Cube Attack on a Trojan-Compromised Hardware Implementation of Ascon

Basel Halak
University of Southampton
Southampton, United Kingdom
basel.halak@soton.ac.uk

Jorge Duarte-Sanchez
Southampton, United Kingdom
jeds1n18@southamptonalumni.ac.uk

2020 IEEE 33rd International System-on-Chip Conference (SOCC) | 978-1-7281-8746-4/20/$31.00 ©2020 IEEE | DOI: 10.1109/SOCC49529.2020.9524771

***Abstract*— Ascon algorithm was selected in 2019, in the CAESAR competition as the first option for lightweight applications as an alternative to AES-GCM for authenticated encryption. As with other encryption algorithms, Ascon relies on some parameters and security assumptions to guarantee its security. For example, if the number of rounds of the initialization phase of the encryption is reduced, the key can be obtained using a cube attack. In this work we describe how by inserting a hardware trojan with low overhead in a hardware implementation of Ascon, it is possible to reduce the number of rounds of its initialization stage and perform a cube attack in order to obtain the key in 94 seconds on average.**

***Keywords*— Hardware Trojan, Authenticated Encryption, Hardware Security, Ascon**

I. Introduction

With the increasing complexity of modern integrated circuits, it is more and more common to use components from third parties and to outsource manufacturing processes. This can represent a security issue because untrusted third parties can have access to the design of a circuit and insert hardware trojans to perform malicious operations.

A Hardware Trojan (HT) is a modification of a circuit that performs a malicious operation to disrupt services, leak sensitive data or degrade the performance of a system. HTs can be inserted in the hardware description code (RTL) or netlist of an Intellectual Property module (IP) during the design or system integration phase, or in the layout of an integrated circuit (IC) before manufacturing. HTs have typically two main components: the payload that is the circuit that performs the malicious operation, and the trigger circuit that activates the HT. Payload and trigger circuits are designed to have a minimum impact on the original circuit to make the detection of the HT more difficult. Additionally, to avoid being detected during functional test, HTs are normally triggered under conditions that rarely occur during the normal operation of the circuit but that the attacker knows so that he can perform the malicious operation.

Protection against HTs can be approached using prevention and detection techniques [1]. Prevention techniques aim to make insertion of HT more difficult by obfuscating the design [2] or by filling with functional cells the empty areas of the layout of an IC which are used to insert HTs [3, 4]. Detection techniques can be used in pre-silicon and post-silicon stages of the IC development. Pre-silicon detection techniques detect HT in the RTL or netlist of IPs by analysing suspicious networks with low activation [5, 6] or inputs with low impact on the circuit outputs. Formal verification techniques can also detect HT by ensuring that the IP implementation represents exactly its specification [7, 8]. Post-silicon detection techniques detect HT once the IC has been manufactured, and they can be invasive or non-invasive. In invasive methods, an IC is de-packed, de-layered and compared against a golden (reference) IC to detect changes in the layout which indicates the presence of HTs. Non-invasive techniques include functional testing and evaluation of the side-channel effects of the HT [9, 10].

HTs can compromise the security of encryption cores by, for example, leaking the key directly or indirectly, or by violating security assumptions that can expose other sensitive information [11-13]. The most evident way of getting the key in a cryptographic core is by inserting a HT that exposes the key directly to the output of the cipher using a multiplexor. This type of HT can be very effective but is obvious and could be detected easily due to the large number of gates required. To make the HT less evident and smaller, and therefore much harder to detect, a HT can modify other parameters of the cipher such that it is possible to obtain the key using additional processing [14, 15].

In this work we describe how by inserting a hardware trojan with low overhead in a hardware implementation of Ascon, it is possible to reduce the number of rounds of the initialization stage and perform a cube attack in order to obtain the key in 94 seconds on average.

II. Background

A. Ascon Algorithm

Ascon is an algorithm for Authenticated Encryption with Associated Data (AEAD) which was selected in 2019 in the CAESAR competition as an alternative to AES Galois/Counter Mode block ciphers (AES-GCM) for lightweight applications [16].

AEAD schemes provide confidentiality and authenticity of messages where encrypted information (payload) is used together with optional non-encrypted (associated) data. For example, the associated data can correspond to a packet header which must be accessible (not encrypted) to navigate through a network, while the payload contains sensitive data that must be encrypted to keep confidentiality of the message. In an AEAD scheme, the authenticity of the associated data is achieved by embedding the data in the encryption and decryption operations. This operation is illustrated in Fig. 1. The payload is encrypted using the associated data and three parameters: a secret key (K), a constant initialization vector (IV) and a public number (N). N

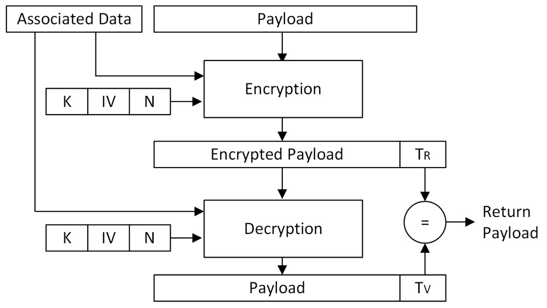

Fig. 1. Authenticated Encryption and Decryption process with associated data using Ascon.

works as a nonce, meaning that each encryption must use a different value of N. The encryption process also generates a tag reference tag (TR). To decrypt, the cipher uses the same three parameters and the associated data to recover the payload and to generate a validation tag (TV). Only if both tags are equal, the authenticity of the associated data is verified, and the cipher returns the plain payload.

An Ascon encryption operation consists of four stages as illustrated in Fig. 2: Initialization, Associated Data, Plaintext and Finalization [17]. The core of Ascon is a 320-bit permutation consisting on three operations: addition of constants, substitution, and linear diffusion. The Initialization and Finalization stages perform 12 rounds of permutations (operation denoted as pa); The Associated data and Plaintext (payload) blocks are processed using 6 rounds of permutations (denoted as p^b).

B. Cube Attack

The security of Ascon relies, among other factors, on the number of rounds of permutation operations. A security analysis of the algorithm is presented in [20] using cube-like, differential and linear cryptanalysis. In that work, the authors describe theoretical and practical key-recovery attacks of round-reduced versions of Ascon. Furthermore, in [18] Z. Li at al. present improvements to the cube-like cryptanalysis of Ascon to achieve a practical key-recovery attack of a reduced initialization phase of 6 and 5 rounds with time complexity of 2^{40} and 2^{24}, respectively.

The cube attack is a cryptoanalysis method applicable to symmetric key algorithms (like Ascon). The attack targets the initialization phase of the algorithm by processing (N, $P \oplus C$) pairs without associated data as depicted in Fig. 3. The basic idea of the cube attack is to recover the key one bit at the time by

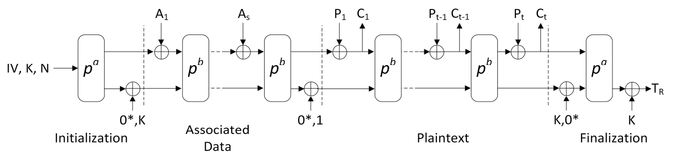

Fig. 2. Mode of operation of Ascon encryption algorithm. After Initialization, each block of the associated data (Ai) is combined with upper r bits of the internal state of the cipher. Then, each block of the payload (Pi) is combined with upper r bits of the state of the cipher to generate the blocks of encrypted payload (Ci). For each block of payload and associated data, the state is updated by performing 6 or 8 rounds of permutations (pb). The tag (T) is generated in the finalization phase combining the k lower bits of the state of the system with the key (K). (Adapted from [17])

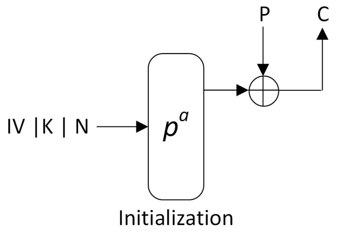

Fig. 3. Basic setup for the Cube Attack on Ascon. The cube attack targets the initialization phase of the algorithm by processing ciphertext blocks C obtained using specific nonce values (N) without associated data. Adapted from [18].

manipulating specific bits of N, called cube variables. For each key bit ki, a set of cube variables is defined and different values of N are generated by setting the cube variables with all their possible binary combinations. These values of N are used to encrypt an empty message (P=0). The resultant encrypted messages (C) form a system of linear equations for ki that can be solved to obtain its value. More details of the cube attack can be found in [18-20].

After evaluating all the cube variables for all the key bits. there might be remaining bits of the key that cannot be obtained. Experimental results show that the number of remaining bits is always less than 2^{14}, therefore it is feasible to find them by exhaustive search [18].

III. Injecting a Hardware Trojan in an Ascon Hardware Implementation

This section presents the design of a HT that exploit the characteristics of the hardware implementation of Ascon to reduce the number of rounds of the permutation operation during the initialization phase of encryption from 12 to 5 so that it is possible to perform a cube attack with a time complexity of 2^{24}.

A. Hardware Implementation of Ascon

In this work we used the CAESAR Hardware API reference implementation available in [21] to identify features of the design that allowed us to insert a hardware trojan. The main module of the implementation is the Cipher Core, which is the implementation of the Ascon cipher itself. The Cipher core has an iterating implementation, this means that the same hardware required to process one round of permutation is reused as many times as the number of rounds is required for each stage of encryption or decryption. Fig. 4 shows a schematic diagram of the Ascon cipher core.

The algorithmic state of the system (*ostate*) is controlled using a 320-bit 3-to-1 multiplexor. In the initialization phase (*sel_state* = 10) *ostate* takes the value of state0 which contains

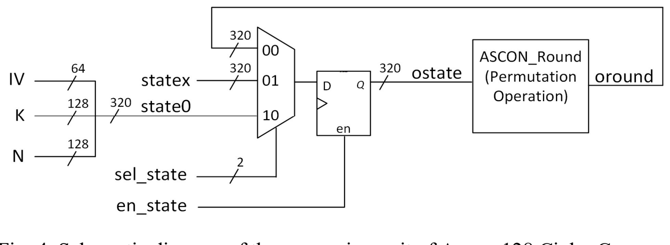

Fig. 4. Schematic diagram of the processing unit of Ascon-128 CipherCore

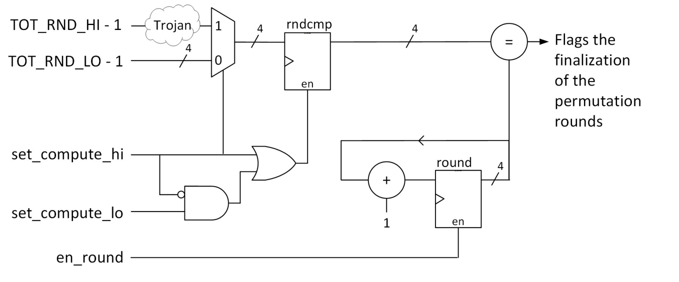

Fig. 5. Trojan insertion in the control circuit to reduce the number of rounds of permutations in Ascon Encryption and Decryption.diagram of the processing unit of Ascon-128 CipherCore

the initialization vector constant (IV), the key, and the nonce (N). Then, the block ASCON_Round performs one round of the permutation operation and puts the result in the signal *oround*. The multiplexor then selects *oround* as the new value of *ostate* (*sel_state* = 00) to calculate the next round of permutation. This process is repeated until completing 12 rounds. When the cipher core is processing ciphertext, plaintext or associated data blocks, *ostate* is updated with the value of *statex* (*sel_state* = 01) which contains the value of the current data block, and then performs six rounds of permutations.

B. *Trojan for Initialization Round Reduction*

In a full unrolled or pipeline implementation of Ascon, the hardware structure is fixed for a given number of permutation rounds [22]. On the other hand, in the iterated implementation of Ascon the number of permutation rounds is controlled using a counter whose output is compared against a constant value equivalent to the required number of rounds as shown in Fig. 5. Therefore, by modifying this part of the circuit, it is possible to reduce the number of rounds.

TOT_RND_HI and TOT_RND_LO are constants defined in the VHDL code of the controller. TOT_RND_HI is used to set the number of rounds in the initialization and finalization phases and its value is 12. TOT_RND_LO is used to set the number of rounds when the cipher is processing the associated data, the plaintext and the ciphertext and its value is 6. *set_compute_hi* and *set_compute_lo* are control signals used to set the value of the register *rndcmp* to TOT_RND_HI or TOT_RND_LO depending on the current state of the system. When the value of the counter round is equal to the value of *rndcmp*, the cipher stops processing rounds of permutations and waits until a new data block arrives.

a) Trojan Payload

The number of rounds of the initialization phase can be changed by inserting a 4-bit multiplexer that selects between the normal expected value TOT_RND_HI - 1 (11), and the desired value, which in this case is 4 in order to execute 5 rounds of permutation as shown in Fig. 6 (a). Since the inputs of the multiplexor are constants and complementary, the same operation can be simplified and implemented using only NOT gates and wires as it is shown in Fig. 6 (b). As a result, the designed HT has a small footprint, which makes it relatively easy to insert in the layout and hard to detect.

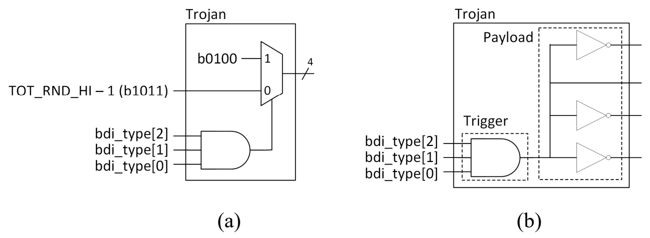

Fig. 6. Trojan for round reduction. (a) Model of the Trojan with a multiplexor and one AND gate. (b) Equivalent circuit of the Trojan replacing the multiplexor with NOT gates and wires.

b) Trojan Trigger

The unused combinations of primary inputs that do not have a specific function can be used as trigger conditions for a HT. The implementation of Ascon includes a wrapper module that implements a hardware API to evaluate all the submissions of the CAESAR competition using the same hardware interface [48]. In the wrapper, the signal *bdi_type* identifies the type of data that is received at the input port (associated data, data, tag, nonce, etc.). However, in the CipherCore entity, *bdi_type* is only used to distinguish between associated data (*bdi_type* = 00X) and data (plaintext or ciphertext blocks) (*bdi_type* = 01X) (X denotes either 1 or 0). This is because other control signals determine the behaviour of the cipher and the type of data that is expected in the input at each stage of operation. Therefore, since the condition *bdi_type* = 111 is never used during normal operation, it can be used as a trigger condition for the Trojan. This can be implemented with a 3-input AND gate as shown in Fig. 6 (b).

C. *Attack Assumptions*

In order to reduce the number of rounds of the initialization phase, the attacker must be able to insert the HT in either the RTL or netlist during the integration phase of the design of a SoC, or in the layout of the circuit before manufacturing.

With the infected cipher already deployed in the field, the attacker must have access to the system that uses the cipher to install a malware (malicious software) to activate the HT and perform the cube attack. To activate the HT, the malware must have the capability of altering the function that sends the nonce (N) to the core so that it writes '111' to *bdi_type* when the nonce value is sent. The malware would be a process running along with other processes in a computing system with an operating system (OS). To perform the attack, the malware must perform encryption operations while the other applications are not using the core. Every time that the operating system grants execution time to the malware, it activates the HT and runs a part of the attack performing as many encryptions as possible before the OS pre-empts and executes a different process.

IV. Experimental Evaluation

A. Setup

To evaluate the feasibility of the proposed attack (round-reduction HT + cube attack), it was implemented in a development board DE1-SoC with a Cyclone V SoC FPGA (5CSEMA5F31C6). Fig. 7 shows a block diagram of the experimental setup. The Ascon cipher (Ciphercore) was implemented in the logic fabric of the FPGA, and the code to perform the cube attack was executed in the ARM processor embedded in the chip.

The code for the cube attack is based on the one used in [18] which is available in [23]. The code was modified to encrypt using the CipherCore through an AXI4 Lite memory- mapped interface, instead of using the software functions of the original code.

B. Results

Table I summarizes the results of 50 cube attacks using random keys after activating the designed Trojan in the CipherCore. The table shows the minimum, average and maximum of the number of the encryption operations required to perform the cube attack, the number of remaining key bits that were not guessed after performing the cube attack, the number of encryptions required to find the remaining bits by exhaustive search, and the total attack time.

In order to evaluate the hardware overhead and the performance impact of the designed Trojan, the CipherCore was synthesized for FPGA and ASIC targets with and without the Trojan. A summary of the synthesis results is shown in Table II.

TABLE I. Summary of the results of 50 cube attacks.

Metric	Min.	Avg.	Max.
Cube Attack Encryptions	11927552	12702188	13762560
Remaining Key Bits	0	3.3	12
Exhaustive Search Encryptions	0	16.2	148
Total Attack Time [s]	89	94.36	103

V. Analysis of Results

A. Trojan Overhead

The results shown in Table II confirm that the designed Trojan has a low overhead; however, these results do not represent precisely the effect of the Trojan. For example, the results show that in FPGA that the Trojan decreases the maximum path delay of the circuit, but since the Trojan is not located in the critical path of the original or the modified circuit, this parameter should not change. This can be explained considering that small modifications in the design can result in different logic optimizations, resource utilization, routing and critical paths because the initial conditions (seed values) of the synthesis algorithm change.

Nevertheless, the results show that, in both FPGA and ASIC, the circuit with Trojan uses one more register than the original circuit. This is because without the Trojan, the register *rndcmp*[0] is optimized away since the two values that it can

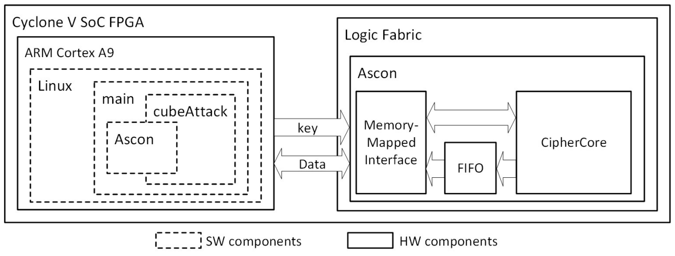

Fig. 7. Setup to perform the cube attack on the Hardware implementation of Ascon

take, TOT_RND_HI[0] or TOT_RND_LO[0], are equal to 1. On the other hand, with the Trojan, the value of TOT_RND_HI can be even (5-1) or odd (12-1), and therefore, TOT_RND_HI[0] can be 0 or 1, which prevents *rndcmp*[0] to be optimized away. This can be avoided if the Trojan reduces the number of rounds to an even number.

B. Trojan Detection

Due to its low overhead, the designed Trojan does not alter significantly the power consumption of the original circuit, therefore, detecting the Trojan by the analysis of its side-channel effects would be challenging, especially if the Trojan is not triggered during testing. Some detection techniques that evaluate changes of path delays could be effective at detecting small trojans by incorporating Built-in Self-Test (BIST) or PUF-based circuits; however, since the designed Trojan creates a path that does not exist in the original circuit (from bdi_type to rndcmp), this path might not be tested and the Trojan could pass undetected.

Pre-silicon techniques like UCI [5] can be used to detect the designed trojan during verification of the circuit because since normally, only functional input combinations are applied to the circuit, the AND gate of the trojan trigger would not be activated and would be flagged as a potential malicious circuit. Similar techniques like VeriTrust [24] and FANCI [6] can also be effective at detecting the trojan because they can detect unused input combinations.

To detect the Trojan using post-silicon detection methods based on functional or structural testing it is important to determine the effect of unused combinations of inputs in the behaviour of the circuit and used them during testing to check if there is any deviation from that expected behaviour.

Finally, runtime techniques can be used to detect the designed Trojan once it is deployed in the field. For example, a

TABLE II. Synthesis Results of the Ciphercore without and with the designed Trojan

Target	Parameter	Without Trojan	With Trojan
ASIC	Combinational Cells	3983	3971
	Registers	535	536
	Buf/Inv	885	869
	Total Dynamic Power [mW]	22.5	22.3
	Maximum Path Delay [ns]	11.5	11.5
FPGA	ALMs	771	771
	Registers	534	535
	Maximum Path Delay [ns]	4.8	4.75

runtime security monitor can keep track of the number of clock cycles that it takes to encrypt a message with a known number of data blocks so that any alteration of the number of rounds would result in an unexpected number of clock cycles and the monitor can block the operation.

VI. Conclusions

In this work we showed how a hardware trojan can be used in combination with a cube attack to get the key of a hardware implementation of the Ascon algorithm in 94 seconds on average with 100% of success. The designed Trojan has a low overhead which could make its detection very challenging, especially during testing of the IC because, without adequate test vectors, the Trojan could remain inactive. However, by including unused input combinations during testing and knowing their effects on the original circuit, it is possible to reveal suspicious behaviour of the circuit to detect the Trojan.

Furthermore, when the cipher is obtained from an untrusted party in the form of an IP module, pre-silicon techniques can be effective to detect trojans inserted in the RTL. Additionally, run-time monitors can also be used to detect suspicious behaviour of the cipher during runtime to deactivate it and prevent leakage of sensitive information.

References

[1] K. Xiao, D. Forte, Y. Jin, R. Karri, S. Bhunia, and M. Tehranipoor, "Hardware Trojans: Lessons Learned after One Decade of Research," ACM Transactions on Design Automation of Electronic Systems(TODAES), vol. 22, pp. 1-23, 2016.

[2] X. T. a. G. Ngo, Sylvain and Bhasin, Shivam and Danger, Jean-Luc and Najm, Zakaria, "Encoding the State of Integrated Circuits : a Proactive and Reactive Protection against Hardware Trojans Horses," Proceedings of the 9th Workshop on Embedded Systems Security, pp. 7:1--7:10, 2014.

[3] K. Xiao, D. Forte, and M. Tehranipoor, "A novel built-in self-authentication technique to prevent inserting hardware trojans," IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, vol. 33, pp. 1778-1791, 2014.

[4] S. T. C. Konigsmark, D. Chen, and M. D. F. Wong, "Information Dispersion for Trojan Defense through High-Level Synthesis," 2018.

[5] M. Hicks, M. Finnicum, S. T. King, M. M. K. Martin, and J. M. Smith, "Overcoming an Untrusted Computing Base: Detecting and Removing Malicious Hardware Automatically," in 2010 IEEE Symposium on Security and Privacy, ed: IEEE, 2010, pp. 159-172.

[6] A. Waksman, M. Suozzo, and S. Sethumadhavan, "FANCI : Identification of Stealthy Malicious Logic Using Boolean Functional Analysis," Ccs 2013, pp. 697 - 708, 2013.

[7] M. Rathmair, F. Schupfer, and C. Krieg, "Applied formal methods for hardware Trojan detection," in 2014 IEEE International Symposium on Circuits and Systems (ISCAS), ed: IEEE, 2014, pp. 169-172.

[8] J. Rajendran, A. M. Dhandayuthapany, V. Vedula, and R. Karri, "Formal Security Verification of Third Party Intellectual Property Cores for Information Leakage," in 2016 29th International Conference on VLSI Design and 2016 15th International Conference on Embedded Systems (VLSID), ed: IEEE, 2016, pp. 547-552.

[9] A. Amelian and S. E. Borujeni, "A Side-Channel Analysis for Hardware Trojan Detection Based on Path Delay Measurement," Journal of Circuits, Systems and Computers, vol. 27, p. 1850138, 2018.

[10] X. Wang, H. Salmani, M. Tehranipoor, and J. Plusquellic, "Hardware Trojan Detection and Isolation Using Current Integration and Localized Current Analysis," in 2008 IEEE International Symposium on Defect and Fault Tolerance of VLSI Systems, ed: IEEE, 2008, pp. 87-95.

[11] C. Krieg, C. Wolf, and A. Jantsch, "Malicious LUT: A stealthy FPGA Trojan injected and triggered by the design flow," Proc. 35th International Conference on Computer-Aided Design – ICCAD '16, pp. 1-8, 2016.

[12] X. Wang, T. Mal-Sarkar, A. Krishna, S. Narasimhan, and S. Bhunia, "Software exploitable hardware Trojans in embedded processor," Proceedings - IEEE International Symposium on Defect and Fault Tolerance in VLSI Systems, pp. 55-58, 2012.

[13] Y. Jin, N. Kupp, and Y. Makris, "Experiences in hardware trojan design and implementation," 2009 IEEE International Workshop on Hardware- Oriented Security and Trust, HOST 2009, pp. 50-57, 2009.

[14] S. Bhasin, J.-L. Danger, S. Guilley, X. T. Ngo, and L. Sauvage, "Hardware Trojan Horses in Cryptographic IP Cores," in 2013 Workshop on Fault Diagnosis and Tolerance in Cryptography, ed: IEEE, 2013, pp. 15-29.

[15] M. Yoshimura, A. Ogita, and T. Hosokawa, "A smart Trojan circuit and smart attack method in AES encryption circuits," in 2013 IEEE International Symposium on Defect and Fault Tolerance in VLSI and Nanotechnology Systems (DFTS), ed: IEEE, 2013, pp. 278-283.

[16] "CAESAR call for submissions," 2014.

[17] C. Dobraunig, M. Eichlseder, F. Mendel, and M. Schl, "Ascon v1.2 Submission to the CAESAR Competition," pp. 1-27, 2016.

[18] Z. Li, X. Dong, and X. Wang, "Conditional Cube Attack on Round-Reduced ASCON", IACR Transactions on Symmetric Cryptology, vol. 2017, pp. 175-202, 2017.

[19] C. Dobraunig, M. Eichlseder, F. Mendel, and M. Schlaffer, "Cryptanalysis of Ascon," ed: Springer, Cham, 2015, pp. 371-387.

[20] I. Dinur and A. Shamir, "Cube attacks on tweakable black boxp Polynomials," Joux A. (eds) Advances in Cryptology - EUROCRYPT 2009. EUROCRYPT 2009. Lecture Notes in Computer Science, vol. 5479, pp. 278-299, 2009.

[21] E. Homsirikamol. (2016, 17/10/2019). Ascon Hardware. Available: https://github.com/IAIK/ascon_hardware/tree/master/caesar_hardware_api_v_1_0_3/ASCON_ASCON

[22] M. Fivez, "Energy Efficient Hardware Implementations of CAESAR Submissions," 2016.

[23] Z. Li, X. Dong, and X. Wang. (2017, 17/10/2019). Conditional Cube Attack on Round-Reduced ASCON. Available: https://github.com/lizhengcn/Ascon_test/blob/master/Attackon5-rASCON.cpp

[24] J. Zhang, Q. Xu, Z. Sun, F. Yuan, and L. Wei, "VeriTrust," 2013 50th ACM/EDAC/IEEE Design Automation Conference (DAC), p. 1, 2013.