Nazish *et al. Cybersecurity* (2023) 6:42
https://doi.org/10.1186/s42400-023-00174-9

Cybersecurity

RESEARCH

Open Access

An efficient permutation approach for SbPN-based symmetric block ciphers

Mir Nazish[1], M. Tariq Banday[1*], Insha Syed[1] and Sheena Banday[1]

Abstract

It is challenging to devise lightweight cryptographic primitives efficient in both hardware and software that can provide an optimum level of security to diverse Internet of Things applications running on low-end constrained devices. Therefore, an efficient hardware design approach that requires some specific hardware resource may not be efficient if implemented in software. Substitution bit Permutation Network based ciphers such as PRESENT and GIFT are efficient, lightweight cryptographic hardware design approaches. These ciphers introduce confusion and diffusion by employing a 4×4 static substitution box and bit permutations. The bit-wise permutation is realised by simple rerouting, which is most cost-effective to implement in hardware, resulting in negligible power consumption. However, this method is highly resource-consuming in software, particularly for large block-sized ciphers, with each single-bit permutation requiring multiple sub-operations. This paper proposes a novel software-based design approach for permutation operation in Substitution bit Permutation Network based ciphers using a bit-banding feature. The conventional permutation using bit rotation and the proposed approach have been implemented, analysed and compared for GIFT and PRESENT ciphers on ARM Cortex-M3-based LPC1768 development platform with KEIL MDK used as an Integrated Development Environment. The real-time performance comparison between conventional and the proposed approaches in terms of memory (RAM/ROM) footprint, power, energy and execution time has been carried out using ULINKpro and ULINKplus debug adapters for various code and speed optimisation scenarios. The proposed approach substantially reduces execution time, energy and power consumption for both PRESENT and GIFT ciphers, thus demonstrating the efficiency of the proposed method for Substitution bit Permutation Network based symmetric block ciphers.

Keywords Lightweight cryptography, PRESENT, GIFT, SbPN, Cortex-M, LPC1768, Bit-band memory

Introduction

The Internet of things (Ashton 2009) is an ever-growing network of uniquely identifiable smart connected devices that sense, communicate and share information using heterogeneous networks. Smart IoT applications (Rejeb et al. 2022) are set to bring remarkable benefits to human lives by digitising the day-to-day used physical assets. However, IoT, a fragmented technology, encompasses heterogeneous-natured devices with several limitations and challenges hindering its widespread adoption (Nazish and Banday 2018). Because of the generic constraints associated with these devices in terms of area, bandwidth, memory, power and battery life, together with the financial limitations, security has often been an afterthought, resulting in minimal space left for the crypto implementation. This restricts the application of conventional and standardised crypto primitives for securing IoT devices. Nevertheless, lightweight cryptography attempts to design efficient primitives to mitigate most existing threats while proving less resource intensive.

*Correspondence:
M. Tariq Banday
sgrmtb@yahoo.com
[1] Department of Electronics and Instrumentation Technology, University of Kashmir, Srinagar 190006, India

Most cryptography revolves around the block cipher design due to their profound use in designing pseudorandom number generators, key establishment protocols, MAC, hash and encryption primitives. The design target metrics considered in lightweight design are mainly area (slice or flip-flop count), memory footprint (RAM/ROM), latency, power and energy consumption. However, the heterogeneous nature of IoT devices and the trade-offs between various metrics make it quite challenging to design a 'one design fits all' lightweight block cipher. Therefore, specific featured lightweight cryptographic primitives have been designed for diverse smart IoT applications. For example, PRINCE (Borghoff, et al. 2012) and Midori (Banik et al. 2015) are low-latency and low-energy block primitives, respectively. On the other hand, PRESENT (Bogdanov et al. 2007), SIMON (Beaulieu et al. 2015) and GIFT (Lee 1989) are hardware-efficient primitives, with ITUBee (Karakoç et al. 2013) and SPECK (Beaulieu et al. 2015) being examples of software-friendly block ciphers.

A sub-class of SPN network-based ciphers known as SbPN primitives, such as PRESENT and GIFT block ciphers, are remarkably efficient in hardware because of their s-boxes and bit-permutations. Both use lightweight 4×4 s-boxes to offer confusion, whereas, for diffusion, bit-permutations involving zero gate count are used. However, in software, bit-permutations are the most inefficient in terms of the instruction count and execution time. As such, to make the SbPN-based primitives' software efficient, several methods have been used, such as table-based implementations (Heys 2020), bit-slicing (Kwan 2000), fix-slicing (Adomnicai et al. 2020) in addition to the use of the bit manipulation instructions (Lee 1989). However, even though these implementation techniques provide impressive results, they have some form of complexity in terms of large memory footprint or high energy and power consumption requirements. Moreover, they are not always cost-effective, limiting their use for securing low-end embedded devices. Thus, rather than getting bogged down with addressing security concerns for lightweight IoT devices from a hardware or only software perspective, designing ciphers from a use-case perspective with optimum efficiency in both is the ultimate requirement for smart IoT applications. Thus, efforts must be put to use the appropriate confusion and diffusion implementation techniques that are both hardware and software efficient to have an optimum and balanced crypto design suitable for several applications.

This paper provides a novel method to offer bit-permutation-based diffusion for permutation operation in Substitution bit Permutation Network based ciphers using Bit-Banding feature of contemporary ARM Cortex-M processors. The proposed approach has been implemented, analysed and compared using GIFT and PRESENT ciphers.

The paper is structured as follows: "Background" section summarises the SbPN-based PRESENT and GIFT block primitives and outlines their existing implementation techniques. This section also explains the bit-band feature available with the ARM Cortex-M processors and lists various software-efficient compiler optimisation techniques. "Related work" section presents the literature survey of the software efficient block cipher implementation techniques. "Proposed work" section discusses the proposed software-efficient implementation technique for performing the permutation in PRESENT and GIFT block primitives. "Implementation" section explains the implementation methodology and provides a comparative analysis of the results obtained for the direct and proposed methods in terms of various performance metrics. In addition, this section reports the code and performance improvements obtained for the proposed method using seven optimisation techniques. Finally, "Results and discussions" section provides the summarised results of the proposed technique.

Background

SbPN ciphers

The lightweight block cipher design uses a round function iterated a specific number of times to achieve an optimum security margin. Furthermore, the design must satisfy Shannon's confusion and diffusion paradigm (Shannon 1945). Diffusion means each output bit should be influenced by each plaintext and key input bit. The confusion ensures the complicacy of this dependency, which ascertains that the relationship between the input and output bits is complex and hard to reverse. Non-linear components such as s-box, boolean functions and non-linear arithmetic operations offer confusion in addition to a small amount of local diffusion. Mainly, linear elements such as Maximum Distance Separable (MDS) matrices, bit-permutations, circular shifts, XOR and swap operations are employed to offer diffusion on a global level. Substitution Permutation Network (SPN) is one of the most used secure block cipher construction schemes, utilising s-box, p-box and XOR to realise a round function. A special class of SPN ciphers is the Substitution bit Permutation Network (SbPN) based primitives. An m/n-SbPN is an n-bit block cipher with each s-box being m-bit wide. These ciphers use only the bit-permutations to realise the linear layer.

Permutations at the bit level find vast applications in cryptography and digital processing for faster security and multimedia operations. Bit-permutation has been used in several famous ciphers such as DES, Serpent, PRESENT, GIFT and many more. Permutations of two

and six types have been employed in the hardware-oriented Serpent (Biham et al. 1998) and DES (Biryukov and Cannière 2006) ciphers, respectively. Bit-permutations can be invertible or non-invertible. Compression and expansion p-boxes (Forouzan et al. 2015) are examples of non-invertible permutations. In compression p-boxes, the bit-wise permutation is performed so that the diffused output bits are less in number than the input bits. As a result, several input bits are not mapped to the output. This is useful when the next round needs fewer bits than the previous one. On the other hand, in expansion p-boxes, several input bits are mapped to more than one output bit, which results in a more significant number of diffused output bits than the input bits. This is used in ciphers where the next round needs more bits than the previous one. The irreversible compression and expansions p-boxes are not utilised in SPN or SbPN ciphers, which instead use straight invertible p-boxes with an equal number of input and output bits. The following section details the hardware-efficient SbPN ciphers-PRESENT and GIFT and summarises their existing implementation techniques.

PRESENT block cipher

PRESENT is one of the premier hardware-efficient lightweight block ciphers proposed by Bogdanov et al. in 2007. It is an SbPN-based block cipher, having a fixed block size of 64 bits and variable key lengths of 80- or 128-bits. Figure 1 depicts the encryption process of the PRESENT64/80 cipher consisting of 31 rounds followed by a final post-key-whitening stage. Each round consists of a keyed XOR (addroundkey) and keyless substitution (s-BoxLayer), and permutation (p-Layer) sub-stages.

AddRoundKey: The key scheduling algorithm takes an 80-bit shared key as the input and generates 64-bit sub-round keys using a simple round function involving circular shift, s-box and round constant addition operations. Each sub-round key is bit-wise XORed with the 64-bit input state.

S-BoxLayer: The XORed output is applied as input to 16 invertible 4×4 static s-boxes. Each s-box takes 4 bits (X) as input and yields the confused 4-bit output (S[X]). Apart from confusion, these s-boxes offer a local diffusion. Table 1 lists all the s-box output values corresponding to the 16 inputs in hexadecimal notation.

P-Layer: The 64-bit output from 16 s-boxes is applied as an input to the p-layer that performs a bit-wise diffusion. The bit at index location *i* is shifted to location *P(i)* as per the diffusion Table 2.

Being an SPN cipher, each sub-operation in a round function is invertible. As such, decryption is the reverse of the encryption process. It involves static 4×4 inverse s-box and inverse p-layer realised using a 16-byte

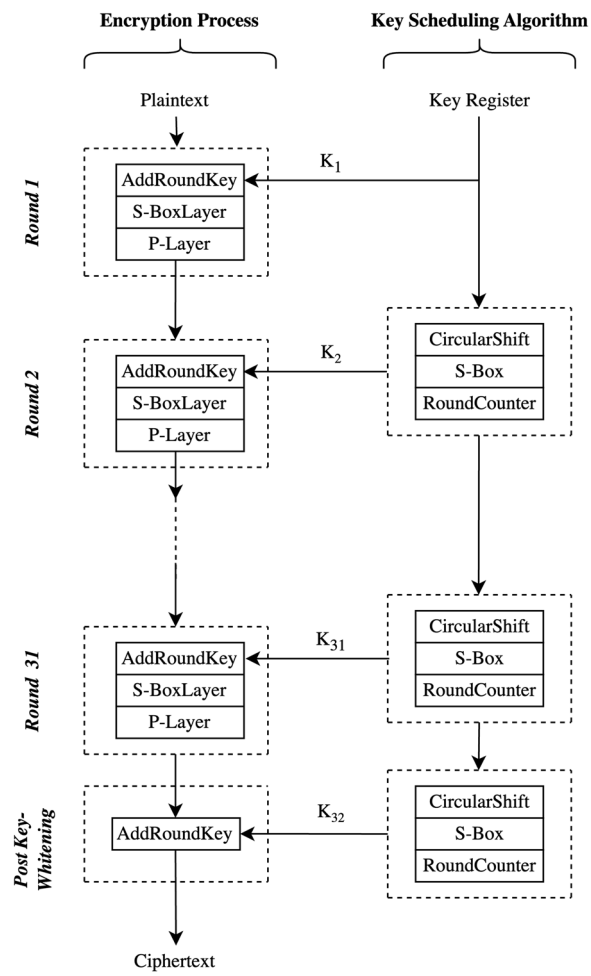

Fig. 1 Encryption process of the PRESENT block cipher

lookup table and 64-bit bit-wise reverse diffusion, respectively. The sub-round keys generated by the key scheduling algorithm are applied in reverse order.

The following section summarises several implementation methods for performing permutation in the PRESENT Block Cipher:

- *Direct Method:* In the direct method, the bit-wise permutation of the s-box layer output is realised using the bit-rotation method. Each bit permutation requires four sub-operations comprising the generation of the mask, masking (AND), shifting and XOR.
- *Wide Table Method:* The wide-table method combines a single s-box and a p-box to form a combined SP lookup table for simultaneous confusion and diffusion. The input to a single SP-wide table is a 4-bit state output from the add round key stage, which acts as the index to a specific memory location. The 64-bit value at the specified location

Table 1 S-box of the PRESENT block cipher

X	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
S[*X*]	C	5	6	B	9	0	A	D	3	E	F	8	4	7	1	2

Table 2 Permutation table of the PRESENT block cipher

i	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
P(*i*)	0	16	32	48	1	17	33	49	2	18	34	50	3	19	35	51
i	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
P(*i*)	4	20	36	52	5	21	37	53	6	22	38	54	7	23	39	55
i	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47
P(*i*)	8	24	40	56	9	25	41	57	10	26	42	58	11	27	43	59
i	48	49	50	51	52	53	54	55	56	57	58	59	60	61	62	63
P(*i*)	12	28	44	60	13	29	45	61	14	30	46	62	15	31	47	63

forms the output corresponding to the 4-bit input state. Sixteen such wide SP tables are required to offer 64-bit diffusion. All the sixteen 64-bit outputs corresponding to 16 nibbles are combined to form the permuted output of one round function.

- *Combined Wide Table Method:* The combined-wide table method combines two s-boxes and one p-box to form a lookup table of 8×64 bytes. The 8-bit input to the two s-boxes forms the input to the combined SP table that points to a specific memory location. The 64-bit value corresponding to this location forms the output. For permuting a 64-bit state, eight such SP tables are required. The eight 64-bit values are XORed together to yield permuted state.

All the methods mentioned above are software efficient; however, these have drawbacks in terms of memory footprint, instruction count and timing requirements. The direct method is a memory-efficient technique. However, this incurs a substantial overhead due to the requirement of several mask, shift and XOR sub-operations for the ciphers with large block sizes. The table-based methods are known for their high-speed execution. These have comparatively fewer instruction requirements than the narrow-table or direct approach. However, these highly memory-intensive methods require 32 and 16 SP tables for encryption and decryption in wide and combined-wide table implementation methods, respectively. Furthermore, these methods employ several bit mask, shift and XOR operations to apply a lookup operation on a specified data nibble or byte.

GIFT block cipher

Although PRESENT is a hardware-oriented cipher, it is not much resistant to linear cryptanalytic attacks. Also, it utilises a high branch numbered s-box that proves costlier in terms of area footprint. Therefore, Banik et al. worked towards designing a comparatively lightweight and more secure cipher and finally came up with an improved version of PRESENT in 2017 named the GIFT block cipher. Unlike PRESENT, which uses an s-box with branch number 3, GIFT uses a reduced branch numbered two s-box that proves more area and cost-efficient and is more resistant against the linear cryptanalytic attack.

GIFT is an SbPN-based symmetric block cipher with two versions: GIFT 64/128 and GIFT 128/128, having a fixed key length of 128 bits with varying block sizes and rounds. For block-size of 64 and 128 bits, 28 and 40 rounds are used, respectively.

GIFT64/128 (Fig. 2) encryption process utilises a key-alternating construction with two keyless (subcells and permbits) and one keyed (addroundkey) sub-stage:

Subcells: The 64-bit input state is applied nibble-wise to the 4×4 static s-box to offer optimum confusion and a small amount of diffusion. Each nibble 'X' is replaced with 'S[X]' using the pre-defined s-box mapping shown in Table 3.

Permbits: This layer performs a bit-wise 64-bit permutation on the output bits of 16 parallel s-boxes. A bit at index position *i* is shifted to the *P(i)* bit position as per the permutation table given in Table 4.

AddRoundKey: The diffused state bits from the permbits stage are XORed with the sub-round key bits and round constants. Each sub-round key generated from the key scheduling algorithm using simple extraction and circular shift operations is 32 bits in size. Therefore, only 32 bits out of the 64-bit state are bit-wise XORed with

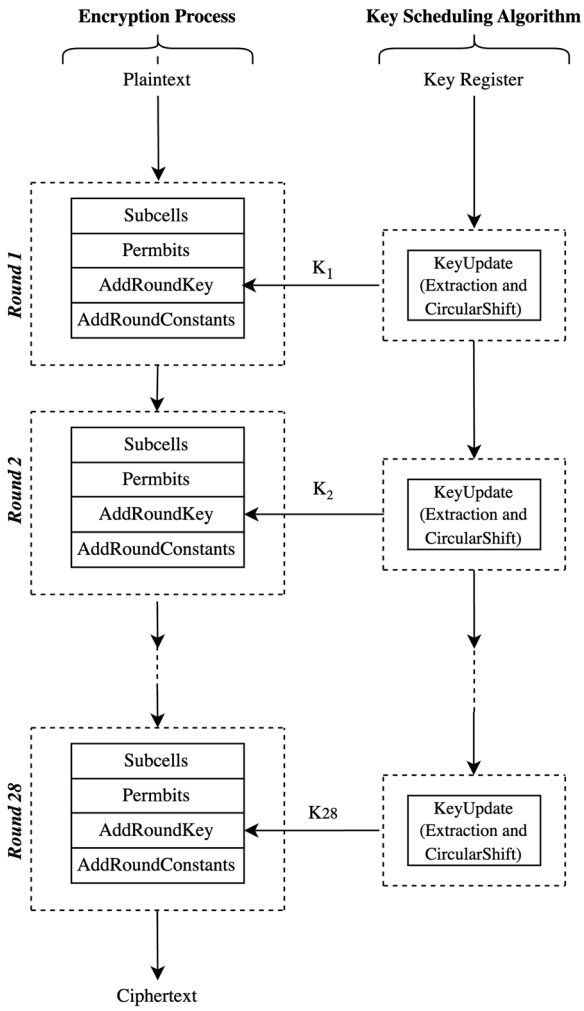

Fig. 2 Encryption process of the GIFT block cipher

the sub-round key for greater hardware efficiency. This saves the computational costs associated with the XOR operations, making the cipher efficient in hardware and software.

AddRoundConstants: Six input state bits are bit-wise XORed with six round constants. In addition, bit (b_{63}) is XORed with '1'.

GIFT decryption involves using a 4×4 static inverse s-box and inverse p-layer to offer confusion and diffusion in the cipher. The round keys generated from the key generation algorithm are applied in reverse order to obtain the original message.

The following methods exist for performing permutation in the GIFT Block Cipher:

- *Direct Method:* In the direct Implementation, the permutation layer takes the output state from the s-box as input. It performs diffusion using the bit-rotation method involving several mask generation, masking, shifting and XOR sub-operations.
- *Bit-Slicing Method*: In the bit-slicing technique, diffusion is performed using masking, shift and XOR steps simultaneously on bits in a given slice. This, in turn, amounts to the requirement of multiple such operations for permuting bits in multiple slices, resulting in a higher cycle count and delayed execution. Bit-sliced permutation can also be performed by transposing and then subjecting each slice to different row-swapping operations determined by the slice number.
- *Fix-Slicing Method*: In the fix-slicing method, the first slice is not subjected to any diffusion operation, whereas the rest of the three slices undergo row-wise and column-wise rotations.

Table 3 S-box of the GIFT block cipher

X	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
S[*X*]	1	A	4	C	6	F	3	9	2	D	B	7	5	0	8	E

Table 4 Permutation table of the GIFT block cipher

i	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
P(*i*)	0	17	34	51	48	1	18	35	32	49	2	19	16	33	50	3
i	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
P(*i*)	4	21	38	55	52	5	22	39	36	53	6	23	20	37	54	7
i	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47
P(*i*)	8	25	42	59	56	9	26	43	40	57	10	27	24	41	58	11
i	48	49	50	51	52	53	54	55	56	57	58	59	60	61	62	63
P(*i*)	12	29	46	63	60	13	30	47	44	61	14	31	28	45	62	15

The direct implementation method proves to be a memory-efficient technique. But, this incurs a substantial overhead due to the requirement of several mask, shift and XOR sub-operations for the ciphers with large block sizes. The bit-sliced-based computational process is more straightforward and faster because the plaintext block is divided into multiple slices. Also, it permits the processing of multiple blocks in parallel. However, it can prove inappropriate for low-end IoT devices that usually work with much smaller payloads. Also, substantial overhead is associated with the diffusion layer, as bits must be transposed in the slice individually rather than in large chunks, making it computationally intensive. Even though bit-slicing can improve speed, the overheads associated with packing and unpacking data at the start and the end of the encryption and decryption processes make the process quite resource-consuming for ultra-lightweight devices. Furthermore, this method uses several general-purpose registers to store the transposed bits of a given message. Unfortunately, low-end IoT processors often have a minimal number of such registers, thereby increasing the number of load and store instructions that degrade the overall performance. Moreover, the bit-sliced permutation in GIFT cipher involves multiple mask, shift, and XOR operations, thereby incurring large computational overhead regarding the number of cycles. The fix-slicing technique saves multiple operations by replacing the transposition and row-switching operations with row and column rotations, thereby increasing the speed of the cipher. It also takes advantage of the barrel shifter capability available with the ARM Cortex architecture for performing multi-bit rotations in a single clock cycle, thereby making the implementation of the linear layer less costly. However, the round keys and round constants need to be modified as per the new bit positions, which incurs additional computational overheads.

Bit-band memory

ARM Cortex-M (Banday 2018; Rouf et al. 2022) are 32-bit processors primarily designed for deeply embedded microcontrollers and IoT market spaces. These low-powered processors feature several energy modes, barrel shifters and pipelined architectures. This makes them suitable for diverse low-power and low-latency IoT applications. Furthermore, these processors are based on ARMv7 instruction set architecture and support the Thumb-2 instruction set, which includes a mix of 16- and 32-bit instructions, making them highly suitable for high-performance and memory-deficient IoT applications (Schwabe and Stoffelen 2017; Kim, et al. 2022). In addition, ARM Cortex-M processors have optional support for bit manipulation using the bit-banding feature. Unlike other processors, which include separate bit-manipulation processors or use specific instructions to perform bit-level manipulations that increase the overall design cost, these processors incorporate a unique feature of bit-banding that uses two memory regions, bit-band and bit-band alias, to support bit-wise operations. Regular access to the bit-band region results in a word read or write operation. On the other hand, normal read or write to the bit-band alias region results in single-bit access in the corresponding bit-band region. This is because each bit in the bit-band region is mapped to a word (more specifically, to the least significant bit of the 32-bit word) in the corresponding bit-band alias area.

In ARM Cortex-M3 processors, two bit-band regions are set aside for performing the bit-band operations. These are located in the starting 1 MB of SRAM and the first 1 MB of the peripheral regions with base addresses as 0X20000000 and 0X40000000, respectively. The corresponding two bit-band alias regions are in SRAM, and peripheral regions with base addresses 0X22000000 and 0X42000000, respectively. Each bit-band alias region is 32 MB in size because each bit mapping in the bit-band region requires a word (32 bits) in the bit-band alias region.

Bit-banding offers several advantages. First, it simplifies the bit write and read operations by working directly on the appropriate bit-band alias location corresponding to a specific bit in the bit-band region. It performs bit manipulation in a single cycle. Unlike the conventional bit-modification involving read, modify and write sub-tasks, bit-banding permits atomic and uninterrupted error-free bit operations. This also prevents conflicts in the case of multiple tasks using shared memory (Yiu 2014). Also, single-bit manipulation operation is realised using a single load or store instruction, which results in faster bit manipulations (Bai 2015). Further, it simplifies the execution of several conditional branching operations by reading a specific bit-band alias location instead of reading and masking 32 bits in the bit-band region, thereby speeding the branching decisions (Tahir and Javed 2017).

Compiler optimisation

One of the design approaches to achieve software efficiency is to employ optimisation techniques available with the compilers. This method makes the design either code-efficient with reduced RAM and ROM utilisation or can help enhance the execution speed. In addition, using optimisation techniques can help run programs faster without changing the code. The compiler uses precomputation of values, inlining functions, unrolling loops, reordering code statements, and many more to produce a much faster binary. However, the downside with the inclusion of the compiler optimisation techniques is that it can make the program hard to debug. With lower

optimisation levels, detailed information about the program can be viewed, which can then be used to track down the bugs in the code. On the other hand, this feature becomes more restricted with higher optimisation levels, which hinders debugging to a greater extent. However, these levels permit high-speed or low code footprint optimisations. Thus, it is recommended to use lower or no optimisations while developing the algorithm and switch to higher optimisations once the code is released.

Several compiler optimisation options are available with the KEIL MDK Integrated Development Environment (Table 5). They either optimise the program for code size or performance, and opting for one metric degrades the other. Furthermore, depending on the type of application and the constraints involved, one can use a particular optimisation level(s).

Related work

Ruby Lee (Lee 1989) used the EXTRACT and DEPOSIT bit manipulation instructions available with the PA-RISC Precision Architecture processors to perform bit permutations. The results reported the requirement of only two instructions for performing a one-bit permutation, thus resulting in a 50% reduction in instruction count compared to the bit-rotation method that requires four instructions to perform a single-bit permutation.

Eli Biham (1997) presented a high-speed software-friendly bit-sliced implementation of the DES block cipher that resulted in two times increase in its execution speed. Furthermore, an average requirement of 100 gates has been reported for the hardware implementation of one s-box.

Matthew Kwan et al. (2000) propounded the bit-slicing term and used this method to improvise Biham's work with 56 gates required for a single s-box implementation.

Matsui et al. (2007) provided improvised results for the AES block cipher implemented using the bit-sliced Intel Core2 processor architecture. The results report reduced execution time requirements for the proposed implementation compared to the table-based AES implementation.

Bogdanov et al. (2007) proposed PRESENT, a hardware-oriented block cipher for highly constrained devices. It has a fixed block size of 64 bits and a variable key size of 80 and 128 bits. Both versions use 31 SPN rounds with a post-key-whitening step used at the end.

In his thesis, Poschmann (2009), provides the code and speed-optimized implementation of the PRESENT block cipher for diverse platforms with 8-, 16- and 32-bit processors. It also uses the narrow table approach for the s-box implementation, which despite being efficient in software, is prone to cache timing attacks.

Benadjila et al. (2014) performed bit-sliced implementation of several block ciphers using SIMD instructions and vectorisation features available with the Intel×86 platforms. The results report increased speed gain for the analysed ciphers, including PRESENT.

Papapagiannopoulos et al. (2014) implemented various block primitives in a bit-sliced manner on the ATtiny family of AVR platforms. Improved results have been reported for the PRESENT cipher by utilising the

Table 5 Advantages and drawbacks of various compiler optimisation levels

Compiler optimization level	Advantages	Drawbacks
-O0	High correlational view between source and generated codes Quick build and compile Easy Debugging Best for prototyping	No Optimisations enabled Large Code-size Higher execution-time
-O1	Better debug view Good stack utilisation	High Memory Requirements High Execution-time
-O2	High Speed	Larger Code-size
-O3	High Speed	Larger Code-size Poor correlational view between source and generated codes
-Ofast	High Speed	Larger Code-size Poor correlational view between source and generated codes May perform optimisations that are not standard compliant
-Oz	Reduced Memory Footprint	Slower execution
-Os	Balanced speed and memory usage	Moderate performance

s-box implementation of Boyar and Peralta (2010) realised using 14 instructions.

Banik et al. (2017) proposed the GIFT block cipher in 2017, having variable block sizes of 64- and 128-bits with a fixed key length of 128-bits. The number of rounds is variable, 28 for 64-bit and 40 for 128-bit block-sized versions. Results prove the GIFT cipher is more hardware-efficient and secure than the PRESENT block cipher.

Tiago et al. (2017) presented a timing attack-resistant masked implementation of the PRESENT block cipher. Furthermore, the implementation involves decomposing the linear layer and realising the s-box in a bit-sliced manner using optimised boolean functions. On 32-bit ARM Cortex processors, an 8% improvement in execution speed is reported for the cipher, requiring 2100 cycles compared to that provided by FELICS.

Dinu et al. (2019) evaluated crypto ciphers in terms of a figure of merit calculated from various metrics such as time, RAM and ROM footprint. Nineteen block primitives have been comparatively analysed on AVR, MSP430 and ARM, which are 8-, 16- and 32-bit platforms, respectively. In the case of the PRESENT block cipher, a time-efficient implementation has been carried out utilising the combined substitution and permutation tables.

Adomnicai et al. (2020) proposed a software-friendly implementation technique for the GIFT block cipher named fix-slicing. The method uses a few rotations realised using the barrel shifter feature available with the ARM Cortex-M3 processors. The results report faster execution speed requiring 800 and 1300 cycles for GIFT-64 and GIFT-128, respectively, compared to AES and PRESENT ciphers requiring 1617 and 2116 cycles, respectively.

Adomnicai et al. (2020) applied the fix-slicing technique to the AES block cipher. Compared to the bit-sliced AES, the results report a 52% reduction in diffusion operations using the fix-sliced AES implementation technique, requiring only 81 cycles for a single-byte encryption on 32-bit processors.

Further, many software efficient ciphers such as RECTANGLE (Zhang et al. 2015), a 4/64 SbPN cipher with structure similar to GIFT and PRESENT ciphers have been proposed. RECTANGLE uses shift rows to realise the diffusion layer, which is more software friendly than bit rotation method used in direct implementation methods for PRESENT and GIFT block ciphers. However, as far as its security is concerned, not much analysis has been reported regarding how the linear and differential trials are propagated in the RECTANGLE cipher. Also, its key scheduling algorithm is more complex than PRESENT and GIFT primitives. Furthermore, four rounds are required to attain full diffusion in RECTANGLE cipher, whereas the same is attained in only three rounds in case of the PRESENT and GIFT block ciphers.

Although the works mentioned above have attempted to make the cipher implementation efficient in software to a certain extent, however, the associated overheads in terms of larger instruction count, higher memory, power and time requirements along with the inclusion of specific bit-manipulation instructions, a significant increase in the cost of the development platforms, makes it financially and resourcefully a non-viable option to secure the constrained smart IoT applications. This necessitates designing a novel, software-friendly, cost-effective implementation technique for securing diverse low-energy and high-performant low-latency IoT applications. Further, the digital world around us is mostly embedded in nature and as such using only software efficient or mere hardware efficient crypto primitives cannot be considered as a favourable design approach for securing the low-end devices. There is a need to address the security concerns of the smart embedded applications from a holistic approach that should consider both the hardware and software aspects. This paper proposes a novel software efficient implementation method for hardware efficient SbPN ciphers to make these primitives more accessible for use in a wide range of embedded devices, particularly those with limited resource availability.

Proposed work

This paper proposes a novel software-friendly implementation technique for performing the bit-wise permutation in the SbPN ciphers by employing the 'Bit-Banding' feature of ARM Cortex-M processors. An easy, efficient, and high-speed software-efficient mapping between the bit-band alias regions is performed to achieve bit-level diffusion.

All the steps involved in the encryption round function of the PRESENT64/80 cipher (Fig. 3), except the diffusion, are performed in a manner similar to the direct implementation method. First, the 64-bit XORed output is provided to 16 (4×4) s-boxes that provide the confused 64-bit output state. This forms the input to the diffusion layer (P). Then, the diffusion layer in the proposed bit-banding approach is implemented as per the pseudocode (Algorithm 1) using the following steps:

Step 1 Initialise the permutation table, P_t (as shown in Table 2) and store it in memory.

Step 2 Declare two bit-band memory areas, P and Q, each 8 bytes wide.

Step 3 Store 64-bit output state from the sixteen s-boxes in the 'P' bit-band memory of SRAM.

In ARM Cortex-M3-based LPC1768 IoT hardware platform, the SRAM bit-band region starts from 0X200000000×20,000,000. However, the locations from

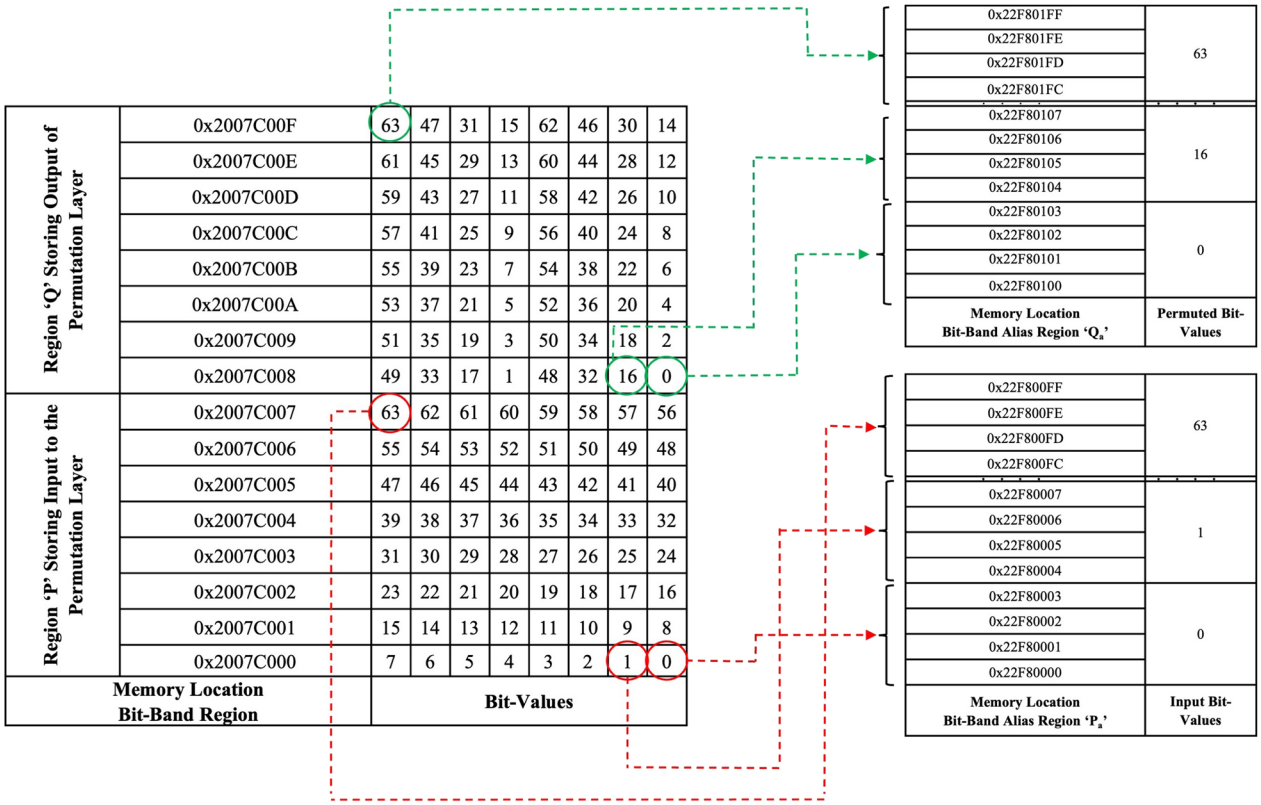

Fig. 3 Bit-band method of performing permutation for the PRESENT block cipher

0X20000000 to 0X2007BFFF are reserved. The input state bits to the linear layer are stored starting from location 0X2007C000. Sixteen nibbles require eight memory byte locations for storage. As such, bits b_0 and b_{63} occupy LSB of 0X2007C000 and MSB of 0X2007C007 memory locations in the bit-band region.

Algorithm 1 : Pseudocode for Bit-Band Permutation

```
function BB_PERM(X)                                   ▷ input: 64 bit state X
    Initialize: P_t                                   ▷ initialize permutation table
    Declare: P, Q   ▷ declare P and Q as two bit-band memory areas of 64-bits each
    P ← X                                             ▷ store 64-bit state X in bit-band memory area P
    for i = 0 to 63 do
        Q_a + P_t(i) × 4 ← P_a + (i × 4)              ▷ perform bit-permutation for each bit
    end for                                           ▷ P_a and Q_a are bit-band alias for P and Q respectively
    return Q                                          ▷ output: 64-bit state Q
end function
```

Step 4 Use the following mapping formula to fill in the Q_a bit-band alias memory locations with the permuted state bits.

$$Q_a + P_t(i) \times 4 \leftarrow P_a + (i \times 4)$$

where P_a and Q_a are the base addresses of the bit-alias region storing the input and output of the permutation layer, respectively, P_t is the array of permutation values (as given in Table 2), and i represents the bit number varying from 0 to 63.

In the bit-band alias region, each bit is represented by 32-bit; storing a 64-bit state requires $64 \times 4 = 256$ bytes in the 'P_a' bit-band alias region with b_0 stored at 0X22F80000 through 0X22F80003 and b_{63} occupying 0X22F801EC to 0X22F801EF memory locations. Similarly, in the 'Q_a' bit-band alias area, 0X22F80100 to 0X22F80103 memory locations store the permuted value b_0. Again, the b_{63} permuted bit occupies four locations starting from 0X22F801FC bit-band alias memory location. In this mapping process, the Q bit-band region from 0X2007C008 to 0X2007C00F, corresponding to the Q_a bit-band alias area, gets automatically filled with the permuted output state.

In addition to the mapping formula, a scatter file is exclusively used to direct the linker to set aside the particular SRAM regions for the permutation function to avoid memory conflicts during program execution.

Step 5 Return the 64-bit permuted output state from the Q bit-band area for further processing by the following rounds.

Similarly, the decryption phase (Fig. 4) involves the following mapping formula between the inverse permutation layer's input and output state bits.

$$Q_a + (i \times 4) \leftarrow P_a + P_t(i) \times 4$$

The proposed method for performing the bit-banding-based permutation has been illustrated for the PRESENT block primitive. GIFT block cipher (Figs. 5, 6) follows the bit-banding approach similar to that used for the PRESENT block cipher.

Implementation

Methodology

The PRESENT and GIFT lightweight block primitives have been choosen to evaluate the proposed permutation because both the ciphers offer a good balance between efficiency, security and hardware simplicity. These are used to secure RFID tags, wireless sensor networks and any low-end embedded smart IoT applications for which resource intensive ciphers like AES are not usually preferable (Bogdanov et al. 2007). Further GIFT cipher inspired by PRESENT cipher has been implemented in part or full in many of the NIST lightweight AEAD candidates such as GIFT-CoFB (Banik et al. 2019a), SUNDAE-GIFT (Banik et al. 2019b), HYENA (Avik Chakraborti 2019a), ESTATE (Avik Chakraborti et al. 2020), LOCUS and LOTUS (Avik Chakraborti et al. 2019b).

The PRESENT and GIFT block ciphers have been implemented on a 32-bit ARM Cortex-M3-based LPC1768 development board. It has 64kB and 512kB of RAM and ROM, respectively and operates with a core clock frequency of 100 MHz. The availability of onboard 20-pin JTAG, 10-pin and 20-pin Cortex connectors permits real-time debugging and tracing of the programs. KEIL MDK has been used on the host side as an integrated development environment to observe, analyse, verify and optimise the algorithms. The algorithms' flashing, debugging and tracing have

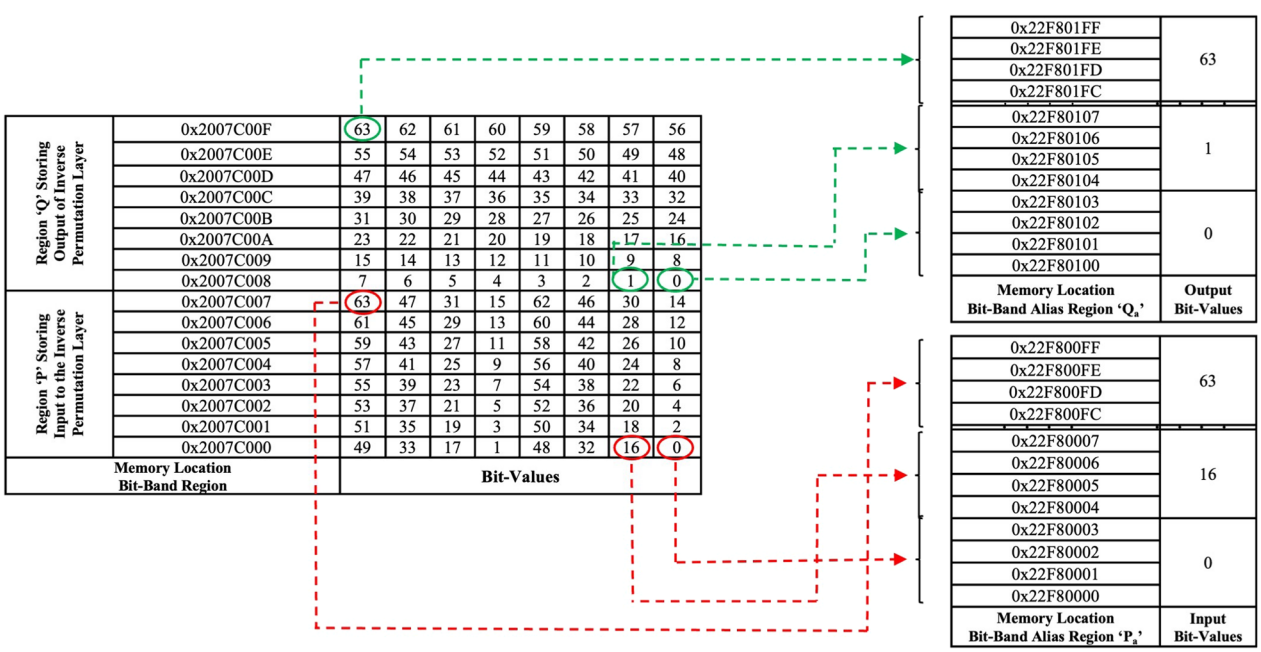

Fig. 4 Bit-band method of performing inverse permutation for the PRESENT block cipher

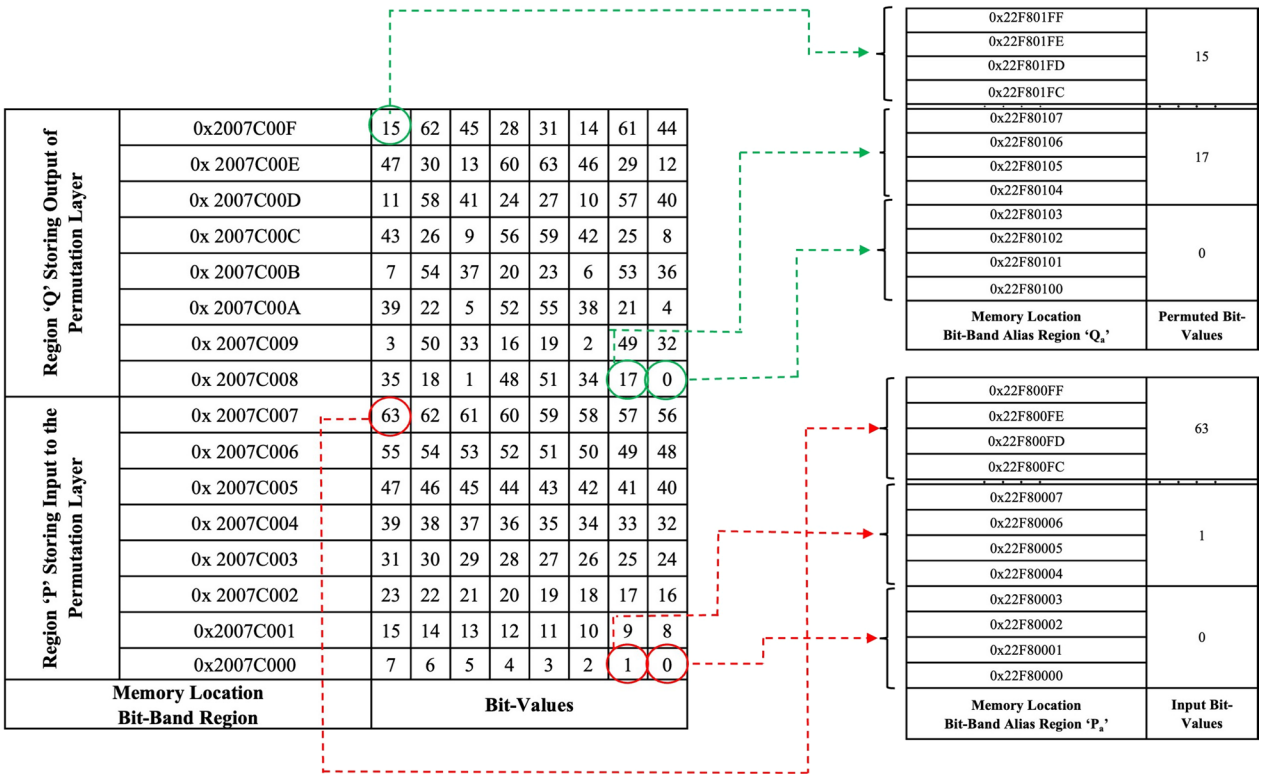

Fig. 5 Bit-band method of performing permutation for the GIFT block cipher

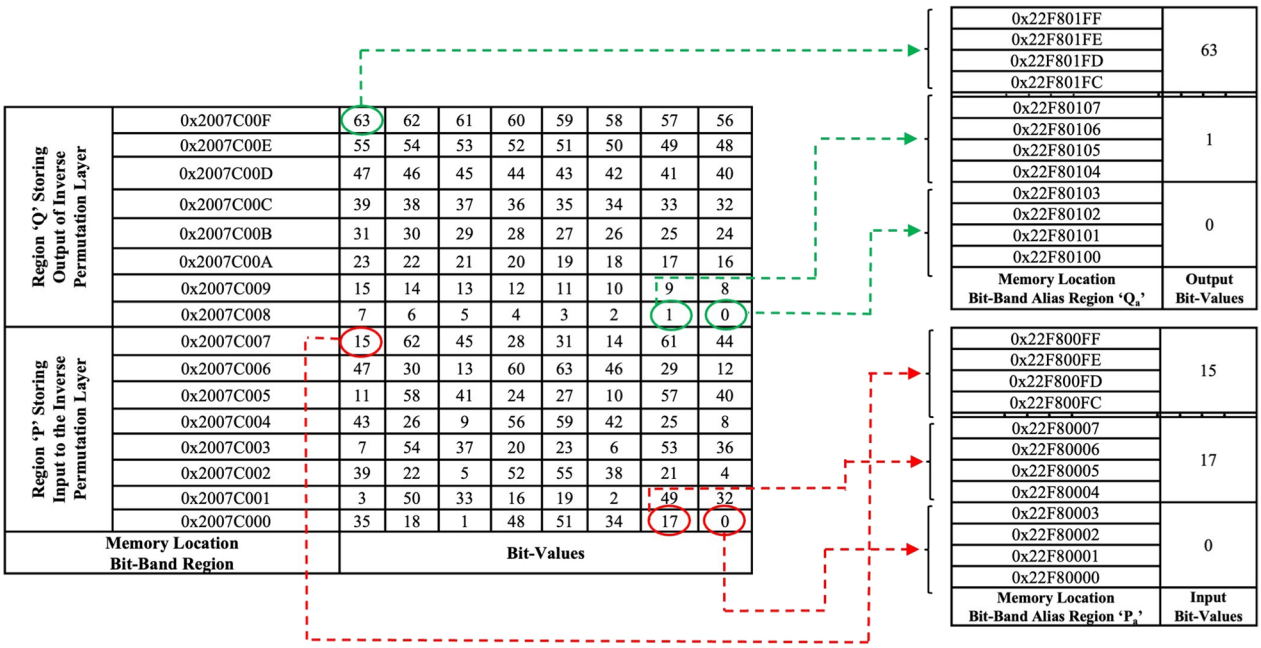

Fig. 6 Bit-band method of performing inverse permutation for the GIFT block cipher

been carried out using advanced debug adapters from ARM, namely ULINKpro and ULINKplus. The RAM and ROM memory usage of the primitives can be calculated using either debug adapters. Power measurement has been explicitly performed using ULINKplus debug adapter. Also, the streaming trace capability with ULINKpro permits complete module and function-level instruction tracing for longer, thus providing detailed execution timing information. Moreover, the energy consumption of the ciphers has been calculated as Energy (in μJ) = Power (in mW) * Time (in ms). Moreover, several compiler optimisation techniques have been used to increase the code and speed efficiency of the direct and proposed implementation methods. In addition, a highly optimised set of libraries known as micro-lib has been used that helps reduce the overall flash footprint of the block cipher primitives to a marginal extent.

Results and discussions

The simple mapping between the bit-band and bit-band alias regions with the preclusion of multiple masks, shift, and XOR operations make the bit-band permutation method the most time-efficient.

Table 6 tabulates the results for various performance metrics such as power, energy, execution time and memory (RAM and ROM) utilisation for the PRESENT block cipher implemented using the proposed bit-banding technique and the direct method.

The percentage difference in various lightweight metrics for the bit-band and direct implementation methods has been reported for a better comparative analysis. In addition, separate computations for the encryption and decryption phases of the PRESENT block primitive have been listed. For the encryption part, the maximum improvement has been reported for the execution time, with the bit-band technique requiring 68.96% less time than the direct method. This is followed by 42.43% and 17.82% reductions in energy and power consumption, respectively. The only downside of the bit-band technique is a comparatively higher memory requirement in terms of RAM and ROM footprints. Similar trends have been observed for the decryption results, with the bit-band method outperforming the direct method by 27.31%, 45.15% and 82.53% improvements in power, energy and time requirements. However, the bit-band method entails a slightly larger memory size than the direct approach.

Figure 7 presents the encryption results of various evaluation metrics for the direct and bit-band implementations of the PRESENT block cipher, executed with the different optimisation levels (as listed in Table 5) available with the KEIL IDE. This evaluation has been made to evaluate the performance of the proposed technique in different compiler optimization levels.

Significant improvements have been obtained for the bit-band method with 68.58% and 86.63% reduction in power and energy consumption by utilising –O2 compared to the –O0 technique. Furthermore, a speed improvement of 56.45% has been attained using the time-optimized –O3 level. Moreover, the overhead with the bit-band-based permutation technique has subsided by 46.02% with the -Ozimage optimisation level. In the case of the direct method, the -O2 level improves power and energy consumption by 53.163% and 78.917%, respectively, compared to the –O0 level. Also, with the –Os balanced level, a 69.14% reduction in execution time has been observed. Finally, more than a 50% decrease in memory footprint is obtained using the –Ozimage optimization level.

Figure 8 presents the decryption results for the PRESENT cipher run with different compiler optimization levels (Table 5). In the case of the bit-band method, 84.83% and 93.97% improvements in power and energy consumption have been observed with the -O2 level compared to the –O0 level. A 78.49% less time for decryption is reported with the –O1 level. The –Oz image size reduces the memory requirements by half. For the direct method, as compared to the –O0 level, a 70.01% and 89.94% decrease in power and energy consumption has been obtained using the –O2 level. A 51.29% decrease

Table 6 Performance comparison of the proposed implementation technique for the PRESENT block cipher

Implementation Technique	Encryption			Decryption		
	Direct	Bit-Band	Bit-Band Vs Direct (%age difference)	Direct	Bit-Band	Bit-Band Vs Direct (%age difference)
Power (mW)	3.478	2.858	17.82	2.621	1.905	27.31
Energy (uJ)	5.132	2.954	42.43	2.903	1.592	45.15
RAM (Bytes)	600	536	10.67	616	528	14.29
ROM (Bytes)	5896	6192	– 5.02	5912	6192	– 4.74
Execution-Time (ms)	2.385	0.7403	68.96	1.611	0.28146	82.53

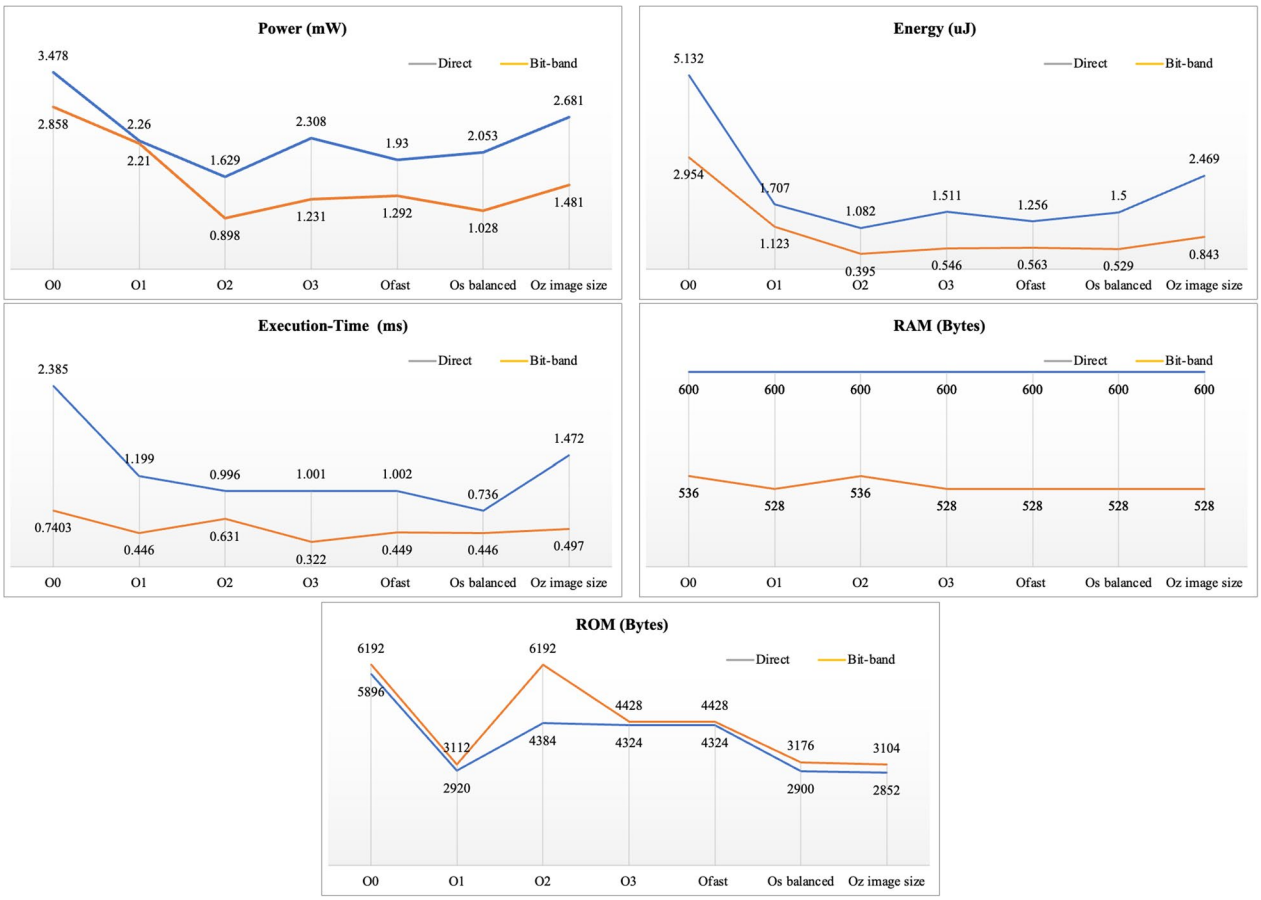

Fig. 7 Performance comparison between direct and proposed implementation techniques for the PRESENT block cipher (encryption) utilising different compiler optimisation levels

in memory requirements has been possible with the- Oz image level. Above 70% reduction in decryption time is made with –O2, -O3 and –Ofast optimization levels.

Table 7 presents the performance evaluation results in terms of various metrics for the GIFT block cipher, implemented using the direct and the proposed bit-band methods on the LPC1768 development board. It also enumerates the percentage difference in various lightweight metrics for the bit-band and direct implementation techniques.

The execution time is reported to show maximum improvement, with the bit-band method requiring 56.42% less time than the direct method. This is followed by a 14.76% and 4.25% reduction in energy and power requirements, respectively. Again all these improvements in the bit-band method are at the cost of a relatively higher memory footprint than the direct method. For the decryption part, a similar trend is followed with 1.11%, 11.28% and 10.28% improvements in power, energy and time performance metrics. However, the memory size is comparatively larger in the bit-band than in the direct method.

Figure 9 depicts the comparative encryption results for various lightweight metrics of the direct and bit-band implementations for the GIFT block cipher run with different optimisation levels (Table 5).

Remarkable improvements have been attained for all metrics of the bit-band method, with 90.38% and 98.57% reductions in power and energy consumption reported with the -O2 technique compared to the –O0 technique. In addition, the execution time has been reduced by 70.90% using high-speed –O3 and –Ofast techniques. Moreover, a more than 50% decrease in the memory footprint has been achieved using the most code efficient –Ozimage optimisation level. For the direct method, with the –O3 level, 84.39%, 97.73% and 89.89% reductions in power, energy and time utilisation have been reported in comparison with the –O0 results. Also, a 32.752% reduced memory size has been achieved using the –Oz image size level.

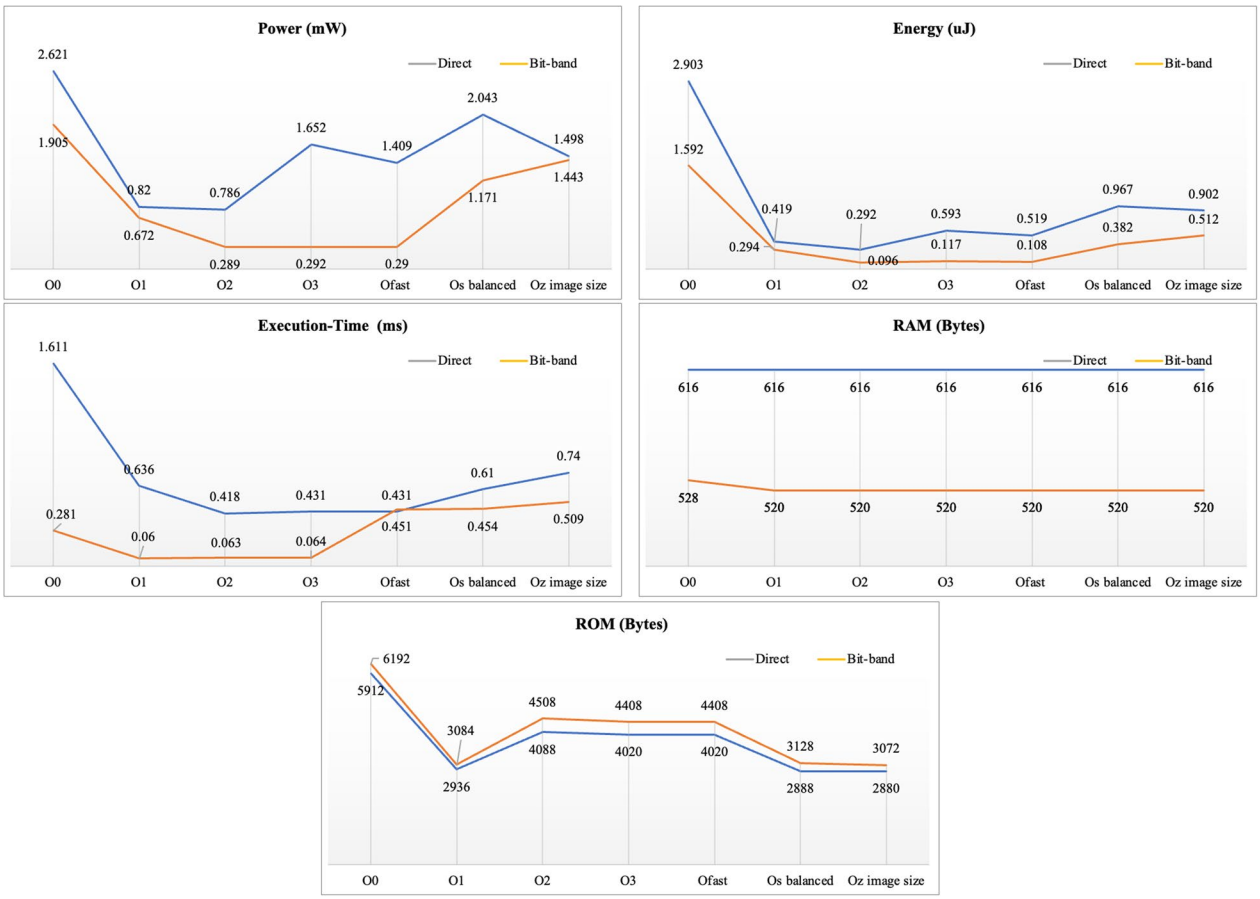

Fig. 8 Performance comparison between direct and proposed implementation techniques for the PRESENT block cipher (decryption) utilising different compiler optimisation levels

Table 7 Performance comparison of the proposed implementation technique with the direct method for the GIFT block cipher

Implementation Technique	Encryption			Decryption		
	Direct	Bit-Band	Bit-Band Vs Direct (%age difference)	Direct	Bit-Band	Bit-Band Vs Direct (%age difference)
Power (mW)	3.204	3.068	4.25	3.308	3.271	1.11
Energy (uJ)	9.119	7.773	14.76	10.016	8.887	11.28
RAM (Bytes)	512	1064	– 107.81	1408	1952	– 38.64
ROM (Bytes)	2064	7124	– 245.16	2188	2720	– 24.31
Execution-time (ms)	5.126	2.234	56.42	3.03	2.72	10.28

Figure 10 shows the direct and bit-band decryption results for the GIFT cipher using various optimisation levels (Table 5).

For the proposed method, the -O3 level reports a 60.62% and 93.1% decrease in power and energy requirements to the –O0 level. Also, a 24.85% reduction in memory size is possible with the –Oz image level. The –O0, -O3, -Ofast and –Oz image levels report almost the same decryption times. For the direct method, in comparison to the –O0 level, an 85.77% reduction in decryption time is attained using the high-speed –O3 and –Ofast optimisation levels. 34.19% reduction in memory footprint has been reported for the –Oz image level. 67.18% and 95.33% decrease in power consumption have been possible with the –Ofast optimisation level.

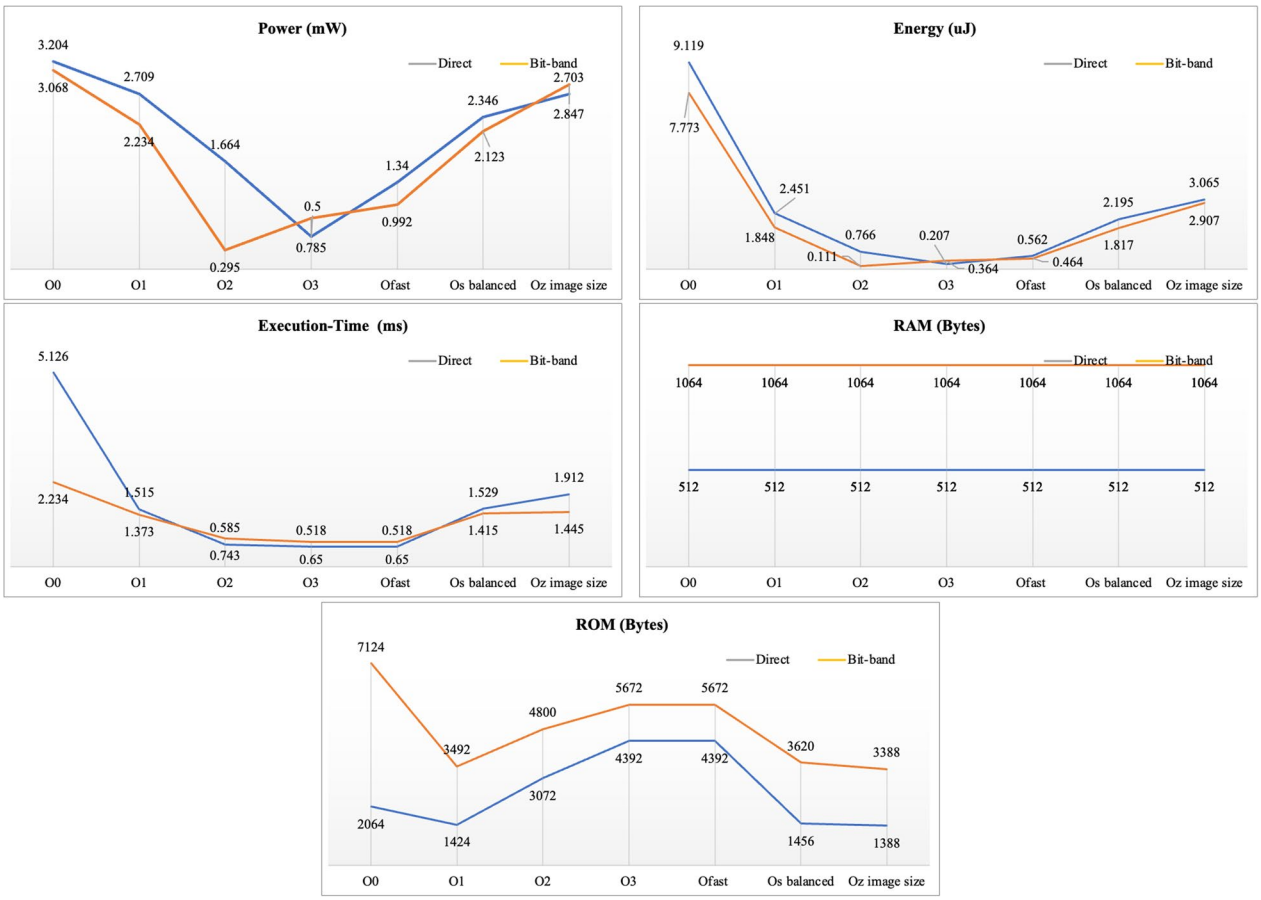

Fig. 9 Performance comparison between direct and proposed implementation techniques for the GIFT block cipher (encryption) utilising different compiler optimisation levels

From the results obtained, it can be inferred that the proposed bit-banding method for performing permutations in the PRESENT and GIFT SbPN ciphers is highly efficient in energy, power and execution time. In the direct method, each sub-operation involved in the bit-rotation method adds to the instruction count, increasing multiple instruction fetch, decode, execute and write-back operations. This is more apparent in lightweight SbPN ciphers with large block sizes and a larger number of rounds. For PRESENT and GIFT primitives with block size = 64, the input to the diffusion layer is large. Each bit transposition requires at least four sub-operations, viz., mask generation, AND or masking, shifting by a specified number of bits and XORing the diffused bit state with the original input state. This amounts to $4 \times 64 = 256$ such operations for realising a single round permutation. For the PRESENT cipher with 31 rounds involving permutation operation on the 64-bit state, $64 \times 4 \times 31$, such operations must be carried out by a low-end IoT device. Similarly, for the 28-round GIFT 64/128 cipher, $64 \times 4 \times 28$ sub-operations are required.

Contrary to this high instruction count and resource-exhaustive bit-rotation method, bit-banding is a software-efficient linear layer implementation technique. Moreover, this method does not involve using bit-manipulation instructions to perform the diffusion, offering a cost-saving option for low-end IoT processors. Instead, a simple mapping between the bit-band and its corresponding bit-band alias region is necessary to perform the bit-wise permutation, thus not only saving the chip space on processors but also leading to faster execution time and reduced power and energy consumption. Implementing the diffusion layer using the proposed technique not only reduces the instruction count, but also results in a significant decrease in all the lightweight design metrics namely power, energy and timing requirements, making the use of SbPN ciphers ideal for low-cost, low-power and low latency applications. Above all, since the proposed method is only an implementation strategy and does not modify the structure of the primitives, therefore,

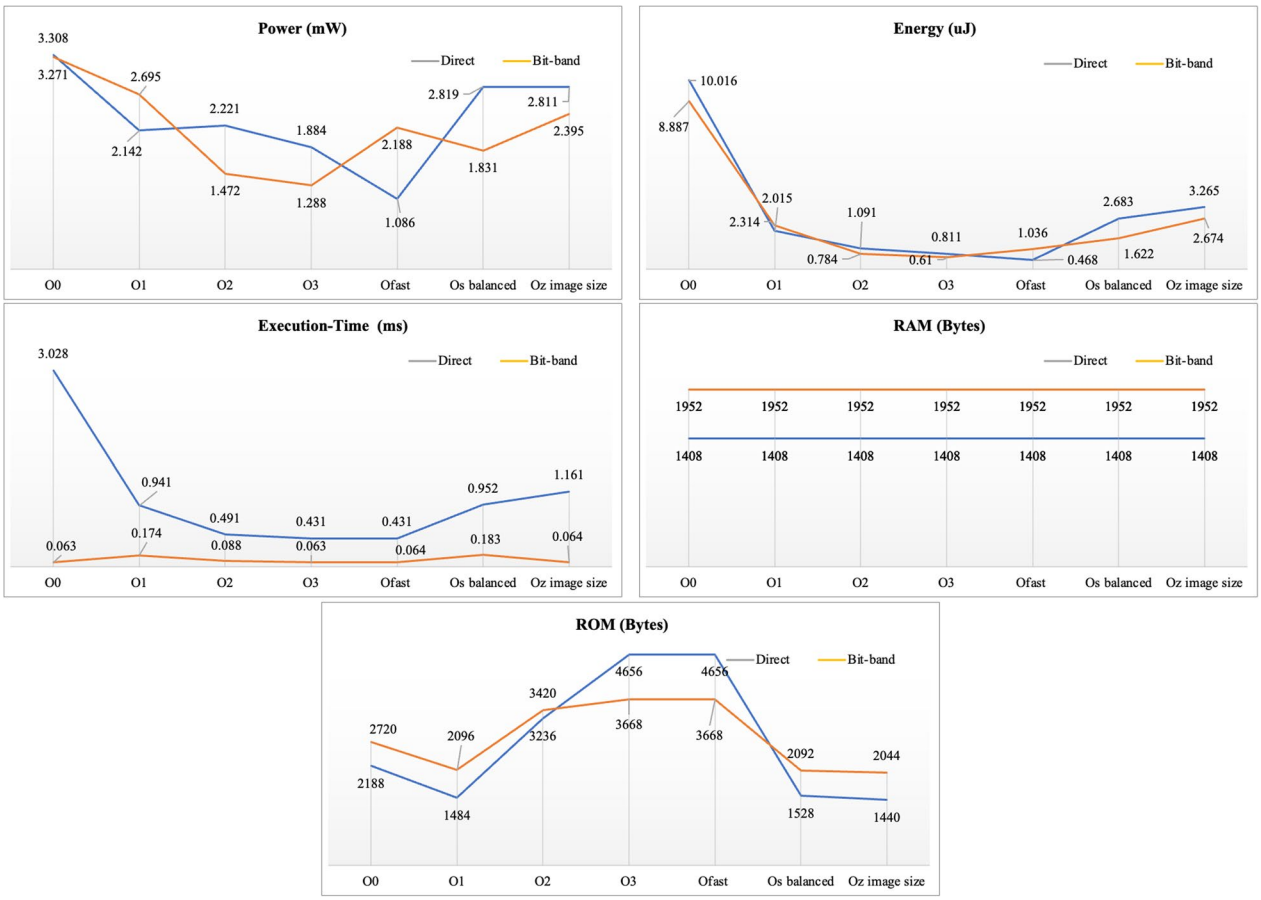

Fig. 10 Performance comparison between direct and proposed implementation techniques for the GIFT block cipher (decryption) utilising different compiler optimisation levels

it does not alter the security margins of any SbPN based primitives.

The fallout of the bit-banding method is the comparatively large memory requirements. Since each bit in the bit-band region corresponds to 32 bits in the bit-band alias region of SRAM. As such, the input and output state to the permutation layer of 64-bit width together occupies $2 \times 64 \times 4 = 512$ bytes in the bit-band alias memory. This makes bit-band permutation less memory efficient than the direct method; however, this memory requirement is much smaller than what is available with most IoT devices.

Conclusion

This paper presents a highly software-efficient method for performing bit-permutation-based diffusion using the bit-manipulation bit-banding technique with the leading edge ARM Cortex-M processors. A simple mapping between the bit-band and its corresponding bit-band alias region is necessary to perform the bit-wise permutation, thus saving the chip space on processors and leading to faster execution time with reduced power and energy consumption. Compared with the direct implementation methods for the PRESENT and GIFT ciphers, the bit-banding technique reports substantial reductions in power, energy and time requirements. All these improvements result from decreased instruction count and a fast mapping between the bit-band and bit-band alias regions. The only drawback of this method is an increase in the memory footprint, which is not much of a concern for ARM Cortex-M-based smart IoT devices. Furthermore, the proposed technique has been subjected to various compiler optimisation techniques available with the KEIL MDK IDE. The results have shown that with -O2 level, GIFT and PRESENT block ciphers significantly improved energy and power efficiency, whereas -O3 and -Ofast have sped up the cipher designs by a considerable mark. Moreover, high code efficiency is attained with '-Ozimage size' optimisation but at the cost of an increase in execution time. Although the proposed technique has been implemented to improve the

software efficiency of two SbPN primitives—PRESENT and GIFT, it is equally applicable for all such SbPN-based primitives.

Acknowledgements
The University Grants Commission, Government of India, supported the research work in the form of a Junior Research Fellowship (190520461818).

Authors' contributions
The authors read and approved the final manuscript.

Funding
The University Grants Commission, Government of India, supported the research work in the form of a Junior Research Fellowship (190520461818).

Availability of data and materials
The data supporting this study's findings are available from the corresponding author upon reasonable request.

Declarations

Competing interests
The authors declare that they do not have any conflict of interest.

Human and animal rights
This article does not contain any studies with human participants or animals performed by any authors.

Received: 28 March 2023 Accepted: 5 July 2023
Published online: 05 October 2023

References

Ashton K (2009) That 'Internet of Things' Thing. RFID J 22:97–114
Adomnicai A, Najm Z, Peyrin T (2020) Fixslicing: a new GIFT representation. IACR Trans Cryptogr Hardw Embed Syst 402:427. https://doi.org/10.46586/tches.v2020.i3.402-427
Adomnicai A, Peyrin T (2020) Fixslicing AES-like ciphers. IACR Trans Cryptogr Hardw Embed Syst:402–425
Avik Chakraborti MN, Datta N, Jha A (2019) HyENA, NIST lightweight cryptography project. https://csrc.nist.gov/Projects/Lightweight-Cryptography/Round-1-Candidates
Avik Chakraborti CML, Datta N, Jha A, Mancillas-LopezAvik Chakraborti C, Datta N, Jha A, Mridul Nandi YS (2020) ESTATE: a lightweight and low energy authenticated encryption mode. IACR Trans Symmetric Cryptol:350–389
Avik Chakraborti CML, Datta N, Jha A, Mridul Nandi YS (2019) LOTUS-AEAD and LOCUS-AEAD, Technical report, First-round submission to the NIST Lightweight Cryptography Competition
Bai Y (2015) Practical microcontroller engineering with ARM® technology. Wiley
Banday MT (2018) A study of current trends in the design of processors for the Internet of Things. ACM Int Conf Proc Ser. https://doi.org/10.1145/3231053.3231074
Banik S et al (2015) Midori : a block cipher for low energy (extended version). Int Conf Theory Appl Cryptol Inf Secur 9453:411–436
Banik S, Pandey SK, Peyrin T, Sasaki Y, Sim SM, Todo Y (2017) GIFT: a small present. In: Lecture notes in computer science (including subseries lecture notes in artificial intelligence and lecture notes in bioinformatics), vol 10529 LNCS, 2017, pp 321–345
Banik S et al (2019a) Gift-cofb v1.0. NIST lightweight cryptography project. pp 1–30. https://csrc.nist.gov/Projects/lightweight-cryptography/round-2-candidates
Banik S et al (2019b) Sundae-gift. Submiss. to Round 1, vol 1, pp 1–22
Beaulieu R, Shors D, Smith J, Treatman-Clark S, Weeks B, Wingers L (2015) Simon and speck: block ciphers for the internet of things. In: Proceedings of the 52nd annual design automation conference on—DAC '15, no. July. pp 1–6. http://dl.acm.org/citation.cfm?doid=2744769.2747946
Benadjila R, Guo J, Lomné V, Peyrin T (2014) Implementing lightweight block ciphers on x86 architectures. In: Lecture notes in computer science (including subseries lecture notes in artificial intelligence and lecture notes in bioinformatics), vol 8282 LNCS, pp 324–351
Biham E (1997) A fast new DES implementation in software. In: Lecture notes in computer science (including subseries lecture notes in artificial intelligence and lecture notes in bioinformatics), vol 1267, pp 260–272
Biham E, Anderson R, Knudsen L (1998) Serpent: a new block cipher proposal. In: Lecture notes in computer science (including subseries lecture notes in artificial intelligence and lecture notes in bioinformatics), vol 1372, pp 222–238
Biryukov A, Cannière C (2006) Data encryption standard (DES). Encycloped Cryptogr Secur 3:129–135
Bogdanov A, Knudsen LR, Leander G, Paar C, Poschmann A (2007) PRESENT : an ultra-lightweight block cipher. Cryptogr Hardw Embed Syst 2007:10–13
Bogdanov A, Knudsen LR, Leander G, Paar C, Poschmann A (2007) PRESENT : an ultra-lightweight block cipher. In: Proc. 9th international workshop on cryptographic hardware and embedded systems (CHES 2007), Vienna, Austria, pp 450–466
Borghoff J et al (2012) PRINCE: a low-latency block cipher for pervasive computing applications. Lect Not Comput Sci Inlude Subser Lect Not Artif Intell Lect Not Bioinform 7658(10):208–225. https://doi.org/10.1007/978-3-642-34961-4_14
Boyar J, Peralta R (2010) A new combinational logic minimization technique with applications to cryptology. In: Lecture notes in computer science (including subseries lecture notes in artificial intelligence and lecture notes in bioinformatics), vol 6049 LNCS, no 2, pp 178–189
Dinu D, Le Corre Y, Khovratovich D, Perrin L, Großschädl J, Biryukov A (2019) Triathlon of lightweight block ciphers for the Internet of things. J Cryptogr Eng 9(3):283–302. https://doi.org/10.1007/s13389-018-0193-x
Forouzan D, Behrouz A, Mukhopadhyay D (2015) Cryptography and network security. Mc Graw Hill Education (India) Private Limited New York, NY, USA
Heys HM (2020) A tutorial on the implementation of block ciphers: software and hardware applications. In: IACR Cryptol. ePrint Arch, p 1545. https://eprint.iacr.org/2020/1545
Karakoç F, Demirci H, Harmanci AE (2013) ITUbee: a software oriented lightweight block cipher. Lect Not Comput Sci 8162:16–27. https://doi.org/10.1007/978-3-642-40392-7_2
Kim H et al (2022) SPEEDY on Cortex–M3: efficient software implementation of SPEEDY on ARM Cortex–M3. Lect Notes Comput Sci 13218:434–444. https://doi.org/10.1007/978-3-031-08896-4_23
Kwan M (2000) Reducing the gate count of Bitslice DES. IACR Cryptol. ePrint Arch., vol. 2000, p 51. http://dblp.uni-trier.de/db/journals/iacr/iacr2000.html#Kwan00
Lee RB (1989) Precision architecture. Comput Long Beach Calif 22(1):78–91. https://doi.org/10.1109/2.19825
Matsui M, Nakajima J (2007) On the power of bitslice implementation on intel core2 processor. In: Cryptographic hardware and embedded systems: CHES 2007, vol. 4727 LNCS, Berlin, Heidelberg: Springer Berlin Heidelberg, pp 121–134
Nazish M, Banday MT (2018) Green Internet of Things: a study of technologies, challenges and applications. In: 2018 international conference on automation and computational engineering (ICACE), pp 210–215. https://doi.org/10.1109/ICACE.2018.8686976
Papapagiannopoulos K (2014) Radio Frequency identification: security and privacy issues, vol 8651. Springer, Cham
Poschmann A (2009) Lightweight cryptography: cryptographic engineering for a pervasive world. Ph. D. Thesis, no. February, pp 1–197. http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.182.1450
Reis TBS, Aranha DF, López J (2017) PRESENT runs fast: efficient and secure implementation in software. Lect Not Comput Sci 10529:644–664. https://doi.org/10.1007/978-3-319-66787-4_31
Rejeb A, Rejeb K, Simske S, Treiblmaier H, Zailani S (2022) The big picture on the internet of things and the smart city: a review of what we know and what we need to know. Internet Things 19:100565. https://doi.org/10.1016/j.iot.2022.100565
Rouf M, Nazish M, Sultan I, Banday MT (2022) Implementation of area and power optimised ARM cortex-M cores on FPGA. In: 2022 smart technologies, communication and robotics (STCR), pp 1–6. https://doi.org/10.1109/STCR55312.2022.10009282

Schwabe P, Stoffelen K (2017) All the AES you need on cortex-M3 and M4. In: Lecture Notes in computer science (including subseries lecture notes in artificial intelligence and lecture notes in bioinformatics), vol 10532 LNCS, pp 180–194
Shannon CE (1945) A mathematical theory of cryptography. https://www.iacr.org/museum/shannon45.html
Tahir M, Javed K (2017) ARM microprocessor systems cortex-M architecture, programming, and interfacing. CRC Press
Yiu J (2014) The definitive guide to ARM Cortex-M3 and cortex-M4 processors. 3rd edition. Newnes, Cambridge, pp 45–55, vol 4, no 1. Elsevier
Zhang W, Bao Z, Lin D, Rijmen V, Yang B, Verbauwhede I (2015) RECTANGLE: a bit-slice lightweight block cipher suitable for multiple platforms. Sci China Inf Sci. https://doi.org/10.1007/s11432-015-5459-7

Publisher's Note