

**DEVICES FOR COMPUTER-BASED GENERATING OF A MIXING MATRIX
FOR CRYPTOGRAPHIC PROCESSING OF DATA, ENCRYPTING DEVICES,
METHODS FOR COMPUTER-BASED GENERATING OF A MIXING
MATRIX FOR CRYPTOGRAPHIC PROCESSING OF DATA
AND ENCRYPTING METHODS**

Technical Field

[0001] Embodiments relate to devices for computer-based generating of a mixing matrix for cryptographic processing of data, encrypting devices, methods for computer-based generating of a mixing matrix for cryptographic processing of data and encrypting methods.

Background

[0002] A mixing layer in a cipher or any symmetric key cryptography primitive aims at providing diffusion. For example, diffusion is the property of an operation such that changing one bit (or byte) of the input will change many adjacent or near-by bits (or bytes) after the operation.

Summary

[0003] In various embodiments, a device for computer-based generating of a mixing matrix for cryptographic processing of data may be provided. The device may include: a square matrix determination circuit configured to determine a square matrix with non-

zero elements only in one diagonal or sub-diagonal and in one row or column; an exponentiation determination circuit configured to determine an exponentiation of the square matrix as a candidate for a mixing matrix; a criterion determination circuit configured to determine whether the candidate fulfils a pre-determined criterion; and a candidate determination circuit configured to determine the candidate as the mixing matrix if the candidate fulfils a pre-determined criterion.

[0004] In various embodiments an encrypting device may be provided. The encrypting device may include a mixing circuit configured to use as a mixing matrix a square matrix that can be written as an exponentiation of a square matrix with non-zero elements only in one diagonal or sub-diagonal and in one row or column.

[0005] In various embodiments an encrypting device may be provided. The encrypting device may include a mixing circuit configured to apply a mixing matrix by repeatedly applying a square matrix with non-zero elements only in one diagonal or sub-diagonal and in one row or column.

[0006] In various embodiments a method for computer-based generating of a mixing matrix for cryptographic processing of data may be provided. The method may include: determining a square matrix with non-zero elements only in one diagonal or sub-diagonal and in one row or column; determining an exponentiation of the square matrix as a candidate for a mixing matrix; determining whether the candidate fulfils a pre-determined criterion; and determining the candidate as the mixing matrix if the candidate fulfils a pre-determined criterion.

[0007] In various embodiments an encrypting method may be provided. The method may include using as a mixing matrix a square matrix that can be written as an

exponentiation of a square matrix with non-zero elements only in one diagonal or sub-diagonal and in one row or column.

[0008] In various embodiments an encrypting method may be provided. The method may include applying a mixing matrix by repeatedly applying a square matrix with non-zero elements only in one diagonal or sub-diagonal and in one row or column.

Brief Description of the Drawings

[0009] In the drawings, like reference characters generally refer to the same parts throughout the different views. The drawings are not necessarily to scale, emphasis instead generally being placed upon illustrating the principles of various embodiments. In the following description, various embodiments of the invention are described with reference to the following drawings, in which:

FIG. 1 shows a device for computer-based generating of a mixing matrix for cryptographic processing of data in accordance with various embodiments;

FIG. 2 shows an encrypting device in accordance with various embodiments;

FIG. 3 shows an encrypting device in accordance with various embodiments;

FIG. 4 shows a flow diagram illustrating a method for computer-based generating of a mixing matrix for cryptographic processing of data in accordance with various embodiments;

FIG. 5 shows a flow diagram illustrating an encrypting method in accordance with various embodiments;

FIG. 6 shows a flow diagram illustrating an encrypting method in accordance with various embodiments;

FIG. 7 shows a flow diagram illustrating a method for computer-based generating of a mixing matrix for cryptographic processing of data in accordance with various embodiments;

FIG. 8 shows a flow diagram illustrating a method for using a mixing matrix for cryptographic processing of data in accordance with various embodiments;

FIG. 9 shows a diagram illustrating the use of key arrays according to various embodiments;

FIG. 10 shows a flow diagram illustrating an overview of a single round of LED in accordance with various embodiments;

FIG. 11 shows a diagram illustrating a hardware architecture in accordance with various embodiments;

FIG. 12 shows a diagram illustrating a framework in accordance with various embodiments;

FIG. 13 shows a flow diagram illustrating one round of a PHOTON permutation according to various embodiments;

FIG. 14 shows a flow diagram illustrating a method according to various embodiments;

FIG. 15 shows a diagram illustrating hardware according to various embodiments; and

FIG. 16 shows a data transmission system according to various embodiments.

Description

[0010] The following detailed description refers to the accompanying drawings that show, by way of illustration, specific details and embodiments in which the invention may be practiced. These embodiments are described in sufficient detail to enable those skilled in the art to practice the invention. Other embodiments may be utilized and structural, logical, and electrical changes may be made without departing from the scope of the invention. The various embodiments are not necessarily mutually exclusive, as some embodiments may be combined with one or more other embodiments to form new embodiments.

[0011] The device for computer-based generating of a mixing matrix for cryptographic processing of data may include a memory which is for example used in the processing carried out by the device for computer-based generating of a mixing matrix for cryptographic processing of data. The encrypting device may include a memory which is for example used in the processing carried out by the encrypting device. A memory used in the embodiments may be a volatile memory, for example a DRAM (Dynamic Random Access Memory) or a non-volatile memory, for example a PROM (Programmable Read Only Memory), an EPROM (Erasable PROM), EEPROM (Electrically Erasable PROM), or a flash memory, e.g., a floating gate memory, a charge trapping memory, an MRAM (Magnetoresistive Random Access Memory) or a PCRAM (Phase Change Random Access Memory).

[0012] In an embodiment, a "circuit" may be understood as any kind of a logic implementing entity, which may be special purpose circuitry or a processor executing software stored in a memory, firmware, or any combination thereof. Thus, in an

embodiment, a "circuit" may be a hard-wired logic circuit or a programmable logic circuit such as a programmable processor, e.g. a microprocessor (e.g. a Complex Instruction Set Computer (CISC) processor or a Reduced Instruction Set Computer (RISC) processor). A "circuit" may also be a processor executing software, e.g. any kind of computer program, e.g. a computer program using a virtual machine code such as e.g. Java. Any other kind of implementation of the respective functions which will be described in more detail below may also be understood as a "circuit" in accordance with an alternative embodiment.

[0013] A device according to various embodiments may be an RFID (radio-frequency identification) tag. An RFID tag may be considered as a computer. A method according to various embodiments may a method carried out by an RFID tag. According to various embodiments, cryptography may be understood as one or more of the following: hashing, ciphering, encrypting, and the inverse processes where feasible.

[0014] The word "exemplary" is used herein to mean "serving as an example, instance, or illustration". Any embodiment or design described herein as "exemplary" is not necessarily to be construed as preferred or advantageous over other embodiments or designs.

[0015] Various embodiments are described for devices, and various embodiments are described for methods. It will be understood that a property described for a method may also hold for the respective device, and vice versa.

[0016] FIG. 1 shows a device 100 for computer-based generating of a mixing matrix for cryptographic processing of data in accordance with various embodiments. The device 100 may include a square matrix determination circuit 102 configured to

determine a square matrix with non-zero elements only in one diagonal or sub-diagonal and in one row or column. The device 100 may further include an exponentiation determination circuit 104 configured to determine an exponentiation of the square matrix as a candidate for a mixing matrix. The device 100 may further include a criterion determination circuit 106 configured to determine whether the candidate fulfils a pre-determined criterion. The device 100 may further include a candidate determination circuit 108 configured to determine the candidate as the mixing matrix if the candidate fulfils a pre-determined criterion. The square matrix determination circuit 102, the exponentiation determination circuit 104, the criterion determination circuit 106 and the candidate determination circuit 108 may be coupled by a coupling 110, for example by an electrical coupling or by an optical coupling, for example a cable or a bus.

[0017] According to various embodiments, the one diagonal or sub-diagonal may be the first upper sub-diagonal. According to various embodiments, a diagonal may be an anti-diagonal. According to various embodiments, a sub-diagonal may be an anti-sub-diagonal.

[0018] According to various embodiments, the one row or column may be the last row.

[0019] According to various embodiments, the exponentiation may be an exponentiation to the power of the number of rows of the square matrix.

[0020] According to various embodiments, the pre-determined criterion may be a criterion based on a maximum distance separable (MDS) code criterion.

[0021] According to various embodiments, the one diagonal or sub-diagonal may be filled with ones (in other words: all entries in the one diagonal or sub-diagonal may have the value “1”).

[0022] FIG. 2 shows an encrypting device 200 in accordance with various embodiments. The encrypting device 200 may include a mixing circuit 202 configured to use as a mixing matrix a square matrix that can be written as an exponentiation of a square matrix with non-zero elements only in one diagonal or sub-diagonal and in one row or column.

[0023] According to various embodiments, the one diagonal or sub-diagonal may be the first upper sub-diagonal. According to various embodiments, a diagonal may be an anti-diagonal. According to various embodiments, a sub-diagonal may be an anti-sub-diagonal.

[0024] According to various embodiments, the one row or column may be the last row.

[0025] According to various embodiments, the exponentiation may be an exponentiation to the power of the number of rows of the square matrix.

[0026] According to various embodiments, the exponentiation may fulfill a pre-determined criterion based on a maximum distance separable code criterion.

[0027] According to various embodiments, the one diagonal or sub-diagonal may be filled with ones.

[0028] FIG. 3 shows an encrypting device 300 in accordance with various embodiments. The encrypting device 300 may include a mixing circuit 302 configured to

apply a mixing matrix by repeatedly applying a square matrix with non-zero elements only in one diagonal or sub-diagonal and in one row or column.

[0029] According to various embodiments, the one diagonal or sub-diagonal may be the first upper sub-diagonal. According to various embodiments, a diagonal may be an anti-diagonal. According to various embodiments, a sub-diagonal may be an anti-sub-diagonal.

[0030] According to various embodiments, the one row or column may be the last row.

[0031] According to various embodiments, the mixing matrix circuit 302 may be configured to repeatedly apply the mixing matrix by a number of times equal to the power of the number of rows of the square matrix.

[0032] According to various embodiments, the mixing matrix may fulfill a pre-determined criterion based on a maximum distance separable code criterion.

[0033] According to various embodiments, the one diagonal or sub-diagonal may be filled with ones.

[0034] FIG. 4 shows a flow diagram 400 illustrating a method for computer-based generating of a mixing matrix for cryptographic processing of data in accordance with various embodiments. In 402, a square matrix with non-zero elements only in one diagonal or sub-diagonal and in one row or column may be determined. In 404, an exponentiation of the square matrix as a candidate for a mixing matrix may be determined. In 406, it may be determined whether the candidate fulfils a pre-determined criterion. In 408, the candidate may be determined as the mixing matrix if the candidate fulfils a pre-determined criterion.

[0035] According to various embodiments, the one diagonal or sub-diagonal may be the first upper sub-diagonal. According to various embodiments, a diagonal may be an anti-diagonal. According to various embodiments, a sub-diagonal may be an anti-sub-diagonal.

[0036] According to various embodiments, the one row or column may be the last row.

[0037] According to various embodiments, the exponentiation may be an exponentiation to the power of the number of rows of the square matrix.

[0038] According to various embodiments, the pre-determined criterion may be a criterion based on a maximum distance separable code criterion.

[0039] According to various embodiments, the one diagonal or sub-diagonal may be filled with ones.

[0040] FIG. 5 shows a flow diagram 500 illustrating an encrypting method in accordance with various embodiments. In 502, a square matrix that can be written as an exponentiation of a square matrix with non-zero elements only in one diagonal or sub-diagonal and in one row or column may be used as a mixing matrix.

[0041] According to various embodiments, the one diagonal or sub-diagonal may be the first upper sub-diagonal. According to various embodiments, a diagonal may be an anti-diagonal. According to various embodiments, a sub-diagonal may be an anti-sub-diagonal.

[0042] According to various embodiments, the one row or column may be the last row.

[0043] According to various embodiments, the exponentiation may be an exponentiation to the power of the number of rows of the square matrix.

[0044] According to various embodiments, the exponentiation may fulfill a pre-determined criterion based on a maximum distance separable code criterion.

[0045] According to various embodiments, the one diagonal or sub-diagonal may be filled with ones.

[0046] FIG. 6 shows a flow diagram 600 illustrating an encrypting method in accordance with various embodiments. In 602, a mixing matrix may be applied by repeatedly applying a square matrix with non-zero elements only in one diagonal or sub-diagonal and in one row or column.

[0047] According to various embodiments, the one diagonal or sub-diagonal may be the first upper sub-diagonal. According to various embodiments, a diagonal may be an anti-diagonal. According to various embodiments, a sub-diagonal may be an anti-sub-diagonal.

[0048] According to various embodiments, the one row or column may be the last row.

[0049] According to various embodiments, the mixing matrix may be repeatedly applied by a number of times equal to the power of the number of rows of the square matrix.

[0050] According to various embodiments, the mixing matrix may fulfill a pre-determined criterion based on a maximum distance separable code criterion.

[0051] According to various embodiments, the one diagonal or sub-diagonal may be filled with ones.

[0052] According to various embodiments, a way of building mixing matrices in cryptography may be provided. A mixing layer in a cipher or any symmetric key cryptography primitive may aim at providing diffusion. For example, diffusion may be the property of an operation such that changing one bit (or byte) of the input will change many adjacent or near-by bits (or bytes) after the operation. This may represent one of the vital components of any symmetric key cryptography primitive.

[0053] A commonly used method for implementing diffusion in cryptography is to multiply the input vector by a mixing matrix. The matrix must therefore have good diffusion capabilities which is often achieved with the "maximum distance separable" (MDS) property. According to various embodiments, it is looked for $(d \times d)$ MDS mixing matrices, but the design techniques may be easily applied to non-square matrices.

[0054] In the following, maximum distance separable (MDS) matrices will be described.

[0055] The "maximum distance separable" property may be explained with coding theory. Let F be a field, then we denote F^n as the set of n -tuples of elements of F . Naturally, F^n may form a vector space over F . We define the hamming weight of a vector $Hw(u)$, $u \in F^n$ as the number of non-zero elements of u . The definition of hamming distance between two vectors u and v is $Hw(u, v) = Hw(u - v)$. The hamming distance function $\omega_d : F^n \rightarrow N$ forms a metric on F^n .

[0056] A linear $[n, k, d]$ code C over F is a k -dimensional subspace of F^n , where the hamming distance between any two distinct vectors is at least d . All linear codes obey the Singleton bound, $d \leq n - k + 1$. If the upper bound is met, $d = n - k + 1$, then the code may be said to be "maximum distance separable" or MDS.

[0057] For a given $[n, k, d]$ linear code C , a generator matrix in the form $G_{(k,n)}$ may be a matrix with rows which form a basis for the code; or in other words, the span of G forms the vectorspace C . The generator matrix is in "reduced echelon form" if it can be written as a partition matrix $G = [I_{(k,k)} | M_{(k,n-k)}]$ where, I is the identity matrix. The arbitrary matrix M is termed "maximal distance separable" iff (in other words: if and only if) every square submatrix of M is non-singular. Or in other terms the matrix M is MDS iff every square submatrix has a non-zero determinant.

[0058] In the following, MDS matrices in cryptography will be described.

[0059] While ways of building MDS matrices are already known, many other constraints may be desired to be included when considering implementations in the real world. Indeed, in many practical scenarios, the coefficients of the mixing matrices will have significant impact on the speed or the hardware area required. "Circulant matrices" may be used as a way to implement MDS matrices with good performances (by performances high speed and/or small area may be meant). More precisely, a circulant matrix is a matrix where each row vector is rotated one element to the right relative to the preceding row vector. Formally, a $(d \times d)$ circulant matrix spanned by vector $(C_1, \dots, C_d)$ takes the form

$$\begin{bmatrix} C_1 & C_d & C_{d-1} & \dots & C_4 & C_3 & C_2 \\ C_2 & C_1 & C_d & \dots & C_5 & C_4 & C_3 \\ \dots & \dots & \dots & \dots & \dots & \dots & \dots \\ C_{d-2} & C_{d-3} & C_{d-4} & \dots & C_1 & C_d & C_{d-1} \\ C_{d-1} & C_{d-2} & C_{d-3} & \dots & C_2 & C_1 & C_d \\ C_d & C_{d-1} & C_{d-2} & \dots & C_3 & C_2 & C_1 \end{bmatrix}.$$

[0060] For example, the Advanced Encryption Standard (AES) uses a circulant matrix with vector (3 1 1 2), that is

[3 2 1 1]

[1 3 2 1]

[1 1 3 2]

[2 1 1 3],

where the coefficients (3 1 1 2) have been chosen to provide good performance.

[0061] In the following, MDS matrices construction in accordance with various embodiments will be described.

[0062] According to various embodiments, a method for computing MDS matrix may be provided. A mixing matrix according to various embodiments may be a submatrix (called "start matrix") raised to the power d . More precisely, it may be started from a $(d \times d)$ matrix A and the final mixing matrix will be $M = A^d$. The start matrix A may be chosen to have the exemplary form

$$\begin{bmatrix} 0 & C_1 & 0 & \dots & 0 & 0 & 0 \\ 0 & 0 & C_2 & \dots & 0 & 0 & 0 \\ & & & \dots & & & \\ 0 & 0 & 0 & \dots & C_{d-2} & 0 & 0 \\ 0 & 0 & 0 & \dots & 0 & 0 & C_{d-1} \\ Z_1 & Z_2 & Z_3 & \dots & Z_{d-2} & Z_{d-1} & Z_d \end{bmatrix}$$

where the values $(C_1, \dots, C_{d-1})$ and $(Z_1, \dots, Z_d)$ are coefficients that may be chosen such that the final mixing matrix M is indeed MDS. Moreover the coefficients may be desired

to provide good performance as much as possible (for example, their hamming weight may be desired to be as small as possible).

[0063] This exemplary form of A may allow to compute A (and therefore $M = A^d$) on the input vector element per element, in a serial way, with very good performance. Being able to compute the mixing matrix in a serial way may reduce the area required for the computation and therefore may improve the performances in the case of memory-constraint devices. Indeed, the subresult memorized during the computation may have only the size of one element of the matrix (whereas for non-serial computation, one may require to maintain a subresult of d-1 elements).

[0064] In commonly used MDS matrix building techniques, the serial decomposition of the mixing matrix may lead to very bad coefficients, therefore implying bad performances. With the method according to various embodiments, it may directly be ensured that the coefficients for the serial decomposition of the mixing matrix are good.

[0065] In the following, an example with AES parameters according to various embodiments will be described.

[0066] As an example, with the same parameters as for the AES (i.e. a 4*4 matrix in the $GF(2^8)$ field with the same irreducible polynomial), the start matrix A may be used:

[0 1 0 0]

[0 0 1 0]

[0 0 0 1]

[1 2 1 4]

which may give the final MDS mixing matrix $M = A^4$:

[1 2 1 4]

[4 9 6 17]

[17 38 24 66]

[66 149 100 11]

[0067] This may imply that the smallest implementation of the AES could be made even smaller if this new mixing matrix was used instead of the AES one (while they may have the same diffusion quality since both are MDS).

[0068] In the following, commercial applications will be described.

[0069] The MDS mixing matrix building method according to various embodiments has commercial applications in constraint-environment cryptography, such as RFID. Indeed, such devices may only spend a very limited amount of memory dedicated to security and cryptography. Therefore, any method that may allow to save some hardware area (and thus the power consumption) may be crucial and may be highly sought after by the industry. The method according to various embodiments may improve the hardware area for many symmetric key cryptography primitives, while maintaining the same diffusion properties as the previously known MDS matrix building techniques.

[0070] FIG. 7 shows a flow diagram 700 illustrating a method for computer-based generating of a mixing matrix for cryptographic processing of data in accordance with various embodiments. In 702, a matrix A may be filled with zeros. In 704, new low hamming weight coefficients may be tried on only one line and one diagonal of the matrix A. In 706, it may be tested, if A^n is MDS. In 708, it may be checked whether the matrix A^n is MDS. If no, it may be continued with 702. If yes, it may be continued with 710. In 710, the matrix A may be outputted.

[0071] FIG. 8 shows a flow diagram 800 illustrating a method for using a mixing matrix for cryptographic processing of data in accordance with various embodiments. In 802, the vector $a(i)$ may be initialized with the input $in(i)$, in other words: $a(i) = in(i)$ for all i . In 804, x may be computed as $x = Z_1 a(1) + \dots + Z_n a(n)$. In 806 $a(i-1)$ may be computed and set by $a(i-1) = C_i a(i)$. In 808, it may be checked that the step 806 is repeated for $i=1$ to n . In 810, $a(n)$ may be set by $a(n) = x$. In 812, it may be checked that the steps 804 to 810 are repeated n times. In 814, the output may be computed by $out(i) = a(i)$. n may be the number of columns (and the number of rows) of the starting matrix.

[0072] In the following, devices and methods according to various embodiments, which may be referred to as the “LED Block Cipher”, will be described.

[0073] According to various embodiments, a block cipher LED may be provided. While dedicated to compact hardware implementation, and offering the smallest silicon footprint among comparable block ciphers, the cipher according to various embodiments is designed to simultaneously tackle three additional, and independent, goals. As will be described below, the role of an ultra-light (in fact non-existent) key schedule is explored. Furthermore, the resistance of ciphers, and LED for example, to related-key attacks is considered: according to various embodiments, very simple yet interesting AES-like security proofs for LED regarding related-key or single-key attacks may be provided. Furthermore, while providing a block cipher that is very compact in hardware, a reasonable performance profile for software implementation may be maintained.

[0074] Over past years many new cryptographic primitives have been proposed for use in RFID tag deployments, sensor networks, and other applications for highly-

constrained devices. The pervasive deployment of tiny computational devices brings with it many interesting, and potentially difficult, security issues.

[0075] Chief among recent developments has been the evolution of lightweight block ciphers where an accumulation of advances in method design, together with an increased awareness of the likely application, has helped provide important developments. To some commentators the need for yet another lightweight block cipher proposal will be open to question. However, in addition to the fact that many proposals were recently shown to present some weaknesses, there may be still more to be said on the subject and it may be observed that it is in the “second generation” of work that designers might learn from the progress, and omissions, of “first generation” proposals. And while new proposals might only slightly improve on successful initial proposals in terms of a single metric, e.g. area, they might, at the same time, overcome other important security and performance limitations. According to various embodiments, it may be returned to the design of lightweight block ciphers and Light Encryption Device, LED, may be described. During the design according to various embodiments, several key observations may have been upper-most in mind.

[0076] Practically all modern block cipher proposals have reasonable security arguments; but few offer much beyond (potentially thorough) ad hoc analysis. Here it may be hoped to provide a more complete security treatment than is usual. For example, related-key attacks are often dismissed from consideration for the application areas that may use such constrained devices, e.g. RFID tags. In practice this is perfectly reasonable. However, researchers may continue to derive cryptanalytic results in the related-key model and there has been some research on how to modify or strengthen existing key

schedules. Having provable levels of resistance to such attacks would be a bonus and it would help avoid confusion developing in the cryptographic literature.

[0077] In addition, attention according to various embodiments may be focused on the performance of the method on the tag. However, there may be constraints when a method is also going to be implemented in software. This is something that has already been discussed with the design of KLEIN, and in the design of LED according to various embodiments, it may be aimed at very compact hardware implementation while maintaining some software-friendly features.

[0078] The block cipher according to various embodiments may be based on AES-like design principles and this may allow to derive very simple bounds on the number of active Sboxes during a block cipher encryption. Since the key schedule is very simple, this analysis may be done in a related-key model as well; i.e. the bounds may apply even when an attacker tries to mount a related-key attack. And while AES-based approaches are well-suited to software, they may not always provide the lightest implementation in hardware. But using commonly known techniques, it may be aimed to resolve this conflict. While block ciphers may be an important primitive, and arguably the most useful in a constrained environment, there has also been much progress in the design of stream ciphers and even in lightweight hash functions.

[0079] In the following, a design approach and specifications according to various embodiments will be described.

[0080] According to various embodiments, an AES-like design may be chosen as a starting point for a clean and secure design. The design of LED may inevitably have many parallels with this established approach, and features such as Sboxes,

ShiftRows, and (a variant of) MixColumns will all feature and take their familiar roles.

[0081] According to various embodiments, for the key schedule it may be chosen to do-away with the "schedule", i.e. the user-provided key may be used repeatedly as is. As well as giving obvious advantages in hardware implementation, it may allow for simple proofs to be made for the security of the scheme even in the most challenging attack model of related keys. At first sight the re-use of the encryption key without variation may appear dangerous, certainly to those familiar with slide attacks and some of their advanced variants. But it is to be noted that such a simple key schedule may not be without precedent though the treatment here may be more complete than previously.

[0082] The LED cipher according to various embodiments will be described below. It may be a 64-bit block cipher with two primary instances taking 64 and 128 bits keys. The cipher state may be conceptually arranged in a (4x4) grid where each nibble represents an element from $GF(2^4)$ with the underlying polynomial for field multiplication given by $X^4 + X + 1$.

[0083] In the following, Sboxes according to various embodiments will be described. LED cipher may re-use the present Sbox which has been widely adopted in many lightweight cryptographic methods. The action of this box in hexadecimal notation may be given by the following table

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

[0084] In the following, MixColumnsSerial according to various embodiments will be described. A commonly used tactic may be re-used to define an MDS matrix for linear diffusion that is suitable for compact serial implementation. The MixColumnsSerial layer according to various embodiments may be viewed as four applications of a hardware-friendly matrix A with the net result being equivalent to using the MDS matrix M where

$$(A)^4 = \begin{pmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 4 & 1 & 2 & 2 \end{pmatrix}^4 = \begin{pmatrix} 4 & 2 & 1 & 1 \\ 8 & 6 & 5 & 6 \\ B & E & A & 9 \\ 2 & 2 & F & B \end{pmatrix} = M.$$

[0085] According to various embodiments, the basic component of LED may be a sequence of four identical rounds used without the addition of any key material. This basic unit, that may also be referred to as "step", may make it easy to establish security bounds for the construction.

[0086] In the following, a specification of LED according to various embodiments will be described.

[0087] For a 64-bit plaintext m the 16 four-bit nibbles $m_0 || m_1 || \dots || m_{14} || m_{15}$ may be arranged (conceptually) in a square array:

$$\begin{bmatrix} m_0 & m_1 & m_2 & m_3 \\ m_4 & m_5 & m_6 & m_7 \\ m_8 & m_9 & m_{10} & m_{11} \\ m_{12} & m_{13} & m_{14} & m_{15} \end{bmatrix}$$

[0088] This may be the initial value of the cipher STATE and it is to be noted that the state (and the key) are loaded row-wise rather than in the column-wise fashion we have come to expect from the AES, because this is a hardware friendly choice.

[0089] The key may be viewed nibble-wise and loaded nibble-by-nibble into one or two arrays, K_1 and K_2 , depending on the key length. The primary definition according to various embodiments may be for 64- or 128-bit keys, but shorter key lengths, e.g. the popular choice of 80 bits, may be padded to give a 128-bit key thereby giving a 128-bit key array. By virtue of the order of loading the tables, any key that is padded (with zeros) to give a 64- or 128-bit key array will effectively set unused nibbles of the key array to 0.

$$\begin{bmatrix} k_0 & k_1 & k_2 & k_3 \\ k_4 & k_5 & k_6 & k_7 \\ k_8 & k_9 & k_{10} & k_{11} \\ k_{12} & k_{13} & k_{14} & k_{15} \end{bmatrix} \quad \text{for 64-bit keys giving } K_1$$

$$\begin{bmatrix} k_0 & k_1 & k_2 & k_3 \\ k_4 & k_5 & k_6 & k_7 \\ k_8 & k_9 & k_{10} & k_{11} \\ k_{12} & k_{13} & k_{14} & k_{15} \end{bmatrix} \begin{bmatrix} k_{16} & k_{17} & k_{18} & k_{19} \\ k_{20} & k_{21} & k_{22} & k_{23} \\ k_{24} & k_{25} & k_{26} & k_{27} \\ k_{28} & k_{29} & k_{30} & k_{31} \end{bmatrix} \quad \text{for 128-bit keys giving } K_1 || K_2$$

[0090] According to various embodiments, the operation

$\text{addRoundKey}(\text{STATE}, K_i)$ may combine nibbles of subkey K_i with the state, respecting array positioning, using bitwise exclusive-or. There may be no key schedule, or rather this may be the sum total of the key schedule, and the arrays K_1 and, where appropriate, K_2 may repeatedly be used without modification. Encryption may be

described using the previously mentioned `addRoundKey (STATE, Ki)` and a second operation, `step (STATE)`. This is illustrated in FIG. 9.

[0091] FIG. 9 shows a diagram 900 illustrating the use of key arrays according to various embodiments. The use of key arrays K_1 and K_2 in LED showing both a 64-bit key array (in 902) and a 128-bit key array (in 904) is illustrated.

[0092] According to various embodiments, the number of steps during encryption may be determined by whether there are one or two key arrays:

```

for i = 1 to 8 do {
    addRoundKey (STATE, K1)
    step (STATE)
}
addRoundKey (STATE, K1)

```

for 64-bit key arrays

```

for i = 1 to 6 do {
    addRoundKey (STATE, K1)
    step (STATE)
    addRoundKey (STATE, K2)
    step (STATE)
}
addRoundKey (STATE, K1)

```

for 128-bit key arrays

[0093] According to various embodiments, the operation `step (STATE)` may include four rounds of encryption of the cipher state. Each of these four rounds may use the operations `AddConstants`, `SubCells`, `ShiftRows`, and `MixColumnsSerial` in sequence as illustrated in FIG. 10.

[0094] FIG. 10 shows a flow diagram 1000 illustrating an overview of a single round of LED in accordance with various embodiments. In 1002, `AddConstants` is illustrated. In 1004, `SubCells` is illustrated. In 1006, `ShiftRows` is illustrated. In 1008, `MixColumnsSerial` is illustrated.

[0095] In the following, `AddConstants` according to various embodiments will be described. A round constant may be defined as follows. At each round, the six bits (rc_5 , rc_4 , rc_3 , rc_2 , rc_1 , rc_0) may be shifted one position to the left with the new value to rc_0 being computed as $rc_5 \oplus rc_4 \oplus 1$. The six bits may be initialized to zero, and updated before use in a given round. The constant, when used in a given round, may be arranged into an array as follows:

$$\begin{bmatrix} 0 & (rc_5 || rc_4 || rc_3) & 0 & 0 \\ 1 & (rc_2 || rc_1 || rc_0) & 0 & 0 \\ 2 & (rc_5 || rc_4 || rc_3) & 0 & 0 \\ 3 & (rc_2 || rc_1 || rc_0) & 0 & 0 \end{bmatrix}$$

[0096] According to various embodiments, the round constants may be combined with the state, respecting array positioning, using bitwise exclusive-or. The values of the (rc_5 ; rc_4 ; rc_3 ; rc_2 ; rc_1 ; rc_0) constants for each round are given further below.

[0097] In the following, `SubCells` according to various embodiments will be described. Each nibble in the array state may be replaced by the nibble generated after using the present Sbox.

[0098] In the following, `ShiftRow` according to various embodiments will be described. Row i of the array state may be rotated i cell positions to the left, for $i = 0; 1; 2; 3$.

[0099] In the following, `MixColumnsSerial` according to various embodiments will be described. Each column of the array state may be viewed as a column vector and

replaced by the column vector that results after post-multiplying the vector by the matrix M (see description above).

[00100] According to various embodiments, the final value of the state may provide the ciphertext with nibbles of the "array" being unpacked in the obvious way. Test vectors for LED are provided further below.

[00101] In the following, security analysis according to various embodiments is described.

[00102] According to various embodiments, the LED block cipher may be simple to analyze and this may allow to precisely evaluate the necessary number of rounds to ensure proper security.

[00103] The scheme according to various embodiments is meant to be resistant to classical attacks, but also to the type of related-key attacks that have been effective against AES-256 and other ciphers. Furthermore, the security of LED will be described in a hash function setting, i.e. when it is used in a Davies-Meyer construction or some other block-cipher-based compression function construction. In other words, attackers will be considered that have full access to the key(s) and try to distinguish the fixed permutations from randomly chosen ones.

[00104] According to various embodiments, a conservative number of rounds for LED may be chosen. For example, when using a 64-bit key array 32 AES-like rounds that are grouped as eight "big" add-key/apply-permutation steps that are each composed of four AES-like rounds, may be used. Further, security margins may be even more conservative if one definitively disregards related-key attacks; as will be seen with the following proofs.

[00105] In the following, the key schedule according to various embodiments will be described.

[00106] The LED key schedule according to various embodiments may have been chosen for its simplicity and security. Because it is very simple to analyze, it may allow to directly derive a bound on the minimal number of active Sboxes, even in the scenario of related-key attacks. The idea may be to first compute a bound on the number of active big steps (each composed of 4 AES-like rounds). Then, using the commonly used 4-round proofs for the AES, one can show that one active big step will contain at least 25 active Sboxes. It is to be noted that this bound is tight as 4-round differential paths containing exactly this number of active Sboxes are known.

[00107] When not considering related-key attacks, it may directly be obtained that any differential path for LED may contain at least $\lfloor r/4 \rfloor \cdot 25$ active Sboxes. For related-key attacks, it may be desired to distinguish between the different key-size versions.

[00108] In the following, a 64-bit key version according to various embodiments will be described. If it is assumed that differences are inserted in the key input, then every subkey K_1 in the 64-bit key variant of LED may be active. Therefore, it may be easily seen that it may be impossible to force two consecutive non active big steps and thus it may be ensured that for every two big steps at least one is active. Overall, this shows that any related-key differential path contains at least $\lfloor r/8 \rfloor \cdot 25$ active Sboxes.

[00109] In the following, a 128-bit key version according to various embodiments will be described. If it is assumed that differences are inserted in the key input, then it may be desired to separate two cases. If the two independent parts K_1 and K_2 composing the key

both contain a difference, then it may be ended up with exactly the same reasoning as for the 64-bit key variant: at least $\lfloor r/8 \rfloor \bullet 25$ active Sboxes may be active. If only one of the two independent parts composing the key contains a difference, then subkeys with and without differences may alternatively be incorporated after each big step. The non-active subkeys impact on the differential paths may completely be void and thus in this case, LED may be viewed as being composed of even bigger steps of 8 AES-like rounds instead. Then, the very same reasoning may apply again: it may be impossible to force two consecutive of these new bigger steps to be inactive and therefore we have at least $\lfloor r/16 \rfloor \bullet 50$ active Sboxes ensured for any differential path (since the best differential path for 8 rounds trivially may include 50 active Sboxes).

[00110] In Table 1, the results obtained for the two main versions of LED are summarized, both for single-key attacks and related-key attacks. It is to be noted that the bounds on the number of active Sboxes are tight as differential paths meeting them are known (for example the truncated differential path for each active big step can simply be any of the 4-round path for AES-128 with 25 active Sboxes).

[00111] For LED-128, since two independent key parts are used, the first and last key addition (which may always the first key part K_1) may be peeled off. Thus, an attacker may remove one big step on each side of the cipher, for a total of 8 rounds, with a complexity of 264 tries on K_1 . This may partially explain why the versions of LED using two independent key parts have 16 more rounds than for LED-64.

	LED-64		LED-128	
	SK	RK	SK	RK
minimal number of active Sboxes	200	100	300	150
differential path probability	2^{-400}	2^{-200}	2^{-600}	2^{-300}
differential probability	2^{-256}	2^{-128}	2^{-384}	2^{-192}
linear approximation probability	2^{-400}	2^{-200}	2^{-600}	2^{-300}
linear hull probability	2^{-256}	2^{-128}	2^{-384}	2^{-192}

Table 1. Minimal number of active Sboxes, upper bounds on the best differential path probability, best differential probability, best linear approximation probability and best linear hull probability for the 64-bit key array and 128-bit key array versions of LED (in both the single-key (SK) and related-key (RK) settings).

[00112] It may be interesting to compare the key schedule according to various embodiments to a WHIRLPOOL-like key schedule, even if it has recently been shown to be flawed. For WHIRLPOOL, the subkey derivation may be composed of the same round function as for the internal block cipher. Therefore, in order to get an equivalent speed with LED for comparison purposes, it would be desired to reduce the number of rounds of the WHIRLPOOL-based proposal by a factor of two. For example, for a 64-bit key, a WHIRLPOOL-based scheme could be derived with 16 rounds in both the key schedule and the internal cipher (instead of 32 rounds for 64-bit key LED). This new cipher may however provide worse proofs than LED: the number of provable active Sboxes for related-key attacks may stay the same (100 Sboxes), but in the case of single-key attacks the number may be divided by a factor 2 compared to LED-64 (100 Sboxes). This profile may seem moreover less interesting as single-key attacks may have much more practical

impact than related-key threats. In the case of LED, the security margin against single-key attacks may be conservative. This very same reasoning may also apply when comparing to a commonly used key schedule.

[00113] In the following, differential/linear cryptanalysis according to various embodiments will be described.

[00114] According to various embodiments, since LED is an AES-like cipher, the extensive work done in the past years on AES may be re-used. A bound on the best differential path probability (where all differences on the input and output of all rounds may be specified) or even the best differential probability (where only the input and output differences may be specified), may be computed for both single-key or related-key settings.

[00115] As the best differential transition probability of the PRESENT Sbox is 2^{-2} , using the previously proven minimal number of active Sboxes it may be deduced that the best differential path probability on 4 active rounds of LED is upper bounded by $2^{-2.25} = 2^{-50}$. By adapting commonly known work, we may also be shown that the maximum differential probability for 4 active rounds of LED according to various embodiments is upper bounded by

$$\max \left\{ \max_{1 \leq u \leq 15} \sum_{j=1}^{15} \{DP^S(u, j)\}^5, \max_{1 \leq u \leq 15} \sum_{j=1}^{15} \{DP^S(j, u)\}^5 \right\}^4 = 2^{-32}$$

where $DP^S(i; j)$ may stand for the differential probability of the Sbox to map the difference i to j . The duality between linear and differential attacks may allow to apply

the same approaches to compute a bound on the best linear approximation or even the best linear hulls.

[00116] Since it has previously been proven that all rounds may be active in the single-key scenario and half of them may be active in the related-key scenario, the upper bounds on the best differential path probability, the best differential probability, the best linear approximation probability and the best linear hull probability for each version of LED according to various embodiments may easily be proven. This is summarized in Table 1.

[00117] In the following, cube testers and algebraic attacks will be described.

[00118] According to various embodiments, the most recent developed cube testers and its zero-sum distinguishers have been applied to the LED fixed-key permutation according to various embodiments; the best that may be found within practical time complexity is at most 3 rounds. It is to be noted that, in case of AES, "zero-sum" property may also be referred as "balanced", found by the AES designers, in which 3-round balanced property may be shown. Up to now, there is no balanced property found for more than 3 AES rounds.

[00119] According to various embodiments, the PRESENT Sbox used in LED may have algebraic degree 3 and it may be checked that $3 \cdot \lfloor r/4 \rfloor \cdot 25 \ggg 64$ for all LED variants according to various embodiments. Moreover, the PRESENT Sbox may be described by $e = 21$ quadratic equations in the $v = 8$ input/output-bit variables over $\text{GF}(2)$. The entire system for a fixed-key LED permutation therefore may include $(16 \cdot r \cdot e)$ quadratic equations in $(16 \cdot r \cdot v)$ variables. For example, in the case of the 64-bit key version according to various embodiments, it may be ended up with 10752 equations in 4096 variables. In comparison, the entire system for a fixed-key AES

permutation may include 6400 equations in 2560 variables. While the applicability of algebraic attacks on AES may remain unclear, those numbers may tend to indicate that LED according to various embodiments may offer a higher level of protection.

[00120] In the following, other cryptanalysis according to various embodiments will be described.

[00121] The slide attack may be a block cipher cryptanalysis technique that may exploit the degree of self-similarity of a permutation. In the case of LED according to various embodiments, all rounds may be made different thanks to the round-dependent constants addition, which may make the slide attack impossible to perform.

[00122] Integral cryptanalysis may be a technique first applied on SQUARE that may be efficient against block ciphers based on substitution-permutation networks, like AES or LED. The idea may be to study the propagation of sums of values and this may be why it may be in general quite powerful on ciphers using bijective components only. As for AES, the best integral property may be found on three rounds, or four rounds with the last mixing layer removed. Thus, two big LED steps may avoid any such observation.

Considering the large amount of rounds of LED according to various embodiments, integrals attacks may be very unlikely to be a threat.

[00123] Rotational cryptanalysis may study the evolution of a rotated variant of some input words through the round process. It was proven to be quite successful against Addition-Rotation-XOR (ARX) block ciphers and hash functions. LED according to various embodiments is an Sbox-oriented block cipher and any rotation property in a cell may be directly removed by the application of the Sbox layer. Even if one looks for a rotation property of cell positions, this may be unlikely to lead to an attack since the

constants used in a LED round may all be distinct and any position rotation property between columns or lines may be removed after the application of two rounds.

[00124] In the following, LED according to various embodiments in a hash function setting will be described.

[00125] Studying a block cipher in a hash function setting may be a good security test since it may be very advantageous for the attacker. In this scenario he will have full control on all inputs. In the so-called known-key or chosen-key models, the attacker may have access or even choose the key(s) used, and its goal may be then to find some input/output pairs having a certain property with a complexity lower than what is expected for randomly chosen permutation(s). For example, the property may be that the input and output differences or values are fixed to a certain subset of the whole domain.

[00126] In the following, Rebound and Super-Sbox attacks will be described. The recent rebound attack and its improved variants (start-from-the-middle attack and Super-Sbox cryptanalysis) have much improved the best known attacks on many hash functions, especially for AES-based schemes. The attacker may first prepare a differential path and then use the available freedom degrees to the most costly part of the trail (often in the middle) so as to reduce the overall complexity. The costly part may be called the controlled rounds, while the rest of the trail may be the uncontrolled rounds and they may be verified probabilistically. The rebound attack and its variants may allow the attacker to nicely use the freedom degrees so that the controlled part may be as big as possible. At the present time, the most powerful technique in the known-key setting may allow the attacker to control three rounds and no method is known to control more rounds, even if the key is chosen by the attacker.

[00127] In order to ease the analysis, it may be assumed pessimistically that the attacker may control four rounds, that is one full active big step, with a negligible computation/memory cost (even if one may find a method to control four AES-like rounds in the chosen-key model, it may not apply here since no key is inserted during four consecutive rounds). In the case of 64-bit key LED, the attacker may control two independent active big steps and later merge them by freely fixing the key value. However, even in this advantageous scenario for the attacker it may be ensured that at least two big steps may be active and uncontrolled, and this may seem sufficient to resist distinguishing attacks. Indeed, for two active big steps of LED, the upper bound for the best differential path probability and the best linear approximation probability (respectively the best differential probability and the best linear hull probability) may be 2^{-100} (respectively 2^{-64}).

[00128] For the 128-bit key version, it may be again imagined that the attacker to control and merge two active big steps with a negligible computation/memory cost. Even if so, with the same reasoning it may be ensured that at least four big steps will be active and uncontrolled, and again this may seem sufficient since for four active big steps of LED, the upper bound for the best differential path probability and the best linear approximation probability (respectively the best differential probability and the best linear hull probability) may be 2^{-200} (respectively 2^{-128}).

[00129] In the following, integral attacks will be described. The known-key variant of integral attacks from may be adapted to the LED internal permutation according to various embodiments. However, this attack may only reach seven rounds with

complexity 228, which may be worse than what can be obtained with previous rebound-style attacks.

[00130] In the following, performance and comparison of devices and methods according to various embodiments will be described.

[00131] In the following, the hardware implementation according to various embodiments will be described.

[00132] According to various embodiments, a Mentor Graphics ModelSimXE 6.4b and Synopsys DesignCompiler A-2007.12-SP1 may be used for functional simulation and synthesis of the designs to the Virtual Silicon (VST) standard cell library UMCL18G212T3, which may be based on the UMC L180 0.18_μm 1P6M logic process with a voltage of 1.8 V. A Synopsys Power Compiler version A-2007.12-SP1 may be used to estimate the power consumption of ASIC implementations according to various embodiments. For synthesis and for power estimation, the compiler may be advised to keep the hierarchy and use a clock frequency of 100 KHz, which may be a widely used operating frequency for RFID applications. It is to be noted that the wire-load model used, though it is the smallest available for this library, still may simulate the wire-load of a circuit with a size of around 10; 000 GE.

[00133] For describing the hardware efficiency of the LED family according, LED-64 and LED-128 may be implemented in VHDL and their post-synthesis performance may be simulated. As can be seen in FIG. 11, a serialized design according to various embodiments may include seven modules: MCS, State, AK, AC, SC, Controller, and Key State.

[00134] FIG. 11 shows a diagram 1100 illustrating a hardware architecture in accordance with various embodiments. In 1102 (in the left of FIG. 11), a serial hardware architecture of LED is shown, and in 1104 (in the right of FIG. 11), A with its sub-components is shown.

[00135] State may include a 4-4 array of flip-flop cells storing 4 bits each. Every row may constitute a shift-register using the output of the last stage, i.e. column 0, as the input to the first stage (column 3) of the same row and the next row. Using this feedback functionality ShiftRows may be performed in 3 clock cycles with no additional hardware costs. Further, since MixColumnsSerial is performed on column 0, also a vertical shifting direction may be desired for this column. Consequently, columns 0 and 3 may include flip-flop cells with two inputs (6 GE; GE may stand for „gate equivalents), while columns 1 and 2 may be realized with flip-flop cells with only one input (4.67 GE)

[00136] The key may be stored in Key State, which may include a 4-bit wide simple shift register of the appropriate length, i.e. 64 or 128. It is to be noted that the absence of a key-schedule of LED may have two advantages: it may allow 1) to use the most basic, and thus cheapest, flip-flops (4.67 GE per bit); and 2) to hardwire the key in case no key update is required. In the latter case additional combinational logic may be desired to select the appropriate key chunk, which may reduce the savings to 278 GE and 577 GE for LED-64 and LED-128, respectively. For arbitrary key lengths the area requirements may grow by 4.67 GE per bit. An LED-80 with the same parameters as PRESENT-80 may thus require approximately 1; 040 GE with a flexible key and only around 690 GE if we fix the key.

[00137] MCS may calculate the last row of A in one clock cycle. The result may be stored in the State module, that is in the last row of column 0, which may have been shifted upwards at the same time. Consequently, after 4 clock cycles the MixColumnsSerial operation may be applied to an entire column. Then the whole state array may be rotated by one position to the left and the next column may be processed. As an example of the hardware efficiency of MCS, A may be depicted in the upper and its sub-components in the lower right part of FIG. 11. In total only 40 GE and 20 clock cycles may be required to perform MCS, which may be 4 clock cycles slower but 85% smaller than a serialized implementation of the AES MixColumns. If it is taken into account that AES may operate on 8 bits and not like LED on 4 bits, the area savings may be still more than 40%.

[00138] AK may perform the AddRoundKey operation by XORing the roundkey every fourth round. For this reason the input to the XNOR gate may be gated with a NAND gate.

[00139] AC may perform one part of the AddConstant operation by XORing the first column of the round constant matrix (a simple arithmetic 2-bit counter) to the first column of the state matrix. For this reason, the input to the XNOR gate may be gated with a NAND gate. In order to use a single control signal for the addition of the round constants, which span over the first two columns, the addition of the second column of the round constant matrix to the second column of the state array may be performed in the State module.

[00140] SC may perform the SubCells operation and may include a single instantiation of the corresponding Sbox. According to various embodiments, an optimized Boolean

representation of the PRESENT Sbox, which only requires 22.33 GE, may be used. It may take 16 clock cycles to perform AddConstant and SubCells on the whole state.

[00141] Controller may use a Finite State Machine (FSM) to generate all control signals required. The FSM may include one idle state, one init state to load the initial values, one state for the combined execution of AC and SC, 3 states for ShR (ShiftRows) and two states for MCS (one for processing one column and another one to rotate the whole state to the left). Several LFSR-based counters may be provided: 6-bit for the generation of the second column of the round constants matrix, 4-bit for the key addition scheduling and 2-bit for the transition conditions of the FSM. Besides, a 2-bit arithmetic counter may be provided for the generation of the first column of the round constants matrix. Its LSB (least significant bit) may also be used to select either the 3 MSB (most significant bits) $rc_5 || rc_4 || rc_3$ or the 3 LSB $rc_2 || rc_1 || rc_0$ of the 6-bit LFSR-based counter. In total the control logic may sum up to 199 GE.

[00142] According to various embodiments, it may require 39 clock cycles to perform one round of LED, resulting in a total latency of 1248 clock cycles for LED-64 and 1872 clock cycles for LED-128. The estimated power consumption at a frequency of 100 KHz and a supply voltage of 1.8V may be 1.67 μ W for LED-64 (1.11 μ W with a hard-wired key) and 2.2 μ W for LED-128 (1.11 μ W). At low frequencies, as for example for low-cost applications, the power consumption may be dominated by its static part, which may be proportional to the amount of transistors involved. Furthermore, the power consumption may strongly depend on the used technology and may greatly vary with the simulation method. To address these issues and to reflect the time-area-power trade-off inherent in any hardware implementation a commonly known figure of merit (FOM) may

be used. In order to have a fair comparison, the power values in Table 2 have been omitted and only cycles per block, throughput at 100 KHz (in kilo bits per second), the area requirements (in GE), and FOM (in nano bits per clock cycle per GE squared) are compared. Table 2 compares our results to previous works, sorted after key flexibility and increasing security levels. As can be seen, the devices according to various embodiments are the smallest when compared to other block ciphers with similar key and block size.

Algorithm	Ref.	key size	block size	cycles/ block (@100 KHz)	T'put [μm]	Tech. [μm]	Area [GE]	FOM [$\frac{bits \times 10^9}{clk \cdot GE^2}$]
Flexible Keys								
DESL	[27]	56	64	144	44.4	0.18	1,848	130
LED-64		64	64	1,248	5.13	0.18	966	55
KLEIN-64	[18]	64	64	N/A	N/A	0.18	1,981	N/A
LED-80*		80	64	1,872	3,42	0.18	1,040	32
PRESENT-80	[38]	80	64	547	11.7	0.18	1,075	101
KATAN64	[11]	80	64	255	25.1	0.13	1054	226
KLEIN-80	[18]	80	64	N/A	N/A	0.18	2,097	N/A
LED-96*		96	64	1,872	3,42	0.18	1,116	27
KLEIN-96	[18]	96	64	N/A	N/A	0.18	2,213	N/A
mCrypton	[28]	96	64	13	492.3	0.13	2,681	685
SEA	[29]	96	96	93	103	0.13	3,758	73
LED-128		128	64	1,872	3.42	0.18	1,265	21
PRESENT-128	[35]	128	64	559	11.45	0.18	1,391	59
HIGHT	[21]	128	64	34	188	0.25	3,048	203
AES	[33]	128	128	226	56.64	0.13	2,400	98
DESXL	[27]	184	64	144	44.4	0.18	2,168	95
Hard-wired Keys								
LED-64		64	64	1,280	5.13	0.18	688	108
PRINTcipher-48	[25]	80	48	768	6.25	0.18	402	387
KTANTAN64	[11]	80	64	255	25.1	0.13	688	530
LED-80*		80	64	1,872	3,42	0.18	690	72
LED-96*		96	64	1,872	3,42	0.18	695	71
LED-128		128	64	1,872	3.42	0.18	700	70
PRINTcipher-96	[25]	160	96	3072	3.13	0.18	726	59

* estimated values.

Table 2. Hardware implementation results of some block ciphers

[00143] In the following, software implementation will be described.

[00144] Two implementations of LED according to various embodiments have been made; one for reference and clarity with the second being optimized for performance (by using table lookups). The measurements were taken on an Intel(R) Core(TM) i7 CPU Q 720 clocked at 1.60GHz.

[00145] In the optimized implementation, the LED state may be represented as a single 64-bit word and eight lookup tables each with 28 64-bit entries may be provided. This may be similar to many AES implementations, except two consecutive nibbles (2_4 bits) may be treated as a unit for the lookup table. Hence SubCells, ShiftRows and MixColumnsSerial may all be achieved using eight table lookups and XORs.

Overall, it may be desired to access $8 \times 32 \times 2 = 512$ 32-bit words of memory (or $8 \times 32 = 256$ 64-bit words of memory). In contrast, an AES implementation with four tables of 28 entries may require $(16+4) \times 10 = 200$ accesses. This may suggest that LED-64 according to various embodiments may be about 2.5 times slower than AES on 32-bit platforms with table-based implementations, and similarly LED-128 according to various embodiments may be 3.8 slower than AES, which may be consistent with measurements carried out.

	reference implementation	table-based implementation
LED-64	7154 cycles/byte	57 cycles/byte
LED-128	9410 cycles/byte	86 cycles/byte

Table 3. Software implementation results of LED.

[00146] In the above, the block cipher **LED** according to various embodiments has been described. According to various embodiments, devices and methods may be used in constrained hardware implementations and may provide one of the smallest block ciphers in the literature for comparable choices of parameters while striving to maintain a competitive performance of the method.

[00147] In the following, round constants according to various embodiments will be described. The generating methods of the round constants have been described above.

Below are the list of (rc5; rc4; rc3; rc2; rc1; rc0) encoded to byte values for each round, with rc0 being the least significant bit.

Rounds	Constants
1-10	01, 03, 07, 0F, 1F, 3E, 3D, 3B, 37, 2F
11-20	1E, 3C, 39, 33, 27, 0E, 1D, 3A, 35, 2B
21-30	16, 2C, 18, 30, 21, 02, 05, 0B, 17, 2E
31-40	1C, 38, 31, 23, 06, 0D, 1B, 36, 2D, 1A
41-48	34, 29, 12, 24, 08, 11, 22, 04

[00148] In the following, test vectors according to various embodiments will be described. Test vectors for LED with 64-bit and 128-bit key arrays are given below.

	plaintext	key	ciphertext	
LED-64	0 0 0 0	0 0 0 0	E F B B	
	0 0 0 0	0 0 0 0	B D 6 D	
	0 0 0 0	0 0 0 0	3 6 8 B	
	0 0 0 0	0 0 0 0	8 9 9 9	
	0 1 2 3	0 1 2 3	F D D 6	
	4 5 6 7	4 5 6 7	F B 9 8	
	8 9 A B	8 9 A B	4 5 F 8	
	C D E F	C D E F	1 4 5 6	
LED-128	0 0 0 0	0 0 0 0	0 0 0 0	1 6 5 0
	0 0 0 0	0 0 0 0	0 0 0 0	B 2 3 0
	0 0 0 0	0 0 0 0	0 0 0 0	E 8 F 9
	0 0 0 0	0 0 0 0	0 0 0 0	2 8 B B
	0 1 2 3	0 1 2 3	0 1 2 3	3 1 3 1
	4 5 6 7	4 5 6 7	4 5 6 7	C 2 3 1
	8 9 A B	8 9 A B	8 9 A B	2 0 5 C
	C D E F	C D E F	C D E F	3 6 6 4

[00149] In the following, devices and methods according to various embodiments, which may be referred to as the “PHOTON Lightweight Hash Functions Family”, will be described.

[00150] RFID security is currently one of the major challenges cryptography has to face, often solved by protocols assuming that an on-tag hash function is available. According to various embodiments, the PHOTON lightweight hash functions family is provided, available in many different flavors and suitable for extremely constrained devices such as passive RFID tags. Various embodiments use a sponge-like construction as domain extension method (or algorithm) and an AES-like primitive as internal

unkeyed permutation. This may allow to obtain the most compact hash function known so far (about 1120 GE for 64-bit collision resistance security), reaching areas very close to the theoretical optimum (derived from the minimal internal state memory size).

Moreover, the speed achieved by PHOTON also may compare to its competitors. This may be mostly due to the fact that unlike for previously proposed schemes, various embodiments are very simple to analyze and tight AES-like bounds on the number of active Sboxes may be derived. This kind of AES-like primitive is usually not well suited for ultra constrained environments, but according to various embodiments, a new method for generating the column mixing layer in a serial way is provided, lowering drastically the area required. Furthermore, the sponge framework is extended in order to offer interesting trade-offs between speed and preimage security for small messages, the classical use-case in hardware.

[00151] RFID tags may be likely to be deployed widely in many different situations of everyday life and they may represent a great business opportunity for various markets. However, this rising technology also may provide new security challenges that the cryptography community has to handle. RFID tags may be used to fight product counterfeiting by authenticating them and on the other hand, it may be desired to guarantee the privacy of the users.

[00152] These two security aspects have already been studied and, interestingly, in most of the privacy-preserving RFID protocols proposed, a hash function is required. Informally, such a primitive is a function that takes an arbitrary length input and outputs a fixed-size value. While no secret is involved in the computation, one would like that finding collisions (two distinct messages hashing to the same value) or (second)-

preimages (a message input that hashes to a given challenge output value) is computationally intractable for an attacker. More precisely, for an n -bit ideal hash function it may be expected to perform between $2^{n/2}$ and 2^n computations in order to find a collision and a (second)-preimage respectively. While not as mature as block-ciphers, the research on hash functions saw a rapid development lately, mainly due to the recent groundbreaking attacks on standardized primitives. At the present time, most of the attention of the symmetric key cryptography academic community are focused on the SHA-3 competition organized by NIST, which should provide a potential replacement of the MD-SHA family.

[00153] In parallel, nice advances have also been made in the domain of lightweight symmetric key primitives in the last years. Protocol designers now have at disposal PRESENT, a 64-bit block-cipher with 80-bit key whose security has already been much analyzed and that can be as compact as 1075 GE. Stream-ciphers may not be outcast with implementations with 80-bit security requiring about 1300 GE and 2600 GE reported for GRAIN and TRIVIUM respectively, two candidates selected in the final eSTREAM hardware portfolio. However, the situation is not as bright in the case of hash functions.

[00154] The community lacks very compact hash functions. Standardized primitives such as SHA-1 or SHA-2 are much too large to fit in very constrained hardware (8120 and 10868 GE for 80 and 128-bit security respectively) and even compact-oriented proposals such as MAME require 8100 GE for 128-bit security. While hardware may be an important criteria in the selection process, one can not expect the SHA-3 finalists to be much more compact. At the present time, the most compact SHA-3 finalist is KECCAK for which 20000 GE implementations have been reported for 128-bit security (smaller

versions of KECCAK that have not been submitted to the competition provide for example 64-bit security with 5090 GE). It is to be noted that a basic RFID tag may have a total gate count of anywhere from 1000 to 10000 gates, with only 200 to 2000 gates budgeted for security.

[00155] This compactness problem in hash method may be partly due to the fact that it widely depends on the memory registers required for the computation. Most hash functions proposed so far are software-oriented and output at least 256 bits in order to be out of reach of any generic collision search in practice. While such an output size may make sense where high level and long-term security are needed, RFID use-cases could bear much smaller security parameters. For example lightweight hash functions may be instantiated using literature-based constructions with the compact block-cipher PRESENT. With SQUASH, Shamir proposed a compact keyed hash function inspired by the Rabin encryption scheme that processes short messages (at most 64-bit inputs) and that provides 64 bits of preimage security, without being collision resistant. Furthermore, the hash function QUARK has been published, using sponge functions as domain extension method, and an internal permutation inspired from the stream-cipher GRAIN and the block-cipher KATAN. Using sponge functions as operating mode may be another step towards compactness. Indeed, classical n -bit hash function constructions like the MD-SHA family may utilize a Merkle-Damgard domain extension method with a compression function h built upon an n -bit block-cipher E in Davies-Meyer mode ($h(CV;K) = E_K(CV) \oplus CV$). Avoiding any feed-forward like for sponge constructions may save memory registers at the cost of an invertible iterative process which induces a

lower (second)-preimage security for the same internal state size. All in all, designers may desire to deal with a trade-off between security and memory requirements.

[00156] According to various embodiments, a hardware-oriented hash functions family (which may be called “PHOTON”) may be provided. It may be chosen to use the sponge functions framework in order to keep the internal memory size as low as possible. However, the squeezing process of sponge functions may be really slow in practice when one would like to hash small messages. Therefore, this framework has been extended so as to provide very interesting trade-offs in hardware where hashing small messages is the classical use-case. The internal permutations of PHOTON may be seen as AES-like primitives especially derived for hardware: a columns mixing layer according to various embodiments may be computed in a serial way while maintaining optimal diffusion properties. Overall, as shown in Table 8 below, not only PHOTON may be easily the smallest hash function known so far, but it may also achieve excellent area/throughput trade-offs.

[00157] In terms of security, it may be interesting to use AES-like permutations as all the previous cryptanalysis performed on AES and on AES-based hash functions may be fully leveraged. Moreover, very simple bounds on the number of active Sboxes for 4 rounds of the permutation may be derived. These bounds being tight, it may confidently be set an appropriate number of rounds that ensures a comfortable security margin.

[00158] In the following, design choices according to various embodiments will be described.

[00159] In tag-based applications, high security primitives, such as a 512-bit output hash function, may not be required. In contrary, 64 or 80-bit security may often be

appropriate considering the cost of an RFID tag. Moreover, a designer may use exactly the level that he expects from his primitive, so as to avoid any waste of area or computing power. This may be the reason why to chose to precisely instantiate several security levels for PHOTON, ranging from 64-bit preimage resistance security to 128-bit collision resistance security.

[00160] In the following, extended sponge functions will be described.

[00161] Sponge functions have been introduced as a way of building hash functions from a fixed permutation (later more applications were proposed). The internal state S , composed of the c -bit capacity and the r -bit bitrate, may be first initialized with some fixed value. Then, after having appropriately padded the message to hash, one simply and iteratively may process all r -bit message chunks by xoring them to the bitrate part of the internal state and then applying the $(c + r)$ -bit permutation P . Once all message chunks have been handled by this absorbing phase, one successively may output r bits of the final hash value by extracting r bits from the bitrate part of the internal state and then applying the permutation P on it (which may be referred to as squeezing process).

[00162] A sponge function may have proven to be indifferentiable from a random oracle up to $2^{c/2}$ calls to the internal permutation P . More precisely, for an n -bit sponge hash function with capacity c and bitrate r , when the internal primitive is modeled as a random permutation, one may obtains $\min\{2^{n/2}; 2^{c/2}\}$ as collision resistance bound and $\min\{2^n; 2^{c/2}\}$ as (second)-preimage bound. However, in the case of preimage, there may exist a gap between this bound and the best known generic attack. Therefore, the following complexities may be expected in the generic case (The 2^{n-r} term for preimage

may come from the fact that in order to invert the hash function, the attacker will have to invert the squeezing process. The best known generic attack to solve this "multiblock constrained-input-constrained-output problem" requires 2^{n-r} computations):

- Collision: $\min\{2^{n/2}; 2^{c/2}\}$;
- Second-preimage: $\min\{2^n; 2^{c/2}\}$; and
- Preimage: $\min\{2^n; \max\{2^{n-r}; 2^{c/2}\}\}$

[00163] Moreover, sponge functions may be used as a Message Authentication Code with $\text{MACK}(M) = H(K||M)$, where $K \in \{0; 1\}^k$ may stand for the key and M for the message. It has been shown that as long as the amount of message queries is limited to 2^a with $a \ll c/2$, then no attack better than exhaustive key search may exist if $c \geq k + a + 1$.

[00164] Sponge functions may be chosen in order to minimize the amount of memory registers in hardware since they offer speed/area trade-offs. Indeed, the only memory required for the internal state is $c + r$ bits, while for a classical Davies-Meyer construction using an m -bit block cipher with a k -bit key input one needs to store $2m + k$ bits because of the feed-forward. For an equivalent collision security level ($m = c$) and by minimizing the area (r and k are very small), the sponge function may require only about halve the memory.

[00165] However, sponge functions may also possess some drawbacks for hardware scenarios. For example, in order to minimize the memory area, a sponge with a capacity equal to the hash output size, and a very small bitrate may be chosen. It has been identified that in most RFID applications the user will not hash a large amount of data,

i.e. in general less than 256 bits. Consider for example the electronic product code (EPC) number, which is a 96-bit string that is meant to identify globally any tag/product. In this case of small messages, sponge functions with a small bitrate r may be slow since one needs to call $(\lceil n/r \rceil - 1)$ times the internal permutation to complete the final squeezing process. This may for example be the case with QUARK, that has a throughput of 1.47 kbps for very long messages which drops to 0.63 kbps for 96-bit inputs. To summarize, sponge functions in fact may provide area/ small message speed trade-offs.

[00166] Various embodiments avoid this issue by extending the sponge framework by allowing the number r of bits extracted during each iteration of the squeezing process to be different than the bitrate r' . This is depicted in FIG. 12.

[00167] FIG. 12 shows a diagram 1200 illustrating a framework in accordance with various embodiments. In the diagram of the extended sponge framework, the domain extension method used by the PHOTON hash functions family, absorbing is illustrated in 1202, and squeezing is illustrated in 1204.

[00168] Increasing r' may directly reduce the time spent in the squeezing process, but may also reduce the preimage security. On the contrary, decreasing r' may improve the preimage bound at the cost of a speed drop for small messages. As long as the preimage security remains in an acceptable bound, this configuration may be very interesting in most scenarios where tiny inputs only are to be hashed. More precisely, in this model according to various embodiments, the best known generic attacks require the following amount of computations:

- Collision: $\min\{2^{n/2}; 2^{c/2}\};$

- Second-preimage: $\min\{2^n; 2^{c/2}\};$ and

- Preimage: $\min\{2^n; \max\{2^{n-r'}; 2^{c/2}\}\}$

and now there may be at disposal an area/ small message speed/ preimage security trade-off.

[00169] In most tag-based applications the collision resistance may not be a requirement, while the one-wayness of the function must be ensured. However, as described above, the sponge construction does not maintain the preimage security at the full level of its capacity c . This may be due to the output process of the sponge operating mode. Of course, performing a Davies-Meyer like feed-forward just after the final truncation would do the job, but that may also double the memory area required (which may be what is to be avoided). The nice trick of squeezing in the sponge functions framework may allow to avoid any feed-forward while somehow rendering the process non-invertible, up to some extent (see commonly used multiblock constrained-input constrained-output problem). One solution to reach the full capacity preimage security may be to add one more squeezing iteration, thus increasing the output size of the hash by r' bits. Then, the best known generic preimage attack for this $(n + r')$ -bit hash function may run in

$$\min\{2^{n+r'}, \max\{2^n, 2^{c/2}\}\} \geq 2^n.$$

[00170] According to various embodiments, the extended sponge model may offer area / small message speed / preimage security / hash output size trade-offs and this flexibility may be kept so that the PHOTON function may be adapted to the use-case that is to be handled. In any case, the exemplary shape of the extended sponge for PHOTON may be a capacity c equal or slightly larger than the hash output n and a very small bitrate

r. The output bitrate r' may be the same as r , unless it is looked for better preimage security (by reducing r') or improving the small message efficiency (by increasing r').

[00171] In the following, an AES-like internal permutation according to various embodiments will be described.

[00172] An AES-like function may be defined to be a fixed key permutation P applied on an internal state of d^2 elements of s bits each, which can be represented as a $(d \cdot d)$ matrix. P may be composed of N_r rounds, each containing four layers as depicted in FIG. 13: AddConstants (AC), SubCells (SC), ShiftRows (ShR), MixColumnsSerial (MCS).

[00173] FIG. 13 shows a flow diagram 1300 illustrating one round of a PHOTON permutation according to various embodiments. In 1302, AddConstants is illustrated, in 1304, SubCells is illustrated, in 1306, ShiftRows is illustrated, and in 1308, MixColumnsSerial is illustrated.

[00174] Informally, AddConstants may include adding fixed values to the cells of the internal state, while SubCells may apply an s -bit Sbox to each of them. ShiftRows may rotate the position of the cells in each of the rows and MixColumnsSerial may linearly mix all the columns independently.

[00175] AES-like permutations according to various embodiments may offer much confidence in the design strategy as one can leverage previous cryptanalysis works done on AES and on AES-like hash functions. Moreover, AES-like permutations may allow to derive very simple proofs on the number of active Sboxes over four rounds of the primitive. More precisely, if the matrix underlying the MixColumnsSerial layer is Maximum Distance Separable (MDS), then one may immediately show that at least

$(d+1)^2$ Sboxes will be active for any 4-round differential path. This bound may be tight, and there are already known differential paths with only $(d+1)^2$ active Sboxes for four rounds (which will be use below for security analysis purposes). Moreover, it is to be noted that the permutations according to various embodiments are fixed-key, so related-key attacks or any issue that might arise from the construction of a key-schedule may be get rid of.

[00176] In the following, AddConstants according to various embodiments will be described. The constants may have been chosen such that each of the N_r round computations are different, and such that the classical symmetry between columns in AES-like designs may be destroyed (without the AddConstants layer, an input with all columns equal may maintain this property through any number of rounds). Also, the round constants may be generated by a combination of very compact Linear Feedback Shift Registers. For performance reasons, only the first column of the internal state may be involved.

[00177] In the following, SubCells according to various embodiments will be described. According to various embodiments, the Sboxes show good hardware quality. 4-bit Sboxes may be very compact in hardware while the acceptable upper limit on the cell size is $s = 8$. It may be desired to avoid to use an Sbox size s which is odd, because this may leads to odd message block size or capacity when d is also odd. This may leave $s = 4; 6; 8$, but reusing some already trusted and well analyzed components may increase the confidence in the security of the scheme and may save a lot of time for cryptanalysts. Finally, two types of Sboxes may be used: the 4-bit PRESENT Sbox SBOXPRE and the

8-bit AES Sbox SBOXAES, the latter being only utilized for high security levels (at least 128 bits of collision resistance). It is to be noted also that $s = 4; 8$ may allow simpler and faster software implementations.

[00178] In the following, ShiftRows according to various embodiments will be described. The choice of the ShiftRows constants may be very simple for PHOTON since the internal state may always be a square of cells. Therefore, row i may classically be rotated by i positions to the left.

[00179] In the following, MixColumnsSerial according to various embodiments will be described. The matrix underlying the AES MixColumns function may be a circulant matrix with low hamming weight coefficients. Even if those coefficients and the irreducible polynomial used to create the Galois field for the AES MixColumns function have been chosen so as to improve the hardware footprint of the cipher, it may not be implemented in an extremely compact way. One of the main reason may be that the byte-serial implementation of this function is not compact. Said in other words, if we write the AES MixColumns matrix as the composition of d operations each updating a single byte at a time in a serial way, then the coefficients of these d matrices may be very bad for small area implementations.

[00180] In order to solve this issue, the problem may be took the other way round. Let A be the matrix that updates the last cell of the column with a linear combination of all of them and then rotates the column cells by 1 position towards the top. The MixColumnsSerial layer according to various embodiments may be composed of d applications of this matrix to the input column vector. More formally, let

$X = (x_0, \dots, x_{d-1})^T$ be an input column vector of MixColumnsSerial and

$Y = (y_0, \dots, y_{d-1})^T$ be the corresponding output. Then, we have $Y = A^d \times X$, where A is a

$(d \times d)$ matrix of the form:

$$A = \begin{pmatrix} 0 & 1 & 0 & 0 & \dots & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & \dots & 0 & 0 & 0 & 0 \\ \vdots & & & & & & & \vdots & \\ 0 & 0 & 0 & 0 & \dots & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & \dots & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & \dots & 0 & 0 & 0 & 1 \\ Z_0 & Z_1 & Z_2 & Z_3 & \dots & Z_{d-4} & Z_{d-3} & Z_{d-2} & Z_{d-1} \end{pmatrix}$$

where coefficients $(Z_0, \dots, Z_{d-1})$ may be chosen freely. By $\text{CircSerial}(Z_0, \dots, Z_{d-1})$ such a

matrix may be denoted. It is desirable that the final matrix A^d is MDS, so as to maintain as much diffusion as for the AES initial design strategy. For each square size d that may be picked according to various embodiments of PHOTON, MAGMA may be used to test all the possible values of $Z_0, \dots, Z_{d-1}$ and the most compact candidate making A^d an MDS matrix may be picked. Furthermore, the irreducible polynomial with compactness may be chosen as main criterion.

[00181] For design strategy comparison purposes, the AES case may be taken as an example. By using the mixing layer design method according to various embodiments, it may be possible to find the matrix $A = \text{CircSerial}(1; 2; 1; 4)$ which gives the following MDS final matrix:

$$(A)^4 = \begin{pmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 1 & 2 & 1 & 4 \end{pmatrix}^4 = \begin{pmatrix} 1 & 2 & 1 & 4 \\ 4 & 9 & 6 & 17 \\ 17 & 38 & 24 & 66 \\ 66 & 149 & 100 & 11 \end{pmatrix}$$

[00182] The smallest AES hardware implementation may require 2400 GE, for which 263 GE are dedicated to MixColumns. It may be possible to implement MixColumns of AES in a byte-by-byte fashion, which may require only 81 GE to calculate one byte of the output column. However, since AES uses a circulant matrix, at least three additional 8-bit registers (144 GE), may be required to hold the output, plus additional control logic, which may increase the area requirements significantly. That is why usually a serial MixColumns is not used, but rather one column is processed at a time.

[00183] It is to be noted that in general the choice of non-zero constants for any $d \times d$ MDS matrix on s -bit cells may have only a minor impact of the area consumption, since a multiplication by x consists of w XOR gates, where w denotes the Hamming weight of the irreducible polynomial used. At the same time, $(d-1) \cdot s$ XOR gates may be required to sum up the d individual terms of s bits each. Multiplying with the constants above accounts for only 21.3 GE out of the 74 GE required. In fact, the efficiency of the approach according to various embodiments lies in the shifting property of A , since this allows to re-use the existing memory with neither temporary storage nor additional control logic required.

[00184] All in all, using the approach according to various embodiments may provide a tweaked AES cipher with the very same diffusion properties as the original one (the matrix being MDS), but that can fit in only 2210 GE, a total saving of around 8%. Moreover, for the deciphering process, a slightly modified hardware may be used in order

to unroll the MixColumnsSerial, further reducing the area footprint of such a PHOTON-based cipher.

[00185] One may think that the software implementations may suffer from this new layer. While the goal is to make a hardware-oriented primitive, it is to be remarked that most AES software implementations are precomputed tables-based (applying both the Sbox and the MixColumns coefficients at the same time) and the very same method may be applied to PHOTON. This may be confirmed by a software implementations, whose benchmarks are given further below.

[00186] In the following, the PHOTON Hash Function Family according to various embodiments will be described.

[00187] In the following, the PHOTON family of hash functions according to various embodiments will be described. Each variant will be fully defined by its hash output size $64 \leq n \leq 256$, its input and its output bitrate r and r' respectively. Therefore each function is denoted PHOTON- $n/r/r'$. The internal state size $t = (c + r)$ may depend on the hash output size and may take only 5 distinct values: 100, 144, 196, 256 and 288 bits. As a consequence, only 5 internal permutations P_t may be defined, one for each internal state size.

[00188] In order to cover a wide spectrum of applications, five different flavors of PHOTON may be provided, one for each internal state size: PHOTON-80/20/16, PHOTON-128/16/16, PHOTON-160/36/36, PHOTON-224/32/32 and PHOTON-256/32/32 may use internal permutations P_{100} , P_{144} , P_{196} , P_{256} and P_{288} respectively.

It is to be noted that the first proposal is special in the sense that it is designed for the

specific cases where 64-bit preimage security is considered to be sufficient (By sponge keying and using a commonly known security bound, PHOTON-80/20/16 also may provide a secure 64-bit key MAC as long as the number of messages to be computed is lower than 215. For a secure 64-bit key MAC handling more messages (up to 227), one may for example go for a very similar PHOTON-80/8/8 version that also uses P_{100} . This version with capacity $c = 92$ may require the same area as PHOTON-80/20/16 but may be slower.). In contrary, the last proposal may provide a high security level of 128-bit collision resistance, thus making it suitable for generic applications.

[00189] In the following, the domain extension method according to various embodiments will be described.

[00190] The message M to hash may be first padded by appending a "1" bit and as many zeros (possibly none) such that the total length is a multiple of the bitrate r and finally l message blocks $m_0, \dots, m_{l-1}$ of r bits each may be obtained. The t -bit internal state S may be initialized by setting it to the value $S_0 = IV = \{0\}^{t-24} \parallel_{n=4}^{\parallel r \parallel r}$, where $\parallel$ may denote the concatenation and each value may be coded on 8 bits. For implementation purposes, it may be noted that each byte may be interpreted in big-endian form.

[00191] Then, as for the classical sponge strategy, at iteration i the message block m_i may be absorbed on the most significant part of the internal state S_i and then the permutation P_t may be applied, i.e. $S_{i+1} = P_t(S_i \oplus (m_i \parallel \{0\}^c))$.

[00192] Once all l message blocks have been absorbed, the hash value may be built by concatenating the successive r' -bit output blocks z_i until the appropriate output size n is reached:

$$hash = z_0 || \dots || z_{l'-1}$$

where l' may denote the number of squeezing iterations, that is $l' = \lceil n/r' \rceil - 1$. More precisely, z_i may be the r' most significant bits of the internal state S_{l+i} and one may have $S_{l+i+1} = P(S_{l+i})$ for $0 \leq i < l'$. If the hash output size is not a multiple of r' , one may truncate $z_{l'-1}$ to $n \bmod r'$ bits.

[00193] In the following, the internal permutations according to various embodiments will be described.

[00194] The internal permutations P_t may be defined, where $t \in [100; 144; 196; 256; 288]$. The internal state of the N_r -round permutation may be viewed as a $(d \times d)$ matrix of s -bit cells and the corresponding values depending of t are given in Table 4. It is to be noted that a cell size of 4 bits may be used, except for the largest version for which 8-bit cells may be used, and that the number of rounds may always be $N_r = 12$, whatever the value of t may be. The internal state cell located at row i and column j may be denoted $S[i; j]$ with $0 \leq i, j < d$.

	t	d	s	N_r	$IC_d(\cdot)$	irr. polynomial	Z_t coefficients
P_{100}	100	5	4	12	[0, 1, 3, 6, 4]	$x + 1$	(1, 2, 9, 9, 2)
P_{144}	144	6	4	12	[0, 1, 3, 7, 6, 4]	$x + 1$	(1, 2, 8, 5, 8, 2)
P_{196}	196	7	4	12	[0, 1, 2, 5, 3, 6, 4]	$x + 1$	(1, 4, 6, 1, 1, 6, 4)
P_{256}	256	8	4	12	[0, 1, 3, 7, 15, 14, 12, 8]	$x + 1$	(2, 4, 2, 11, 2, 8, 5, 6)
P_{288}	288	6	8	12	[0, 1, 3, 7, 6, 4]	$x^4 + x^3 + x + 1$	(2, 3, 1, 2, 1, 4)

Table 4. The parameters of the internal permutations P_t .

[00195] One round may be composed of four layers (see FIG. 13): AddConstant (AC), SubCell (SC), ShiftRows (ShR) and MixColumnsSerial (MCS).

[00196] In the following, AddConstant according to various embodiments will be described. At round number v (starting the counting from 1), first a round constant $RC(v)$ to each cell $S[i, 0]$ of the first column of the internal state may be xored. Then, distinct internal constants $IC_d(i)$ to each cell $S[i, 0]$ of the same first column may be xored.

Overall, for round v one may have $S'[I, 0] = S[i, 0] \oplus RC(v) \oplus IC_d(i)$ for all $0 \leq i < d$.

[00197] The round constants may be $RC(v) = [1, 3, 7, 14, 13, 11, 6, 12, 9, 2, 5, 10]$. The internal constants may depend on the square size d and on the row position i . They are given in Table 4. Further below, all the constants for all square sizes, round numbers, row positions and how they have been generated will be given.

[00198] In the following, SubCells according to various embodiments will be described. This layer may apply an s -bit Sbox to each of the cells of the internal state, i.e. $S'[i, j] = SBOX(S[i, j])$ for all $0 \leq i, j < d$. In the case of 4-bit cells, the PRESENT Sbox $SBOX_{PRE}$ may be used while for the 8-bit cells case the AES Sbox $SBOX_{AES}$ may be used. Both are given further below.

[00199] In the following, ShiftRows according to various embodiments will be described. As for the AES, for each row i this layer may rotate all cells to the left by i column positions. Namely, $S'[i, j] = S[i, (j + i) \bmod d]$ for all $0 \leq i; j < d$.

[00200] In the following, MixColumnsSerial according to various embodiments will be described. The final mixing layer may be applied to each of the columns of the internal state independently. For each column j input vector $(S[0, j], \dots, S[d-1, j])^T$, the matrix $A_t = \text{CircSerial}(Z_0, \dots, Z_{d-1})$ may be applied d times. That is, for all $0 \leq j < d$:

$$(S'[0, j], \dots, S'[d-1, j])^T = A_t^d \times (S[0, j], \dots, S[d-1, j])^T$$

where the coefficients $Z_0, \dots, Z_{d-1}$ are given in Table 4. In the case of 4-bit cells, the irreducible polynomial, $x^4 + x + 1$, may be chosen, while for the 8-bit case the AES one, i.e. $x^8 + x^4 + x^3 + x + 1$ may be chosen. The matrices A_t together with the overall

MixColumnsSerial matrices A_d^t for each internal state size t are given further below.

Note that all A_d^t matrices are Maximum Distance Separable. One may wonder why we it is not proposed a version with $d = 9$ and $s = 4$. The reason is that there is no matrix fulfilling the desired "serial MDS" properties for those parameters, whatever the irreducible polynomial chosen.

[00201] In the following, security analysis of devices and methods according to various embodiments will be described.

[00202] The sponge-like domain extension method according to various embodiments may allow to fully trust the security of the PHOTON hash function as long as the internal

permutation P_t does not present any structural aw whatsoever. In other words, the cryptanalysis work may be made easy since in order to cover any instantiation of PHOTON, one just has to study the security of the five internal permutations P_{100} , P_{144} , P_{196} , P_{256} and P_{288} . It is to be noted that we do not have to consider an adversary bounded to 2^t computations, since for the flat sponge claim no resistance is provided for attacks requiring a workload of more than $2^{c/2}$ operations. To be exact, it is not provided a little bit more than $2^{c/2}$ security for preimage resistance, so in the case of PHOTON it only the internal permutation may be desired to be indistinguishable from a randomly chosen permutation on the same domain $\{0,1\}^t$ up to $\max\{2^{n-r}, 2^{c/2}\}$ operations.

[00203] The PHOTON hash function security may be extremely conservative: a number of rounds may be chosen with a comfortable security margin such that this assumption on the internal permutations may be fulfilled. However, even if any attacker could find a structural flaw on P_t , may not be likely to become an issue for the whole hash function. This argument may be true for a “small-r” sponge-like shape according to various embodiments. Indeed, the amount of freedom degrees available at the input of each internal permutation call during the absorbing phase may be extremely small. Thus, even if a aw is found for the internal permutation, the amount of freedom degrees may be so thin that utilizing this flaw will very likely turn out to be intractable. The utilization of freedom degrees has always be one the most powerful cryptanalyst tool (for MD-SHA

family of hash functions or even for sponge-like hash functions), thus reducing as much as possible this ability may greatly increase the confidence in PHOTON's security.

[00204] The PHOTON hash function may be very simple to analyze so as to make the cryptanalyst work as easy as possible. In the following, different attack scenarios will be described for the internal permutations P_t in which the attacker may be considered to have all freedom degrees possible, that is all t bits of internal state.

[00205] In the following, differential/linear cryptanalysis will be described.

[00206] PHOTON's internal primitives may be AES-like permutations. Therefore, by reusing the work done in the past years on AES, it may be easy to compute a bound on the best differential path probability (where all differences on the input and output of all rounds may be specified) or even the best differential probability (where only the input and output differences may be specified).

[00207] A bound on the number of active Sboxes (i.e. Sboxes with non-zero difference) may be obtained for four rounds of the PHOTON internal permutation by simply adapting a wide-trail strategy to the parameters according to various embodiments: since matrices underlying the MixColumnsSerial layer according to various embodiments are MDS, at least $(d + 1)^2$ Sboxes may be active for any non-null differential path. It is to be noted that this bound may be tight since it may be known four-round differential paths with $(d + 1)^2$ active Sboxes, as will be described further below. As the best differential probability of the PRESENT Sbox is 2^{-2} , and for the AES Sbox it is 2^{-6} , it may be deduced that the best differential path probability on four rounds

of an internal permutation of PHOTON is upper bounded by $2^{-2 \cdot (d+1)^2}$ when $s = 4$ and by $2^{-6 \cdot (d+1)^2}$ when $s = 8$.

[00208] Since the diffusion in PHOTON may be achieved with 2 rounds, a simple freedom degrees utilization (like commonly known for SHA-0) may be likely to allow the control of two rounds. However, as shown further below, more involved methods were recently introduced that may allow to control up to three rounds. Even if it may be considered that those methods can someday be pushed up to four rounds, this may not endanger P_t since eight rounds provide at least $2 \cdot (d + 1)^2$ active Sboxes. Therefore the best differential path on eight rounds has probability upper bounded by $2^{-4 \cdot (d+1)^2} < 2^{-t}$ when $s = 4$ and by $2^{-12 \cdot (d+1)^2} < 2^{-t}$ when $s = 8$.

[00209] It also can be shown that the maximum differential probability for 4 rounds of a PHOTON internal permutation is upper bounded by

$$\max \left\{ \max_{1 \leq u \leq 2^s - 1} \sum_{j=1}^{2^s - 1} \{DP^S(u, j)\}^{d+1}, \max_{1 \leq u \leq 2^s - 1} \sum_{j=1}^{2^s - 1} \{DP^S(j, u)\}^{d+1} \right\}^d$$

where $DP^S(i; j)$ may stand for the differential probability of the Sbox to map the difference i to j .

[00210] The duality between linear and differential attacks may allow to apply the same approaches to compute a bound on the best linear approximation or even the best linear hulls. In Table 5, the upper bounds on the best differential path probability, the best

differential probability, the best linear approximation probability and the best linear hull probability for four rounds of the five PHOTON permutations are summarized.

	P_{100}	P_{144}	P_{196}	P_{256}	P_{288}
differential path probability	2^{-72}	2^{-98}	2^{-128}	2^{-162}	2^{-294}
differential probability	2^{-50}	2^{-72}	2^{-98}	2^{-128}	2^{-246}
linear approximation probability	2^{-72}	2^{-98}	2^{-128}	2^{-162}	2^{-294}
linear hull probability	2^{-50}	2^{-72}	2^{-98}	2^{-128}	2^{-234}

Table 5. Upper bounds on the best differential path probability, best differential probability, best linear approximation probability and best linear hull probability for four rounds of the five PHOTON internal permutations.

[00211] Another important security argument may also be applicable to PHOTON: the internal permutations may be fixed and allow no key input. Of course, this may come at a cost of lowered efficiency, but on the other hand it may avoid any attack leveraging a weakness in the key schedule. This may be important as it was shown that such a control can lead for example to theoretical attacks against AES-192 and AES-256 in the related-key model, or distinguishers for the full WHIRLPOOL compression function.

[00212] In the following, rebound and super-Sbox attacks will be described.

[00213] The original rebound attack and its improved variants (start-from-the-middle attack and Super-Sbox cryptanalysis) have been introduced recently and provide the currently best known methods to analyze AES-like permutations in a hash functions setting, where no secret is involved. It therefore may be important to test what is the resistance of the PHOTON hash functions regarding those new tools.

[00214] At the present time, in order to distinguish a fixed-key AES-like permutation from an ideal one, the best results may be obtained by using a non-full active differential path, while the freedom degrees will be utilized in a Super-Sbox fashion. This may allow to reach 8 rounds and in FIG. 14 the differential path considered are given.

[00215] FIG. 14 shows a flow diagram 1400 illustrating a method according to various embodiments. An 8-round differential path for PHOTON internal permutations is shown, for example the case $d = 8$, $n_L^{\text{in}} = n_L^{\text{out}} = n_R^{\text{out}} = 5$ and $n_R^{\text{in}} = 4$. In 1402, the input data may be shown, in 1404 the data after round 1, in 1406 the data after round 2, in 1408 the data after round 3, in 1410 the data after round 4, in 1412 the data after round 5, in 1414 the data after round 6, in 1416 the data after round 7, and in 1418 the data after round 8.

[00216] It is to be noted that this trail may actually be meeting the bound on the minimal number of active Sboxes for four rounds (round 3 to round 6), thus confirming its quality. Moreover, this trail may perfectly fit for a rebound-kind of attack, since most of its complexity is located at the same place, right in the middle of the path, which may allow the attacker to concentrate all available freedom degrees on this precise part.

[00217] A commonly known non-full linear framework may be extended: n_L^{in} may be called the number of active Sboxes after application of the first round and n_L^{out} after the second in the path from FIG. 14. Identically, n_R^{in} and n_R^{out} may denote the number of active Sboxes after application of the fifth and sixth rounds respectively. Because of the MDS property there may be the constraints that

$$n_L^{\text{out}} + n_R^{\text{in}} \geq d+1, n_L^{\text{in}} + n_L^{\text{out}} \geq d+1$$

and

$$n_R^{in} + n_R^{out} \geq d+1$$

By using the commonly known freedom degrees counting method, it may be checked that

$$2^{s(n_L^{in} + n_L^{out} + n_R^{in} + n_R^{out} - 2d)}$$

solution pairs can be found for this differential path. This may always be greater to one if the previous constraints are fulfilled. Moreover, with the tweaked Super-Sbox method, a solution may be found with one operation on average for the 3 controlled round in the middle (depicted in dashed lines) and with a very limited minimal cost of 2^s computations and memory. Overall, obtaining a solution for the complete path will cost the attacker

$$2^{s(2d - n_L^{in} - n_R^{out})}$$

operations because of the uncontrolled differential transitions in round 2 and 6.

Considering the limited-birthday paradox problem, with a generic method (modeling the permutation as a black box) a solution pair may be found with the same input/output properties with

$$\max\{2^{sd(d - n_L^{in})/2}, 2^{sd(d - n_R^{out})/2}, 2^{sd(d - n_L^{in} - n_R^{out})}\}$$

computations.

[00218] According to various embodiments, the best n_L^{in} , n_L^{out} , n_R^{in} , n_R^{out} values

may be chosen in order to minimize the distinguishing attack cost and the results are

given in Table 6. As long as they satisfy the constraints, n_L^{out} and n_R^{in} may have no

impact on the complexity except for the controlled part (which always may be ensured to

have a minimal cost of 2^S). Moreover, it may be observed that the best results may be obtained when $n_L^{\text{in}} = n_R^{\text{out}}$. Overall, all versions of PHOTON may provide a very comfortable security margin against state-of-the-art rebound-style attacks since only 8 rounds can be reached. It is to be noted that there is no key input for the PHOTON internal permutation and this may avoid any potential improvement based on a weak key schedule or by the increased amount of freedom degrees provided by the key. Moreover, the recently introduced internal differential attack that was applied on Grøstl may not work here since only one single internal permutation may be used (whereas the internal differential attack looks for properties between two very related permutations).

	P_{100}	P_{144}	P_{196}	P_{256}	P_{288}
$n_L^{\text{in}}, n_R^{\text{out}}$	4	5	6	7	5
computations	2^8	2^8	2^8	2^8	2^{24}
memory	2^4	2^4	2^4	2^4	2^8
generic	2^{10}	2^{12}	2^{14}	2^{16}	2^{16}

Table 6. Computation and memory complexities of the application of rebound-like attacks to the internal permutations of PHOTON. The attacks allow to distinguish 8 rounds of the permutation from an ideal permutation of the same size.

[00219] In the following cube testers and algebraic attacks for devices and methods according to various embodiments will be described.

[00220] The most recent developed cube testers and its zero-sum distinguishers have been applied to the PHOTON permutations, the best that could be found within practical

time complexity is at most 3 rounds for all PHOTON variants. It is to be noted that, in case of AES, "zero-sum" property is also referred as "balanced", found by the AES designers, in which 3-round balanced property is shown. There is no balanced property found for more than 3 AES rounds.

[00221] According to various embodiments, there are two types of Sboxes used in PHOTON, the PRESENT Sbox and the AES Sbox, each of algebraic degree 3 and 7 respectively. Above, it has been shown that the number of active Sbox is at least $(d+1)^2$ for any consecutive 4-round trail of PHOTON. One can easily check that $3 \text{ (resp. } 7) \cdot (d+1)^2 \cdot 3 \gg t$ for all PHOTON variants.

[00222] Concerning algebraic attacks, the PRESENT Sbox may be described by $e = 21$ quadratic equations in the $v = 8$ input/output-bit variables over $GF(2)$, while the AES Sbox may be described by $e = 40$ quadratic equations in the $v = 16$ input/output-bit variables over $GF(2)$. The entire system for the internal permutations of PHOTON therefore consists of $(d^2 \cdot N_r \cdot e)$ quadratic equations in $(d^2 \cdot N_r \cdot v)$ variables. For example, in the case of P_{144} used for PHOTON-128/16/16, it may end up with 9072 equations in 3456 variables. In comparison, the entire system for a fixed-key AES permutation may include 6400 equations in 2560 variables. While the applicability of algebraic attacks on AES may remain unclear, those numbers tend to indicate that PHOTON offers a comparable level of protection.

[00223] In the following, other cryptanalysis of device and methods according to various embodiments will be described.

[00224] The slide attack may originally be a block cipher cryptanalysis technique, but may be applied to sponge-like hash functions. The idea may be to exploit the degree of self-similarity of a permutation. In the case of PHOTON, all rounds of the internal permutation may be made different thanks to the round-dependent constants addition. Thus the slide attack may be impossible to perform at the permutation level. Moreover, the slide attack at the operating mode level may be impossible to apply here since the padding rule from PHOTON may force the last message block to be different from zero (which may prevent any sliding event).

[00225] Rotational cryptanalysis was proven to be quite successful against Addition-Rotation-XOR (ARX) primitives and advances were made on some SHA-3 candidates. The idea may be to study the evolution of a rotated variant of some input words through the round process. However, PHOTON may be an Sbox-oriented hash function and any rotation property in a cell may be directly removed by the application of the Sbox layer. One could look for rotation of cell positions in the internal state, but this may be unlikely to lead to an attack since the constants used in a PHOTON round may be all distinct and any position rotation property between columns or lines may be removed after the application of two rounds.

[00226] Integral attacks may be quite efficient against AES-based permutations and the known-key variant may be adapted to the PHOTON internal permutation cases. However, those attacks may only reach seven rounds with complexity $2^{s(2d-1)}$, which may be worse than what can be obtained with rebound-style attacks described above.

[00227] In the following, performances and comparison of devices and methods according to various embodiments will be described.

[00228] In the following, the tools used will be described. Furthermore, the hardware architectures and the optimizations done will be detailed. Furthermore, results according to various embodiments will be compared to previous work.

[00229] In the following, design flow according to various embodiments will be described.

[00230] According to various embodiments, Mentor Graphics ModelSimXE 6.4b and Synopsys DesignCompiler A-2007.12-SP1 may be used for functional simulation and synthesis of the designs to the Virtual Silicon (VST) standard cell library

UMCL18G212T3, which is based on the UMC L180 0.18_μm 1P6M logic process with a exemplary voltage of 1.8 V. Synopsys Power Compiler version A-2007.12-SP1 may be used to estimate the power consumption of our ASIC implementations. For synthesis and for power estimation the compiler may be advised to keep the hierarchy and use a clock frequency of 100 KHz, which may be a widely used operating frequency for RFID applications. It is to be noted that the wire-load model used, though it is the smallest available for this library, still may simulate the exemplary wire-load of a circuit with a size of around 10 000 GE.

[00231] In the following, architectures according to various embodiments will be described.

[00232] To show substantiate the hardware efficiency of the PHOTON family according to various embodiments, the flavors specified above have been implemented in VHDL and their post-synthesis performance has been simulated. Two architectures have been designed, one is fully serialized, i.e. performing operations on one cell per clock cycle, and aims for the smallest area possible. The second one is a d times parallelization

of the first architecture, thus performing operations on one row in one clock cycle, resulting in a significant speed-up. As can be seen in FIG. 15, the serialized design according to various embodiments includes six modules: MCS, State, IO, AC, SC, and Controller.

[00233] FIG. 15 shows a diagram 1500 illustrating hardware according to various embodiments. In 1502, a serial hardware architecture of PHOTON according to various embodiments is shown (left in FIG. 15). As an example for its component A_t it is depicted A_{100} with its sub-components in 1504 (right in FIG. 15).

[00234] According to various embodiments, IO may allow to 1) initialize the implementation with an all '0' vector, 2) input the IV, 3) absorb message chunks, and 4) forward the output of the State module to the AC module without further modification. Instead of using two Multiplexer and an XOR gate, two NAND and one XOR gate may be used, thereby reducing the gate count required from $s \cdot 7.33$ to $s \cdot 4.67$ GE.

[00235] According to various embodiments, State may include a $d \cdot d$ array of flip-flop cells storing s bits each. Every row may constitute a shift-register using the output of the last stage, i.e. column 0, as the input to the first stage (column $d - 1$) of the same row and the next row. Using this feedback functionality ShiftRows may be performed in $d - 1$ clock cycles with no additional hardware costs. Further, since MixColumnsSerial may be performed on column 0, also a vertical shifting direction may be desired for this column. Consequently, columns 0 and $d-1$ may consist of flip-flop cells with two inputs (6 GE), while columns 1 to $d - 2$ may consist of flip-flop cells with only one input (5 GE). The

overall gate count for this module may be $s \cdot d \cdot ((d - 2) \cdot 5 + 2 \cdot 6)$ GE and for all flavors it may occupy the majority of the area required (between 65 and 77.5%).

[00236] According to various embodiments, MCS may calculate the last row of A_t in one clock cycle. The result may be stored in the State module, that is in the last row of column 0, which may have been shifted upwards at the same time. Consequently, after d clock cycles, the MixColumnsSerial operation may be applied to an entire column. Then the whole state array may be rotated by one position to the left and the next column may be processed. In total $d \cdot (d + 1)$ clock cycles may be desired to perform MCS. As an example of the hardware efficiency of MCS, A100 is depicted in the upper and its sub-components in the lower right part of FIG. 15. Using the library according to various embodiments, for a multiplication by 2, 4 and 8, 2.66 GE, 4.66 GE, and 7 GE may be desired when using the irreducible polynomial $x + 1$, respectively. Therefore the choice of the coefficients may have only a minor impact on the overall gate count, as the majority may be required to sum up the intermediate results. For example, in the case of A100, 56 out of 75.33 GE may be desired for the XOR sum. The gate counts for the other matrices are: 80 GE, 99 GE, 145 GE, and 144 GE for A144, A196, A256, and A288, respectively.

[00237] According to various embodiments, AC may perform the AddConstant operation by XORing the sum of the round constant RC with the current internal constant IC. Furthermore, since AC may only be applied to the first column, the input to the XNOR gate may be gated with a NAND gate. Instead of using an AND gate in

combination with an XOR gate, this approach may allow to reduce the area required from $s \cdot 6.67$ to $s \cdot 6$ GE.

[00238] According to various embodiments, SC may perform the SubCells operation and may include a single instantiation of the corresponding Sbox. For $s = 4$, an optimized Boolean representation of the PRESENT Sbox may be used, which may only desire 22.33 GE and for $s = 8$, Canright's representation of the AES Sbox may be used which may desire 233 GE. It may take $d \cdot d$ clock cycles to perform AddConstant and SubCells on the whole state.

[00239] According to various embodiments, Controller may use a Finite State Machine (FSM) to generate all control signals required. Furthermore, also the round constants and the internal constants may be generated within this module, as their values are used for the transition conditions of the FSM. The FSM may include one idle state, one state for the combined execution of AC and SC, $d-1$ states for ShR (ShiftRows) and two states for MCS (one for processing one column and another one to rotate the whole state to the left). Its gate count may vary depending on d : 197 GE, 210 GE, 235 GE, and 254 GE for $d = 5; 6; 7; 8$, respectively.

[00240] In the following, results and comparisons will be described.

[00241] It may be assumed that the message is padded correctly and the IVs are loaded at the beginning of the operation. Then it may require $d \cdot d + (d-1) + d \cdot (d+1)$ clock cycles to perform one round of the permutation P, resulting in a total latency of $12 \cdot (2 \cdot d \cdot (d+1) - 1)$ clock cycles. Table 8 below compares the results according to various embodiments to previous works, sorted after increasing security levels. As can be seen, the devices and methods according to various embodiments may compete well in

terms of area requirements, since they may be 18% to 75% smaller compared to previous proposals with a similar security level.

[00242] According to various embodiments, PHOTON, the most lightweight hash function family known so far, very close to the theoretical optimum, may be provided. According to various embodiments, devices and methods may be on the AES design strategy, and a new mixing layer building method that is perfectly fit for small area scenarios is provided. This may allow to directly leverage the extensive work done on AES and AES-like hash functions so as to provide good confidence in the security of the scheme. Finally, PHOTON is not only the smallest, but it also achieves excellent area/throughput trade-offs and very acceptable performances have been obtained.

[00243] In Table 7, software implementation performances for the PHOTON variants according to various embodiments are shown. The processor used for the benchmarks is an Intel(R) Core(TM) i7 CPU Q 720 cadenced at 1.60GHz. Moreover, for comparison purposes, the speed of an AES permutation (without the key schedule) and a modified version of it with a serially computable MDS matrix instead (the 4x4 matrix A given above) is also shown. The table-based implementations reach the same speed for both versions.

	reference implementation	table-based implementation
PHOTON-80/20/16	19648	483
PHOTON-128/16/16	40189	834
PHOTON-160/36/36	28557	635
PHOTON-224/32/32	53561	730
PHOTON-256/32/32	21570	169

Table 7. Software performances in cycles per byte of the PHOTON variants for long messages.

[00244] In the following, PHOTON hardware performance according to various embodiments will be described.

[00245] Area requirements may be provided in GE, while the latency may be given in clock cycles for only the internal permutation P (or the internal block-cipher E), and the whole hash function H. Further metrics may be throughput in kbps and a figure of merit (FOM). In order to have a comparison for a real-world application and a best case scenario, the latter two metrics may be provided for 96-bit messages plus padding (that is a 96-bit message is padded with "1" and as many "0"s as required) and for 'long' messages. The parameters n , c , r and r' may stand for the hash output size, the capacity, the input bitrate and the output bitrate respectively. Finally, the column "Pre" may give the provided preimage resistance security and "Col" the claimed collision resistance security.

Name	Ref.	Parameters				Security		Performance							
		n	c	r	r'	Pre	Col	Area [GE]	Latency [clk]		Throughput [kbps]		FOM [nb/clk/GE ²]		
									P/E	H	long	96-bit	long	96-bit	
64-bit security (preimage only)															
SQUASH	[68]	64	x	x	x	64	0	2646	31800	31800	0.2	0.15	0.29	0.14	
DM-PRESENT-80	[16]	64	64	80	x	64	32	1600	547	547	14.63	8.78	57.13	28.56	
DM-PRESENT-80	[16]	64	64	80	x	64	32	2213	33	33	242.42	145.45	495.01	247.50	
DM-PRESENT-128	[16]	64	64	128	x	64	32	1886	559	559	22.90	22.90	64.37	64.37	
DM-PRESENT-128	[16]	64	64	128	x	64	32	2530	33	33	387.88	387.88	605.98	605.98	
PHOTON-80/20/16		80	80	20	16	64	40	865	708	3540	2.82	1.51	37.73	20.12	
PHOTON-80/20/16		80	80	20	16	64	40	1168	132	660	15.15	8.08	111.13	59.27	
64-bit security															
U-QUARK	[4]	128	128	8	8	120	64	1379	544	8704	1.47	0.63	7.73	3.31	
U-QUARK	[4]	128	128	8	8	120	64	2392	68	1088	11.76	5.04	20.56	8.81	
H-PRESENT-128	[16]	128	128	64	x	128	64	2330	559	559	11.45	8.59	21.09	15.82	
H-PRESENT-128	[16]	128	128	64	x	128	64	4256	32	32	200.00	150.00	110.41	82.81	
ARMADILLO2-B	[6]	128	128	64	x	128	64	4353	256	256	25.00	18.75	13.19	9.90	
ARMADILLO2-B	[6]	128	128	64	x	128	64	6025	64	64	100.00	75.00	27.55	20.66	
PHOTON-128/16/16		128	128	16	16	112	64	1122	996	7968	1.61	0.69	12.78	5.48	
PHOTON-128/16/16		128	128	16	16	112	64	1708	156	1248	10.26	4.4	35.15	15.06	
80-bit security															
D-QUARK	[4]	160	160	16	16	144	80	1702	704	7040	2.27	0.85	7.85	2.94	
D-QUARK	[4]	160	160	16	16	144	80	2819	88	880	18.18	6.42	22.88	8.58	
ARMADILLO2-C	[6]	160	160	80	x	160	80	5406	320	320	25.00	15.00	8.55	5.13	
ARMADILLO2-C	[6]	160	160	80	x	160	80	7492	80	80	100.00	60.00	17.82	10.69	
PHOTON-160/36/36		160	160	36	36	124	80	1396	1332	6660	2.70	1.03	13.87	5.28	
PHOTON-160/36/36		160	160	36	36	124	80	2117	180	900	20	7.62	44.64	17.01	
112-bit security															
S-QUARK	[4]	224	224	32	32	192	112	2296	1024	7168	3.13	0.94	5.93	1.78	
S-QUARK	[4]	224	224	32	32	192	112	4640	64	448	50.00	15.00	23.22	6.97	
PHOTON-224/32/32		224	224	32	32	192	112	1736	1716	12012	1.86	0.56	6.19	1.86	
PHOTON-224/32/32		224	224	32	32	192	112	2786	204	1428	15.69	4.71	20.21	6.06	
128-bit security															
ARMADILLO2-E	[6]	256	256	128	x	256	128	8653	512	0	25.00	18.75	3.34	2.50	
ARMADILLO2-E	[6]	256	256	128	x	256	128	11914	128	0	100.00	75.00	7.05	5.28	
PHOTON-256/32/32		256	256	32	32	224	128	2177	996	7968	3.21	0.88	6.78	1.85	
PHOTON-256/32/32		256	256	32	32	224	128	4362	156	1248	20.51	5.59	10.78	2.94	

Table 8. Overview of parameters, security level, and performance of several lightweight hash functions.

[00246] While for practical applications - and for example for ultra-constrained devices, such as passive RFID-tags, that may be fields where PHOTON may be applied - the peak and average power consumption is important, power figures are not included in Table 8 for the following reasons: (1) the power consumption may strongly depend on the technology used and (2) may not be compared between different technologies in a fair manner. Furthermore, (3) simulated power figures strongly depend on the simulation method used, and the effort spent. Instead, it is just briefly listed the simulated power figures for our proposals here: 1.59, 2.29, 2.74, 4.01, and 4.55 μW for serialized implementation of PHOTON-80/20/16, PHOTON-128/16/16, PHOTON-160/36/36, PHOTON-224/32/32, and PHOTON-256/32/32, respectively. The d-parallel implementations require 2.7, 3.45, 4.35, 6.5, and 8.38 μW , respectively. Thus it may be concluded that all PHOTON flavors may be suitable for passive RFID tags.

[00247] In the following, PHOTON Sboxes according to various embodiments will be described.

[00248] For cells of $s = 4$ bits, the SubCells layer may use the PRESENT Sbox (in hexadecimal display):

$$SBOX_{PRE} = [0xc, 0x5, 0x6, 0xb, 0x9, 0x0, 0xa, 0xd, 0x3, 0xe, 0xf, 0x8, 0x4, 0x7, 0x1, 0x2].$$

[00249] For cells of $s = 8$ bits, the SubCells layer may use the AES Sbox (in hexadecimal display):

$SBOX_{AES} =$

[0xca, 0x82, 0xc9, 0x7d, 0xfa, 0x59, 0x47, 0xf0, 0xad, 0xd4, 0xa2, 0xaf, 0x9c, 0xa4, 0x72, 0xc0, 0xb7, 0xfd, 0x93, 0x26, 0x36, 0x3f, 0xf7, 0xcc, 0x34, 0xa5, 0xe5, 0xf1, 0x71, 0xd8, 0x31, 0x15, 0x04, 0xc7, 0x23, 0xc3, 0x18, 0x96, 0x05, 0x9a, 0x07, 0x12, 0x80, 0xe2, 0xeb, 0x27, 0xb2, 0x75, 0x09, 0x83, 0x2c, 0x1a, 0x1b, 0x6e, 0x5a, 0xa0, 0x52, 0x3b, 0xd6, 0xb3, 0x29, 0xe3, 0x2f, 0x84, 0x53, 0xd1, 0x00, 0xed, 0x20, 0xfc, 0xb1, 0x5b, 0x6a, 0xcb, 0xbe, 0x39, 0x4a, 0x4c, 0x58, 0xcf, 0xd0, 0xef, 0xaa, 0xfb, 0x43, 0x4d, 0x33, 0x85, 0x45, 0xf9, 0x02, 0x7f, 0x50, 0x3c, 0x9f, 0xa8, 0x51, 0xa3, 0x40, 0x8f, 0x92, 0x9d, 0x38, 0xf5, 0xbc, 0xb6, 0xda, 0x21, 0x10, 0xff, 0xf3, 0xd2, 0xcd, 0x0c, 0x13, 0xec, 0x5f, 0x97, 0x44, 0x17, 0xc4, 0xa7, 0x7e, 0x3d, 0x64, 0x5d, 0x19, 0x73, 0x60, 0x81, 0x4f, 0xdc, 0x22, 0x2a, 0x90, 0x88, 0x46, 0xee, 0xb8, 0x14, 0xde, 0x5e, 0x0b, 0xdb, 0xe0, 0x32, 0x3a, 0x0a, 0x49, 0x06, 0x24, 0x5c, 0xc2, 0xd3, 0xac, 0x62, 0x91, 0x95, 0xe4, 0x79, 0xe7, 0xc8, 0x37, 0x6d, 0x8d, 0xd5, 0x4e, 0xa9, 0x6c, 0x56, 0xf4, 0xea, 0x65, 0x7a, 0xae, 0x08, 0xba, 0x78, 0x25, 0x2e, 0x1c, 0xa6, 0xb4, 0xc6, 0xe8, 0xdd, 0x74, 0x1f, 0x4b, 0xbd, 0x8b, 0x8a, 0x70, 0x3e, 0xb5, 0x66, 0x48, 0x03, 0xf6, 0x0e, 0x61, 0x35, 0x57, 0xb9, 0x86, 0xc1, 0x1d, 0x9e, 0xe1, 0xf8, 0x98, 0x11, 0x69, 0xd9, 0x8e, 0x94, 0x9b, 0x1e, 0x87, 0xe9, 0xce, 0x55, 0x28, 0xdf, 0x8c, 0xa1, 0x89, 0x0d, 0xbf, 0xe6, 0x42, 0x68, 0x41, 0x99, 0x2d, 0x0f, 0xb0, 0x54, 0xbb, 0x16].

[00250] In the following, PHOTON mixing matrices according to various

embodiments will be described. $x^4 + x + 1$ may be used as the irreducible polynomial for

multiplication in $GF(2^4)$, and this may apply to all permutations with 4-bit cells, e.g.,

A_{100} , A_{144} , A_{196} , and A_{256} . $x^8 + x^4 + x^3 + x + 1$, as in AES, may be used for

multiplication in $GF(2^8)$ as used in A_{288} .

$$(A_{100})^5 = \begin{pmatrix} 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 1 & 2 & 9 & 9 & 2 \end{pmatrix}^5 = \begin{pmatrix} 1 & 2 & 9 & 9 & 2 \\ 2 & 5 & 3 & 8 & 13 \\ 13 & 11 & 10 & 12 & 1 \\ 1 & 15 & 2 & 3 & 14 \\ 14 & 14 & 8 & 5 & 12 \end{pmatrix}$$

$$(A_{144})^6 = \begin{pmatrix} 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 1 & 2 & 8 & 5 & 8 & 2 \end{pmatrix}^6 = \begin{pmatrix} 1 & 2 & 8 & 5 & 8 & 2 \\ 2 & 5 & 1 & 2 & 6 & 12 \\ 12 & 9 & 15 & 8 & 8 & 13 \\ 13 & 5 & 11 & 3 & 10 & 1 \\ 1 & 15 & 13 & 14 & 11 & 8 \\ 8 & 2 & 3 & 3 & 2 & 8 \end{pmatrix}$$

$$(A_{196})^7 = \begin{pmatrix} 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 1 & 4 & 6 & 1 & 1 & 6 & 4 \end{pmatrix}^7 = \begin{pmatrix} 1 & 4 & 6 & 1 & 1 & 6 & 4 \\ 4 & 2 & 15 & 2 & 5 & 10 & 5 \\ 5 & 3 & 15 & 10 & 7 & 8 & 13 \\ 13 & 4 & 11 & 2 & 7 & 15 & 9 \\ 9 & 15 & 7 & 2 & 11 & 4 & 13 \\ 13 & 8 & 7 & 10 & 15 & 3 & 5 \\ 5 & 10 & 5 & 2 & 15 & 2 & 4 \end{pmatrix}$$

$$(A_{256})^8 = \begin{pmatrix} 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 2 & 4 & 2 & 11 & 2 & 8 & 5 & 6 \end{pmatrix}^8 = \begin{pmatrix} 2 & 4 & 2 & 11 & 2 & 8 & 5 & 6 \\ 12 & 9 & 8 & 13 & 7 & 7 & 5 & 2 \\ 4 & 4 & 13 & 13 & 9 & 4 & 13 & 9 \\ 1 & 6 & 5 & 1 & 12 & 13 & 15 & 14 \\ 15 & 12 & 9 & 13 & 14 & 5 & 14 & 13 \\ 9 & 14 & 5 & 15 & 4 & 12 & 9 & 6 \\ 12 & 2 & 2 & 10 & 3 & 1 & 1 & 14 \\ 15 & 1 & 13 & 10 & 5 & 10 & 2 & 3 \end{pmatrix}$$

$$(A_{288})^6 = \begin{pmatrix} 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 2 & 3 & 1 & 2 & 1 & 4 \end{pmatrix}^6 = \begin{pmatrix} 2 & 3 & 1 & 2 & 1 & 4 \\ 8 & 14 & 7 & 9 & 6 & 17 \\ 34 & 59 & 31 & 37 & 24 & 66 \\ 132 & 228 & 121 & 155 & 103 & 11 \\ 22 & 153 & 239 & 111 & 144 & 75 \\ 150 & 203 & 210 & 121 & 36 & 167 \end{pmatrix}$$

[00251] In the following, PHOTON constants according to various embodiments will be described.

[00252] According to various embodiments, the round constants may have been generated by a 4-bit linear feedback shift register with maximum cycle length, while the internal constants ICd(i) may have been generated by shift registers with a cycle length of

d. It is to be noted that this design decision may be made with hardware implementations in mind. Since one may desire these counters for the control logic, the generation of the round constant and the internal constants may be basically for free. For all variants, shift registers may be used with $l = 3$ bits, except for $d = 8$, where we $l = 4$ may be used. Let us denote the internal state of the shift register with $X_r = (x_{l-1}, \dots, x_1, x_0)$, where $x_i = \{0, 1\}$, and $X_0 = (0, \dots, 0, 0)$. Then in each update iteration the new content of the shift register may be given by $X_{r+1} = (x_{l-2}, \dots, x_0, FB(X_r))$, where $FB(X_r)$ may be the feedback function. For the round constants $FB(X_r) = x_3 \text{ XNOR } x_2$ may be chosen, while the choices for the feedback functions for the internal constants may be like shown in Table 9. Tables 10 to 13 display constants for all square sizes, round numbers and row positions.

d	5	6	7	8
$FB(X_r)$	$x_2 \text{ NOR } x_1$	$\text{NOT } x_2$	$x_2 \text{ XNOR } x_1$	$\text{NOT } x_3$
$IC_d(\cdot)$	$[0, 1, 3, 6, 4]$	$[0, 1, 3, 7, 6, 5]$	$[0, 1, 2, 5, 3, 6, 4]$	$[0, 1, 3, 7, 15, 14, 12, 8]$

Table 9. Feedback functions for internal constants generation.

round row	1	2	3	4	5	6	7	8	9	10	11	12
0	1	3	7	14	13	11	6	12	9	2	5	10
1	0	2	6	15	12	10	7	13	8	3	4	11
2	2	0	4	13	14	8	5	15	10	1	6	9
3	7	5	1	8	11	13	0	10	15	4	3	12
4	5	7	3	10	9	15	2	8	13	6	1	14

Table 10. Constants for $d = 5$.

round row	1	2	3	4	5	6	7	8	9	10	11	12
0	1	3	7	14	13	11	6	12	9	2	5	10
1	0	2	6	15	12	10	7	13	8	3	4	11
2	2	0	4	13	14	8	5	15	10	1	6	9
3	6	4	0	9	10	12	1	11	14	5	2	13
4	7	5	1	8	11	13	0	10	15	4	3	12
5	5	7	3	10	9	15	2	8	13	6	1	14

Table 11. Constants for $d = 6$.

round row	1	2	3	4	5	6	7	8	9	10	11	12
0	1	3	7	14	13	11	6	12	9	2	5	10
1	0	2	6	15	12	10	7	13	8	3	4	11
2	3	1	5	12	15	9	4	14	11	0	7	8
3	4	6	2	11	8	14	3	9	12	7	0	15
4	2	0	4	13	14	8	5	15	10	1	6	9
5	7	5	1	8	11	13	0	10	15	4	3	12
6	5	7	3	10	9	15	2	8	13	6	1	14

Table 12. Constants for $d = 7$.

round row	1	2	3	4	5	6	7	8	9	10	11	12
0	1	3	7	14	13	11	6	12	9	2	5	10
1	0	2	6	15	12	10	7	13	8	3	4	11
2	2	0	4	13	14	8	5	15	10	1	6	9
3	6	4	0	9	10	12	1	11	14	5	2	13
4	14	12	8	1	2	4	9	3	6	13	10	5
5	15	13	9	0	3	5	8	2	7	12	11	4
6	13	15	11	2	1	7	10	0	5	14	9	6
7	9	11	15	6	5	3	14	4	1	10	13	2

Table 13. Constants for $d = 8$.

[00253] In the following, test vectors according to various embodiments will be described. Below are test vectors for all described flavours of PHOTON. The absorbing and squeezing position of the state array is underlined.

	IV	m	$P(m)$
PHOTON-80/20/16	<u>0 0 0 0 0</u> 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 4 1 4 1 0	0 0 0 0 0	<u>3 3 D 5 F</u> 6 2 9 B 9 5 C 4 8 1 6 5 C E 7 B 7 7 0 C
PHOTON-128/16/16	<u>0 0 0 0 0 0</u> 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 2 0 1 0 1 0	0 0 0 0	<u>9 5 F C 3 C</u> E 2 2 A 2 A 6 3 2 D 6 F E B 4 E 0 B 6 2 5 9 2 D 8 D 0 3 2 9
PHOTON-160/36/36	<u>0 0 0 0 0 0 0 0</u> <u>0 0 0 0 0 0 0 0</u> 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 2 8 2 4 2 4	0 0 0 0 0 0 0 0 0 0	<u>1 F 0 D 4 A 1</u> <u>D D 0 A 3 1 D</u> E C F 5 B 6 9 B 6 6 E 0 C 8 F 6 4 4 C E E E 9 0 2 0 F 4 3 A 9 D E 7 4
PHOTON-224/32/32	<u>0 0 0 0 0 0 0 0 0 0</u> 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 3 8 2 0 2 0	0 0 0 0 0 0 0 0 0 0	<u>1 7 3 0 4 2 4 2</u> 9 C F 2 6 E 1 0 8 D 3 D 9 C F 9 0 0 E 2 7 B D C C 6 2 9 B 3 D 1 A F 4 1 F 1 C B 7 4 8 3 F C C 0 8 9 1 6 B 8 2 C
PHOTON-256/32/32	<u>00 00 00 00 00 00</u> 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 40 44 44	00 00 00 00	<u>AA 15 0E E5 4A 60</u> FD 24 5D 5E 15 B2 F8 78 85 4A 53 3E 80 50 B3 C0 F9 9A 8C 6F 55 0C 3D CF B1 E4 74 CB 61 7B

[00254] The following table shows the savings in terms of space according to various embodiments.

	Area of the smallest AES			
	MixColumns	can be optimized combinational logic sum	cannot be optimized sequential	total
State of the Art	263	633	1767	2400
Our Proposal	74	444	1767	2211
Savings	71.86%	29.86%	0.00%	7.88%

[00255] All figures may be in Gate Equivalents (GE).

[00256] On GE may be the silicon area of the chip divided by the area of a NAND-gate with the lowest driving strength. The smallest hardware realization of AES, the industry standard, requires 2400 GE.

[00257] 1767 GE out of 2400 GE (= 74%) may be required for sequential logic (i.e. storage) and hence may not be optimized. Out of the remaining 633 GE for combinational logic Mixcolumns occupies 263 GE, which is the largest component. According to various embodiments, the area of MixColumns may be reduced by almost 72 %. As a consequence, the overall chip area for a highly optimized AES chip may be further decreased by 8 %.

[00258] FIG. 16 shows a data transmission system 1600 according to various embodiments. A first computing device 1602 may desire to send information to a second computing device 1612. For example, the first computing device 1602 and the second computing device 1612 each may be a computer, and the data to be transmitted may include or may be a data file or may include or may be data input by a user. In another example, the first computing device 1602 may be an RFID tag and the second computing device 1612 may be a RFID reader, and the data to be transmitted may include or may be

RFID related data, for example data identifying the RFID tag. In yet another example, the first computing device 1602 may be an RFID reader and the second computing device 1612 may be an RFID tag, and the data to be transmitted may include or may be RFID reader related data, for example data identifying (and/ or authorizing) the RFID reader.

[00259] In the first computing device 1602, original data 1604 may be provided. For example, the original data 1604 may include or may be data stored in a file, or may include or may be data input by a user, or may include or may be data hardwired in a circuit of the first computing device 1602. The first computing device 1604 may perform any of the above described methods according to various embodiments to generate send data 1608 (like indicated by a first arrow 1606) which is to be sent to the second computing device 1612 (like indicated by a second arrow 1610). The send data 1608 may be or may include hashed data based on the original data 1604, or may be or may include ciphered data based on the original data 1604, or may include or may be encrypted data based on the original data 1604.

[00260] The second computing device 1612 may receive the data sent by the first computing device 1602, and may process these received data 1614 like indicated by a third arrow 1616 to determine processed data 1618. This processing may include any of the above described methods according to various embodiments.

[00261] While the invention has been particularly shown and described with reference to specific embodiments, it should be understood by those skilled in the art that various changes in form and detail may be made therein without departing from the spirit and scope of the invention as defined by the appended claims. The scope of the invention is

thus indicated by the appended claims and all changes which come within the meaning and range of equivalency of the claims are therefore intended to be embraced.

Claims

What is claimed is:

1. A device for computer-based generating of a mixing matrix for cryptographic processing of data, the device comprising:

a square matrix determination circuit configured to determine a square matrix with non-zero elements only in one diagonal or sub-diagonal and in one row or column;

an exponentiation determination circuit configured to determine an exponentiation of the square matrix as a candidate for a mixing matrix;

a criterion determination circuit configured to determine whether the candidate fulfils a pre-determined criterion; and

a candidate determination circuit configured to determine the candidate as the mixing matrix if the candidate fulfils a pre-determined criterion.
2. The device of claim 1,

wherein the one diagonal or sub-diagonal is the first upper sub-diagonal.
3. The device of claim 1 or 2,

wherein the one row or column is the last row.
4. The device of any one of claims 1 to 3,

wherein the exponentiation is an exponentiation to the power of the number of rows of the square matrix.

5. The device of any one of claims 1 to 4,
wherein the pre-determined criterion is a criterion based on a maximum distance separable code criterion.
6. The device of any one of claims 1 to 5,
wherein the one diagonal or sub-diagonal is filled with ones.
7. An encrypting device, comprising:
a mixing circuit configured to use as a mixing matrix a square matrix that can be written as an exponentiation of a square matrix with non-zero elements only in one diagonal or sub-diagonal and in one row or column.
8. The encrypting device of claim 7,
wherein the one diagonal or sub-diagonal is the first upper sub-diagonal.
9. The encrypting device of claim 7 or 8,
wherein the one row or column is the last row.
10. The encrypting device of claims 7 to 9,

wherein the exponentiation is an exponentiation to the power of the number of rows of the square matrix.

11. The encrypting device of any one of claims 7 to 10,
wherein the exponentiation fulfils a pre-determined criterion based on a maximum distance separable code criterion.
12. The encrypting device of any one of claims 7 to 11,
wherein the one diagonal or sub-diagonal is filled with ones.
13. An encrypting device, comprising:
a mixing circuit configured to apply a mixing matrix by repeatedly applying a square matrix with non-zero elements only in one diagonal or sub-diagonal and in one row or column.
14. The encrypting device of claim 13,
wherein the one diagonal or sub-diagonal is the first upper sub-diagonal.
15. The encrypting device of claim 13 or 14,
wherein the one row or column is the last row.
16. The encrypting device of any one of claims 13 to 15,

wherein the mixing matrix circuit is configured to repeatedly apply the mixing matrix by a number of times equal to the power of the number of rows of the square matrix.

17. The encrypting device of any one of claims 13 to 16,
wherein the mixing matrix fulfils a pre-determined criterion based on a maximum distance separable code criterion.
18. The encrypting device of any one of claims 13 to 17,
wherein the one diagonal or sub-diagonal is filled with ones.
19. A method for computer-based generating of a mixing matrix for cryptographic processing of data, the method comprising:
determining a square matrix with non-zero elements only in one diagonal or sub-diagonal and in one row or column;
determining an exponentiation of the square matrix as a candidate for a mixing matrix;
determining whether the candidate fulfils a pre-determined criterion; and
determining the candidate as the mixing matrix if the candidate fulfils a pre-determined criterion.
20. The method of claim 19,
wherein the one diagonal or sub-diagonal is the first upper sub-diagonal.

21. The method of claim 19 or 20,
wherein the one row or column is the last row.
22. The method of any one of claims 19 to 21,
wherein the exponentiation is an exponentiation to the power of the number of
rows of the square matrix.
23. The method of any one of claims 19 to 22,
wherein the pre-determined criterion is a criterion based on a maximum distance
separable code criterion.
24. The method of any one of claims 19 to 23,
wherein the one diagonal or sub-diagonal is filled with ones.
25. An encrypting method, comprising:
using as a mixing matrix a square matrix that can be written as an exponentiation
of a square matrix with non-zero elements only in one diagonal or sub-diagonal
and in one row or column.
26. The method of claim 25,
wherein the one diagonal or sub-diagonal is the first upper sub-diagonal.

27. The method of claim 25 or 26,
wherein the one row or column is the last row.
28. The method of any one of claims 25 to 27,
wherein the exponentiation is an exponentiation to the power of the number of
rows of the square matrix.
29. The method of any one of claims 25 to 28,
wherein the exponentiation fulfils a pre-determined criterion based on a maximum
distance separable code criterion.
30. The method of any one of claims 25 to 29,
wherein the one diagonal or sub-diagonal is filled with ones.
31. An encrypting method, comprising:
applying a mixing matrix by repeatedly applying a square matrix with non-zero
elements only in one diagonal or sub-diagonal and in one row or column.
32. The method of claim 31,
wherein the one diagonal or sub-diagonal is the first upper sub-diagonal.
33. The method of claim 31 or 32,
wherein the one row or column is the last row.

34. The method of any one of claims 31 to 34,
wherein the mixing matrix is repeatedly applied by a number of times equal to the power of the number of rows of the square matrix.
35. The method of any one of claims 31 to 34,
wherein the mixing matrix fulfils a pre-determined criterion based on a maximum distance separable code criterion.
36. The method of any one of claims 31 to 35,
wherein the one diagonal or sub-diagonal is filled with ones.