Journal of Scientific & Industrial Research
Vol. 82, January 2023, pp. 142-150
DOI: 10.56042/jsir.v82i1.69938

DeeR-Hash: A lightweight hash construction for Industry 4.0 / IoT

Deena Nath Gupta* & Rajendra Kumar

Jamia Millia Islamia, New Delhi, Delhi 110 025, India

Received 2 August 2022; revised 18 September 2022; accepted 06 October 2022

Industry 4.0 and IoT are emerging computing environments for low energy devices. Implementing complex security mechanisms in such environment is challenging. A lightweight and energy aware hashing provides high security to the devices under these environments. Earlier hash algorithms such as SHA and MD5 were very complex and hence are not suitable for the energy constrained devices. Similar hashing algorithm is needed for low energy devices as well. The authors proposed a sponge based hashing algorithm that is capable of providing a security up to second preimage attack to the devices communicating under such constrained environments. The methodology of the proposed design is derived from some existing lightweight hash constructions such as Photon, Quark, Gluon, and Spongent. The steps in the algorithm of DeeR-Hash include the steps for DeeRSponge and DeeRStateUpdate as well. To construct the sponge for the proposed hashing, the authors had taken the value of b, r, and c as 80, 2, and 78 respectively. After implementing the algorithm in a tag-reader scenario, the authors find that it is taking only 483 GE for 80-bits digest and is suitable for a lightweight cryptographic environment. The avalanche effect produced by the proposed algorithm further strengthens the security claim of the authors. Comparing other related work in this area, the authors claim that the required area in ASIC in lowest.

Keywords: Collision resistance, Cryptography, Energy aware, Lightweight, Security

Introduction

The main focus of Industry 4.0 / IoT is to provide a secure computing environment for the resource constrained devices in a lightweight fashion. This includes a small registration mechanism, a low energy mutual authentication, and a lightweight encryption scheme. Apart from all these, storage capacity and transmission range are also the driving factors of Industry 4.0/IoT. The scope of this article is restricted to lightweight encryption scheme only. A lightweight encryption scheme can be a block or stream cipher mechanism. The computer science research community is blessed with a lot of work in this field from many researchers from all over the globe. Some of the well-known lightweight ciphers are GRAIN, TRIVIUM, PRESENT, HIGH, HUMMINGBIRD, BEAN, CLEFIA, KATAN, and KTANTAN.[1–6]

Although the above-mentioned ciphers are meant for constrained environment, they don't fully satisfy the requirements of Industry 4.0/IoT. There are many devices in Industry 4.0 /IoT those hardly get a power source to recharge themselves. This means that they have to manage their whole life with the stored battery power only. Keeping this constraint in mind, the research community decides to design the security mechanisms including only the operations those required very less amount of computation power. For this purpose, two methodologies are commonly used. One is Pseudo Random Number Generator (PRNG) and the other is Sponge based Hash Construction.[7,8] The common approach used in both of the methods is shift operation. Because of the fact that the shift operation works with lowest computing power requirement, the above mentioned two methods are suitable for Industry 4.0/IoT upon satisfying the lightweight requirements of the environment.

The development of PRNG and sponge based hash construction is somewhat similar. In this article, the authors are mainly focusing the development of sponge based hash constructions. Hashing is a very old methodology for securing the messages. It takes the bits from input message to perform a specific operation. After completing its operation with all the input bits, it produces a digest as output. This digest, known as message digest or hash value, is a fixed-size value for any size of input message bits. The sponge construction serves the purpose very effectively. A sponge construction is subdivided into two parts, the first one is absorbing phase and the second is squeezing phase.[9]

The absorbing phase is responsible for implementing a strong confusion mechanism. Unlike

*Author for Correspondence
E-mail: prof.dev.cse@gmail.com

the older hash techniques those uses S-boxes, the sponge construction works only with shift and XOR operations. Different hash constructions use different combinations of shift registers for their update process. Two types of shift registers are in use, one is Linear Feedback Shift Registers (LFSR) and the second one is Non-Linear Feedback Shift Registers (NLFSR). The output from a LFSR can be guessed easily. The same is not true for a NLFSR. Also, a LFSR can work with low power, whereas the NLFSR needs more computing power.[10] Keeping both the factors in mind, the researchers started using a mix of both the flavors.

Another important factor of using a sponge construction is the security against preimage and collisions. The width of a sponge state 'b' is subdivided into two parts; one is the rate 'r' and second is the capacity 'c'. Where the capacity should be smaller than the size of hash value and the rate should be as small as possible. The collision resistance of a sponge construction under the flat sponge claim is $2^{\min(cp,n)/2}$, the (second) preimage resistance is $2^{\min(cp/2,n)}$.The main attraction of a sponge construction over the Merkle-Damgard construction is its resistance over the second preimage attack.

Related Work

The authors studied several well-known sponge-based hash designs. The most popular among them are Photon, Quark, Gluon, Spongent, and Hash-One.[6,11–13] Photon is hardware oriented lightweight hash design based on sponge construction. Its internal state is presented as a matrix whose inputs are either 4-bits or 8-bits. Photon uses a fix key for permutation likewise in AES. The 12 steps of Photon include AddConstant, SubCells, ShiftRows, and AES like MixColoumSerial. The five variants of Photon are PHOTON-80/20/16, PHOTON-128/16/16, PHOTON-160/36/36, PHOTON-224/32/32, and PHOTON-256/32/32.

Quark is a lightweight hash design mainly based on the functionality of Grain and Katan. Three versions of Quark are proposed namely, U-Quark, S-Quark, and D-Quark. The lightest of them are U-Quark. Quark updates its internal state 4b times with the help of two NFSR and one LFSR. One linear Boolean function is used to update the LFSR while three non-Boolean functions are used to update the NFSR. Apart from all these, one non-Boolean function is used to update both the NFSR simultaneously. The design of Gluon is based on two stream ciphers F-FCSR-v3 and X-FCSR-v2.[14] A world ring FCSR based on a main shift register and a carry register is used in the design of Gluon. The three variants of Gluon are Gluon-128/8, Gluon 160/16 and Gluon 224/32. Spongent is based on PRESENT like permutation. Five variants of Spongent have been proposed namely, SPONGENT-88/88/40, SPONGENT-128/128/64, SPONGENT-160/160/80, SPONGENT-224/224/112, and SPONGENT-256/256/128.

The Hash-One claim to operate with lowest Gate Equivalent (GE) among the above mentioned lightweight sponge based hash constructions. The authors studied the methodology used in Hash-One and found some serious security flaws in it. Although the Hash-One claims to be the lightest one yet it is not suitable to be used in the applications those needs high security, for example, life critical healthcare IoT and real time industrial IoT. Some of the raised issues from the Hash-One are as follows:

1. What if someone manage to know the initial entries in S? The entries of P and Q will be known.
2. If someone knows the array positions of P and Q those are used in function then the values of P_f, Q_f, and L_f will be calculated easily so that the values of T_1 and T_2.
3. If someone knows the steps involved in the algorithm, he/she can easily hash his/her own message. This message might be used to initiate the communication with the internal node/device.
4. The 324 times state update for first and last message bits and 162 times state update for all other internal bits will take a lots of computation power and hence this is not suitable for the energy constrained devices.

To make up these shortfalls, the authors proposed a new hash design named DeeR-Hash. DeeR-Hash is lightweight yet highly secure hash algorithm to be used in cryptographic applications under Industry 4.0/IoT. DeeR-Hash is fast enough to be used in real time industrial applications as well as highly secured to be used in life critical healthcare applications. The name DeeR-Hash comes from the initials of its inventers Deena Nath Gupta and Rajendra Kumar.

Methodology

In this section, the authors will describe everything about scheme, methodology and architecture of the proposed hash construction. There will be total nine random selections in sender side. The receiver uses these values to compute hash over the same value. For example, suppose a Tag is having the ID of Reader at

the time of registration. When it comes into the periphery of a Reader, the Reader will give its Hashed ID to Tag along with other credentials. The Tag will use these credentials over the Reader's ID it is already having. If the computed hash value at Tag side will match the Hashed ID given by Reader, then only the verification will be successful.

Scheme

The proposed design is based on the fact that security is the utmost priority of Industry 4.0/IoT. The lower area requirement is welcomed but not at the cost of security. For the same, it is worth to notice that randomization is a proven concept of uncertainty. Keeping this in mind, the authors proposed their scheme for DeeR-Hash. The authors first take a set of eight well-known mathematical constants. One will be taken randomly for one complete hashing process. A list containing eight mathematical constants is shown in Table 1 for reference.

The binary equivalent (up to 80-bits) of the chosen mathematical constant will be placed in an array, authors named it D here. This array will be subdivided into two arrays E and F of 40-bits each. The first 40-bits of D will be placed as it is into E and the remaining 40-bits will be placed into F. Again, four random selections will be performed on each of the arrays E and F. This will result in four array positions from E as well as from F. Suppose, V_1,V_2,V_3, and V_4 are the randomly selected array positions from E and U_1, U_2, U_3, and U_4 are the randomly selected array positions from F. These selections will be effective for one complete hashing process only.

After successfully running of the entire algorithm, the sender will have packet containing following items:

1. The 8-bit binary equivalent of chosen number of Mathematical Constant (MC) from the list.
2. A total of 64-bits (32-bits for E and 32-bits for F) related to selected array positions. Each position number will be converted to its 8-bit binary equivalent.
3. The computed hash value H of 80-bits length.

The packet send to receiver will look like:

[MC (8-bits) || $V_1V_2V_3V_4U_1U_2U_3U_4$ (64-bits) || H (80-bits)]

The receiver of the packet will use first 8-bits to identify the mathematical constant used in the hashing process. Next 64-bits will give the details about array positions those are used to calculate the values of E_f and F_f. Remaining 80-bits will be used to match the computed hash value.

Proposed Method

The design of DeeR-Hash is highly motivated from the well-known lightweight hash design Hash-One. Although the authors followed the working style of Hash-One, they included their own security mechanism at each desired level. Also, the number of state update is reduced to significant amount of time in DeeR-Hash to make it suitable for low energy devices. The design of DeeR-Hash follows the true definition of a sponge construction, i.e. variable length input and arbitrary length output. The algorithm to construct DeeR-Hash is given here for easiness in understanding.

Algorithm: DeeR-Hash

Step1- Select one constant randomly from the given list of eight mathematical constants

Step2- Randomly choose four positions from array E[40] and four positions from array F[40]

Step3- Call DeeRSponge

Step4- Send the packet to receiver

Algorithm: DeeRSponge

Initialization Phase:

Step1- Make the length of message multiple of r (=2), do padding of one bit if required

Absorbing Phase:

Step2- Perform XOR of M_1 with D_{39} and M_2 with D_{79} and store the resultant at D_{39} and D_{79} respectively

Step3- Increment M by 2 and call

Table 1 — Mathematical constants used for initialization

Sr. No.	Name	Value
1	Archimedes' constant π	3.14159 26535 89793 23846 26433 83279 50288
2	Euler's number e	2.71828 18284 59045 23536 02874 71352 66249
3	Pythagoras' constant √2	1.41421 35623 73095 04880 16887 24209 69807
4	Theodorus' constant √3	1.73205 08075 68877 29352 74463 41505 87236
5	The Euler–Mascheroni constant γ	0.57721 56649 01532 86060 65120 90082 40243
6	The Feigenbaum constants δ	4.66920 16091 02990 67185 32038 20466 20161
7	Apéry's constant ζ(3)	1.20205 69031 59594 28539 97381 61511 44999
8	The golden ratio φ	1.61803 39887 49894 84820 45868 34365 63811

DeeRStateUpdate

Step4- Repeat Step2 with incremented values of M

Step5- Repeat Step3 and Step4 till Message length

Squeezing Phase:

Step7- Do h[1]$\leftarrow S_{79}$

Step8- Increment M by 1 (k=2 to 80)

Step9- Call DeeRStateUpdate

Step10- Do h[k]$\leftarrow S_{79}$

Step11- Concatenate H=h[1]||h[2]||h[3]||----||h[80]

Algorithm: DeeRStateUpdate

Step1- Convert the value of MC in its binary (take first 80 bits)

Step2- Make an array D of size 80 with the values from Step1

Step3- Make two arrays E and F of sizes 40 and 40 respectively

Step4- Calculate Ef by using the following function:

$$E_f(V_1,V_2,U_3,V_4) = (V_1V_2 \oplus V_1V_4 \oplus V_2U_3 \oplus U_3V_4 \oplus U_3 \oplus V_4) \oplus 1$$

Step5- Calculate Ff by using the following function:

$$F_f(U_1,U_2,V_3,U_4) = (U_2V_3 \oplus U_2U_4 \oplus U_1V_3 \oplus U_1U_4 \oplus U_2 \oplus U_4)$$

Step6- Calculate Lf by using the following function:

$$L_f(V_1,U_1,V_{25}) = (V_1 \oplus U_1 \oplus V_{25})$$

Step7-Calculate the value of T_1 and T_2 as follows:

$$T_1 = E_f \oplus L_f$$

$$T_2 = F_f \oplus L_f$$

Step8- Update the array E. Pop the leftmost bit, shift all the bits to one position left, and insert T_1 at the rightmost bit position. Similarly update the array F by inserting T_2 at the rightmost bit position

Step9- Update S by concatenating the updated E and F

The variables T1 and T2 are generated from the state update process. During the absorption phase the values of T1 and T2 will be updated using linear and non-linear shift registers. This update will take place exactly half of the message length times, i.e. [Length of Message/2] to produce a high level of confusion. On the other hand, in the squeezing phase, the values of T1 and T2 will be updated for complete hash length times to produce the proper diffusion.

Architecture

DeeR-Hash is a sponge based lightweight hash design. The algorithm named DeeRSponge illustrates the complete process right from the initialization till the squeezing phase through absorbing phase. The r value taken here is 2. Hence, in the initialization phase, padding is required (if necessary) to make the length of massage a multiple of 2. The total width of a sponge state taken here is b = 80-bits subdivided into rate r = 2-bits and capacity c = 78-bits. This will provide a hash value of n=80-bits satisfying the basic formula of the sponge function [b = (r + c) ≥ n]. The proposed construction for sponge based hash design DeeRSponge is illustrated in Fig. 1.

The 39th bit and 79th bit of array D is taken out to be the two values of r. The value at these array positions will be XORed with the initial two bits of message. The resultant is stored at the same array position, 39 and 79. The state update procedure will be called and next two message bits will be XORed with the new values at the array position 39 and 79. This process will be repeated till the end of all message bits. This complete process ensures the mixing of all message bits with the state bits in absorbing phase. In the squeezing phase, the value at position 79 of array D will be returned as it is for h1. The state update procedure will be called. Again, the changed last bit of array D will be return for h2. This process will continues till the size of state bit, here it is 80. After a complete process of DeeRSponge, the mechanism will produce a hash of size 80-bits for the input message of any size.

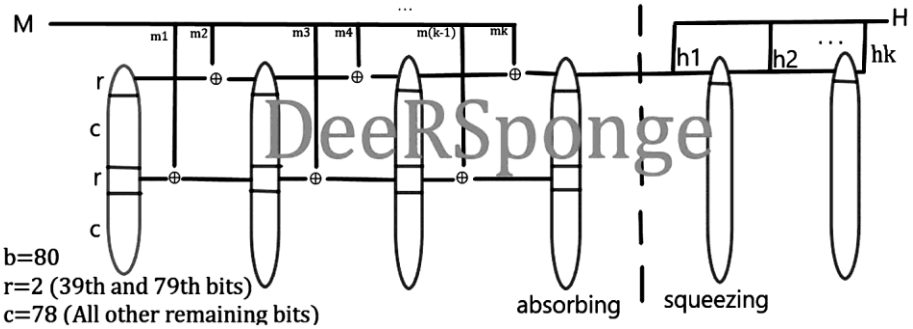

Fig. 1 — Proposed DeeRSponge construction

The update function is a combination of two non-linear update functions along with one linear update function. Here, the non-linear update function works with four variables while the linear update function works with only three variables. The randomly chosen array positions will act like the inputs for the non-linear update functions. Suppose that the array positions chosen from array E are V_1, V_2, V_3, and V_4, and the array positions chosen from array F are U_1, U_2, U_3, and U_4. The formula for updating the non-linear functions will look like:

Calculate Ef by using the following function:

$$E_f(V_1,V_2,U_3,V_4) = (V_1V_2 \oplus V_1V_4 \oplus V_2U_3 \oplus U_3V_4 \oplus U_3 \oplus V_4) \oplus 1$$

Calculate Ff by using the following function:

$$F_f(U_1,U_2,V_3,U_4) = (U_2V_3 \oplus U_2U_4 \oplus U_1V_3 \oplus U_1U_4 \oplus U_2 \oplus U_4)$$

After calculating the values for non-linear update functions, the linear update function will be calculated by using the second value of both the arrays along with the 25^{th} value of array E. The formula for updating the linear function will look like:

Calculate Lf by using the following function:

$$L_f(V_1,U_1,V_{25}) = (V_1 \oplus U_1 \oplus V_{25})$$

These calculated values of E_f, F_f, and L_f will be used to calculate the values of temporarily variables T_1, and T_2. The formula for calculating the values of T_1, and T_2 will look like:

Calculate the value of T_1 and T_2 as follows:

$$T_1 = E_f \oplus L_f$$

$$T_2 = F_f \oplus L_f$$

The value of T_1 and T_2 will then be inserted at the last array positions of array E and array F respectively. All the values will be shifted one bit left and the left most value will be removed from the arrays. This whole process will produce an updated array D of 80-bits. This update will be performed before each computation. The same update procedure will be followed for absorption as well as squeezing phase.

Results and Discussion

The authors analyze the proposed design for the hardware and the software implementation both. The lightweight criteria for a hash design appropriate to be used in Industry 4.0 / IoT is taken care of along with the less complex nature of algorithm as required for a low energy device.[15–17] The proposed design choses shift registers for confusion and diffusion because of the fact that the shift operations take lowest power for execution. On the contrary, substitution and permutation boxes are not only complex but also need lots of power to run. The shift registers can be seen as unsecured linear computing model. For this, the authors use two NLFSRs along with one LFSR. NLFSRs work with four variables whose positions are randomly chosen and the LFSR works with three variables from selected fixed positions. Two different arrays are used in this process. The selections are made from both the arrays.

For the constrained devices, almost all the applications uses Area Specific Integrated Circuits (ASIC) rather than microcontrollers because the ASICs are very much cheaper (180 nm) than programmed microcontrollers. The usual measurement for area is μm^2 but the comparisons are done on the basis of Gate Equivalent (GE) used because of the fact that earlier depends on some inherent properties of circuits such as fabrication and others. The two-input NAND gate with lowest driving strength of the corresponding technology turns out to be equivalent to one GE. Here, we will see the area requirements for different elements used in our design.[5]

The design consists of selecting one element randomly from a list of eight elements. This could be achieved by using an "8 × 1" multiplexer. For the random selection of four different elements from an array of 40 elements, the authors divided the array into five equal parts of 8 elements and then leaving the middle partition as it is. The authors select one element from remaining four partitions by implementing four "8×1" multiplexers. This will give first array position from 0 to 7, second from 8 to 15, third from 24 to 31 and fourth from 32 to 39. Two similar arrangements will be needed for both the arrays E and F. It results into a total of three "8 × 1" multiplexers. The area requirement for implementing one "8×1" multiplexer is 26 GE. Hence, implementing three 8 × 1 multiplexer will need 78 GE in total. The logic circuits used in different constructions are shown in Fig. 2.

The shift registers is made up of flip-flops. The design of DeeR-Hash used D flip-flops. The area requirement for a D flip-flop is 5 GE. Hence, the total area required by an 80-bit shift register would be 80 × 5 = 400 GE approximately. The state update process merges two arrays into one to get the desired array positions for XOR operation. For this, the design uses a 2 × 1 multiplexer. The area requirement for a 2 × 1 multiplexer is 4 GE. An XOR operation require 4 GE, in this design two XOR operations are needed

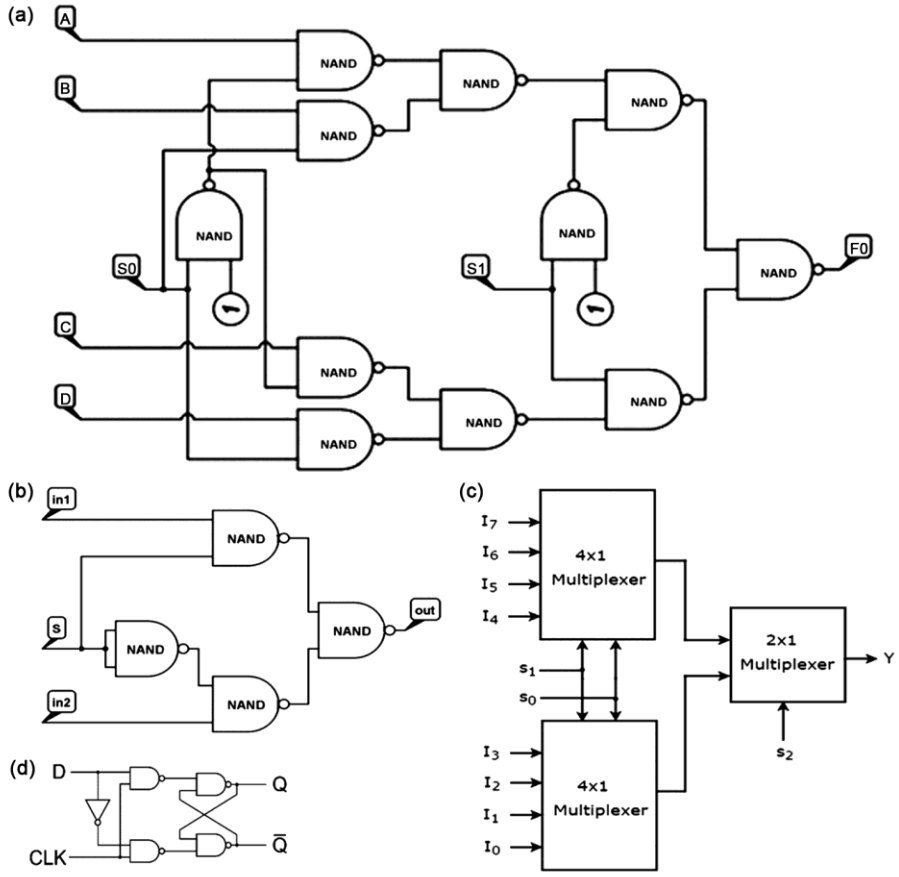

Fig. 2 — Different logic circuit used in DeeR-Hash: (a) A 4 × 1 multiplexer using 2 input NAND gates only, (b) A 2 × 1 multiplexer using 2 input NAND gates only, (c) An 8 × 1 multiplexer using two 4 × 1 and one 2 × 1 multiplexer, (d) A D flip-flop using 2 input NAND gates only

Table 2 — Hardware performance of DeeR-Hash against other available counterparts

Hash function	N	c	r	Preimage	Collision	Second Preimage	Process (μm)	Area (GE)	Cycle
DeeR-Hash	160	158	2	158	79	79	0.18	984	1
Hash-One	160	160	1	160	80	80	0.18	1006	324/162
SPONGENT	176	160	16	144	80	80	0.13	1329	3960
D-QUARK	176	160	16	160	80	80	0.13	2190	90
PHOTON	160	160	36	124	80	80	0.18	1396	1332
Gluon	160	160	16	160	80	80	0.18	2799	50

simultaneously for XORing message bits with T_1 and T_2, hence it needs 8 GE for XOR operations. The linear function performs two XOR operations; hence 8 GE will be used by this.

Also, two non-linear functions are using AND & XOR operations for their calculation. The calculated area requirement for each of the non-linear functions is 13 GE. Hence, two non-linear update will take 26 GE. The total sum for the complete operation will be (400 + 4 + 8 + 8 + 26 =) 405 GE. Adding this with the total gate requirement of multiplexers (78 GE), the overall total will come to be 483 GE only. For a 160-bit hashing, the proposed mechanism will require (800 + 4 + 8 + 8 + 26 + 26 + 112 =) 984 GE. This area requirement is the lowest till date. The values showing the comparison of GE used by different well-known lightweight hash designs are given in Table 2.

Analysis of the Update Function

Randomization is a proven solution for uncertainty. Uncertainty brings yet another level of security to the cryptographic algorithms. This is said that, an algorithm is suitable to be used in any cryptographic application only when even the makers don't know

about the next move. Once deployed, the algorithm should not be predictable in any circumstances. Keeping this in mind, the authors used a list of eight mathematical constants randomly to choose one for a specific operation. This protects the initial value of the 80-bit D array to be known by anyone. Similarly, the values of 40-bit E and 40-bit F arrays will be unknown.

The random selection of array positions to get the input for update function E_f and F_f will provide another level of randomization. This protects the inputs of the functions to be known unlike in Hash-One where the makers know the exact inputs of the function. If the adversary knows the inputs of a function, he/she can easily predict the outputs well in advance. In DeeR-Hash no one knows the inputs to be taken for update functions till the process begins the randomization. Also, these chosen values will be effective for one operation only.

The temporarily variables are calculated from XORing the outputs of NFSRs with the output of LFSR. This will ensure the proper mixing of input bits with state bits. The chosen combination of shift registers will provide adequate security with low power consumption. Two input bits are taken at a time for this XOR operation. This will help reduce the operation by half the times of message length, again requiring almost half of required energy. Also, the update function will run after each absorbing and squeezing phase to produce a high quality of confusion and diffusion.

Computed hash and their hex equivalent for different input messages are presented in Table 3. It can be seen that first six inputs are slightly differing while other four are of variable lengths. One can clearly see that DeeR-Hash produces fix length hash on variable length input sizes. Also, there is an avalanche effect on the produced hash values, i.e. almost all the values got changed on a slight change in input messages.

Security Analysis

The resistance of the proposed model from the preimage, collision, and second preimage attacks are already discussed in the introduction part of Bertoni *et al*.[9] As the design follows the sponge based criteria, the DeeR-Hash gets all the benefits related to security from the sponge only. In this section, the authors will discuss about other security requirements of the update function.

Cube Attack

The update function of DeeR-Hash works with two NLFSR and one LFSR. The XOR operation between an NLFSR and a LFSR gives it a similar processing as in a Galois FCSR (Feedback with Carry Shift Registers). Cube attackers attack the functions with weak algebraic structure. High degree components are used to make a Galois FCSR and hence its algebraic structure is so strong that a cube attacker has no impact on the update functions. Hence, the authors can conclude that the DeeR-Hash is protected with the cube attacks.

Table 3 — Computed Hash and their hex equivalent

Sr. No.	Input Massage	Computed Hash	Hex Equivalent
1	Deena Nath	0001110011010001101001110001001111000101000000010000011010010011100010 0010010111	1CD1A713C50106938897
2	deena Nath	1001011110011111100011011001011011011011001110110100100100110111100000 0001001011	979F8D96DB3B4937804B
3	deenanath	0011110010111011110001010000110001111001100110001101001001001001011000 1001000110	3CBBC50C7998D2496246
4	dEenanath	1011100100011001111101010011110010111100111010101110100001000011101111 1001000100	B919F53CBCEAE843BE44
5	deEnanath	0111100110101110110101101001111011110111100010000000011100001101010100 1010000100	79AED69EF788070D5284
6	deeNanath	1110010110101011010000010100011001110100011100000000111100011001101011 1 0001011011	E5AB414674701E335C5B
7	Deena	0010110000110000110000001101101000100000111011110111010101010111010100 1010111011	2C30C0DA20EF755752BB
8	Deena Nath Gupta	0001101111010011111111100111000110010101001010100001110100111011000010 1101011101	1BD3FE71952A1D3B0B5D
9	Chandra	1111101000011110011111100010111011101001101101001110001110111101101101 1001111000	FA1E7E2EE9B4E3BDB678
10	Rajendra	0101001110111001010111110000011111001100010111001010010001001000100011 1101101111	53B95F07CC5CA4488F6F

The Hell and Johansson Attack

The feedback values T1 and T2 are generated by using four randomly chosen values (three from one sub array and one from other sub array) putting in a non-linear function and performing XOR with its output to the output of a linear function that works on three different values (two from one sub array and one from other sub array). These feedback bits will not affect the next round as the inputs for the next round will be independent from these values. Hence, the Hell and Johansson attack will not be effective.

Differential Attack

The two specific X points on the graph of an FCSR, the all-zero point turned into itself and the all-one point turned into it, could be used to create slide distinguishers.Due to the way the sponge construction creates its internal states, the all-one point for f could not be attained when employed inside the sponge construction. However, if a specific start value is not assigned to a specific word of the FCSR's main register, the all-zero point for f may be reached. That is why the authors used eight different mathematical constants to randomly initialize the 80-bit array used in the construction of DeeR-Hash.

Conclusions

The proposed instance of DeeR-Hash is an 80-bit digest. It works on an estimated 483 GE only, which is lowest till date. For the comparison purpose, the area requirement for a 160-bit instance is also given. The 160-bit digest of DeeR-Hash required an estimated 984 GE only, again lowest among its counterparts. The proposed hash construction is a lightweight hash design based on the sponge construction that fulfills all security requirements and energy requirements of Industry 4.0 / IoT. Hence, the authors can conclude that the proposed instance of the lightweight hash design DeeR-Hash is suitable to be used in the lightweight cryptographic algorithms.

Scholars can choose different mathematical constants to seed the initial arrays. The authors presented a method that selects four out of five available sub arrays; the scholars can make their own method of selecting array positions. Also, the mathematics proposed by authors can be framed differently. Scholars can check other equations those might produce better results. The packet formation to be sent to receiver end may also be reconfigured. This will also add on the security of the whole hashing process. The authors fixed the value for rate as 2; other scholars may choose different values like 4 or any other even number and can check for the differences in output.

References

1 Ågren M, Hell M, Johansson T & Meier W, Grain-128a: a new version of Grain-128 with optional authentication, *Int J Wirel Mob Comput*, **5(1)** (2011) 48, https://doi.org/10.1504/ijwmc.2011.044106.

2 Chakraborti A, Chattopadhyay A, Hassan M & Nandi M, TriviA and uTriviA: two fast and secure authenticated encryption schemes, *J Cryptogr Eng*, **8(1)** (2018) 29–48, https://doi.org/10.1007/s13389-016-0137-2.

3 Duan X, Cui Q, Wang S, Fang H & She G, Differential power analysis attack and efficient countermeasures on PRESENT, *Proc 8th IEEE Int Conf Commun Soft Netw* (IEEE) (2016), 8–12, https://doi.org/10.1109/ICCSN.2016.7586627.

4 Engels D, Saarinen M J O, Schweitzer P & Smith E M, The hummingbird-2 lightweight authenticated encryption algorithm', in *Int workshop on radio frequency identification: Security and privacy issues* (Springer-Verlag Berlin Heidelberg) 2012, 19–31. https://doi.org/10.1007/978-3-642-25286-0_2.

5 Saravanan P, Rani S S, Rekha S S & Jatana H S, An Efficient ASIC Implementation of CLEFIA Encryption/Decryption Algorithm with Novel S-Box Architectures, *2019 IEEE 1st Int Conf Energy Syst Inf Process*, (IEEE) (Chennai, India) 2019, 1–6, https://doi.org/10.1109/ICESIP46348.2019.8938329.

6 Aumasson J P, Henzen L, Meier W & Naya-Plasencia M, Quark: A lightweight hash, *J Cryptol*, **26(2)** (2013) 313–339, https://doi.org/10.1007/s00145-012-9125-6.

7 Gupta A, Srivastava A, Anand R & Tomažič T, Business application analytics and the internet of things: The connecting link, in *Transforming the Internet through Machine Learning, IoT, and Trust Modeling*, edited by G Shrivastava, S-L Peng, H Bansal, K Sharma & M Sharma, (New Age Analytics, Apple Academic Press) 2020, 249–273, https://doi.org/10.1201/9781003007210.

8 Gupta A, Asad A, Meena L & Anand R, IoT and RFID-based smart card system integrated with health care, electricity, QR and banking sectors, in *Artificial Intelligence on Medical Data*, *Lecture Notes in Computational Vision and Biomechanics*, edited by M Gupta, S Ghatak, A Gupta & A L Mukherjeem, vol 37 (Springer, Singapore) 2023**,** 253–265, https://doi.org/10.1007/978-981-19-0151-5_22

9 Bertoni G, Daemen J, Peeters M & Assche G V, Sponge Functions, *ECRYPT Hash Workshop* (May 2007, Barcelona, Spain) (9) 2007**,** 1–22.

10 Peinado A & Fúster-Sabater A, Generation of pseudorandom binary sequences by means of linear feedback shift registers (LFSRs) with dynamic feedback, *Math Comput Model*, **57(11-12)** (2013) 2596–2604.

11 Guo J, Peyrin T & Poschmann A, The PHOTON family of lightweight hash functions, *Lect Notes Comput Sci*, **6841** (2011) 222–239, https://doi.org/10.1007/978-3-642-22792-9_13.

12 Berger T P, D'Hayer J, Marquet K, Minier M & Thomas G, The GLUON family: A lightweight hash function family

based on fcsrs, in *Progress in Cryptology - AFRICACRYPT 2012, AFRICACRYPT 2012, Lecture Notes in Computer Science*, vol 7374, edited by A Mitrokotsa & S Vaudenay, (Springer, Berlin, Heidelberg) 2012, 306–323, https://doi.org/10.1007/978-3-642-31410-0_19.

13 Manayankath S, Srinivasan C, Sethumadhavan M & Megha M P, Hash-One: a lightweight cryptographic hash function, *IET Inf Secur*, **10(5)** (2016) 225–231, https://doi.org/10.1049/iet-ifs.2015.0385.

14 Arnault F, Berger T P, Lauradoux C & Minier M, X-FCSR - A new software-oriented stream cipher based upon FCSRs, in *Progress in Cryptology – INDOCRYPT 2007, INDOCRYPT 2007, Lecture Notes in Computer Science*, vol 4859, edited by K Srinathan, C P Rangan & M Yung (Springer, Berlin, Heidelberg) 2007, 341–350, https://doi.org/10.1007/978-3-540-77026-8_26.

15 Anand R, Sinha A, Bhardwaj A & Sreeraj A, Flawed security of social network of things, in *Handbook of Research on Network Forensics and Analysis Techniques* edited by G Shrivastava, P Kumar, B B Gupta, S Bala & N Dey (IGI Global) 2018, 65–86, https://doi.org/10.4018/978-1-5225-4100-4.ch005.

16 Gupta R, Shrivastava G, Anand R & Tomažič T, IoT-based privacy control system through android, in *Handbook of E-business Security*, 1st Edn, edited by J M R S Tavares, B K Mishra, R Kumar, N Zaman & M Khari (Auerbach Publications) 2018, 341–363, https://doi.org/10.1201/9780429468254.

17 Al-Turjman F, Yadav S P, Kumar M, Yadav V & T Stephan, *Transforming Management with AI, Big-Data, and IoT* (Springer) 2022, https://doi.org/10.1007/978-3-030-86749-2.