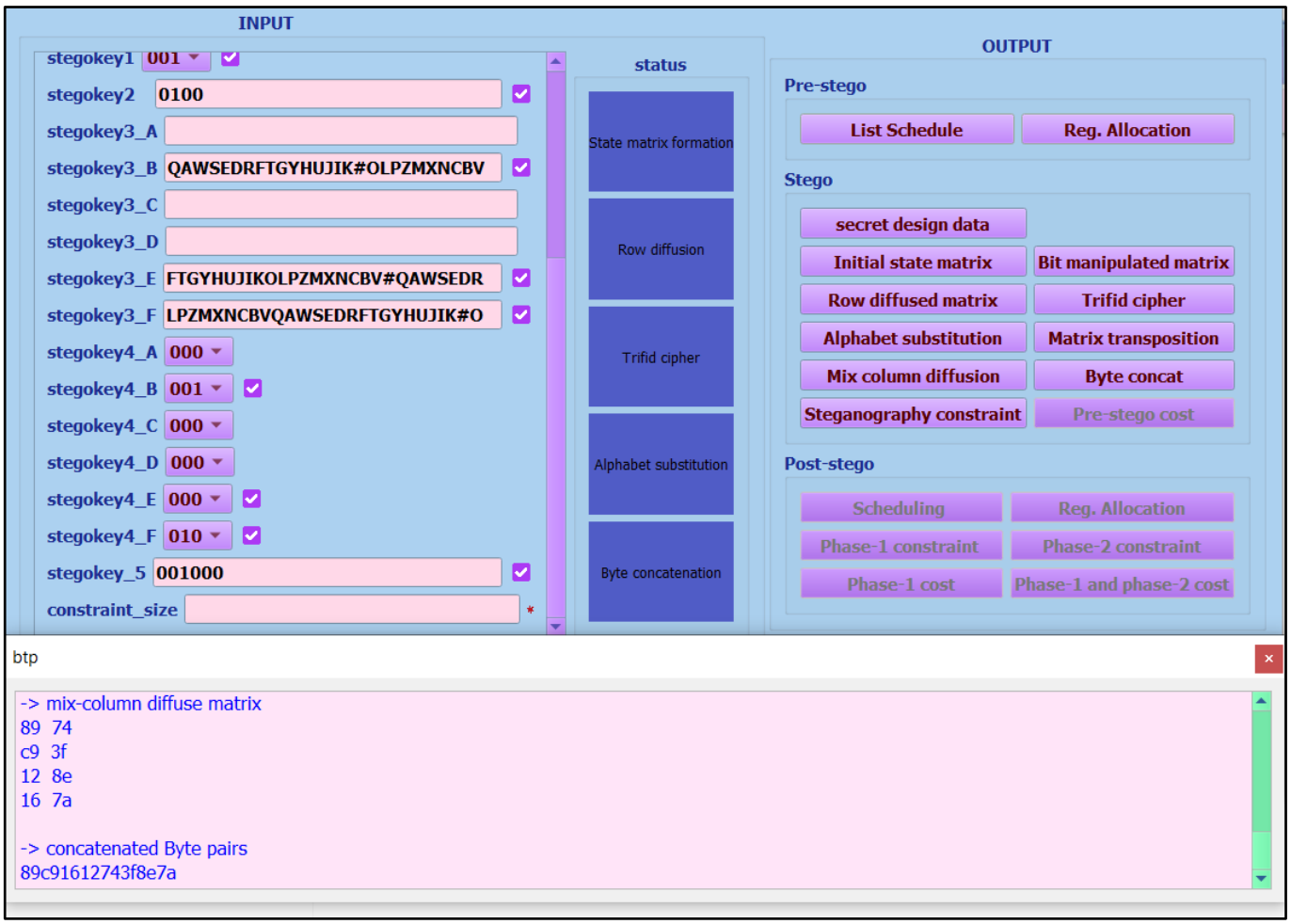

Fig. 16. *Snapshot of the tool post feeding all five stego-keys*

order to involve adding of edges <S0, S10> and <S4, S10> in the CIG. It can either be allocated to the Orange register or the Yellow register in the same control step. In the demonstration, the possibility of allocating S10 to the Yellow register has been exploited; whereas the register allocation scheme implemented in the tool exploits the possibility of allocating S10 to the Orange register (i.e. the register/colour number 4), as shown in the output window of Fig. 17. Further, the FU vendor re-allocation post embedding 1s is shown onto the output window by clicking on the button “scheduling” under the post-stego section of the output panel. As shown in the ‘post-stego operation scheduling’ table in the output window, the parameter written at the left to the operator shows the operation number and the parameter written at the right to the operator show the FU vendor type. Thus the crypto-based steganography approach can be simulated and analysed using the crypto-stego tool developed by the authors. This tool is useful for case studies of various kind of DSP hardware accelerator applications such as finite impulse response (FIR) filter, infinite impulse response (IIR) filter, discrete wavelet transform (DWT), auto-regression filter (ARF) etc. In addition, the tool evaluates and shows the design cost pre and post-embedding steganography information into the design.

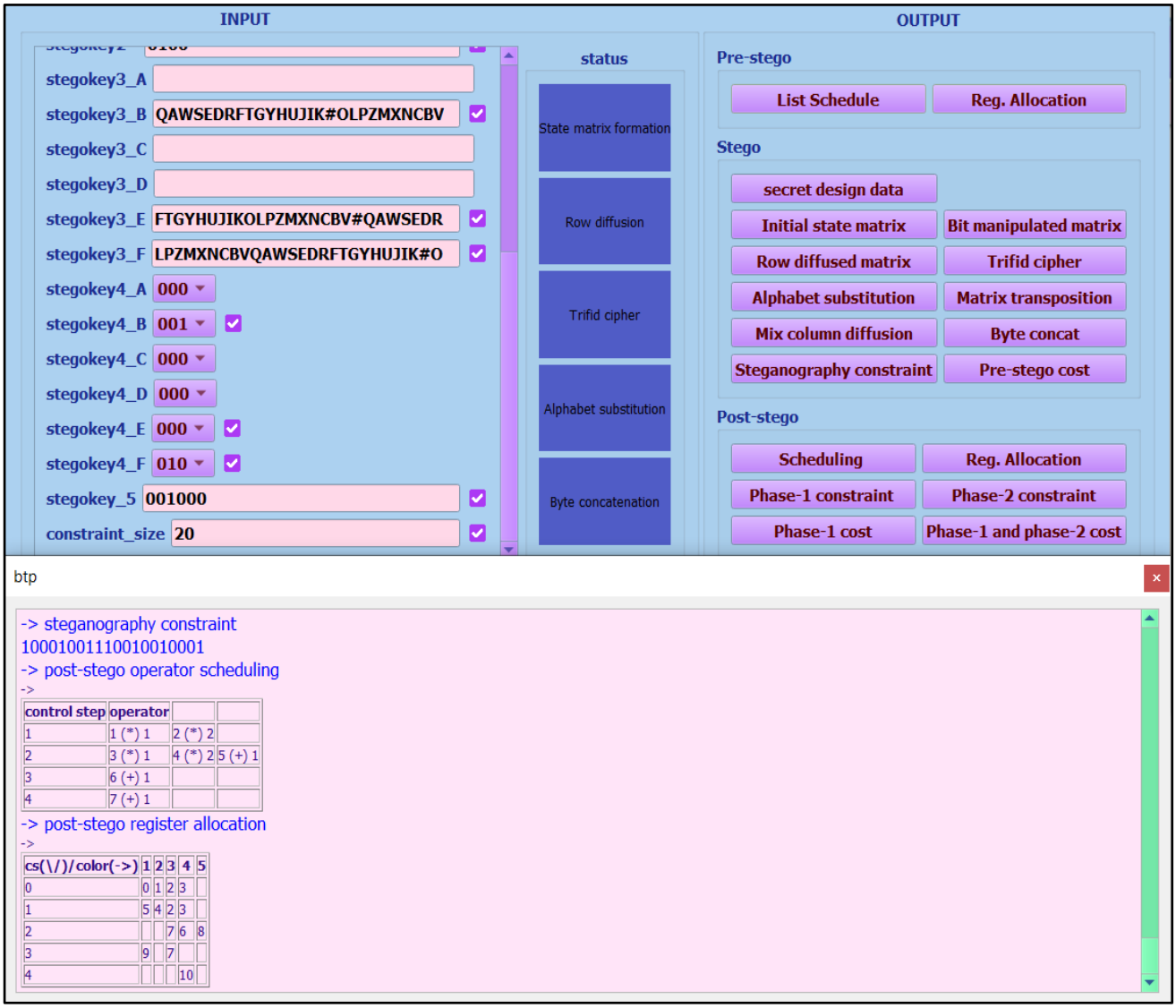

Fig. 17. *Snapshot of the tool post feeding stego-constraints size*

2.5. Case Studies on DSP Hardware Accelerator Applications

(Sengupta and Rathor, 2019b) analysed crypto-based hardware steganography approach for various DSP hardware accelerators viz. 8-point DCT, FIR, JPEG IDCT, MPEG, JPEG sample and EWF. The analysis has been performed by assessing security and design cost of the crypto-based hardware steganography approach. The security analysis has been performed in terms of strength of authorship proof (digital evidence) and the size of stego-key. Further, the strength of authorship proof and the key-size have been compared with a related cotemporary approach (Sengupta and Bhadauria, 2016). Additionally, the design cost of the crypto-based hardware steganography approach has been analysed by comparing it with a non-stego-embedded (baseline version) counterpart. The detailed discussion on security and design cost analysis are as follows (Sengupta and Rathor, 2019b):

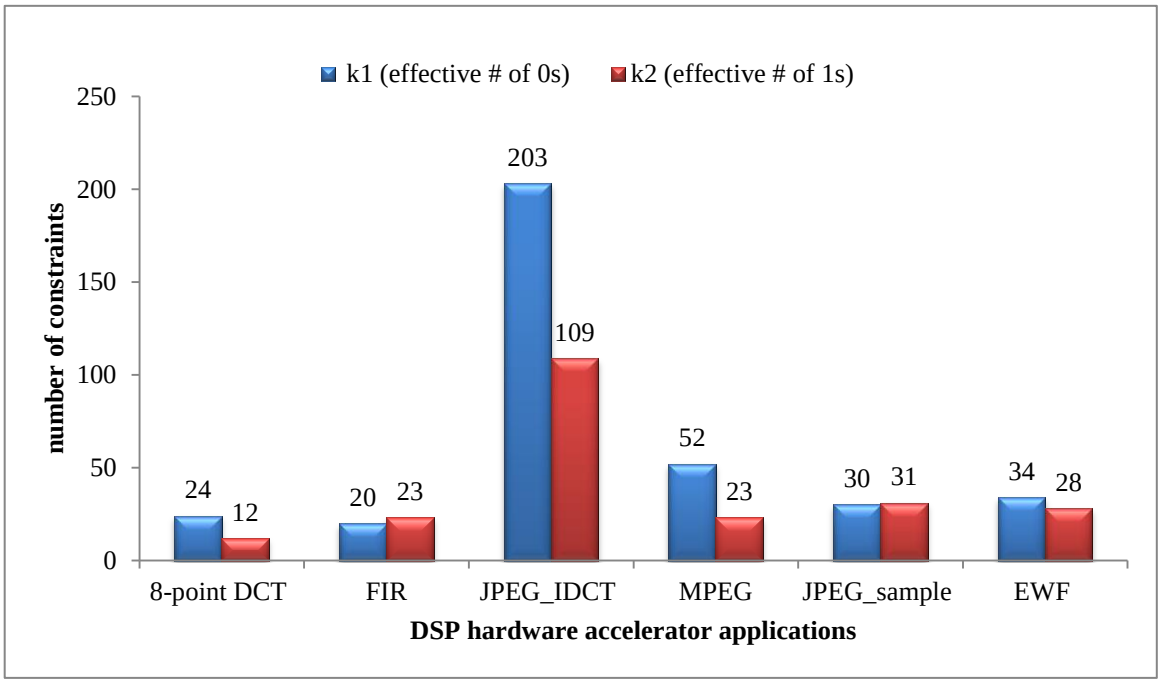

Fig. 18. *Total stego-constraints (k1 + k2) embedded into DSP applications(Sengupta and Rathor, 2019b)*

1. Security Analysis

As discussed earlier, the crypto-based hardware steganography approach aims to secure hardware accelerators against counterfeiting, cloning and false claim of authorship threats. The security against false claim of authorship is ensured using probability of coincidence metric. The strength of authorship proof increases with the decrease in probability of coincidence. Further, the probability of coincidence also indicates the robustness of the embedded stego-mark. A low value of probability of coincidence indicates that higher amount of steganography information (digital evidence) is embedded into the design, thus enhancing the robustness of the stego-mark. This also enhances the resilience against the counterfeiting and cloning threats. This is because the higher robustness of the stego-mark ensures the fail-proof detection of counterfeiting and cloning. Hence, probability of coincidence is an important metric to analyse the security of the crypto-based hardware steganography approach. The mathematical formulation of probability of coincidence is given below (Sengupta and Rathor, 2019b):

$$Pc = \left(1 - \frac{1}{h}\right)^{k1} * \left(1 - \frac{1}{\pi_{j=1}^{m} N(Uj)}\right)^{k2} \qquad (10)$$

Where, ‘Pc’ indicates the probability of coincidence, ‘h’ indicates the number of colours/registers in the CIG before steganography and ‘k1’ indicates the number of stego-constraints embedded during the register allocation phase (i.e. number of 0s embedded). Further, ‘k2’ indicates the number of stego-constraints embedded during the FU vendor allocation phase (i.e. effective number of 1s embedded), ‘N(Uj)’ indicates the number of resources of FU type Uj and ‘m’ indicates the total types of

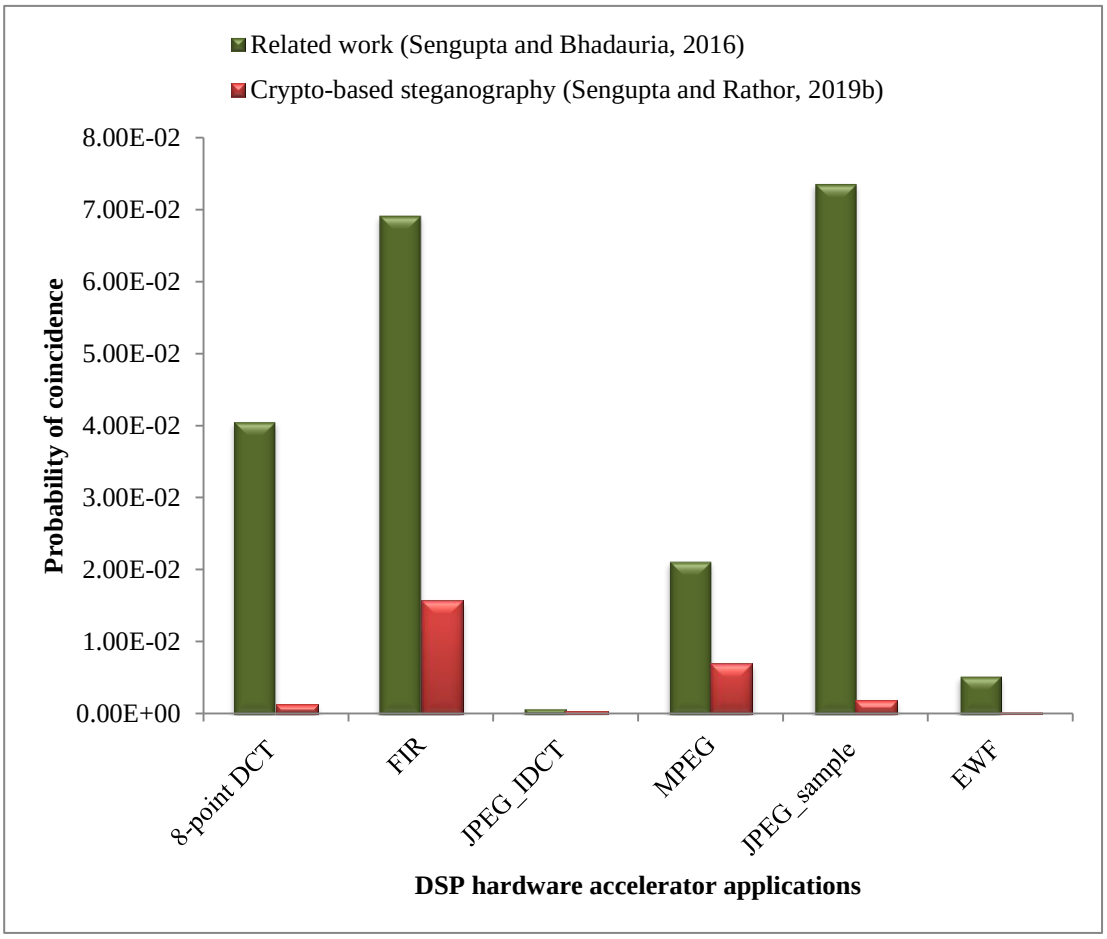

Fig. 19. *Comparison of crypto-steganography approach with watermarking in terms of proof of authorship (Pc) (Sengupta and Rathor, 2019b)*

FU resources required in the design of a hardware accelerator. Lower the Pc metric, lower is the probability of coincidently detecting same stego-mark in an unsecured (non-stego-embedded) version, which also signifies the false positive rate. Therefore a designer always aims to achieve lower value of Pc metric. The Pc value reduces with the increase in the number of stego-constraints k1+k2 (i.e. number of 0s and 1s embedded into register and FU vendor allocation phase respectively). For various DSP applications, the stego-constraints embedded in the crypto-based steganography approach during both register allocation and FU vendor allocation phase are shown in Fig. 18. Further, Fig. 19 shows the probability of coincidence value of the crypto-based steganography approach and compares with a contemporary security approach (Sengupta and Bhadauria, 2016) for the same number of constraints implanted into the register allocation phase. As evident from the figure, the crypto-based steganography approach achieves very low value of Pc because of embedding of stego-constraints into two different phases of HLS process. More explicitly, the crypto-based steganography approach more deeply (and uniformly) embeds the steganography information (digital evidence) than the contemporary approach because of embedding into the FU vendor allocation phase also. Moreover, the contemporary approach performed embedding of secret constraints into a single phase only i.e. register allocation phase. Hence,

Table 7. *Stego-key size in bits for different DSP applications*

DSP applications	Key Size (Stego-strength) in Bits					
	Stego-key1	Stego-key2	Stego-key3	Stego-key4	Stego-key5	Total Key size
8-point DCT	3	10	564	18	15	610
FIR	3	16	564	18	24	625
JPEG_IDCT	3	80	564	18	120	785
MPEG	3	14	564	18	21	620
JPEG_sample	3	32	564	18	48	665
EWF	3	22	564	18	33	640

the crypto-based steganography approach embeds higher digital evidence into the design of hardware accelerators and achieves stronger proof of authorship in contrast to the contemporary approach.

Additionally, the crypto-based steganography approach employs a number of security mechanisms to generate stego-constraints. Further, a very large size stego-key has been involved in the process of stego-constraints generation. This involvement of stego-key renders the back engineering of stego-constraints highly intricate for an attacker, hence s/he fails to regenerate or extract the stego-constraints to prove ownership. This provides very high security to the generated stego-constraints. Therefore, only a designer or vendor who is aware of the scientific algorithm and stego-keys involved in the stego-constraints generation process can regenerate the stego-constraints during detection. Hence, the piracy of the stego-constraints by an attacker does not help him/her as s/he cannot prove as his/her meaningful right over them. Table 7 shows the size of individual sub-keys (i.e. stego-key1 to stego-key5) and the total size of stego-key for various DSP applications. The size of individual sub-keys indicates the contribution in security by different major intermediate steps of stego-constraints generation process. Further, as evident from the table that the crypto-based steganography approach (Sengupta and Rathor, 2019b) requires a very large size stego-key, whereas the contemporary approach (Sengupta and Bhadauria, 2016) does not involve any crypto-key to generate secret constraints. Hence the crypto-based steganography approach offers very high security in terms of larger key size, complex involvement of various security properties in the stego-constraints generation algorithm and as well as higher strength of authorship proof (digital evidence) in contrast to the contemporary approach (Sengupta and Bhadauria, 2016).

2. Design Cost Analysis

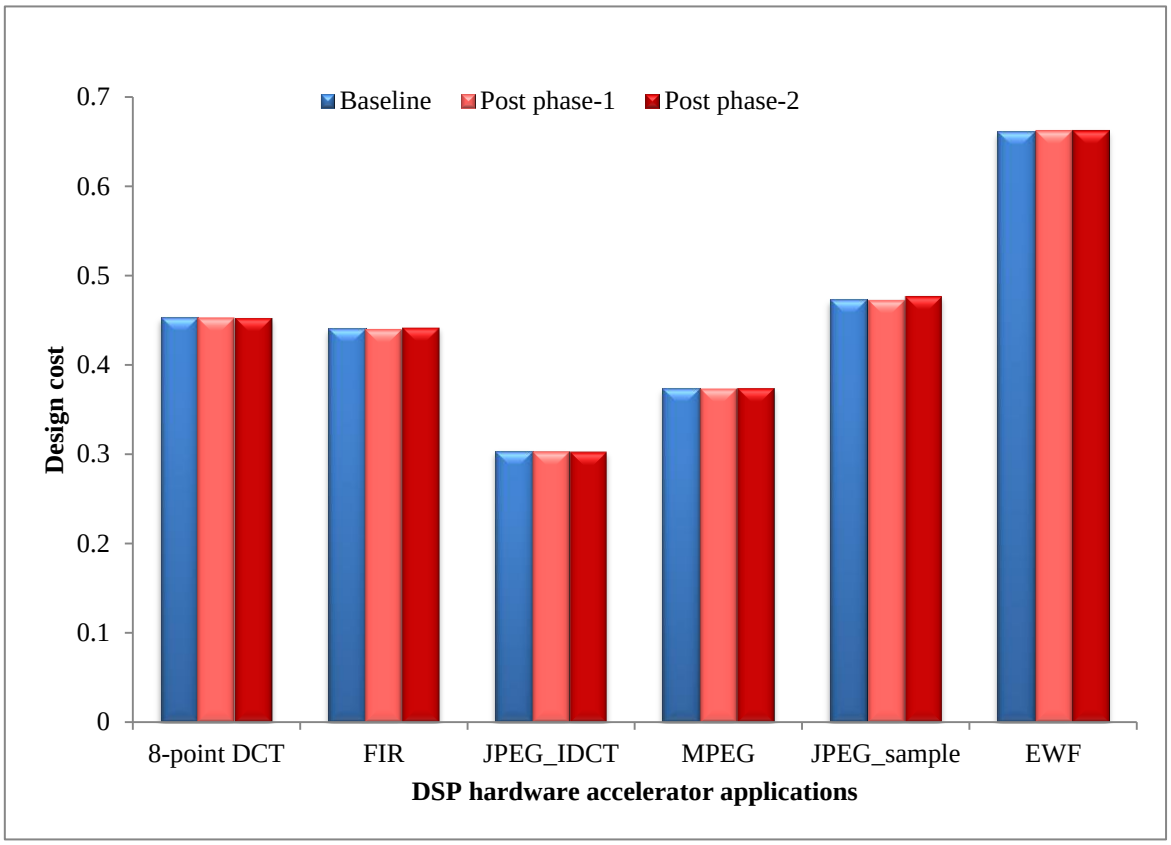

Fig. 20. *Design cost comparison of crypto-steganography approach with respect to baseline (Sengupta and Rathor, 2019b)*

The employment of a security mechanism to secure a design against various hardware threats should be realistic. In other words, employing security mechanism should not incur excessive design overhead. Otherwise, the security mechanism will be lesser effective even if it offers higher security. Therefore, the design cost of the crypto-based steganography approach needs to be analysed. Following equation is used to evaluate the design cost:

$$C_d(U_i) = \rho_1 \frac{L_d}{L_m} + \rho_2 \frac{A_d}{A_m} \qquad (11)$$

Where, $C_d(U_i)$ is the design cost on FU constraints U_i, L_d and L_m are the design latency at specified FU constraints and maximum design latency respectively, A_d and A_m are the design area at specified FU constraints and maximum area respectively and ρ_1, ρ_2 are the weights which are kept at 0.5 to fix equal priority for both.

The design cost of the crypto-based steganography approach (Sengupta and Rathor, 2019b) post-phase 1 (i.e. register allocation phase) and post-phase 2 (i.e. FU vendor allocation phase) have been shown in Fig. 20. Further, the design cost of baseline (i.e. cost before embedding steganography) is also shown in Fig. 20. It is obvious from the figure that the design cost post-phase 1 and phase 2 either remains same as the baseline design cost or increases by a very marginal value. This signifies that the almost zero design overhead is incurred because of embedding crypto-based dual phase steganography, which is a desirable feature of any

security algorithm employed. The reason behind incurring a negligible design overhead for some applications is the requirement of extra registers to embed all secret edge constraints into the CIG. As the size of DSP application increases, the chances of register overhead significantly reduces because a larger DSP application already comprises of a larger number of registers; and there is a very high probability that the available registers would accommodate all the secret edge constraints without incurring any register overhead. In conclusion, it can be inferred that the crypto-based steganography approach (Sengupta and Rathor, 2019b) works more efficiently for larger DSP applications (i.e. incurs zero overhead and simultaneously offering higher security).

2.6. Conclusion

Hardware accelerators play a crucial role in modern age electronics systems. This chapter focused on the DSP hardware accelerators which are typically employed to speed-up the processing of DSP applications such as image processing, audio and video processing applications etc. Because of wide utility of DSP hardware accelerators in electronics products, their security perspective has also been highlighted in this chapter. To secure DSP hardware accelerators against counterfeiting, cloning and false claim of authorship threats, crypto-based hardware steganography approach has been discussed in this chapter. The crypto-based hardware steganography approach exploits various cryptographic and non-cryptographic properties to generate stego-constraints through stego-encoder process. Further, involvement of stego-keys in the constraints generations process enhances the security level. In addition, stego-information has been embedded into two distinct phases of HLS, resulting into deeper, more uniform and more distributed embedding of security evidence. Comparative perspective of crypto-based steganography contemporary security approaches has also been discussed in this chapter.

At the end of this chapter, a reader understands the following concepts:

1. Utility of DSP hardware accelerators
2. Advantages of hardware steganography over hardware watermarking
3. Stego-encoder or stego-constraints generation process in crypto-based steganography
4. Various security properties/mechanisms such as bit manipulation, row diffusion, Trifid cipher based encryption, alphabet substitution, mix column diffusion, byte concatenation and bit mapping.
5. Demonstration of crypto-based steganography on 4-point DCT hardware accelerator
6. Detection process of crypto-based steganography
7. Highlights of a *crypto-stego tool* developed by the authors of crypto-based steganography approach
8. Case studies of crypto-based steganography on various DSP hardware accelerators

2.7. Questions and Exercise

1) What is the difference between steganography and watermarking?
2) What is entropy threshold?
3) Explain the process to computer entropy threshold.
4) What is a hardware accelerator?
5) What is the significance of hardware accelerator in the context of DSP and image processing applications?
6) Give some examples of hardware accelerators.
7) How do you control the amount of secret stego-information added in a DSP design?
8) What is a stego-encoder?
9) What inputs does a stego-encoder accept in a crypto-steganography process?
10) Why is HLS crucial for designing a hardware accelerator?
11) How is a single-phase watermarking different than a multi-phase watermarking?
12) Mention the cryptographic and non-cryptographic security properties incorporated in the stego-constraints generation process.
13) Explain the mapping rules used in stego-embedding process of a DSP hardware accelerator.
14) What is the cover design data in hardware steganography?
15) Calculate the stego-key strengths used in crypto-driven hardware steganography.
16) What is the role of Trifid cipher in hardware steganography?
17) How is state matrix formation important in secret stego-constraint generation process?
18) How do you determine the number of layers of Trifid cipher application?
19) What is the security property used in S-Box?
20) What is the security property used in row/column diffusion?
21) How do you decide the key-size in Trifid cipher based encryption?
22) What is the role of byte concatenation in crypto-driven hardware steganography?
23) How does a designer choose stego-constraint strength before implanting into a hardware accelerator?
24) Explain the FU vendor allocation process.
25) Calculate the total stego-key size of a 8-point DCT core.

References

R. Schneiderman (2010), 'DSPs evolving in consumer electronics applications, '*IEEE Signal Process. Mag*., vol. 27(3), pp. 6–10.

H. R. Mahdiany, A. Hormati and S. M. Fakhraie (2001), 'A hardware accelerator for DSP system design,' in Proc. ICM, pp. 141-144.

C. Pilato, S. Garg, K. Wu, R. Karri and F. Regazzoni (2018), 'Securing hardware accelerators: a new challenge for high-level synthesis,' *IEEE Embedded Syst. Lett*., vol. 10(3), pp. 77-80.

A. Sengupta (2017), 'Hardware Security of CE Devices [Hardware Matters],'*IEEE Consumer Electronics Mag,* vol. 6(1), pp. 130-133.

A. Sengupta (2016), 'Intellectual Property Cores: Protection designs for CE products,'*IEEE Consumer Electronics Mag*, vol. 5, no. 1, pp. 83-88.

A. Sengupta, S. Bhadauria (2016), 'Exploring Low Cost Optimal Watermark for Reusable IP Cores During High Level Synthesis,'*IEEE Access*, vol. 4, pp. 2198-2215,.

A. Sengupta, S. P. Mohanty (2019a), 'Advanced encryption standard (AES) and its hardware watermarking for ownership protection', *Book: 'IP Core Protection and Hardware-Assisted Security for Consumer Electronics'*,e-ISBN: 9781785618000, pp. 317-335.

E. Castillo, U. Meyer-Baese, A. Garcia, L. Parilla, A. Lloris (2007), 'IPP@HDL: Efficient intellectual property protection scheme for IP cores,'*IEEE Trans. Very Large Scale Integr. (VLSI) Syst*., vol. 15, no. 5, pp. 578–590.

M.C. McFarland, A.C. Parker, R. Camposano (1988), 'Tutorial on high-level synthesis', *DAC '88 Proceedings of the 25th ACM/IEEE Design Automation*,vol. 27 (1), pp. 330-336.

S. M. Plaza, I. L. Markov (2015), 'Solving the Third-Shift Problem in IC Piracy With Test-Aware Logic Locking,' *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*,vol. 34(6), pp. 961-971.

A. Sengupta, S. P. Mohanty (2019b), 'IP core and integrated circuit protection using robust watermarking', *Book: 'IP Core Protection and Hardware-Assisted Security for Consumer Electronics*', e-ISBN: 9781785618000, pp. 123-170.

A. Sengupta, D. Roy, S. P. Mohanty (2019), 'Low-Overhead Robust RTL Signature for DSP Core Protection: New Paradigm for Smart CE Design,' *Proc. 37th IEEE International Conference on Consumer Electronics (ICCE)*, pp. 1-6.

D. Roy, A. Sengupta (2019), 'Multilevel Watermark for Protecting DSP Kernel in CE Systems [Hardware Matters],'*IEEE Consumer Electronics Mag*, vol. 8(2), pp. 100-102.

D. Ziener, J. Teich (2008), 'Power signature watermarking of IP cores for FPGAs,'*J. Signal Process. Syst*., vol. 51(1), pp. 123-136.

B. Le Gal, L. Bossuet (2012), 'Automatic low-cost IP watermarking technique based on output mark insertions,' *Design Autom.Embedded Syst*., vol. 16(2), pp. 71-92.

F. Koushanfar et al. (2005), 'Behavioral synthesis techniques for intellectual property protection,' *ACM Trans. Des. Autom. Electron. Syst*., vol. 10(3), pp. 523-545.

A. Sengupta, D. Roy (2017), 'Antipiracy-Aware IP Chipset Design for CE Devices: A Robust Watermarking Approach [Hardware Matters],'*IEEE Consumer Electronics Mag*, vol. 6(2), pp. 118-124.

A. Sengupta, D. Roy (2018), 'Multi-Phase Watermark for IP Core Protection,' *Proc. 36th IEEE International Conference on Consumer Electronics (ICCE)*, pp. 1-3.

A. Sengupta, D. Roy, S. P. Mohanty (2018), 'Triple-Phase Watermarking for Reusable IP Core Protection During Architecture Synthesis, '*IEEE Trans. Comput.-Aided Design Integr. Circuits Syst*, vol. 37(4), pp. 742-755.

A. Sengupta, R. Sedaghat, Z. Zeng (2010), 'A high level synthesis design flow with a novel approach for efficient design space exploration in case of multi-parametric optimization objective,' *Microelectronics Reliability*, vol. 50, Issue: 3, pp. 424-437.

V. K. Mishra, A. Sengupta (2014), 'MO-PSE: Adaptive multi-objective particle swarm optimization based design space exploration in architectural synthesis for application specific processor design,' *Advances in Engineering Software 67*, pp. 111-124.

A. Sengupta and M. Rathor, (2019a), 'IP core steganography for protecting DSP kernels used in CE systems,' *IEEE Trans. Consum. Electron.*, vol. 65(4), pp. 506-515.

A. Sengupta and M. Rathor, (2019b), 'Crypto-based dual-phase hardware steganography for securing IP cores,' *Lett. IEEE Comput. Soc*., vol. 2(4), pp. 32-35.

A. Sengupta, E. R. Kumar and N. P. Chandra (2019), 'Embedding digital signature using encrypted-hashing for protection of DSP cores in CE,' *IEEE Trans. Consum. Electron.*, vol. (3), pp. 398-407.