Noname manuscript No.
(will be inserted by the editor)

A Generic Scan Attack on Hardware based eStream Winners

Sandip Karmakar · Dipanwita Roy Chowdhury

Received: date / Accepted: date

Abstract Scan chains, a design for testability (DFT) feature, are included in most modern-day ICs. But, it opens a side channel for attacking cryptographic chips. We propose a methodology by which we can recover internal states of any stream cipher using scan chains without knowledge of its design. We consider conventional scan-chain design which is normally not scrambled or protected in any other way. In this scenario the challenge of the adversary is to obtain the correspondence of output of the scan chain and the internal state registers of the stream cipher. We present a mathematical model of the attack and the correspondence between the scan chain-outputs and the internal state bits have been proved under this model. We propose an algorithm that through off-line and on-line simulation forms bijection between the above mentioned sets and thus finds the required correspondence. We also give an estimate of the number of off-line simulations necessary for finding the correspondence.

The proposed strategy is successfully applied to eStream hardware based finalists MICKEY-128 2.0, Trivium and Grain-128. To the best of our knowledge, this is the first scan based attack against full round Grain-128 and only the fourth reported cryptanalysis. This attack on Trivium is better than that of the published scan-attack on Trivium. This scan-based attack is also the first reported scan based cryptanalysis against MICKEY-128 2.0.

Keywords Scan Attack · eStream Winners · Side Channel Attack · Grain-128 · Trivium · MICKEY-128 2.0

Indian Institute of Technology, Kharagpur

1 Introduction

Cryptanalysis of stream ciphers has various directions, like, algebraic attacks, statistical attacks, side channel attacks etc. Algebraic or statistical attacks evaluate the mathematical robustness of ciphers. There is another kind of attack which exploits practical implementations, called Side Channel Attacks (SCA). Hardware/software implementations may leak information through power consumption, timing etc. SCA exploits these side channel information to attack systems. Scan chain based attack ([19], [18], [2], [16]) is one such side channel attack. It exploits the design for testability features built in almost all modern day ICs. Scan chain is a testability feature built in devices, where, all the flip-flops (FFs) are connected via a scan chain, data may be input in the device through *scan_in* line and after normal mode of operation scanned out through the *scan_out* line. Through scan chain feature, user may both observe the contents of a FF and set any FF to a desired state. However, both of these features may be exploited to break a system. It has been shown in ([18],[20], [19]) that crypto-systems may be broken, practically, exploiting the scan technique. Scan attacks are thus a threat to cryptographic devices implementations. Evidently, by disabling scan chains this vulnerability may be removed. But, this increases the possibility of defective supply of hardware. However, extensive effort is also given towards countermeasures against scan attacks([10], [13], [17], [11], [3], [15], [12]).

Scan attacks and their countermeasures are studied by many researchers. Literature shows that such kind of attacks are effective against both block ciphers and stream ciphers. Even strong ciphers that are resistant to other existing attacks are vulnerable against scan attack. Scan based attacks on Data Encryption

Standard (DES) and Advanced Encryption Standard (AES) have been shown in [19] and [20] respectively. Till date no mathematical attacks are known against full round Trivium, but [2] mounts a scan based side channel attack against Trivium, while [18] presents an attack on the RC4 stream cipher. However, all these attacks are application specific. Our approach is towards a generic scan-attack that does not require knowledge of design of the cipher. On the other hand a number of schemes are suggested to prevent these attacks ([11], [18], [20]). However, all these countermeasures are application specific. Hence, scan attack is still a threat for cryptographic hardware.

In this paper, we propose a general strategy for attacking any stream cipher using scan based side channels. The proposed strategy is employed to hardware efficient finalists of eStream [1], [4], MICKEY-128 2.0, Trivium, Grain-128 and are shown that all three are vulnerable against this attack in practical scenario.

This paper is organized as follows. Following this introduction, section 2 briefly discusses scan attack and states the proposed methodology of scan based side channel attack on stream ciphers. Section 3 illustrates the proposed attack on MICKEY-128 2.0, Trivium and Grain-128. Finally, section 4 concludes the paper.

2 A Generic Scan Attack on Stream Ciphers

The DFT feature provides the ability to run the system in two modes, normal and test. In the test mode a *scan_in* line allows test input to the IC, which after normal mode of operation of the chip may be scanned out through the *scan_out* line that shows the contents of one register per clock cycle. *Scan_in/scan_out* is a universally accepted feature practised in VLSI test community. But it opens a side channel for attacking crypto-hardware.

A scan based side channel attack works in two phases. In the first phase, the attacker obtains correspondence of the scan chain bit positions with the hardware FFs. This way the attacker is able to obtain the internal state of the cipher. Once, the internal state of the cipher is known, in the next phase, the attacker inverts the state to its initial state to get back the key of the cipher. As the scan chain connects the registers of the IC in an arbitrary manner, the main challenge of an adversary is to determine the exact contents of the state of the cipher from the permutation of FFs in the scan chain. The success of this phase depends on the design of the algorithm. The second phase of attack is concentrated on the possibility of reversibility of cipher states from one state to its previous. However, the second phase is redundant as once the full state of the cipher is known at any cycle, we can predict the next sequence of keystreams easily according to the cipher algorithm.

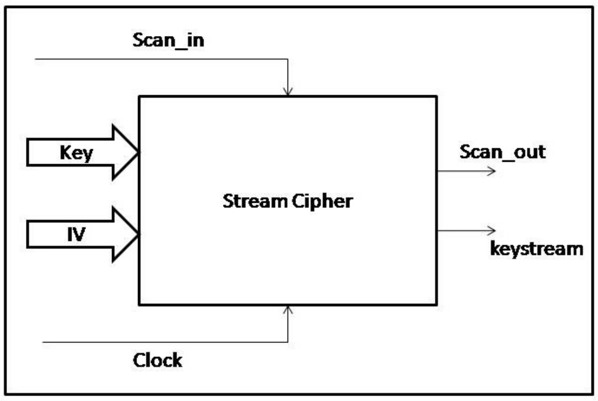

Fig. 1 I/O Interface of Stream Cipher for Scan Attack

In this section, a new scan-based attack is employed successfully on hardware efficient finalists of eStream, MICKEY-128 2.0, Trivium and Grain-128. Our attacking principle require an interface through which we can feed key and IV to the stream cipher as shown in Fig. 1. The proposed methodology for scan-based attack is applicable on any stream cipher when we have the interface, shown in Fig. 1 to the cipher. In this section we propose this general strategy. It may be mentioned that the strategy is equally applicable against block ciphers as well. The attack model is described next.

2.1 Physical and Mathematical Model of Scan Attack

User may input *key* and *IV* bits through the interface (Fig. 1). The hardware implementation has a scan architecture with *scan_in* controllability and *scan_out* observability.

- The user is able to run the cipher for any number of cycles in normal mode by keeping $scan\_in = 0$.
- The state of the internal registers can be obtained through the *scan_out* line when $scan\_in = 1$.
- The order in which scan chain connects the internal registers is *unknown* to the user.

Since, the state of the internal register may be fully known only when the order of getting the bits is known, the challenge of an adversary is to find correspondence of the output bits with the internal registers used in the hardware implementation.

Let, the internal registers of the crypto-chip implementing the crypto-algorithm be the ordered sequence, $R =< r_0, r_1, \ldots, r_{n-1} >$. We take R in such a way that $R =< K, V, X >$, where, K, V are the exact register sequence holding initial private *key* and public *IV* of the algorithm in correct order. X is the rest of the registers in the chip connected to scan chains. Let, the positions $P =< p_0, p_1, \ldots, p_{n-1} >$ denotes the first, second, ... n^{th} position of the *scan_out* in order. Let, $S_j = Content^j(R)$ and $Q_j = Content^j(P)$, where, $Content^j(Y)$ is the *content* of Y after j rounds of operation of the algorithm following the $< key, IV >$ loading phase. Also, let, $s_{ij} = Content^j(r_i)$ and $q_{ij} = Content^j(p_i)$.

Then, by the principle of scan-chains,

$$P = \Pi(R),$$

where, Π represents a permutation operation.

Now, $S_j = Round^j(S_0)$, where,

$$S_0 = Content^0(R) =< k, iv, x >$$

the initial internal state of the chip and exponentiation refers to execution of the *Round* operation of the cipher. *Clearly, knowing S_0 gives away the private key, k, initial vector iv and auxiliary bits x of the algorithm.*

P and $Q_j, \forall j$ are the leakages through the *scan_out* line. We exploit this information to obtain S_0 and thus k.

2.1.1 Obtaining Π^{-1}

The goal to obtain Π^{-1} is equivalent to obtaining a bijection between R and P.

We define an equivalence relation, I on $\{r_0, r_1, \ldots, r_{n-1}\}$ as follows:
For a fixed m and $< key, IV >$ pair,

$$(r_i, r_j) \in I \textit{ if and only if, } s_{il} = s_{jl} \forall l = 0, 1, \ldots, m.$$

Then, we form equivalence classes in I. The set of $r_i, i = 0, 1, \ldots, n-1$ having same value as q_{ij} for $j = 0, 1, 2 \ldots, m$ form an equivalence class E_i. In other words,

$$E_i = \{r_l : s_{lj} = q_{ij} \forall j = 0, 1, \ldots, m\}$$

We then gradually increase m to increase the number of equivalence classes, $||E||$. Clearly, when $||E|| = n$, we have obtained a bijection. Let, for $m = M$ this bijection is obtained. We give in the theorem below the proof that M exists.

Theorem 1: For all i, The sequence $< s_{i0}, s_{i1}, \ldots, s_{im} >$ is pseudo-random for large values of m.

Proof: A good stream cipher generates pseudo-random sequences as output. In other words, $Z_j = C^j(K, V)$ is not polynomial-time distinguishable from a random sequence. If possible let, $\exists$ i, s.t., $< s_{i0}, s_{i1}, \ldots, s_{im} >$ is polynomial-time distinguishable $\forall$ m. Since, $Z_j = C^j(K, V)$, after a number of iterations from the $< key, IV >$ loading phase, say, α, Z_α is a function of $< s_{i0}, s_{i1}, \ldots, s_{i\alpha} >$. Since, $< s_{i0}, s_{i1}, \ldots, s_{i\alpha} >$ is not pseudo-random, after α iterations, we can polynomial-time distinguish $Z_0, Z_1, \ldots, Z_\alpha$ from a random sequence as $Z_j = f(s_{kj} | k = 0, \ldots)$ is a polynomial, for stream ciphers. A contradiction. Hence, after large enough values of m (=α), $< s_{i0}, s_{i1}, \ldots, s_{im} >$ is pseudo-random.

It can be mentioned that there may be *unused state bits* in the algorithm which are directly or indirectly never used in computation of key-streams. These bits can be clearly ignored. Thus, X registers are not required in our attack.

Theorem 2: *N is the minimum number of iterations the cipher algorithm is to be run so that each internal register (consisting of K and V only) have distinct state sequence over the iterations for a fixed $< key, IV >$ pair. N is a characteristic of a crypto-algorithm. N exists for good crypto-systems.*

Proof: $\forall$ i and large enough m, $< s_{i0}, s_{i1}, \ldots, s_{im} >$ is pseudo-random for good crypt-systems. Let, $\beta = max_i(m)$, $i = 0, 1, \ldots, n-1$. Then the probability that after $m > \beta$ iterations, $T = \{< s_{00}, s_{01}, \ldots, s_{0m} >, < s_{10}, s_{11}, \ldots, s_{1m} >, \ldots, < s_{(n-1)0}, s_{(n-1)1}, \ldots, s_{(n-1)m} >\}$ will each generate distinct m-bit Boolean sequences is given by,

$$Pr(U_m) = \frac{(2^m - 1)(2^m - 2) \ldots (2^m - n + 1)}{2^{m(n-1)}}$$
$$= (1 - \frac{1}{2^m})(1 - \frac{2}{2^m}) \ldots (1 - \frac{n-1}{2^m})$$

Clearly, $1 - Pr(U_m) < \epsilon$, for some large value of m and however small ϵ. That is, this uniqueness is guaranteed. Hence, for any good crypto-system, $\exists$ N, for which, $Pr(U_m) = 1$.

There may exist some $< key, IV >$ pairs that do not allow this uniqueness which are ideally not admissible in a good crypto-system. We call such pairs, *weak pair.* Though possible practically, such pairs should be few for any practical crypto-system. A random choice of $< key, IV >$ pair avoids using *weak pairs* and thus reaching uniqueness with high probability.

Theorem 3: *The number of iterations, M, the chip should be run to reach unique state is between $log_2 n$ and N.*

Proof: It is proved that eventually n equivalence classes can be reached, where n represents the number of state bits. The maximum number of iterations the cipher is to be simulated to reach this uniqueness is by definition N. Also, m iterations produces $\frac{n}{2^m}$ equivalence classes. Hence, minimum number of iterations required to reach uniqueness is, $log_2 n$.

Theorem 4: The average number of iterations required to reach n equivalence classes is,

$$\Sigma_{m > log_2 n} m \frac{(2^m - 1)(2^m - 2) \ldots (2^m - n + 1)}{2^{m(n-1)}}.$$

Proof: It follows directly from the value of $Pr(U_m)$ (theorem 2).

2.1.2 Breaking the System

Once, we have obtained Π^{-1}, we know,

$$R = \Pi^{-1}(P).$$

If it is obtained after M rounds of operation, we know, Q_M and thus, due to the bijection, S_M. But,

$$S_M = Round^M(S_0)$$

for that particular secret key and IV which is embedded into the crypto-device. Hence,

$$S_0 = Round^{-M}(S_M) = (Round^{-1}(S_M))^M.$$

If the cipher operation is reversible,i.e., $Round^{-1}$ is known, we get after M inversions $S_0 =< k, v, x >$, from which we get the secret key, k. If, however, $Round^{-1}$ is not known, since,

$$S_{M+1} = Round(S_M)$$

we can always predict the following output key-streams from this knowledge of internal state, thus breaking the system.

2.2 The Generic Scan Attack

In this subsection we present a general methodology to attack any stream cipher using the knowledge of output of scan-chain side channel.

It is a *general* method. It differs from the previously published scan-attacks on ciphers in that it does not use any knowledge of the design of the cipher. For example, in [2], to obtain the correspondence between scan-chain positions and the internal registers the adversary chooses a *particular* $< key, IV >$ pair to input to the chip (Fig. 1) exploiting the design of Trivium. In this proposed methodology, the adversary does not need to know the design of the cipher. *In this procedure, the cipher is treated as a black box having interface like Fig. 1.* A *random* $< key, IV >$ pair is chosen matching certain criteria as described in step 1 later. This random pair is utilized to obtain the correspondence between scan-chain positions and the internal registers of the chip.

The general scan-based side channel attack is based upon the observation that the state register bits of a stream cipher are each expected to be in different states over a time period. For example, if a cipher has states r_0, $r_1 \ldots, r_{n-1}$, over a run of m cycles, r_0, $r_1, \ldots, r_{n-1}$ are expected have gone through different Boolean sequences of length m. In other words, r_0 , $r_1 \ldots, r_{n-1}$ are expected to be (*possibly weak*) pseudo-random sequence generators. Otherwise, if r_0 and r_1 hold same states always, at least one of r_0 or r_1 may be removed from the design without reducing the security of the cipher. Clearly, this is true irrespective of the key and initialization vector used in the cipher. However, *weak pairs* may exist. But,

- It is highly unlikely that the distinction be *not* made for large m (theorem 2), owing to the pseudo-randomness described earlier. In fact, as we have shown N exists for all crypto-systems.
- There is always a key, IV combination that may distinguish between two state registers. Otherwise, those two states may be merged to one or one state may be ignored.

The scan-based attack can, therefore, be framed on any stream cipher through the following two steps:

- **Step 1 - Preprocessing Phase (Π^{-1}):** The first step of our attack is done offline both in software and in hardware with the chip to be analyzed. It consists of three subphases. The first phase requires the knowledge of the cipher algorithm. The second phase uses a hardware implementation with the interface given in Fig. 1. This chip has the same scan-chain configuration as the one the adversary is going to break.
  - **Simulation Phase:** In this step, we simulate the cipher in software. The cipher is initialized with *any* specific key and initialization vector. It is then run for m cycles. The Boolean sequences generated by the state registers $< r_0, r_1, \ldots, r_{n-1} >$ of the cipher are noted in a table. If the sequences generated by $< r_0, r_1, \ldots, r_{n-1} >$ are unique (all different from others) we have the Boolean sequences to distinguish among the state register bits of the stream cipher. On the other hand, if, the sequences generated are not unique, we change the key, initialization vector combination or increase number of rounds of simulation until M to obtain unique sequences (since, N is not known). Thus, at the end of this phase, we have,
    - A key and an initialization vector.
    - Number of rounds the cipher is to be run, m.
    - The unique Boolean sequences for $r_0, r_1, \ldots, r_{n-1}$ over the m cycles of operation, which is a $n \times m$ table, T_1. The i^{th} column of T_1 denotes the Boolean string generated by r_i over m cycles of operation.
  - **Online Phase:** During this phase the adversary has access to the hardware of the algorithm and

the scan-chain. Due to the interface (Fig. 1) assumed, she can input *any* key and IV pair to the cipher. In this phase, we start to run the hardware of the cipher with the key and initialization vector for m cycles noted during the previous phase. We scan out full states of the cipher for the m cycles of its execution. The scanned-out bit sequences are noted in a table over the m cycles, with i^{th} scanned-out bit in column i over the m cycles. Thus, we create another $n \times m$ table, T_2 with i^{th} column denoting the output of the i-th scanned-out bit position for the m cycles of operation.
 - **Deduction Phase:** This phase finds the correspondence between scan-chain bits and cipher state bits. In this phase, we match each column of T_2, with T_1. Thus obtaining the correspondence of scan-chain bits with cipher state bits. For example, if i-th column of T_2 matches with j-th column of T_1, scan-chain bit position i, p_i corresponds to cipher state bit r_j. Hence, after this step we obtain the full correspondence of the scan-chain bits with the state bits of the cipher. This corresponds to sorting both the tables. Hence, the time complexity of this phase is $O(nlogn)$.

Thus, after this state we have obtained the correspondence between internal states of the cipher and the scan-output positions.
The complexity of the attack is determined by the number of state bits of the cipher, n and the number of rounds the cipher needs to be simulated to reach at unique sequences, m. Let, t_s denote the time required to simulate each round of the cipher and t_o denote the execution time of each round of the cipher including its full state scan-out. Then the time required to obtain the full correspondence is, $O(t_s \times m + t_o \times m + nlogn)$. The number of restarts required during the entire process is, m. Clearly, the whole process is practically executable.

- **Step 2 - Actual Attack:** We now are given a crypto-hardware having the same scan-chain configuration as before with a secret key and public IV input to it. Since the adversary knows Π^{-1} for this hardware, she can obtain the internal state of the cipher after *any* number of iterations. Hence, the content of R after m cycles, S_m is known. If the internal state is invertible, after m inversions we get back $< k, iv, x >$ and thus, the secret key. If, however, the inversion operation is not known we can compute $S_{m+1} = Round(S_m)$, therefore, computing the future key-streams and essentially breaking the system.

Note that here we have considered only a single scan chain. An implementation might contain multiple scan chains. If the scan chains are independent, we choose any one among the scan chains and perform the attack as before. On the other hand if the scan chains are dependent, it becomes hard to determine actual values of the scan flip-flops. For example, if 4 scan-chains are compacted through a XOR as the final scan-output, we need to know exact values of a set of scan chains connected to all the scan flip-flops. From the output of the compacted scan-chain we will guess one of the 2^4 possible values of the 4 scan chains. The attack procedure will then consider one scan-chain, ignore others and follow the algorithm described above to determine the correspondence. This incurs exponential number of operations due to 2^4 guesses per step.

If the crypto-system is embedded in a larger implementation, this attack can be applied in a similar way if the input-output characteristics of the whole system is known. In this case, we can apply our algorithm in a straightforward manner. Note that our algorithm treats the system under investigation as a black box and only needs to know the input-output characteristics of the system. Hence, the algorithm remains the same while the complexity is determined by the number of scan flip-flops in the entire implementation.

The attack is practical for crypto-systems when m is practical number of iterations the system can be run. The off-line simulations are performed in software. Hence, as running for m-iterations is practical, the off-line simulations can be performed in few minutes. The on-line phase requires m cycles of operation of the system. It also requires an unprotected scan-chain access of the system.

3 The Generic Scan Attack on Hardware based eStream Winners

The proposed scheme is employed on the three hardware based eStream winners and shows that all the three i.e., MICKEY-128 2.0, Trivium and Grain-128 are vulnerable to this generic strategy.

3.1 Scan Attack against MICKEY-128 2.0

MICKEY-128 2.0 is an eStream winner. Till date very few cryptanalysis or side channel attacks are reported against MICKEY-128 2.0. Recently, two fault-attacks ([14], [6]) are presented against it. However, to the best of our knowledge, no scan-attack were reported against MICKEY-128 2.0.

The cipher is designed using two registers, linear and nonlinear, with clock based updates. It can be mentioned that the cipher does not use shift registers in its design. The detailed specification of MICKEY-128 2.0 may be found in [5].

Scan-based side channel attack using the general strategy is successfully applied to MICKEY-128 2.0 as follows:

– In the preprocessing phase, 128 bit key and IV are initialized to 1.
– MICKEY is allowed to run for 18 cycles to find out the correspondence between scan-chain bits and the cipher state bits.
– Boolean sequence generated by r and s registers of MICKEY over 18 cycles are found out (Table 1).
– Keystream bits are predicted from known internal states of r and s registers.

The total time required for the attack is, $O(t_s \times 18 + t_o \times 18 + 320log320)$, where t_s is simulation time for one round of the cipher and t_o is the scan-out time. This time is quite low and the attack is practically implementable in a few minutes. So we claim that our general strategy for scan attack is successful against MICKEY-128 2.0.

3.2 Scan Attack on Trivium

Trivium is also one of the eStream winner stream cipher. The detailed description of Trivium may be obtained in [8].

A number of cryptanalysis and side channel attacks are known against Trivium. Most of the attacks on Trivium are on its reduced round versions. A scan-based side channel attack on Trivium is depicted on [18]. The proposed attack in [18] takes 4×288 cycles to obtain the full correspondence of the scan-chain bits with the state register bits.

Our attack based on the general strategy takes only $N = 108$ cycles of operation to retrieve full internal state of Trivium. The attack works as follows:

– In preprocessing phase, 80-bit key and IV are both set to 1.
– The cipher is allowed to run for 108 cycles to obtain correspondence between scan-chain bits and cipher state bits.
– Boolean sequence generated by s register over 108 cycles is found out. A sample list of sequences is given in table 2.
– Thus the internal state of Trivium being known and the cipher operation being reversible [2], we can revert back to the initial configuration and the initial secret key can be obtained.

The complexity of the attack is, $O(t_s \times 108 + t_o \times 108 + 288log288)$, where t_s is simulation time for one round of the cipher and t_o is the scan-out time. This takes only few minutes to realize in practice. Thus, the general strategy of our scan attack also is successful on Trivium.

3.3 Scan Attack on Grain-128

Grain-128 is also an eStream winner. Till date, very few cryptanalysis are reported against Grain-128 ([7] etc.).

The cipher is designed using two registers, linear and nonlinear left shift registers. The most least significant bits (s_{127} and b_{127}) are updated by a linear and a nonlinear feedback per cycle of operation respectively. The keystream is output based on a nonlinear mixing of internal state bits of this cipher. During initialization the keystream bit is not output but is XOR-ed with both linear and nonlinear feedbacks. A detailed specification of Grain-128 may be found in [9].

Scan-based side channel attack using the general strategy can be applied to Grain-128 as follows:

– During preprocessing phase, key and IV are both set to 1. We used 128 bit key and 96 bit IV.
– The cipher is allowed to run for 167 cycles.
– In 167 cycles only the correspondence between scan-chain bits and cipher state bits can be obtained.
– Boolean sequence generated by b and s registers are obtained. A sample list of obtained Boolean sequences is tabulated in table 3.
– Once, the full internal state of Grain is known, we should be able to revert iterations of Grain-128 due to its reversibility property suggested in [7] to retrieve back its original key.

The total time required for the attack is, $O(t_s \times 167 + t_o \times 167 + 256log256)$, where t_s is the simulation time for one round of the cipher and t_o is scan-out time. The time required is quite low and therefore the attack is mountable in a few minutes. Thus, our generic scan attack succeeds against Grain in practical scenario.

3.4 Performance

All the three crypto-systems may thus be seen to be extremely vulnerable against scan-based side channel attack methodology proposed in the previous section. In Tab. 4 we summarize the cost of our proposed attack on the three stream ciphers. Table 4 also compares the performance of the present attack with the existing attacks in terms of number of cycles of operations required. It can be mentioned that only Trivium has a reported scan-attack.

Table 1 Boolean Sequence of MICKEY-128 2.0 State Bits

State	*Sequence*	*State*	*Sequence*	*State*	*Sequence*	*State*	*Sequence*
r_0	010010001101110111	s_0	010010001101110111	r_1	001101110111011111	s_1	001101110111011111
r_2	100110101010101011	s_2	100110101010101011	r_3	011111100101110101	s_3	011111100101110101
r_4	011001011110101001	s_4	011001011110101001	r_5	011010110011100011	s_5	011010110011100011
r_6	101001101001010001	s_6	101001101001010001	r_7	011000100100101100	s_7	011000100100101100
r_8	111010011110000001	s_8	111010011110000001	r_9	010101111110000000	s_9	010101111110000000
r_{10}	111100100011110111	s_{10}	111100100011110111	r_{11}	000100011100101000	s_{11}	000100011100101000
r_{12}	100010001111110100	s_{12}	100010001111110100	r_{13}	111101110110111110	s_{13}	111101110110111110
r_{14}	100100100111101100	s_{14}	100100100111101100	r_{15}	011110010010110110	s_{15}	011110010010110110
r_{16}	111001110100001000	s_{16}	111001110100001000	r_{17}	010100101011100100	s_{17}	010100101011100100
r_{18}	101110010101010010	s_{18}	101110010101010010	r_{19}	111011111011001101	s_{19}	111011111011001101
r_{20}	100111010000110001	s_{20}	100111010000110001	r_{21}	011111001000011100	s_{21}	011111001000011100
r_{22}	101011000100001010	s_{22}	101011000100001010	r_{23}	111001000011100101	s_{23}	111001000011100101
r_{24}	010100110001010010	s_{24}	010100110001010010	r_{25}	011100010101011010	s_{25}	011100010101011010
r_{26}	101010001011001001	s_{26}	101010001011001001	r_{27}	011001110101000100	s_{27}	011001110101000100
r_{28}	001000101011000010	s_{28}	001000101011000010	r_{29}	000100010101000001	s_{29}	000100010101000001
r_{30}	110000000110110111	s_{30}	110000000110110111	r_{31}	010000000010011111	s_{31}	010000000010011111
r_{32}	111110001100011100	s_{32}	111110001100011100	r_{33}	010111110111101010	s_{33}	010111110111101010
r_{34}	001111011010110101	s_{34}	001111011010110101	r_{35}	010101000000001001	s_{35}	010101000000001001
r_{36}	011100111101110011	s_{36}	011100111101110011	r_{37}	001010011111011101	s_{37}	001010011111011101
r_{38}	110111110011011101	s_{38}	110111110011011101	r_{39}	010011011001001010	s_{39}	010011011001001010
r_{40}	101101001100100101	s_{40}	101101001100100101	r_{41}	011010110111110010	s_{41}	011010110111110010
r_{42}	111011100111001010	s_{42}	111011100111001010	r_{43}	000111011111010010	s_{43}	000111011111010010
r_{44}	000011001110101101	s_{44}	000011001110101101	r_{45}	000001000110010110	s_{45}	000001000110010110
r_{46}	010010111111111000	s_{46}	010010111111111000	r_{47}	001101101110111000	s_{47}	001101101110111000
r_{48}	100110100110011000	s_{48}	100110100110011000	r_{49}	011111100010001000	s_{49}	011111100010001000
r_{50}	011001011100010011	s_{50}	011001011100010011	r_{51}	111010110010011010	s_{51}	111010110010011010
r_{52}	110101101001101001	s_{52}	110101101001101001	r_{53}	000000101001000011	s_{53}	000000101001000011
r_{54}	110010011001010110	s_{54}	110010011001010110	r_{55}	000011110001011000	s_{55}	000011110001011000
r_{56}	110011010101011111	s_{56}	110011010101011111	r_{57}	100011000110111100	s_{57}	10001100011011 1100
r_{58}	111101000010011010	s_{58}	111101000010011010	r_{59}	010110110001101001	s_{59}	010110110001101001
r_{60}	011101100101000011	s_{60}	011101100101000011	r_{61}	111000101110110110	s_{61}	111000101110110110
r_{62}	000110011011101000	s_{62}	000110011011101000	r_{63}	110001110000100011	s_{63}	110001110000100011
r_{64}	110000101000010001	s_{64}	110000101000010001	r_{65}	000010011001111011	s_{65}	000010011001111011
r_{66}	010011110001001110	s_{66}	010011110001001110	r_{67}	001101011000100111	s_{67}	001101011000100111
r_{68}	000110111100010011	s_{68}	000110111100010011	r_{69}	010001100010011010	s_{69}	010001100010011010
r_{70}	001100100001101001	s_{70}	001100100001101001	r_{71}	100110010000110100	s_{71}	100110010000110100
r_{72}	111111111000011110	s_{72}	111111111000011110	r_{73}	100101010001111100	s_{73}	100101010001111100
r_{74}	101100110101001101	s_{74}	101100110101001101	r_{75}	011010011011000110	s_{75}	011010011011000110
r_{76}	011011110000110100	s_{76}	011011110000110100	r_{77}	101001011000011110	s_{77}	101001011000011110
r_{78}	011000111100001011	s_{78}	011000111100001011	r_{79}	011010010010010010	s_{79}	011010010010010010
r_{80}	111011110100011010	s_{80}	111011110100011010	r_{81}	000111010110011110	s_{81}	000111010110011110
r_{82}	110001000111111100	s_{82}	110001000111111100	r_{83}	110000110010111010	s_{83}	110000110010111010
r_{84}	010000011001111001	s_{84}	010000011001111001	r_{85}	011110000001001111	s_{85}	011110000001001111
r_{86}	011001111101010000	s_{86}	011001111101010000	r_{87}	101000101111001100	s_{87}	101000101111001100
r_{88}	011000010110100110	s_{88}	011000010110100110	r_{89}	101000001010010011	s_{89}	101000001010010011
r_{90}	001010001000011010	s_{90}	001010001000011010	r_{91}	010111111001111110	s_{91}	010111111001111110
r_{92}	111101010001001100	s_{92}	111101010001001100	r_{93}	010110111000100110	s_{93}	010110111000100110
r_{94}	001111101100010011	s_{94}	001111101100010011	r_{95}	010101011010011010	s_{95}	010101011010011010
r_{96}	001110111101101001	s_{96}	001110111101101001	r_{97}	010101100010100011	s_{97}	010101100010100011
r_{98}	101110100001110001	s_{98}	101110100001110001	r_{99}	111011100000111100	s_{99}	111011100000111100
r_{100}	100111011101101101	s_{100}	100111011101101101	r_{101}	101101000010100001	s_{101}	101101000010100001
r_{102}	111010110001110000	s_{102}	111010110001110000	r_{103}	010101101000111100	s_{103}	010101101000111100
r_{104}	001110100100011010	s_{104}	001110100100011010	r_{105}	110101101110011110	s_{105}	110101101110011110
r_{106}	100000101011111100	s_{106}	100000101011111100	r_{107}	001110011000101101	s_{107}	001110011000101101
r_{108}	110101110001100001	s_{108}	110101110001100001	r_{109}	100000100101000111	s_{109}	100000100101000111
r_{110}	011100010011000011	s_{110}	011100010011000011	r_{111}	011000000100110110	s_{111}	011000000100110110
r_{112}	011010001110001000	s_{112}	011010001110001000	r_{113}	111011111011110011	s_{113}	111011111011110011
r_{114}	110101011101011101	s_{114}	110101011101011101	r_{115}	100000110010111101	s_{115}	100000110010111101
r_{116}	101110010100001101	s_{116}	101110010100001101	r_{117}	001001110110010001	s_{117}	001001110110010001
r_{118}	000100101010001100	s_{118}	000100101010001100	r_{119}	100010010101100110	s_{119}	100010010101100110
r_{120}	011101111011010011	s_{120}	011101111011010011	r_{121}	101010101101001101	s_{121}	101010101101001101
r_{122}	011001100111000110	s_{122}	011001100111000110	r_{123}	001000100010100011	s_{123}	001000100010100011
r_{124}	100100010001110001	s_{124}	100100010001110001	r_{125}	111110001000111100	s_{125}	111110001000111100
r_{126}	010111110100011010	s_{126}	010111110100011010	...	...	...	...

Table 2 Sample Boolean Sequences of Trivium

State	Sequence
s_0	110011111111111111111111111111111111111111111111111111111111111110001110 0000000000000000000000000000000000
s_1	1110011111111111111111111111111111111111111111111111111111111111111000111 0000000000000000000000000000000000
s_2	11110011111111111111111111111111111111111111111111111111111111111111100011 1000000000000000000000000000000000
s_3	11111001111111111111111111111111111111111111111111111111111111111111110001 1100000000000000000000000000000000
s_4	11111100111111111111111111111111111111111111111111111111111111111111111000 1110000000000000000000000000000000
s_5	11111110011111111111111111111111111111111111111111111111111111111111111100 0111000000000000000000000000000000
s_6	111111110011111111111111111111111111111111111111111111111111111111111111100 011100000000000000000000000000000
s_7	1111111110011111111111111111111111111111111111111111111111111111111111111100 01110000000000000000000000000000
s_8	1111111111001111111111111111111111111111111111111111111111111111111111111110 00111000000000000000000000000000
s_9	1111111111100111111111111111111111111111111111111111111111111111111111111111 00011100000000000000000000000000
s_{10}	111111111111001111111111111111111111111111111111111111111111111111111111111 110001110000000000000000000000000
s_{11}	111111111111100111111111111111111111111111111111111111111111111111111111111 111000111000000000000000000000000
s_{12}	111111111111110011111111111111111111111111111111111111111111111111111111111 111100011100000000000000000000000
s_{13}	111111111111111001111111111111111111111111111111111111111111111111111111111 111110001110000000000000000000000
s_{14}	111111111111111100111111111111111111111111111111111111111111111111111111111 111111000111000000000000000000000
s_{15}	111111111111111110011111111111111111111111111111111111111111111111111111111 111111100011100000000000000000000
s_{16}	111111111111111111001111111111111111111111111111111111111111111111111111111 111111110001110000000000000000000
s_{17}	111111111111111111100111111111111111111111111111111111111111111111111111111 111111111000111000000000000000000
s_{18}	111111111111111111110011111111111111111111111111111111111111111111111111111 111111111100011100000000000000000
s_{19}	111111111111111111111001111111111111111111111111111111111111111111111111111 111111111110001110000000000000000
s_{20}	111111111111111111111100111111111111111111111111111111111111111111111111111 111111111111000111000000000000000
s_{21}	111111111111111111111110011111111111111111111111111111111111111111111111111 111111111111100011100000000000000
.	.
.	.
s_{277}	00000000000000000000000000000000000000000000000000000000000000000000000000 0000000000000000000000000001110111
s_{278}	00000000000000000000000000000000000000000000000000000000000000000000000000 0000000000000000000000000000111011
s_{279}	00000000000000000000000000000000000000000000000000000000000000000000000000 0000000000000000000000000000011101
s_{280}	00000000000000000000000000000000000000000000000000000000000000000000000000 0000000000000000000000000000001110
s_{281}	00000000000000000000000000000000000000000000000000000000000000000000000000 0000000000000000000000000000000111
s_{282}	00000000000000000000000000000000000000000000000000000000000000000000000000 0000000000000000000000000000000011
s_{283}	00000000000000000000000000000000000000000000000000000000000000000000000000 0000000000000000000000000000000001
s_{284}	00000000000000000000000000000000000000000000000000000000000000000000000000 0000000000000000000000000000000000
s_{285}	10000000000000000000000000000000000000000000000000000000000000000000000000 0000000000000000000000000000000000
s_{286}	11000000000000000000000000000000000000000000000000000000000000000000000000 0000000000000000000000000000000000
s_{287}	11100000000000000000000000000000000000000000000000000000000000000000000000 0000000000000000000000000000000000

Table 3 Sample Boolean Sequences of Grain-128

State	Sequence
b_0	11111111111111111111111111111111111111111111111111111111111111111111111111111111111 111111111111111111111111111111111111111111111000000000000000000000000000000001111100
b_1	11111111111111111111111111111111111111111111111111111111111111111111111111111111111 1111111111111111111111111111111111111111111100000000000000000000000000000000011111001
b_2	11111111111111111111111111111111111111111111111111111111111111111111111111111111111 11111111111111111111111111111111111111111110000000000000000000000000000000000111110011
b_3	11111111111111111111111111111111111111111111111111111111111111111111111111111111111 111111111111111111111111111111111111111111000000000000000000000000000000001111100111
b_4	11111111111111111111111111111111111111111111111111111111111111111111111111111111111 1111111111111111111111111111111111111111100000000000000000000000000000000011111001111
b_5	11111111111111111111111111111111111111111111111111111111111111111111111111111111111 11111111111111111111111111111111111111110000000000000000000000000000000000111110011111
b_6	11111111111111111111111111111111111111111111111111111111111111111111111111111111111 111111111111111111111111111111111111111000000000000000000000000000000001111100111110
b_7	11111111111111111111111111111111111111111111111111111111111111111111111111111111111 1111111111111111111111111111111111111100000000000000000000000000000000011111001111100
b_8	11111111111111111111111111111111111111111111111111111111111111111111111111111111111 11111111111111111111111111111111111110000000000000000000000000000000000111110011111000
b_9	11111111111111111111111111111111111111111111111111111111111111111111111111111111111 111111111111111111111111111111111111000000000000000000000000000000001111100111110000
b_{10}	1111111111111111111111111111111111111111111111111111111111111111111111111111111111 11111111111111111111111111111111111100000000000000000000000000000000011111001111100000
b_{11}	1111111111111111111111111111111111111111111111111111111111111111111111111111111111 1111111111111111111111111111111111100000000000000000000000000000000000111110011111000000
b_{12}	1111111111111111111111111111111111111111111111111111111111111111111111111111111111 11111111111111111111111111111111110000000000000000000000000000000000001111100111110000000
b_{13}	1111111111111111111111111111111111111111111111111111111111111111111111111111111111 111111111111111111111111111111111000000000000000000000000000000000011111001111100000000
b_{14}	1111111111111111111111111111111111111111111111111111111111111111111111111111111111 1111111111111111111111111111111100000000000000000000000000000000000111110011111000000000
b_{15}	1111111111111111111111111111111111111111111111111111111111111111111111111111111111 11111111111111111111111111111110000000000000000000000000000000000001111100111110000000000
b_{16}	1111111111111111111111111111111111111111111111111111111111111111111111111111111111 111111111111111111111111111111000000000000000000000000000000000011111001111100000000000
.	.
.	.
s_{112}	1111111111111111111111111111111111111111111111111111111000000000000000011111111100000 0001100101111111100110000000000101000110000100001101010100001011100110001010101000
s_{113}	1111111111111111111111111111111111111111111111111111110000000000000000111111111000000 0011001011111111001100000000001010001100001000011010101000010111001100010101010001
s_{114}	1111111111111111111111111111111111111111111111111111100000000000000001111111110000000 0110010111111110011000000000010100011000010000110101010000101110011000101010100010
s_{115}	1111111111111111111111111111111111111111111111111111000000000000000011111111100000000 1100101111111100110000000000101000110000100001101010100001011100110001010101000101
s_{116}	1111111111111111111111111111111111111111111111111110000000000000000111111111000000001 1001011111111001100000000001010001100001000011010101000010111001100010101010001011
s_{117}	1111111111111111111111111111111111111111111111111100000000000000001111111110000000011 0010111111110011000000000010100011000010000110101010000101110011000101010100010110
s_{118}	1111111111111111111111111111111111111111111111111000000000000000011111111100000000110 0101111111100110000000000101000110000100001101010100001011100110001010101000101100
s_{119}	1111111111111111111111111111111111111111111111110000000000000000111111111000000001100 1011111111001100000000001010001100001000011010101000010111001100010101010001011000
s_{120}	1111111111111111111111111111111111111111111111100000000000000001111111110000000011001 0111111110011000000000010100011000010000110101010000101110011000101010100010110001
s_{121}	1111111111111111111111111111111111111111111111000000000000000011111111100000000110010 1111111100110000000000101000110000100001101010100001011100110001010101000101100011
s_{122}	1111111111111111111111111111111111111111111110000000000000000111111111000000001100 1011111111001100000000001010001100001000011010101000010111001100010101010001011000111
s_{123}	1111111111111111111111111111111111111111111100000000000000001111111110000000011001 0111111110011000000000010100011000010000110101010000101110011000101010100010110001110
s_{124}	1111111111111111111111111111111111111111111000000000000000011111111100000000110010 1111111100110000000000101000110000100001101010100001011100110001010101000101100011101
s_{125}	1111111111111111111111111111111111111111110000000000000000111111111000000001100101 1111111001100000000001010001100001000011010101000010111001100010101010001011000111011
s_{126}	1111111111111111111111111111111111111111100000000000000001111111110000000011001011 1111110011000000000010100011000010000110101010000101110011000101010100010110001110111
s_{127}	1111111111111111111111111111111111111111000000000000000011111111100000000110010111 1111100110000000000101000110000100001101010100001011100110001010101000101100011101111

Table 4 Comparison of Attack Complexity

Cipher	Cycles Req.(Proposed)	Cycles Req. (Existing)
MICKEY-128 2.0	18	-
Trivium	108	1152 [2]
Grain-128	167	-

4 Conclusion

In this paper, we have shown that if the state of the art testable design techniques is employed for cryptographic hardware, then the design is vulnerable against scan based side channel attack. We also propose a methodology by which internal state of any stream cipher may be extracted using scan chains in realistic time. The proposed strategy is successfully employed on hardware efficient winners of eStream, MICKEY-128 2.0, Trivium and Grain-128. The attack is practically mountable as it works within few cycles of operations of the ciphers.

References

1. The eSTREAM Project. "http://www.ecrypt.eu.org/stream/" (2004)
2. Agrawal, M., Karmakar, S., Saha, D., Mukhopadhayay, D.: Scan Based Side Channel Attacks on Stream Ciphers and their Counter-measures. Progress in Cryptology - INDOCRYPT 2008 **5365/2008**, 226–238 (2008)
3. Arslan, B., Orailoglu, A.: Circularscan: A scan architecture for test cost reduction. DATE 2002 (2004)
4. Babbage, S., Canniere, C.D., Canteaut, A., Cid, C., Gilbert, H., Johansson, T., Parker, M., Preneel, B., Rijmen, V., Robshaw, M.: The eSTREAM Portfolio. "http://www.ecrypt.eu.org/stream/portfolio.pdf" (2009)
5. Babbage, S., Dodd, M.: The stream cipher MICKEY 2.0. eSTREAM, ECRYPT Stream Cipher Project **2006** (2006)
6. Banik, S., Maitra, S.: A differential fault attack on mickey 2.0. In: G. Bertoni, J.S. Coron (eds.) Cryptographic Hardware and Embedded Systems - CHES 2013, *Lecture Notes in Computer Science*, vol. 8086, pp. 215–232. Springer Berlin Heidelberg (2013)
7. Berzati, A., Canovas, C., Castagnos, G., Debraize, B., Goubin, L., Gouget, A., Paillier, P., Salgado, S.: Fault analysis of GRAIN-128. Hardware-Oriented Security and Trust, IEEE International Workshop on **0**, 7–14 (2009)
8. Canniere, C.D., Preneel, B.: TRIVIUM Specifications. eSTREAM, ECRYPT Stream Cipher Project (2006)
9. Hell, M., Johansson, T., Meier, W.: A Stream Cipher Proposal: Grain-128. eSTREAM, ECRYPT Stream Cipher Project **2006** (2006)
10. Hely, D., Bancel, F., Flottes, M.L., Rouzeyre, B.: Test control for secure scan designs. ETS 2005: Proceedings of the 10th IEEE European Symposium on Test, Washington, DC, USA pp. 190–195 (2005)
11. Hely, D., Bancel, F., Flottes, M.L., Rouzeyre, B.: A secure scan design methodology. DATE 2006: Proceedings of the conference on Design, automation and test in Europe, 3001 Leuven, Belgium pp. 1177–1178 (2006)
12. Hely, D., Bancel, F., Flottes, M.L., Rouzeyre, B.: Secure scan techniques: A comparison. IOLTS 2006: Proceedings of the 12th IEEE International Symposium on On-Line Testing, Washington, DC, USA pp. 119–124 (2006)
13. Hely, D., Flottes, M.L., Bancel, F., Rouzeyre, B., Berard, N., Renovell, M.: Scan design and secure chip. IOLTS 2004: Proceedings of the 10th IEEE International On-Line Testing Symposium, Washington, DC, USA p. 219 (2004)
14. Karmakar, S., Chowdhury, D.R.: Differential fault analysis of mickey-128 2.0. In: Fault Diagnosis and Tolerance in Cryptography (FDTC), 2013 Workshop on, pp. 52–59 (2013). DOI 10.1109/FDTC.2013.8
15. Lee, J., Tehranipoor, M., Patel, C., Plusquellic, J.: Securing scan design using lock and key technique. DFT 2005: Proceedings of the 20th IEEE International Symposium on Defect and Fault Tolerance in VLSI Systems, Washington, DC, USA pp. 51–62 (2005)
16. Liu, Y., Wu, K., Karri, R.: Scan-based attacks on linear feedback shift register based stream ciphers. ACM Trans. Des. Autom. Electron. Syst. **16**(2), 20:1–20:15 (2011). DOI 10.1145/1929943.1929952. URL http://doi.acm.org/10.1145/1929943.1929952
17. Mukhopadhyay, D., Banerjee, S., RoyChowdhury, D., Bhattacharya, B.B.: Cryptoscan: A secured scan chain architecture. ATS 2005: Proceedings of the 14th Asian Test Symposium on Asian Test Symposium, Washington, DC, USA pp. 348–353 (2005)
18. Sengar, G., Mukhopadhyay, D., Chowdhury, D.R.: Secured flipped scan-chain model for crypto-architecture. IEEE Trans. on CAD of Integrated Circuits and Systems **26(11)**, 2080–2084 (2007)
19. Yang, B., Wu, K., Karri, R.: Scan based side channel attack on dedicated hardware implementations of data encryption standard. ITC 2004: Proceedings of the International Test Conference, Washington, DC, USA pp. 339–344 (2004)
20. Yang, B., Wu, K., Karri, R.: Secure scan: A design-for-test architecture for crypto chips. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems **25(10)**, 2287–2293 (2006)