SCREAM & iSCREAM
Side-Channel Resistant Authenticated Encryption with Masking

Vincent Grosso[1*] Gaëtan Leurent[2] François-Xavier Standaert[1†] Kerem Varici[1‡]
François Durvaux[1*] Lubos Gaspar[1‡] Stéphanie Kerckhof[1§]

[1] ICTEAM/ELEN/Crypto Group, Université catholique de Louvain, Belgium.
[2] Inria, EPI Secret, Rocquencourt, France.

Contact e-mail: scream@uclouvain.be

Version 1, March 2014.

Abstract

This document defines the family of authenticated encryption (with associated data) algorithms SCREAM and iSCREAM. They are based on Liskov et al.'s Tweakable Authenticated Encryption (TAE) mode with the new tweakable block ciphers Scream and iScream.

The main desirable features of SCREAM and iSCREAM are:

- A *simple* and *regular* design allowing excellent performances on a wide range of architectures, in particular if *masking* is implemented *as a side-channel countermeasure*;
- Inheriting from TAE, *security beyond the birthday bound*, i.e. a 128-bit security guarantee with up to 2^{128} blocks of data processed with the same 128-bit key;
- *Low overheads* for the authentication mode (e.g. no extra cipher calls to generate masks);
- *Fully parallelisable* authenticated encryption with *minimal ciphertext length*.

In addition, iSCREAM allows *compact implementations* for combined encryption and decryption, by taking advantage of involutive components in its underlying cipher iScream.

* PhD student funded by the ERC Project 280141 (acronym CRASH).
† Associate researcher of the Belgian fund for scientific research (FNRS - F.R.S).
‡ Post-doctoral researcher funded by the ERC Project 280141 (acronym CRASH).
§ PhD student funded by a FRIA grant, Belgium.

1 Design rationale

The following ciphers and encryption modes aim to allow implementations that are secure against *Side-Channel Attacks* (SCAs) such as Differential Power Analysis (DPA) [15] and Electro-Magnetic Analysis (EMA) [8]. We believe they are important threats to the security of modern computing devices for an increasingly wider class of applications. In this context, numerous countermeasures have been introduced in the literature (a good survey can be found in [17]). Our designs will focus on SCA security based on masking (aka secret sharing) for three main reasons. First, it is a thoroughly investigated countermeasure with well established benefits (a security gain that can be exponential in the number of shares [4, 25, 31]) and limitations (the requirement that the leakage of each share is independent of the others [6, 18]). Second, it can take advantage of algorithms tailored for this purpose, as we envision here and illustrated by recent block cipher proposals such as PiCaRo [22], Zorro [10], Robin and Fantomas [11]. Third, it can be implemented efficiently and securely both in software [9, 26, 28] and hardware devices [20, 21]. As a result, and without neglecting the need to combine masking with other countermeasures to reach high physical security levels, we believe it is an important building block in the design of side-channel resistant implementations.

Based on these premises, two important additional criteria are *implementation efficiency and design regularity/simplicity*. The first one is motivated by the fact that more operations inevitably mean more leaking operations that may be exploited by a clever adversary (e.g. such as the algebraic one in [27]). The second one derives from the observation that physically secure implementations are easier to obtain if computations are performed on well aligned data. For example, manipulating bits and bytes such as in the PRESENT block cipher [3] raises additional challenges for the developers (to guarantee that the bit manipulations do not leak more information than the byte ones). As a result, we also aim for implementation efficiency on various platforms, with performances close to the ones of the AES in an unprotected setting, and significantly improved when the masking countermeasure is activated. Concretely, this includes privileging highly parallel designs.

As far as the block cipher used in our proposal is concerned, the LS-designs recently introduced at FSE 2014 appear as natural candidates to reach the previous goals [11] – we will take advantage of their general structure. As for the authenticated encryption mode, two main options are available. The first one is to directly exploit a block cipher based solution, for which the extra operations required for authentication are as linear (hence, easy to mask) as possible. Depending on the desired implementation and security properties (e.g. parallelism, need of decryption, misuse resistance), modes such as OCB [30], OTR [19], COPA [1], or COBRA [2] could be considered for this purpose. Yet, a drawback of such schemes is that they only guarantee birthday security. Alternatively, one can take advantage of the Tweakable Authenticated Encryption (TAE) proposed by Liskov et al. [16], which looses nothing in terms of its advantage of the underlying tweakable block cipher, hence can provide *beyond birthday security* – we will opt for this second solution.

Instantiating TAE requires a tweakable block cipher, which we achieve by extending the previously proposed block ciphers Robin and Fantomas. Our main ingredient for this purpose is the addition of a lightweight tweak/key scheduling algorithm. In this respect, our choices were oriented by the conclusions in [14], where it was observed that allowing *round keys (and tweaks)* to be *derived* "*on-the-fly*" both in encryption and decryption can significantly improve the hardware performances in certain contexts. We also follow the recommendation that an *efficient combination of encryption and decryption* can be highly beneficial to the hardware implementation cost, and therefore propose instances of ciphers based on both involutive and non-involutive components.

Finally, and since we care about physical security issues for which developers anyway have to pay attention to implementation aspects (e.g. masking is useless in case of biased randomness), we will not consider misuse resistance as a goal. For similar reasons, we will propose instances of our ciphers with and without security guarantees against related-key attacks – the later ones being the most relevant for our intended case studies. Yet, we note that combining misuse resistance with a leakage-resilient authenticated encryption mode of operation is an interesting scope for further research. Indeed, both properties generally imply reducing the designs' parallelism.

In the following, we will denote our non-involutive block cipher as Scream, our involutive block cipher as iScream, the TAE based on Scream as SCREAM, and the TAE based on iScream as iSCREAM. The designers have not hidden any weakness in any of these ciphers/modes.

2 Security goals

Our security goals are summarized in Table 1. There is no secret message number. The public message number is a nonce. The cipher does not promise intergrity or confidentiality if the legitimate key holder uses the same nonce to encrypt two different (plaintext,assicoated data) pairs under the same key. The numbers in the table correspond to key guesses to find the secret key for confidentiality, and to online forgery attempts for integrity. Any successful forgery or key recovery should be assumed to completely compromise confidentiality and integrity of all messages.

Table 1: Summary of our security goals.

	SCREAM bits of security	iSCREAM bits of security
Confidentiality of the plaintext	128	128
Integrity of the plaintext	128	128
Integrity of the associated data	128	128
Integrity of the public message number	128	128
Side-channel resistance	masking	masking
Related-key security	optional	optional
Misuse resistance	no	no

The lower part of the table contains qualitative security statements. Side-channel resistant implementations are expected to be achieved with masking. Related-key security is optional and can be obtained with an increased number of rounds. Misuse resistance is not claimed.

3 Specifications

3.1 Tweakable LS-designs

Both Scream and iScream are based on a variant of the LS-designs introduced in [11] that we will denote as Tweakable LS-designs (TLS-designs). They essentially update a n-bit state x by iterating N_s steps, each of them made of N_r rounds. The state is structured as a $l \times s$ matrix, such that $x[i, \star]$ represents a row and $x[\star, j]$ represents a column. The first row contains state bits 0 to $l-1$, the second row contains state bits l to $2l-1, \ldots$ In the following, the number of rounds per step will be fixed to $N_r = 2$. By contrast, the number of steps will vary and will serve as a parameter to adapt

Algorithm 1 TLS-design with l-bit L-boxes and s-bit S-boxes ($n = l \cdot s$)

```
x ← P ⊕ TK(0);                                   ▷ x is a l × s bits matrix
for 0 < σ ≤ N_s do
    for 0 < ρ ≤ N_r do
        r = 2 · (σ − 1) + ρ;                     ▷ Round index
        for 0 ≤ j < l do                         ▷ S-box Layer
            x[⋆, j] = S[x[⋆, j]];
        end for
        x ← x ⊕ C(r);                            ▷ Constant addition
        for 0 ≤ i < s do                         ▷ L-box Layer
            x[i, ⋆] = L[x[i, ⋆]];
        end for
    end for
    x ← x ⊕ TK(σ);                               ▷ Tweakey addition
end for
return x
```

the security margins in Section 5. One significant advantage of TLS-designs is their simplicity: they can be described in a couple of lines, as illustrated in Algorithm 1. In this algorithm, P denotes the plaintext, TK a combination of the master key K and tweak T that we will call tweakey. Finally, S and L are the s-bit S-boxes and l-bit L-boxes that are used in all TLS-designs.

3.2 The tweakable block ciphers Scream and iScream

Scream and iScream are n=128-bit ciphers with s=8-bit S-boxes and l=16-bit L-boxes. Specifying them requires to define these components, together with the round constants. The table representation, algebraic normal form and bitslice implementation of the Scream S-box are given in Appendices A.1 and A.2. The binary representation and 8-bit table representation of the Scream L-box are given in Appendices A.3 and A.4. Similar definitions are given for the iScream S-box and L-box, in Appendix B.1 and B.2. Round constants are defined in Appendix C and are the same for both ciphers. Finally, Scream and iScream have a slightly different tweakey scheduling algorithm as we now detail. They both take the 128-bit key K and the 128-bit tweak T as input.

Scream tweakey scheduling. The tweak is divided into 64-bit halves: $T = t_0 \parallel t_1$. Then, three different tweakeys are used every three steps as follows:

$$\begin{aligned} TK(\sigma = 3i) &= K \oplus (t_0 \parallel t_1), \\ TK(\sigma = 3i+1) &= K \oplus (t_0 \oplus t_1 \parallel t_0), \\ TK(\sigma = 3i+2) &= K \oplus (t_1 \parallel t_0 \oplus t_1). \end{aligned}$$

The tweakeys can also be computed on-the-fly using a simple linear function ϕ, corresponding to multiplication by a primitive element in $GF(4)$ (such that $\phi^2(x) = \phi(x) \oplus x$, and $\phi^3(x) = x$):

$$\begin{aligned} \phi : x_0 \parallel x_1 &\mapsto (x_0 \oplus x_1) \parallel x_0, \\ \tau_0 &= T, \\ \tau_{i+1} &= \phi(\tau_i), \\ TK(i) &= K \oplus \tau_i. \end{aligned}$$

iScream tweakey scheduling. Two different tweakeys are used every two steps as follows:

$$TK(\sigma = 2i) = T \oplus K,$$
$$TK(\sigma = 2i+1) = (T \overset{16}{\lll} 1),$$

where $\overset{16}{\lll}$ is a rotation of one bit applied independently to all the (16-bit) rows of the state.

3.3 The encryption modes SCREAM and iSCREAM

We use the tweakable block ciphers Scream and iScream in the TAE mode proposed in [16]. A plaintext $(P_0, \cdots, P_{m-1})$ is encrypted using a nonce (next denoted as N) – the algorithm produces a ciphertext $(C_0, \cdots, C_{m-1})$ and a tag T. Blocks of associated data $(A_0, \cdots, A_{q-1})$ can optionally be authenticated with the message, without being encrypted. During the decryption process, the ciphertext values, tag and associated data are used to recover the plaintext. If the tag is incorrect, the algorithm returns a null output $\perp$. The nonce is supplied by the user, and every message encrypted with a given key must use a different value for N. The nonce bytesize n_b can be chosen by the user between 1 and 15 bytes (the recommended value is 12 bytes), and every message encrypted with a given key must use the same nonce size. The length of the associated data is limited to 2^{130-8n_b} bytes and the length of the message is limited to 2^{131-8n_b} bytes.

There are three main steps in the mode of operation. First, the associated data is processed by dividing it into 128-bit blocks. Each block is encrypted through the tweakable block cipher and the output values are XORed in order to get the main output of this step (denoted as auth.), as illustrated in Figure 1. If the last block is incomplete, it is padded with a single 1 bit and the rest of the block is filled with zeroes (we denote this padding as $(10^*) := (1000\ldots0)$). If the last block is a full block, it is not padded but the encryption uses a different tweak.

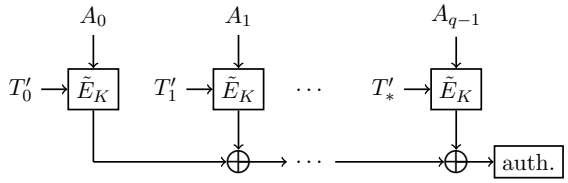

Figure 1: TEA: associated data processing.

Second, plaintext values are encrypted using the tweakable block cipher in order to produce the ciphertext values, as illustrated in Figure 2. The bitlength of the last plaintext block is encrypted to generate a mask, which it is then truncated to the partial block size and XORed with the partial plaintext block. Therefore, the ciphertext length is the same as the plaintext length.

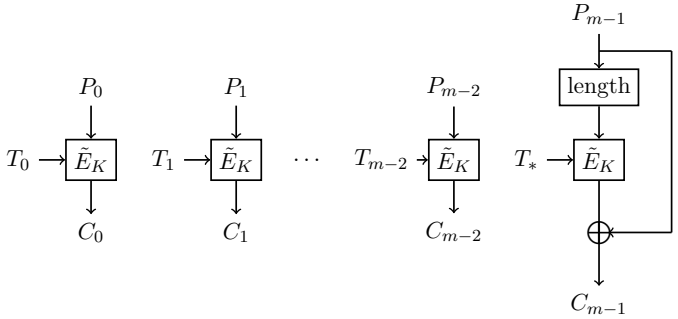

Figure 2: TAE: encryption of the plaintext blocks.

Finally, the tag is generated as represented in Figure 3. That is, the checksum (i.e. the XOR of all plaintext blocks) is first encrypted, and the output of this encryption is then XORed with the output of the associated data processing step (auth.) in order to get the tag.

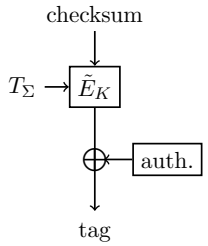

Figure 3: TAE: tag generation.

For the security of the TAE mode, all the calls to the tweakable block cipher must use distinct values of the tweak. In addition, we use some special values for domain separation and we define the tweaks depending on the context. In general, we use tweaks of the form ($N \parallel$ control bits $\parallel c$), where c is a block counter. The control bits and counter are set as follows (with 0^* a zero padding):

Plaintext encryption. c is a $127 - 8n_b$-bit block counter.

- All blocks but the last one use: $T_c = (N \parallel 0 \parallel c)$
- If the last block is a full block, it uses: $T_* = (N \parallel 11 \parallel 000 \parallel 0^*)$
- If the last block is a partial block, it uses: $T_* = (N \parallel 11 \parallel 001 \parallel 0^*)$

Associated data processing. c is a $126 - 8n_b$-bit block counter.

- All blocks but the last one use: $T'_c = (N \parallel 10 \parallel c)$
- If the last block is a full block, it uses: $T'_* = (N \parallel 11 \parallel 010 \parallel 0^*)$
- If the last block is a partial block, it uses: $T'_* = (N \parallel 11 \parallel 011 \parallel 0^*)$

Tag generation.

- If the last plaintext block is a full block, it uses: $T_\Sigma = (N \parallel 11 \parallel 100 \parallel 0^*)$
- If the last plantext block is a partial block, it uses: $T_\Sigma = (N \parallel 11 \parallel 101 \parallel 0^*)$

Our version of the TAE mode is specified in Algorithm 2.

4 Reference implementations

We use the bitslice representation of Scream and iScream in the reference implementations, and consider the 128-bit input values as 8×16 binary matrices on which we apply the step functions. We first process the S-boxes row-wise by using Algorithms 3, 4 and 5 given in Appendix. We then add the round constants before the linear layer, which is performed column-wise using the pre-computed tables given in Appendix A.3.2, A.4.2 and B.2.2. This completes the execution of one round, which is iterated and completed with a tweakey addition to conclude one step. The number of step functions depends on the security level (recommended parameters are in Section 5.3).

Algorithm 2 Tweakable Authenticated Encryption

function TAE(N, A, P)

Initialisation
$c \leftarrow 0$;
$auth. \leftarrow 0$;
$C \leftarrow \varnothing$;
$\Sigma \leftarrow 0$;

TAE: associated data
for $0 \leq i < \lfloor (|A| - 1)/128 \rfloor$ **do**
  $auth. \leftarrow auth. \oplus \tilde{E}(T'_i, A_i)$;
  $c \leftarrow c + 1$;
end for
if $|A| \nmid 128$ **then**
  $A_{i+1} \leftarrow A_{i+1} \,\|\, 10^*$;
  $auth. \leftarrow auth. \oplus \tilde{E}(T'_*, A_{i+1})$;
else
  $auth. \leftarrow auth. \oplus \tilde{E}(T'_*, A_{i+1})$;
end if

TAE: Encryption
for $0 \leq i < \lfloor (|P| - 1)/128 \rfloor$ **do**
  $C \leftarrow C \,\|\, \tilde{E}(T_i, P_i)$;
  $\Sigma \leftarrow \Sigma \oplus P_i$;
  $c \leftarrow c + 1$;
end for
if $|P| \nmid 128$ **then**
  $C \leftarrow C \,\|\, Trunc(\tilde{E}(T_*, |P_{i+1}|)) \oplus P_{i+1}$;
  $\Sigma \leftarrow \Sigma \oplus (P_{i+1} \,\|\, 0^*)$;
else
  $C \leftarrow C \,\|\, \tilde{E}(T_*, |P_{i+1}|) \oplus P_{i+1}$;
  $\Sigma \leftarrow \Sigma \oplus P_{i+1}$;
end if

TAE: Tag generation
$tag \leftarrow \tilde{E}(T_\Sigma, \Sigma) \oplus auth.$;
return C, tag
end function

5 Security analysis

5.1 The tweakable block ciphers Scream and iScream

Scream and iScream are tweakable block ciphers derived from the LS-ciphers Fantomas and Robin. The security analysis in [11] shows that those designs have good properties and follow the wide-trail strategy, which allows deriving simple bounds on the probability of differential and linear trails. However, the security notion for a tweakable block cipher is much stronger than for a standard block cipher. Indeed, the family of keyed permutations indexed by the tweak must be secure against adversaries who can query every member of the family. In particular, she can perform a differential attack between two members of the family with a different tweak value.

In the following we focus our attention on this scenario, and evaluate the best differential trails in this context. This analysis is mostly dependent on the tweakey scheduling algorithms used in Scream and iScream. In Robin and Fantomas, the lack of any key scheduling combined with key additions every round allows very simple trails with a single active S-box per round. In Scream and iScream, we avoid this weakness by using a construction similar to the LED design [12]: the tweakey is used every second round, and we can argue that related-tweak trails over σ steps must have at least $\lfloor \sigma/2 \rfloor$ active steps, where each active step has at least 8 active S-boxes.

The tweakey scheduling of Scream and iScream are designed to allow improved bounds in several scenarios, using two different approaches. On the one hand, the tweakey scheduling of Scream mixes bits corresponding to the same S-box: it allows to reuse some of the analysis for fixed tweak/key in a context with differences in the tweak/key. In particular, the non-involutive L-box has been selected to avoid simple iterative trails with a low number of active S-boxes. On the other hand, iScream uses an involutive L-box, which allows trails over two rounds with only 8 active S-boxes. In order to avoid such simple trails, the tweakey scheduling uses a bit rotation, moving any tweakey

Setting	Steps:	1	2	3	4	5	6	7	8	9	10	11	12
Single-key, fixed-tweak	Scream	8	20	30	**40**								
	iScream	8	16	24	**32**	**40**	**48**						
Single-key, chosen-tweaks	Scream	0	8	14	20	28	**35**						
	iScream	0	8	12	16	24	28	**32**	**40**				
Related-keys, chosen-tweaks	Scream	0	0	8	14	14	22	28	28	**36**			
	iScream	0	0	8	16	16	16	24	**32**	**32**	**32**	**40**	**48**

Table 2: Minimum number of active S-boxes for Scream and iScream.

difference from an S-box to the next one. This solution prevents the use of those simple involutive trails with a difference introduced and canceled by the tweak. As a result, the best trails with an active step between two inactive steps have 12 active S-boxes, rather than 8.

Our results are listed in Table 2, and the detailed analysis is available in Appendix D.

Note that versions of iScream with an odd number of steps are not really meaningful since the last key addition is *before* the last step in these cases. The bounds in grey assume that there is an extra whitening key at the end. They should only be considered as an indication of the way the number of active S-boxes grows. We do not recommend the use of the corresponding parameters.

5.2 The encryption modes SCREAM and iSCREAM

The TAE encryption mode provides a tight security reduction to the security of its underlying tweakable block ciphers that we assume to have 128-bit security (see [16] for the details).

5.3 Suggested and recommended parameters

Based on our security analysis, we suggest the parameters below, corresponding to lightweight security, single-key security and related-key security, for both SCREAM and iSCREAM. For each type of security, we provide two sets of (tight and safe) parameters. Some of these suggestions being redundant, this makes a total of four sets of parameters for SCREAM (with 6, 8, 10 and 12 steps) and four sets of parameters for iSCREAM (with 8, 10, 12 and 14 steps).

Lightweight security. 80-bit security, with a protocol avoiding related keys
Tight parameters: SCREAM with 6 steps, iSCREAM with 8 steps.
Safe parameters: SCREAM with 8 steps, iSCREAM with 10 steps.

Single key security. 128-bit security, with a protocol avoiding related keys
Tight parameters: SCREAM with 8 steps, iSCREAM with 10 steps.
Safe parameters: SCREAM with 10 steps, iSCREAM with 12 steps.

Related-key security. 128-bit security, with possible related keys
Tight parameters: SCREAM with 10 steps, iSCREAM with 12 steps.
Safe parameters: SCREAM with 12 steps, iSCREAM with 14 steps.

Our recommended parameters are the safe ones. Nevertheless, the tight parameters define interesting targets for further cryptanalysis efforts and could lead to additional performance gains.

6 Performance evaluation

In this section, we provide preliminary implementation results for the previous proposals. Our main goal is to confirm their excellent behavior on a wide range of platforms. For this purpose, we selected three implementation contexts that we believe representative of most application scenarios, namely high-end CPUs, small (here, 8-bit) microcontrollers and Application Specific Integrated Circuits (ASICs). We limit ourselves to performances results for the tweakable block ciphers (Scream, iScream), since the overhead of the mode is rather small. We compare the results obtained with the state-of-the-art performances of the AES on the same platforms. For readability, we only provide figures corresponding to single-key security with safe parameters. However, we note that changing the number of steps essentially implies a proportional change in the cycle counts.

6.1 High-end CPUs

An important target for modern cryptography will be high-end CPUs as found in desktop computers and servers, in particular processors from AMD and Intel. We also consider smaller processors found in netbooks, smartphones and tablets, such as the ARM Cortex-A processor and Intel Atom series. They are relatively powerful, and feature a cache hierarchy, various prediction units (branching, cache access), and a high level of parallelism with vector units and super-scalar abilities.

For our implementation of Scream and iScream, we leverage the vector units of those processors. More precisely, we wrote an implementation using the SSSE3 instruction set for x86 devices (supported in most AMD and Intel desktop and server processors since 2006, as well as by the Atom CPUs), and an implementation using the NEON instruction set for ARM processors (supported in most Cortex-A CPUs). Using those SIMD instructions, we can compute bitwise Boolean oprations on 128-bit registers. This allows us to evaluate 128 S-boxes in parallel efficiently. For the L-box layer, we use the vector shuffling instructions (`pshufb` in SSSE3, `vtbl` in NEON) as a 4-bit to 8-bit look-up table. The 16-bit L-box is decomposed as eight 4-bit to 8-bit look-up tables and 6 XORs[1]. This way of evaluating the tables avoids any cache timing attack. We compute 16 instances of Scream or iScream in parallel (with 16 different plaintexts and tweaks), using 16 SIMD registers, every register containing one byte from each copy of the cipher (8 registers for the high-order bytes, and 8 other registers for the low-order bytes). Using SSSE3 instructions, the S-box layer requires 96 instruction to compute two sets of 128 S-boxes in parallel, and the L-box layer requires 280 instructions to evaluate 128 L-boxes. As a point of comparison, the bitsliced AES implementation of Käsper and Schwabe [13] would take respectively 326 and 102 cycles for the same number of S-boxes and linear layers. On the one hand our S-box is much easier to implement in a bitslice way than the AES S-box, since it was one of our design goals. On the other hand, our linear-layer is optimized for a table-based implementation, and more complex than that of the AES. It can still be implemented rather efficiently, but it becomes the dominant factor in this implementation.

As illustrated in Table 3, this strategy leads to implementations that are comparable in speed with the fastest AES implementations. More precisely, Scream-10 and iScream-12 are approximately 35% and 60% slower than the AES implementation of [13], respectively. Note that these overheads are mainly due to the additional rounds needed to securely process the tweak (which also explains the difference with the performances of Robin and Fantomas that are even closer to the AES ones). On CPUs with hardware AES support (such as the AES-NI instructions), AES implementations can be significantly faster, but Scream and iScream still achieve a very good throughput. Moreover, the

[1] This can be further reduced to seven table look-ups for iScream, thanks to the L-box structure.

latest Intel CPUs support 256-bit wide SIMD operations using AVX2 instructions. This should lead to implementations of Scream and iScream about twice as fast as on the *Ivy Bridge* architecture.

Table 3: Implementation results on high-end CPUs. Speed in cycles/byte for long messages.

	Scream-10	iScream-12	AES	
			w/o AES-NI	w/AES-NI
Cortex A15 *Exynos 5*	21.8	26.2	17.8	N/A
Atom *Cedarview*	55	65	17	N/A
Core i7 *Nehalem*	9.4	11.2	6.9	N/A
Core i7 *Ivy Bridge*	7.1	9.1	5.4	1.3

6.2 8-bit microcontrollers

We exploit the methodology and metrics introduced in [7] for comparing the performances of block ciphers in Atmel AVR devices, and refer to this previous work for the details. Summarizing, our implementations store the plaintext, key and tweak in RAM and derive the tweakeys on-the-fly. Performance evaluations are given in Table 4, and compared with two implementations of the AES: the one in [7] (that follows the same guidelines as ours) and the AES furious in [23] (that does not compute the key scheduling on-the-fly), leading to three main conclusions.

Table 4: Implementation results in an Atmel AVR microcontroller.

	ROM words			RAM words	Cycle count	
	code	tables	total		encryption	decryption
AES [7]	1147	512	1659	33	4557	7015
AES furious [23]	800	768	1568	192	3629	4462
Scream-10 (ED)	1173	2048	3221	80	7646	7672
iScream-12 (ED)	951	1024	1975	64	8724	8724
Scream-10 (E)	699	1024	1723	80	7646	-
iScream-12 (E)	571	1024	1595	64	8724	-
Scream-10 (D)	727	1024	1751	80	-	7672
iScream-12 (D)	569	1024	1593	64	-	8724

First, the encryption cycle count of a round of Scream and iScream is inferior to the encryption cycle count of an AES round from [7] (which is observed by multiplying the Scream and iScream cycle counts by 10/20 and 10/24, respectively). It is also very similar to the AES furious encryption cycle count (that excludes the on-the-fly computation of the round keys). Second, we see that the on-the-fly key derivation implies significant overheads in the decryption cycle count of the AES in [7], while the decryption cycle counts of Scream and iScream are similar to the encryption ones. Finally, the ED results illustrates the code size gains obtained thanks to the involutive components in iScream (for which only four 256-byte tables are needed in Appendix B.2.2 vs. eight for Scream in Appendices A.3.2 and A.4.2). Note that since the L-boxes are linear, the (ROM) memory requirements of these implementations could be further reduced at the cost of an increased cycle count, e.g. by computing these operations based on eight 16-byte (or sixteen 16-nibble) tables.

6.3 Application Specific Integrated Circuits

We exploit the methodology and metrics introduced in [14] for comparing the performances of block ciphers in a 65-nanometer low-power CMOS technology, and refer to this previous work for the details. Summarizing, our implementations have a 128-bit datapath, store the plaintext, key and tweak in registers, and derive the keys on-the-fly. We considered architectures implementing both a single round per cycle (denoted as 1R/cycle) and two rounds per cycle (denoted as 2R/cycle). Performance evaluations are given in Table 5 and are compared to the AES implementation in [14]. Current results were only obtained for encryption, but estimates are given for decryption and the encryption/decryption architectures (in italic)[2], leading to the following observations.

Table 5: ASIC Implementation results (65-nanometer low-power CMOS).

	Mode E,D,ED	Area [μm^2]	f_{max} [MHz]	Latency [cycles]	Throughput [Mbps]
AES	E	17921	444	12	4740
(1R/cycle)	D	20292	377	22	2195
	ED	24272	363	≈17	≈2997
Scream-10	E	12951	751	21	4577
(1R/cycle)	*D*	*12951*	*751*	21	*4577*
	ED	*17292*	*751*	21	*4577*
Scream-10	E	17292	446	11	5190
(2R/cycle)	*D*	*17292*	*446*	11	*5190*
	ED	*25974*	*446*	11	*5190*
iScream-12	E	13375	740	25	3789
(1R/cycle)	*D*	*13375*	*740*	25	*3789*
	ED	*13375*	*740*	25	*3789*
iScream-12	E	17024	448	13	4411
(2R/cycle)	*D*	*17024*	*448*	13	*4411*
	ED	*17024*	*448*	13	*4411*

First and most importantly, we see that the combinatorial cost of a single round of Scream or iScream roughly corresponds to half the combinatorial cost of an AES round. This is observed both in the area and maximum frequency of our 2R/cycle architectures, and shows that hardware implementations can further benefit of the compact gate-level description of our S-boxes and L-boxes. As a result, these architectures provide similar throughputs as the AES 1R/cycle one, at a similar cost. Second, and as for the previous AVR implementations, the on-the-fly key derivation in decryption *should* allow avoiding the loss of a factor two in throughput as it is the case for the AES. Finally, and as also observed for the previous AVR implementations, the combination of encryption and decryption *should* come approximately for free for (involutional) iScream.

[2] More precisely, the combinatorial cost of a round (i.e. excluding the registers) is estimated by substracting the cost of the 1R/cycle architectures from the one of the 2R/cycle architectures. We assume the cost of the decryption to be similar to the one of encryption, and their combination to require essentially twice as much combinatorial logic for Scream, while being essentially unchanged for iScream. Cycles count for encryption and decryption are the same.

6.4 Further work

The preliminary results in this section are encouraging in the sense that they put forward performances for Scream and iScream that are comparable to the AES ones, *despite the additional processing of a tweak*. As already mentioned, these performances would be improved with the tight(er) security parameters. Quite naturally, various additional implementation efforts would be worth further investigations. This first includes the optimization of our current implementations and their evaluation on other platforms and architectures. For example, 8051 microcontrollers have bit-addressable memories that would be useful for the table lookup implementation of our S-boxes (in order to efficiently access the columns of the state). The regularity of tweakable LS-designs also make them suitable candidates for compact hardware architectures (i.e. with smaller datapath than the 128-bit ones considered in the previous section). Evaluating the performances of Scream and iScream in recent FPGAs is yet another interesting direction. Overall, we believe that the efficient (both gate-level and table-based) representations of the linear and non-linear operations in these tweakable block ciphers make them promising and versatile candidates for both lightweight and high performance computing. Besides, the extension of these preliminary results towards the authenticated encryption modes is important to analyze as well. Yet, the fully parallelisable nature of TAE should imply very limited overheads compared to their underlying ciphers, as confirmed by first estimations with our reference (non-optimized) implementations. Eventually, assessing the performances and side-channel resistance of masked implementations of SCREAM and iSCREAM is expected to further amplify these positive observations (as for Robin and Fantomas).

7 Intellectual property

To the extent permitted under law, the designers have released all copyright neighboring rights related to the tweakable block ciphers Scream and iScream, and the authenticated encryption algorithms SCREAM and iSCREAM to the public domain. The submitters do not hold any patent related to the designs, nor will apply for any. To the best of their knowledge, the TAE mode of operation is free of patents. Yet, as acknowledged by its authors, it can be viewed as a paraphrase or re-statement of the OCB encryption mode proposed by Rogaway et al. [30], which is patented (US patents 7,046,802, 7,200,227, 7,949,129, and 8,321,675). We note that versions of OCB using a tweakable block cipher, and related patents, are more recent than the publication of TAE by Liskov *et al.* [16]. Referring to Phil Rogaway's webpage [29], "*there are further patents in the authenticated encryption space. I would single out those of Gligor and Donescu (VDG) and Jutla (IBM): 6,963,976, 6,973,187, 7,093,126, and 8,107,620. Do the claims of these patents read against OCB? It is difficult to answer such a question. In fact, I suspect that nobody can give an answer. It seems extremely subjective.*" If any of this information changes, the submitters will promptly (and within at most one month) announce these changes on the `crypto-competitions` mailing list.

8 Consent

The submitters hereby consent to all decisions of the CAESAR selection committee regarding the selection or non-selection of this submission as a second-round candidate, a third-round candidate, a finalist, a member of the final portfolio, or any other designation provided by the committee. The submitters understand that the committee will not comment on the algorithms, except that for each selected algorithm the committee will simply cite the previously published analyses that led to the selection of the algorithm. The submitters understand that the selection of some algorithms

is not a negative comment regarding other algorithms, and that an excellent algorithm might fail to be selected simply because not enough analysis was available at the time of the committee decision. The submitters acknowledge that the committee decisions reflect the collective expert judgments of the committee members and are not subject to appeal. The submitters understand that if they disagree with published analyses then they are expected to promptly and publicly respond to those analyses, not to wait for subsequent committee decisions. The submitters understand that this statement is required as a condition of consideration of this submission by the CAESAR selection committee.

References

[1] E. Andreeva, A. Bogdanov, A. Luykx, B. Mennink, E. Tischhauser, and K. Yasuda, *Parallelizable and authenticated online ciphers*, in ASIACRYPT (1), K. Sako and P. Sarkar, eds., vol. 8269 of Lecture Notes in Computer Science, Springer, 2013, pp. 424–443.

[2] E. Andreeva, A. Luykx, B. Mennink, and K. Yasuda, *COBRA: A parallelizable authenticated online cipher without block cipher inverse*. To appear in the proceedings of FSE 2014.

[3] A. Bogdanov, L. R. Knudsen, G. Leander, C. Paar, A. Poschmann, M. J. B. Robshaw, Y. Seurin, and C. Vikkelsoe, *PRESENT: An ultra-lightweight block cipher*, in CHES, P. Paillier and I. Verbauwhede, eds., vol. 4727 of Lecture Notes in Computer Science, Springer, 2007, pp. 450–466.

[4] S. Chari, C. S. Jutla, J. R. Rao, and P. Rohatgi, *Towards sound approaches to counteract power-analysis attacks*, in Wiener [32], pp. 398–412.

[5] C. Clavier and K. Gaj, eds., *Cryptographic Hardware and Embedded Systems - CHES 2009, 11th International Workshop, Lausanne, Switzerland, September 6-9, 2009, Proceedings*, vol. 5747 of Lecture Notes in Computer Science, Springer, 2009.

[6] J.-S. Coron, C. Giraud, E. Prouff, S. Renner, M. Rivain, and P. K. Vadnala, *Conversion of security proofs from one leakage model to another: A new issue*, in COSADE, W. Schindler and S. A. Huss, eds., vol. 7275 of Lecture Notes in Computer Science, Springer, 2012, pp. 69–81.

[7] T. Eisenbarth, Z. Gong, T. Güneysu, S. Heyse, S. Indesteege, S. Kerckhof, F. Koeune, T. Nad, T. Plos, F. Regazzoni, F.-X. Standaert, and L. van Oldeneel tot Oldenzeel, *Compact implementation and performance evaluation of block ciphers in attiny devices*, in AFRICACRYPT, A. Mitrokotsa and S. Vaudenay, eds., vol. 7374 of Lecture Notes in Computer Science, Springer, 2012, pp. 172–187.

[8] K. Gandolfi, C. Mourtel, and F. Olivier, *Electromagnetic analysis: Concrete results*, in CHES, Çetin Kaya Koç, D. Naccache, and C. Paar, eds., vol. 2162 of Lecture Notes in Computer Science, Springer, 2001, pp. 251–261.

[9] L. Genelle, E. Prouff, and M. Quisquater, *Thwarting higher-order side channel analysis with additive and multiplicative maskings*, in Preneel and Takagi [24], pp. 240–255.

[10] B. Gérard, V. Grosso, M. Naya-Plasencia, and F.-X. Standaert, *Block ciphers that are easier to mask: How far can we go?*, in CHES, G. Bertoni and J.-S. Coron, eds., vol. 8086 of Lecture Notes in Computer Science, Springer, 2013, pp. 383–399.

[11] V. Grosso, G. Leurent, F.-X. Standaert, and K. Varici, *LS-designs: Bitslice encryption for efficient masked software implementations*. To appear in the proceedings of FSE 2014.

[12] J. Guo, T. Peyrin, A. Poschmann, and M. J. B. Robshaw, *The LED block cipher*, in Preneel and Takagi [24], pp. 326–341.

[13] E. Käsper and P. Schwabe, *Faster and Timing-Attack Resistant AES-GCM*, in Clavier and Gaj [5], pp. 1–17.

[14] S. Kerckhof, F. Durvaux, C. Hocquet, D. Bol, and F.-X. Standaert, *Towards green cryptography: A comparison of lightweight ciphers from the energy viewpoint*, in CHES, E. Prouff and P. Schaumont, eds., vol. 7428 of Lecture Notes in Computer Science, Springer, 2012, pp. 390–407.

[15] P. C. Kocher, J. Jaffe, and B. Jun, *Differential power analysis*, in Wiener [32], pp. 388–397.

[16] M. Liskov, R. L. Rivest, and D. Wagner, *Tweakable block ciphers*, J. Cryptology, 24 (2011), pp. 588–613.

[17] S. Mangard, E. Oswald, and T. Popp, *Power analysis attacks - revealing the secrets of smart cards*, Springer, 2007.

[18] S. Mangard, T. Popp, and B. M. Gammel, *Side-channel leakage of masked CMOS gates*, in CT-RSA, A. Menezes, ed., vol. 3376 of Lecture Notes in Computer Science, Springer, 2005, pp. 351–365.

[19] K. Minematsu, *Parallelizable rate-1 authenticated encryption from pseudorandom functions*. To appear in the proceedings of EUROCRYPT 2014.

[20] A. Moradi, A. Poschmann, S. Ling, C. Paar, and H. Wang, *Pushing the limits: A very compact and a threshold implementation of AES*, in EUROCRYPT, K. G. Paterson, ed., vol. 6632 of Lecture Notes in Computer Science, Springer, 2011, pp. 69–88.

[21] S. Nikova, V. Rijmen, and M. Schläffer, *Secure hardware implementation of nonlinear functions in the presence of glitches*, J. Cryptology, 24 (2011), pp. 292–321.

[22] G. Piret, T. Roche, and C. Carlet, *PICARO - a block cipher allowing efficient higher-order side-channel resistance*, in ACNS, F. Bao, P. Samarati, and J. Zhou, eds., vol. 7341 of Lecture Notes in Computer Science, Springer, 2012, pp. 311–328.

[23] B. Pottering, *http://point-at-infinity.org/avraes/*.

[24] B. Preneel and T. Takagi, eds., *Cryptographic Hardware and Embedded Systems - CHES 2011 - 13th International Workshop, Nara, Japan, September 28 - October 1, 2011. Proceedings*, vol. 6917 of Lecture Notes in Computer Science, Springer, 2011.

[25] E. Prouff and M. Rivain, *Masking against side-channel attacks: A formal security proof*, in EUROCRYPT, T. Johansson and P. Q. Nguyen, eds., vol. 7881 of Lecture Notes in Computer Science, Springer, 2013, pp. 142–159.

[26] E. Prouff and T. Roche, *Higher-order glitches free implementation of the AES using secure multi-party computation protocols*, in Preneel and Takagi [24], pp. 63–78.

[27] M. Renauld, F.-X. Standaert, and N. Veyrat-Charvillon, *Algebraic side-channel attacks on the AES: Why time also matters in DPA*, in Clavier and Gaj [5], pp. 97–111.

[28] M. Rivain and E. Prouff, *Provably secure higher-order masking of AES*, in CHES, S. Mangard and F.-X. Standaert, eds., vol. 6225 of Lecture Notes in Computer Science, Springer, 2010, pp. 413–427.

[29] P. Rogaway, *http://www.cs.ucdavis.edu/ rogaway/ocb/*.

[30] P. Rogaway, M. Bellare, and J. Black, *OCB: A block-cipher mode of operation for efficient authenticated encryption*, ACM Trans. Inf. Syst. Secur., 6 (2003), pp. 365–403.

[31] F.-X. Standaert, N. Veyrat-Charvillon, E. Oswald, B. Gierlichs, M. Medwed, M. Kasper, and S. Mangard, *The world is not enough: Another look on second-order DPA*, in ASIACRYPT, M. Abe, ed., vol. 6477 of Lecture Notes in Computer Science, Springer, 2010, pp. 112–129.

[32] M. J. Wiener, ed., *Advances in Cryptology - CRYPTO '99, 19th Annual International Cryptology Conference, Santa Barbara, California, USA, August 15-19, 1999, Proceedings*, vol. 1666 of Lecture Notes in Computer Science, Springer, 1999.

A Scream parameters

A.1 Scream S-box

A.1.1 Table representation

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
00	1E	75	5F	E1	99	FC	89	2F	86	EE	F1	7B	23	52	10	94
10	0C	B7	4D	67	D8	42	C8	D6	C4	6B	AA	BA	3D	A5	00	33
20	53	2D	0B	B8	DA	A8	C5	6C	CA	B6	A4	22	60	07	5D	D7
30	4F	F4	15	32	81	1B	9C	8E	91	3F	E6	F9	70	E9	43	7E
40	8D	F3	CC	65	08	7A	18	AB	16	6A	77	FD	A7	C0	82	04
50	9F	31	DE	E3	49	D0	59	46	54	EF	2E	3C	BB	21	92	B5
60	55	3E	0F	A9	DC	B9	C1	7F	CE	A6	B4	30	72	03	5B	D1
70	4B	E4	13	20	85	1D	9A	8A	97	2C	F6	E8	62	F8	47	6D
80	29	41	68	D5	AC	CB	BE	1A	B0	DB	C7	4E	17	64	26	A0
90	39	83	78	51	ED	76	FF	E2	F2	5C	9D	8F	0A	93	34	05
A0	25	58	7C	CD	AF	DF	B3	19	BD	C2	D2	56	14	71	2A	A3
B0	3A	80	61	44	F5	6E	EB	FB	E7	48	90	8C	06	9E	37	09
C0	98	E5	D9	73	1F	6F	0D	BC	02	7D	63	EA	B1	D4	96	12
D0	88	27	C9	F7	5E	C6	4C	50	40	FA	3B	2B	AE	35	84	A1
E0	01	69	5A	FE	8B	EC	95	28	9B	F0	E0	66	24	57	0E	87
F0	1C	B2	45	74	D3	4A	CF	DD	C3	79	A2	BF	36	AD	11	38

Table 6: Scream S-box, table representation.

A.1.2 Algebraic Normal Form

$y_0 = x_0 + x_1 + x_0x_1 + x_2 + x_1x_2 + x_0x_3 + x_0x_1x_3 + x_0x_2x_3 + x_1x_2x_3 + x_0x_1x_2x_3 + x_2x_4 + x_0x_2x_4 + x_0x_1x_2x_4 + x_0x_3x_4 + x_1x_3x_4 + x_0x_1x_3x_4 + x_2x_3x_4 + x_0x_2x_3x_4 + x_1x_2x_3x_4 + x_5 + x_0x_5 + x_1x_5 + x_3x_5 + x_0x_3x_5 + x_2x_3x_5 + x_1x_2x_3x_5 + x_0x_4x_5 + x_0x_1x_4x_5 + x_0x_1x_2x_4x_5 + x_3x_4x_5 + x_0x_1x_3x_4x_5 + x_2x_3x_4x_5 + x_6 + x_0x_6 + x_3x_6 + x_0x_3x_6 + x_5x_6 + x_3x_5x_6 + x_0x_4x_5x_6 + x_7 + x_0x_7 + x_0x_2x_7 + x_3x_7 + x_0x_2x_4x_7 + x_1x_3x_4x_7 + x_0x_1x_3x_4x_7 + x_2x_3x_4x_7 + x_5x_7 + x_1x_5x_7 + x_0x_1x_5x_7 + x_2x_5x_7 + x_4x_5x_7$

$y_1 = 1 + x_0 + x_2 + x_0x_2 + x_0x_1x_2 + x_0x_3 + x_1x_3 + x_0x_1x_3 + x_2x_3 + x_0x_2x_3 + x_4 + x_2x_4 + x_0x_2x_4 + x_0x_1x_2x_4 + x_0x_3x_4 + x_2x_3x_4 + x_1x_2x_3x_4 + x_2x_5 + x_0x_2x_5 + x_1x_2x_5 + x_4x_5 + x_1x_4x_5 + x_0x_2x_4x_5 + x_3x_4x_5 + x_1x_3x_4x_5 + x_0x_1x_3x_4x_5 + x_2x_3x_4x_5 + x_6 + x_0x_1x_6 + x_2x_6 + x_0x_2x_6 + x_3x_6 + x_1x_3x_6 + x_0x_1x_3x_6 + x_2x_3x_6 + x_1x_3x_4x_6 + x_0x_1x_3x_4x_6 + x_2x_3x_4x_6 + x_1x_5x_6 + x_0x_1x_5x_6 + x_2x_5x_6 + x_4x_5x_6 + x_7 + x_0x_7 + x_2x_7 + x_1x_2x_7 + x_4x_7 + x_0x_4x_7 + x_0x_1x_4x_7 + x_2x_4x_7 + x_3x_4x_7 + x_6x_7$

$y_2 = 1 + x_0x_1 + x_2 + x_0x_2 + x_0x_1x_2 + x_1x_3 + x_0x_1x_3 + x_0x_2x_3 + x_1x_2x_3 + x_0x_1x_4 + x_0x_2x_4 + x_0x_3x_4 + x_2x_3x_4 + x_1x_2x_3x_4 + x_5 + x_0x_5 + x_2x_5 + x_1x_2x_5 + x_4x_5 + x_0x_4x_5 + x_0x_1x_4x_5 + x_2x_4x_5 + x_3x_4x_5 + x_0x_6 + x_1x_3x_6 + x_0x_1x_3x_6 + x_2x_3x_6 + x_0x_2x_4x_6 + x_5x_6 + x_7$

$y_3 = 1 + x_0 + x_0x_2 + x_3 + x_0x_2x_4 + x_1x_3x_4 + x_0x_1x_3x_4 + x_2x_3x_4 + x_5 + x_1x_5 + x_0x_1x_5 + x_2x_5 + x_4x_5$

$y_4 = 1 + x_0x_1 + x_1x_2 + x_0x_1x_2 + x_3 + x_1x_3 + x_0x_1x_3 + x_0x_2x_3 + x_1x_2x_3 + x_4 + x_0x_4 + x_2x_4 + x_3x_4 + x_0x_3x_4 + x_1x_3x_4 + x_0x_1x_3x_4 + x_1x_2x_3x_4 + x_0x_5 + x_1x_5 + x_0x_1x_5 + x_1x_2x_5 + x_0x_4x_5 + x_0x_1x_4x_5 + x_2x_4x_5 + x_3x_4x_5 + x_6 + x_0x_6 + x_1x_3x_6 + x_0x_1x_3x_6 + x_2x_3x_6 + x_0x_2x_4x_6 + x_5x_6 + x_7$

$y_5 = x_0 + x_1x_3 + x_0x_1x_3 + x_2x_3 + x_0x_2x_4 + x_7 + x_6x_7$

$y_6 = x_0 + x_1 + x_0x_1 + x_1x_2 + x_0x_4 + x_0x_1x_4 + x_2x_4 + x_3x_4 + x_5 + x_5x_7$

$y_7 = x_0x_1 + x_2 + x_3 + x_0x_4 + x_6 + x_5x_6$

A.1.3 Bitslice implementation

Algorithm 3 Scream S-box, bitslice implementation

Require: 8 16-bit words $(W_0, \ldots, W_7)$

First 5-bit S-box
$W_2 = W_2 \oplus (W_0 \wedge W_1)$
$W_1 = W_1 \oplus W_2$
$W_3 = W_3 \oplus (W_0 \wedge W_4)$
$W_2 = W_2 \oplus W_3$
$W_0 = W_0 \oplus (W_3 \wedge W_1)$
$W_4 = W_4 \oplus W_1$
$W_1 = W_1 \oplus (W_2 \wedge W_4)$
$W_1 = W_1 \oplus W_0$
Extend-Xor
$W_0 = W_0 \oplus W_5$
$W_1 = W_1 \oplus W_6$
$W_2 = W_2 \oplus W_7$
Constant
$W_3 = \neg(W_3)$
$W_4 = \neg(W_4)$
First 3-bit S-box
$(t_5, t_6, t_7) = (W_5, W_6, W_7)$
$W_5 = W_5 \oplus \neg(t_6) \wedge t_7$
$W_6 = W_6 \oplus \neg(t_7) \wedge t_5$
$W_7 = W_7 \oplus \neg(t_5) \wedge t_6$
Truncate-Xor
$W_5 = W_0 \oplus W_5$
$W_6 = W_1 \oplus W_6$
$W_7 = W_2 \oplus W_7$
Second 5-bit S-box
$W_2 = W_2 \oplus (W_0 \wedge W_1)$
$W_1 = W_1 \oplus W_2$
$W_3 = W_3 \oplus (W_0 \wedge W_4)$
$W_2 = W_2 \oplus W_3$
$W_0 = W_0 \oplus (W_3 \wedge W_1)$
$W_4 = W_4 \oplus W_1$
$W_1 = W_1 \oplus (W_2 \wedge W_4)$
$W_1 = W_1 \oplus W_0$

A.2 Inverse Scream S-box

A.2.1 Table representation

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
00	1E	E0	C8	6D	4F	9F	BC	2D	44	BF	9C	22	10	C6	EE	62
10	0E	FE	CF	72	AC	32	48	8C	46	A7	87	35	F0	75	00	C4
20	73	5D	2B	0C	EC	A0	8E	D1	E7	80	AE	DB	79	21	5A	07
30	6B	51	33	1F	9E	DD	FC	BE	FF	90	B0	DA	5B	1C	61	39
40	D8	81	15	3E	B3	F2	57	7E	B9	54	F5	70	D6	12	8B	30
50	D7	93	0D	20	58	60	AB	ED	A1	56	E2	6E	99	2E	D4	02
60	2C	B2	7C	CA	8D	43	EB	13	82	E1	49	19	27	7F	B5	C5
70	3C	AD	6C	C3	F3	01	95	4A	92	F9	45	0B	A2	C9	3F	67
80	B1	34	4E	91	DE	74	08	EF	D0	06	77	E4	BB	40	37	9B
90	BA	38	5E	9D	0F	E6	CE	78	C0	04	76	E8	36	9A	BD	50
A0	8F	DF	FA	AF	2A	1D	69	4C	25	63	1A	47	84	FD	DC	A4
B0	88	CC	F1	A6	6A	5F	29	11	23	65	1B	5C	C7	A8	86	FB
C0	4D	66	A9	F8	18	26	D5	8A	16	D2	28	85	42	A3	68	F6
D0	55	6F	AA	F4	CD	83	17	2F	14	C2	24	89	64	F7	52	A5
E0	EA	03	97	53	71	C1	3A	B8	7B	3D	CB	B6	E5	94	09	59
F0	E9	0A	98	41	31	B4	7A	D3	7D	3B	D9	B7	05	4B	E3	96

Table 7: Inverse Scream S-box, table representation.

A.2.2 Algebraic Normal Form

$$y_0 = x_0x_1 + x_2 + x_1x_2 + x_0x_3 + x_2x_3 + x_0x_2x_3 + x_1x_2x_3 + x_0x_1x_2x_3 + x_1x_4 + x_2x_4 + x_0x_1x_2x_4 + x_2x_3x_4 + x_0x_2x_3x_4 + x_1x_2x_3x_4 + x_5 + x_1x_2x_5 + x_1x_3x_5 + x_0x_1x_3x_5 + x_1x_2x_3x_5 + x_1x_4x_5 + x_0x_1x_4x_5 + x_2x_4x_5 + x_0x_2x_4x_5 + x_2x_3x_4x_5 + x_0x_2x_3x_4x_5 + x_0x_6 + x_1x_6 + x_0x_1x_6 + x_3x_6 + x_0x_3x_6 + x_1x_3x_6 + x_2x_3x_6 + x_1x_2x_3x_6 + x_4x_6 + x_0x_4x_6 + x_0x_1x_4x_6 + x_2x_4x_6 + x_3x_4x_6 + x_0x_3x_4x_6 + x_5x_6 + x_0x_5x_6 + x_1x_5x_6 + x_3x_5x_6 + x_0x_3x_5x_6 + x_1x_3x_5x_6 + x_4x_5x_6 + x_2x_4x_5x_6 + x_3x_4x_5x_6 + x_2x_3x_4x_5x_6 + x_7 + x_0x_7 + x_1x_7 + x_0x_1x_7 + x_0x_2x_7 + x_0x_1x_2x_7 + x_3x_7 + x_0x_1x_3x_7 + x_2x_3x_7 + x_0x_2x_3x_7 + x_1x_2x_3x_7 + x_4x_7 + x_0x_4x_7 + x_0x_1x_4x_7 + x_2x_4x_7 + x_3x_4x_7 + x_0x_3x_4x_7 + x_1x_2x_3x_4x_7 + x_5x_7 + x_0x_5x_7 + x_0x_1x_5x_7 + x_3x_5x_7 + x_0x_3x_5x_7 + x_1x_3x_5x_7 + x_0x_4x_5x_7 + x_0x_2x_4x_5x_7 + x_0x_3x_4x_5x_7 + x_1x_3x_4x_5x_7 + x_0x_6x_7 + x_0x_1x_6x_7 + x_3x_6x_7 + x_0x_3x_6x_7 + x_1x_3x_6x_7 + x_0x_4x_6x_7 + x_0x_2x_4x_6x_7 + x_0x_3x_4x_6x_7 + x_1x_3x_4x_6x_7$$

$$y_1 = 1 + x_0 + x_1 + x_0x_1 + x_0x_2 + x_0x_1x_2 + x_3 + x_1x_3 + x_0x_1x_3 + x_0x_2x_3 + x_1x_2x_3 + x_0x_4 + x_1x_4 + x_0x_1x_4 + x_2x_4 + x_3x_4 + x_1x_3x_4 + x_1x_2x_3x_4 + x_1x_5 + x_0x_1x_5 + x_2x_5 + x_1x_2x_5 + x_3x_5 + x_1x_3x_5 + x_0x_2x_3x_5 + x_1x_2x_3x_5 + x_0x_4x_5 + x_1x_4x_5 + x_0x_2x_4x_5 + x_3x_4x_5 + x_6 + x_0x_6 + x_1x_6 + x_2x_6 + x_0x_2x_6 + x_3x_6 + x_1x_3x_6 + x_0x_2x_3x_6 + x_1x_2x_3x_6 + x_4x_6 + x_0x_4x_6 + x_2x_4x_6 + x_0x_3x_4x_6 + x_1x_3x_4x_6 + x_2x_3x_4x_6 + x_0x_5x_6 + x_1x_5x_6 + x_4x_5x_6 + x_2x_4x_5x_6 + x_7 + x_0x_7 + x_2x_7 + x_3x_7 + x_0x_3x_7 + x_1x_3x_7 + x_4x_7 + x_2x_3x_4x_7 + x_5x_7 + x_6x_7$$

$$y_2 = 1 + x_0 + x_1 + x_0x_2 + x_1x_2 + x_0x_3 + x_1x_3 + x_0x_1x_3 + x_2x_3 + x_0x_1x_2x_3 + x_0x_4 + x_1x_4 + x_0x_1x_4 + x_0x_1x_2x_4 + x_0x_3x_4 + x_1x_3x_4 + x_1x_2x_3x_4 + x_5 + x_1x_5 + x_2x_5 + x_0x_2x_5 + x_1x_2x_5 + x_3x_5 + x_0x_3x_5 + x_1x_3x_5 + x_0x_1x_3x_5 + x_2x_3x_5 + x_0x_2x_3x_5 + x_1x_4x_5 + x_0x_2x_3x_4x_5 + x_6 + x_0x_6 + x_0x_2x_6 + x_1x_2x_6 + x_1x_3x_6 + x_0x_1x_3x_6 + x_4x_6 + x_0x_1x_4x_6 + x_2x_4x_6 + x_0x_2x_4x_6 + x_3x_4x_6 + x_1x_3x_4x_6 + x_0x_2x_3x_4x_6 + x_0x_5x_6 + x_2x_5x_6 + x_0x_2x_5x_6 + x_0x_1x_2x_5x_6 + x_0x_3x_5x_6 + x_1x_3x_5x_6 + x_0x_1x_3x_5x_6 + x_2x_3x_5x_6 + x_0x_2x_3x_5x_6 + x_1x_2x_3x_5x_6 + x_4x_5x_6 + x_0x_1x_4x_5x_6 + x_0x_2x_4x_5x_6 + x_3x_4x_5x_6 + x_1x_3x_4x_5x_6 + x_1x_2x_3x_4x_5x_6 + x_7 + x_2x_7 + x_1x_2x_7 + x_0x_1x_2x_7 + x_0x_3x_7 + x_1x_3x_7 + x_1x_4x_7 + x_0x_1x_4x_7 + x_0x_2x_4x_7 + x_1x_2x_4x_7 + x_0x_1x_2x_4x_7 + x_1x_3x_4x_7 + x_0x_1x_3x_4x_7 + x_2x_3x_4x_7 + x_1x_2x_3x_4x_7 + x_0x_5x_7 + x_1x_5x_7 + x_0x_1x_5x_7 + x_2x_5x_7 + x_0x_2x_5x_7 + x_3x_5x_7 + x_1x_3x_5x_7 + x_0x_1x_3x_5x_7 + x_2x_3x_5x_7 + x_0x_2x_3x_5x_7 + x_4x_5x_7 + x_0x_1x_4x_5x_7 + x_2x_4x_5x_7 + x_2x_3x_4x_5x_7 + x_0x_2x_3x_4x_5x_7 + x_0x_2x_6x_7 + x_1x_2x_6x_7 + x_0x_1x_2x_6x_7 + x_0x_3x_6x_7 + x_1x_3x_6x_7 + x_2x_3x_6x_7 + x_1x_2x_3x_6x_7 + x_4x_6x_7 + x_0x_4x_6x_7 + x_3x_4x_6x_7 + x_0x_3x_4x_6x_7 + x_2x_3x_4x_6x_7 + x_0x_2x_3x_4x_6x_7 + x_1x_2x_3x_4x_6x_7$$

$$y_3 = 1 + x_0 + x_0x_1 + x_0x_2 + x_0x_1x_2 + x_3 + x_1x_3 + x_0x_1x_3 + x_0x_4 + x_0x_1x_2x_4 + x_1x_3x_4 + x_0x_2x_3x_4 + x_5 + x_1x_5 + x_2x_5 + x_0x_2x_5 + x_1x_2x_5 + x_3x_5 + x_0x_3x_5 + x_1x_3x_5 + x_0x_2x_3x_5 + x_0x_1x_2x_3x_5 + x_4x_5 + x_0x_4x_5 + x_0x_1x_4x_5 + x_2x_4x_5 + x_0x_2x_4x_5 + x_0x_3x_4x_5 + x_1x_3x_4x_5 + x_0x_1x_3x_4x_5 + x_1x_2x_3x_4x_5 + x_1x_6 + x_0x_1x_6 + x_2x_6 + x_1x_2x_6 + x_3x_6 + x_1x_3x_6 + x_1x_2x_3x_6 + x_0x_1x_2x_3x_6 + x_4x_6 + x_0x_1x_4x_6 + x_0x_1x_3x_4x_6 + x_1x_2x_3x_4x_6 + x_5x_6 + x_0x_1x_5x_6 + x_0x_4x_5x_6 + x_0x_2x_4x_5x_6 + x_0x_3x_4x_5x_6 + x_1x_3x_4x_5x_6 + x_2x_3x_4x_5x_6 + x_7 + x_0x_7 + x_1x_7 + x_2x_7 + x_1x_2x_7 + x_0x_1x_2x_7 + x_3x_7 + x_4x_7 + x_0x_4x_7 + x_1x_4x_7 + x_0x_1x_4x_7 + x_2x_4x_7 + x_1x_2x_4x_7 + x_3x_4x_7 + x_0x_2x_3x_4x_7 + x_1x_2x_3x_4x_7 + x_0x_1x_5x_7 + x_0x_3x_5x_7 + x_1x_3x_5x_7 + x_0x_4x_5x_7 + x_0x_2x_4x_5x_7 + x_3x_4x_5x_7 + x_0x_3x_4x_5x_7 + x_1x_3x_4x_5x_7 + x_6x_7 + x_0x_6x_7 + x_0x_1x_6x_7 + x_0x_2x_6x_7 + x_1x_2x_6x_7 + x_0x_1x_2x_6x_7 + x_0x_3x_6x_7 + x_1x_3x_6x_7 + x_0x_1x_3x_6x_7 + x_4x_6x_7 + x_0x_1x_4x_6x_7 + x_1x_3x_4x_6x_7 + x_2x_3x_4x_6x_7 + x_1x_2x_3x_4x_6x_7$$

$$y_4 = 1 + x_0 + x_1 + x_0x_1 + x_2 + x_0x_1x_2 + x_3 + x_0x_1x_3 + x_4 + x_1x_4 + x_0x_1x_4 + x_2x_4 + x_3x_4 + x_0x_3x_4 + x_2x_3x_4 + x_1x_2x_3x_4 + x_0x_5 + x_0x_1x_5 + x_1x_2x_5 + x_1x_3x_5 + x_0x_2x_3x_5 + x_1x_2x_3x_5 + x_0x_4x_5 + x_1x_4x_5 + x_2x_4x_5 + x_0x_2x_4x_5 + x_3x_4x_5 + x_1x_6 + x_2x_6 + x_0x_2x_6 + x_3x_6 + x_0x_3x_6 + x_0x_2x_3x_6 + x_1x_2x_3x_6 + x_4x_6 + x_0x_4x_6 + x_2x_4x_6 + x_0x_3x_4x_6 + x_1x_3x_4x_6 + x_5x_6 + x_0x_5x_6 + x_1x_5x_6 + x_4x_5x_6 + x_2x_4x_5x_6 + x_0x_7 + x_2x_7 + x_3x_7 + x_0x_3x_7 + x_1x_3x_7 + x_4x_7 + x_2x_3x_4x_7 + x_5x_7 + x_6x_7$$

$$y_5 = x_0 + x_0x_2 + x_1x_2 + x_2x_4 + x_5 + x_0x_6 + x_0x_1x_6 + x_2x_6 + x_0x_2x_6 + x_3x_6 + x_0x_3x_6 + x_0x_2x_3x_6 + x_1x_2x_3x_6 + x_0x_2x_4x_6 + x_2x_3x_4x_6 + x_7 + x_0x_7 + x_1x_7 + x_2x_7 + x_3x_7 + x_2x_4x_7 + x_6x_7$$

$y_6 = x_0 + x_1 + x_0x_1 + x_2 + x_3 + x_2x_4 + x_0x_2x_4 + x_5 + x_0x_5 + x_0x_1x_5 + x_2x_5 + x_0x_2x_5 + x_3x_5 + x_0x_3x_5 + x_0x_2x_3x_5 + x_1x_2x_3x_5 + x_0x_2x_4x_5 + x_2x_3x_4x_5 + x_6 + x_0x_7 + x_0x_3x_7 + x_1x_3x_7 + x_2x_3x_4x_7 + x_5x_7$

$y_7 = x_0 + x_1 + x_2x_4 + x_0x_5 + x_1x_5 + x_2x_5 + x_3x_5 + x_2x_4x_5 + x_6 + x_0x_6 + x_0x_3x_6 + x_1x_3x_6 + x_2x_3x_4x_6 + x_5x_6 + x_7$

A.2.3 Bitslice implementation

Algorithm 4 Inverse Scream S-box, bitslice implementation

Require: 8 16-bit words $(W_0, \ldots, W_7)$

Second 5-bit S-box

$W_1 = W_1 \oplus W_0$
$W_1 = W_1 \oplus (W_2 \wedge W_4)$
$W_4 = W_4 \oplus W_1$
$W_0 = W_0 \oplus (W_3 \wedge W_1)$
$W_2 = W_2 \oplus W_3$
$W_3 = W_3 \oplus (W_0 \wedge W_4)$
$W_1 = W_1 \oplus W_2$
$W_2 = W_2 \oplus (W_0 \wedge W_1)$

Truncate-Xor

$W_7 = W_2 \oplus W_7$
$W_6 = W_1 \oplus W_6$
$W_5 = W_0 \oplus W_5$

First 3-bit S-box

$(t_5, t_6, t_7) = (W_5, W_6, W_7)$
$W_5 = W_5 \oplus \neg(t_6) \wedge t_7$
$W_6 = W_6 \oplus \neg(t_7) \wedge t_5$
$W_7 = W_7 \oplus \neg(t_5) \wedge t_6$

Constant

$W_4 = \neg(W_4)$
$W_3 = \neg(W_3)$

Extend-Xor

$W_2 = W_2 \oplus W_7$
$W_1 = W_1 \oplus W_6$
$W_0 = W_0 \oplus W_5$

First 5-bit S-box

$W_1 = W_1 \oplus W_0$
$W_1 = W_1 \oplus (W_2 \wedge W_4)$
$W_4 = W_4 \oplus W_1$
$W_0 = W_0 \oplus (W_3 \wedge W_1)$
$W_2 = W_2 \oplus W_3$
$W_3 = W_3 \oplus (W_0 \wedge W_4)$
$W_1 = W_1 \oplus W_2$
$W_2 = W_2 \oplus (W_0 \wedge W_1)$

A.3 Scream L-box

A.3.1 Binary representation

$$\begin{pmatrix}
0 & 0 & 1 & 1 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 \\
1 & 0 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 0 & 1 & 0 & 1 & 0 \\
1 & 1 & 0 & 0 & 1 & 1 & 0 & 1 & 1 & 1 & 0 & 1 & 1 & 1 & 1 & 0 \\
1 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 1 & 1 & 0 & 1 & 0 & 0 & 1 \\
1 & 0 & 1 & 1 & 0 & 1 & 1 & 0 & 1 & 1 & 1 & 0 & 1 & 0 & 1 & 1 \\
0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 1 & 0 & 1 & 1 & 1 & 0 & 0 \\
0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 1 & 1 & 1 \\
1 & 0 & 1 & 0 & 0 & 1 & 0 & 1 & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\
1 & 1 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 1 & 0 \\
1 & 1 & 1 & 1 & 0 & 1 & 0 & 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\
1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 1 \\
0 & 0 & 1 & 0 & 1 & 1 & 1 & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 0 \\
0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 1 & 1 & 0 \\
1 & 1 & 1 & 1 & 0 & 1 & 1 & 0 & 0 & 1 & 0 & 1 & 1 & 1 & 1 & 0 \\
1 & 1 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 0 \\
1 & 0 & 0 & 0 & 1 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 1
\end{pmatrix}$$

A.3.2 8-bit table representation

$\mathsf{L}(b_0 \parallel b_1) = \big(\mathsf{L}_{1,1}(b_1) \oplus \mathsf{L}_{2,1}(b_0)\big) \parallel \big(\mathsf{L}_{1,2}(b_1) \oplus \mathsf{L}_{2,2}(b_0)\big)$.

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
00	00	38	52	6A	7B	43	29	11	96	AE	C4	FC	ED	D5	BF	87
10	D7	EF	85	BD	AC	94	FE	C6	41	79	13	2B	3A	02	68	50
20	3A	02	68	50	41	79	13	2B	AC	94	FE	C6	D7	EF	85	BD
30	ED	D5	BF	87	96	AE	C4	FC	7B	43	29	11	00	38	52	6A
40	E5	DD	B7	8F	9E	A6	CC	F4	73	4B	21	19	08	30	5A	62
50	32	0A	60	58	49	71	1B	23	A4	9C	F6	CE	DF	E7	8D	B5
60	DF	E7	8D	B5	A4	9C	F6	CE	49	71	1B	23	32	0A	60	58
70	08	30	5A	62	73	4B	21	19	9E	A6	CC	F4	E5	DD	B7	8F
80	FE	C6	AC	94	85	BD	D7	EF	68	50	3A	02	13	2B	41	79
90	29	11	7B	43	52	6A	00	38	BF	87	ED	D5	C4	FC	96	AE
A0	C4	FC	96	AE	BF	87	ED	D5	52	6A	00	38	29	11	7B	43
B0	13	2B	41	79	68	50	3A	02	85	BD	D7	EF	FE	C6	AC	94
C0	1B	23	49	71	60	58	32	0A	8D	B5	DF	E7	F6	CE	A4	9C
D0	CC	F4	9E	A6	B7	8F	E5	DD	5A	62	08	30	21	19	73	4B
E0	21	19	73	4B	5A	62	08	30	B7	8F	E5	DD	CC	F4	9E	A6
F0	F6	CE	A4	9C	8D	B5	DF	E7	60	58	32	0A	1B	23	49	71

Table 8: $\mathsf{L}_{1,1}$.

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
00	00	5C	A9	F5	B3	EF	1A	46	C1	9D	68	34	72	2E	DB	87
10	6D	31	C4	98	DE	82	77	2B	AC	F0	05	59	1F	43	B6	EA
20	E0	BC	49	15	53	0F	FA	A6	21	7D	88	D4	92	CE	3B	67
30	8D	D1	24	78	3E	62	97	CB	4C	10	E5	B9	FF	A3	56	0A
40	24	78	8D	D1	97	CB	3E	62	E5	B9	4C	10	56	0A	FF	A3
50	49	15	E0	BC	FA	A6	53	0F	88	D4	21	7D	3B	67	92	CE
60	C4	98	6D	31	77	2B	DE	82	05	59	AC	F0	B6	EA	1F	43
70	A9	F5	00	5C	1A	46	B3	EF	68	34	C1	9D	DB	87	72	2E
80	A5	F9	0C	50	16	4A	BF	E3	64	38	CD	91	D7	8B	7E	22
90	C8	94	61	3D	7B	27	D2	8E	09	55	A0	FC	BA	E6	13	4F
A0	45	19	EC	B0	F6	AA	5F	03	84	D8	2D	71	37	6B	9E	C2
B0	28	74	81	DD	9B	C7	32	6E	E9	B5	40	1C	5A	06	F3	AF
C0	81	DD	28	74	32	6E	9B	C7	40	1C	E9	B5	F3	AF	5A	06
D0	EC	B0	45	19	5F	03	F6	AA	2D	71	84	D8	9E	C2	37	6B
E0	61	3D	C8	94	D2	8E	7B	27	A0	FC	09	55	13	4F	BA	E6
F0	0C	50	A5	F9	BF	E3	16	4A	CD	91	64	38	7E	22	D7	8B

Table 9: $\mathsf{L}_{1,2}$.

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
00	00	46	F1	B7	A1	E7	50	16	7F	39	8E	C8	DE	98	2F	69
10	67	21	96	D0	C6	80	37	71	18	5E	E9	AF	B9	FF	48	0E
20	7A	3C	8B	CD	DB	9D	2A	6C	05	43	F4	B2	A4	E2	55	13
30	1D	5B	EC	AA	BC	FA	4D	0B	62	24	93	D5	C3	85	32	74
40	70	36	81	C7	D1	97	20	66	0F	49	FE	B8	AE	E8	5F	19
50	17	51	E6	A0	B6	F0	47	01	68	2E	99	DF	C9	8F	38	7E
60	0A	4C	FB	BD	AB	ED	5A	1C	75	33	84	C2	D4	92	25	63
70	6D	2B	9C	DA	CC	8A	3D	7B	12	54	E3	A5	B3	F5	42	04
80	8A	CC	7B	3D	2B	6D	DA	9C	F5	B3	04	42	54	12	A5	E3
90	ED	AB	1C	5A	4C	0A	BD	FB	92	D4	63	25	33	75	C2	84
A0	F0	B6	01	47	51	17	A0	E6	8F	C9	7E	38	2E	68	DF	99
B0	97	D1	66	20	36	70	C7	81	E8	AE	19	5F	49	0F	B8	FE
C0	FA	BC	0B	4D	5B	1D	AA	EC	85	C3	74	32	24	62	D5	93
D0	9D	DB	6C	2A	3C	7A	CD	8B	E2	A4	13	55	43	05	B2	F4
E0	80	C6	71	37	21	67	D0	96	FF	B9	0E	48	5E	18	AF	E9
F0	E7	A1	16	50	46	00	B7	F1	98	DE	69	2F	39	7F	C8	8E

Table 10: $\mathsf{L}_{2,1}$.

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
00	00	4B	AF	E4	33	78	9C	D7	74	3F	DB	90	47	0C	E8	A3
10	12	59	BD	F6	21	6A	8E	C5	66	2D	C9	82	55	1E	FA	B1
20	6F	24	C0	8B	5C	17	F3	B8	1B	50	B4	FF	28	63	87	CC
30	7D	36	D2	99	4E	05	E1	AA	09	42	A6	ED	3A	71	95	DE
40	1B	50	B4	FF	28	63	87	CC	6F	24	C0	8B	5C	17	F3	B8
50	09	42	A6	ED	3A	71	95	DE	7D	36	D2	99	4E	05	E1	AA
60	74	3F	DB	90	47	0C	E8	A3	00	4B	AF	E4	33	78	9C	D7
70	66	2D	C9	82	55	1E	FA	B1	12	59	BD	F6	21	6A	8E	C5
80	B1	FA	1E	55	82	C9	2D	66	C5	8E	6A	21	F6	BD	59	12
90	A3	E8	0C	47	90	DB	3F	74	D7	9C	78	33	E4	AF	4B	00
A0	DE	95	71	3A	ED	A6	42	09	AA	E1	05	4E	99	D2	36	7D
B0	CC	87	63	28	FF	B4	50	1B	B8	F3	17	5C	8B	C0	24	6F
C0	AA	E1	05	4E	99	D2	36	7D	DE	95	71	3A	ED	A6	42	09
D0	B8	F3	17	5C	8B	C0	24	6F	CC	87	63	28	FF	B4	50	1B
E0	C5	8E	6A	21	F6	BD	59	12	B1	FA	1E	55	82	C9	2D	66
F0	D7	9C	78	33	E4	AF	4B	00	A3	E8	0C	47	90	DB	3F	74

Table 11: $\mathsf{L}_{2,2}$.

A.4 Inverse Scream L-box

A.4.1 Binary representation

$$\begin{pmatrix}
0 & 1 & 1 & 1 & 1 & 0 & 0 & 1 & 1 & 1 & 1 & 0 & 0 & 1 & 1 & 1 \\
0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 1 & 1 & 1 & 0 \\
1 & 0 & 0 & 0 & 1 & 0 & 1 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 0 \\
1 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 1 & 1 & 0 \\
1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 1 & 1 \\
0 & 1 & 1 & 0 & 1 & 1 & 1 & 1 & 0 & 1 & 1 & 1 & 0 & 1 & 0 & 1 \\
1 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 1 & 0 & 0 \\
0 & 1 & 1 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 \\
0 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 0 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\
0 & 1 & 1 & 1 & 1 & 1 & 0 & 1 & 1 & 0 & 0 & 1 & 1 & 1 & 0 & 1 \\
0 & 0 & 0 & 1 & 1 & 0 & 1 & 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 0 \\
1 & 0 & 1 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 1 \\
1 & 1 & 1 & 1 & 1 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 1 & 0 \\
1 & 0 & 1 & 0 & 0 & 1 & 1 & 1 & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 0 \\
0 & 1 & 1 & 0 & 1 & 0 & 1 & 1 & 1 & 1 & 0 & 1 & 1 & 1 & 1 & 0 \\
0 & 0 & 0 & 1 & 1 & 0 & 1 & 1 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 1
\end{pmatrix}$$

A.4.2 8-bit table representation

$\mathsf{L}(b_0 \parallel b_1) = \big(\mathsf{L}_{1,1}(b_1) \oplus \mathsf{L}_{2,1}(b_0)\big) \parallel \big(\mathsf{L}_{1,2}(b_1) \oplus \mathsf{L}_{2,2}(b_0)\big)$.

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
00	00	E7	77	90	2A	CD	5D	BA	63	84	14	F3	49	AE	3E	D9
10	DC	3B	AB	4C	F6	11	81	66	BF	58	C8	2F	95	72	E2	05
20	AE	49	D9	3E	84	63	F3	14	CD	2A	BA	5D	E7	00	90	77
30	72	95	05	E2	58	BF	2F	C8	11	F6	66	81	3B	DC	4C	AB
40	29	CE	5E	B9	03	E4	74	93	4A	AD	3D	DA	60	87	17	F0
50	F5	12	82	65	DF	38	A8	4F	96	71	E1	06	BC	5B	CB	2C
60	87	60	F0	17	AD	4A	DA	3D	E4	03	93	74	CE	29	B9	5E
70	5B	BC	2C	CB	71	96	06	E1	38	DF	4F	A8	12	F5	65	82
80	82	65	F5	12	A8	4F	DF	38	E1	06	96	71	CB	2C	BC	5B
90	5E	B9	29	CE	74	93	03	E4	3D	DA	4A	AD	17	F0	60	87
A0	2C	CB	5B	BC	06	E1	71	96	4F	A8	38	DF	65	82	12	F5
B0	F0	17	87	60	DA	3D	AD	4A	93	74	E4	03	B9	5E	CE	29
C0	AB	4C	DC	3B	81	66	F6	11	C8	2F	BF	58	E2	05	95	72
D0	77	90	00	E7	5D	BA	2A	CD	14	F3	63	84	3E	D9	49	AE
E0	05	E2	72	95	2F	C8	58	BF	66	81	11	F6	4C	AB	3B	DC
F0	D9	3E	AE	49	F3	14	84	63	BA	5D	CD	2A	90	77	E7	00

Table 12: $\mathsf{L}_{1,1}$.

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
00	00	9E	04	9A	D1	4F	D5	4B	13	8D	17	89	C2	5C	C6	58
10	05	9B	01	9F	D4	4A	D0	4E	16	88	12	8C	C7	59	C3	5D
20	F6	68	F2	6C	27	B9	23	BD	E5	7B	E1	7F	34	AA	30	AE
30	F3	6D	F7	69	22	BC	26	B8	E0	7E	E4	7A	31	AF	35	AB
40	39	A7	3D	A3	E8	76	EC	72	2A	B4	2E	B0	FB	65	FF	61
50	3C	A2	38	A6	ED	73	E9	77	2F	B1	2B	B5	FE	60	FA	64
60	CF	51	CB	55	1E	80	1A	84	DC	42	D8	46	0D	93	09	97
70	CA	54	CE	50	1B	85	1F	81	D9	47	DD	43	08	96	0C	92
80	AE	30	AA	34	7F	E1	7B	E5	BD	23	B9	27	6C	F2	68	F6
90	AB	35	AF	31	7A	E4	7E	E0	B8	26	BC	22	69	F7	6D	F3
A0	58	C6	5C	C2	89	17	8D	13	4B	D5	4F	D1	9A	04	9E	00
B0	5D	C3	59	C7	8C	12	88	16	4E	D0	4A	D4	9F	01	9B	05
C0	97	09	93	0D	46	D8	42	DC	84	1A	80	1E	55	CB	51	CF
D0	92	0C	96	08	43	DD	47	D9	81	1F	85	1B	50	CE	54	CA
E0	61	FF	65	FB	B0	2E	B4	2A	72	EC	76	E8	A3	3D	A7	39
F0	64	FA	60	FE	B5	2B	B1	2F	77	E9	73	ED	A6	38	A2	3C

Table 13: $\mathsf{L}_{1,2}$.

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
00	00	1E	B9	A7	19	07	A0	BE	A8	B6	11	0F	B1	AF	08	16
10	6A	74	D3	CD	73	6D	CA	D4	C2	DC	7B	65	DB	C5	62	7C
20	7E	60	C7	D9	67	79	DE	C0	D6	C8	6F	71	CF	D1	76	68
30	14	0A	AD	B3	0D	13	B4	AA	BC	A2	05	1B	A5	BB	1C	02
40	7B	65	C2	DC	62	7C	DB	C5	D3	CD	6A	74	CA	D4	73	6D
50	11	0F	A8	B6	08	16	B1	AF	B9	A7	00	1E	A0	BE	19	07
60	05	1B	BC	A2	1C	02	A5	BB	AD	B3	14	0A	B4	AA	0D	13
70	6F	71	D6	C8	76	68	CF	D1	C7	D9	7E	60	DE	C0	67	79
80	86	98	3F	21	9F	81	26	38	2E	30	97	89	37	29	8E	90
90	EC	F2	55	4B	F5	EB	4C	52	44	5A	FD	E3	5D	43	E4	FA
A0	F8	E6	41	5F	E1	FF	58	46	50	4E	E9	F7	49	57	F0	EE
B0	92	8C	2B	35	8B	95	32	2C	3A	24	83	9D	23	3D	9A	84
C0	FD	E3	44	5A	E4	FA	5D	43	55	4B	EC	F2	4C	52	F5	EB
D0	97	89	2E	30	8E	90	37	29	3F	21	86	98	26	38	9F	81
E0	83	9D	3A	24	9A	84	23	3D	2B	35	92	8C	32	2C	8B	95
F0	E9	F7	50	4E	F0	EE	49	57	41	5F	F8	E6	58	46	E1	FF

Table 14: $\mathsf{L}_{2,1}$.

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
00	00	54	BE	EA	D8	8C	66	32	A5	F1	1B	4F	7D	29	C3	97
10	BF	EB	01	55	67	33	D9	8D	1A	4E	A4	F0	C2	96	7C	28
20	E5	B1	5B	0F	3D	69	83	D7	40	14	FE	AA	98	CC	26	72
30	5A	0E	E4	B0	82	D6	3C	68	FF	AB	41	15	27	73	99	CD
40	D6	82	68	3C	0E	5A	B0	E4	73	27	CD	99	AB	FF	15	41
50	69	3D	D7	83	B1	E5	0F	5B	CC	98	72	26	14	40	AA	FE
60	33	67	8D	D9	EB	BF	55	01	96	C2	28	7C	4E	1A	F0	A4
70	8C	D8	32	66	54	00	EA	BE	29	7D	97	C3	F1	A5	4F	1B
80	D8	8C	66	32	00	54	BE	EA	7D	29	C3	97	A5	F1	1B	4F
90	67	33	D9	8D	BF	EB	01	55	C2	96	7C	28	1A	4E	A4	F0
A0	3D	69	83	D7	E5	B1	5B	0F	98	CC	26	72	40	14	FE	AA
B0	82	D6	3C	68	5A	0E	E4	B0	27	73	99	CD	FF	AB	41	15
C0	0E	5A	B0	E4	D6	82	68	3C	AB	FF	15	41	73	27	CD	99
D0	B1	E5	0F	5B	69	3D	D7	83	14	40	AA	FE	CC	98	72	26
E0	EB	BF	55	01	33	67	8D	D9	4E	1A	F0	A4	96	C2	28	7C
F0	54	00	EA	BE	8C	D8	32	66	F1	A5	4F	1B	29	7D	97	C3

Table 15: $\mathsf{L}_{2,2}$.

B iScream parameters

B.1 iScream S-box

B.1.1 iScream S-box

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
00	00	85	65	D2	5B	FF	7A	CE	4D	E2	2C	36	92	15	BD	AD
10	57	F3	37	2D	88	0D	AC	BC	18	9F	7E	CA	41	EE	61	D6
20	59	EC	78	D4	47	F9	26	A3	90	8B	BF	30	0A	13	6F	C0
30	2B	AE	91	8A	D8	74	0B	12	CC	63	FD	43	B2	3D	E8	5D
40	B6	1C	83	3B	C8	45	9D	24	52	DD	E4	F4	AB	08	77	6D
50	F5	E5	48	C5	6C	76	BA	10	99	20	A7	04	87	3F	D0	5F
60	A5	1E	9B	39	B0	02	EA	67	C6	DF	71	F6	54	4F	8D	2E
70	E7	6A	C7	DE	35	97	55	4E	22	81	06	B4	7C	FB	1A	A1
80	D5	79	FC	42	84	01	E9	5C	14	93	33	29	C1	6E	A8	B8
90	28	32	0C	89	B9	A9	D9	75	ED	58	CD	62	F8	46	9E	19
A0	CB	7F	A2	27	D7	60	FE	5A	8E	95	E3	4C	16	0F	31	BE
B0	64	D3	3C	B3	7B	CF	40	EF	8F	94	56	F2	17	0E	AF	2A
C0	2F	8C	F1	E1	DC	53	68	72	44	C9	1B	A0	38	9A	07	B5
D0	5E	D1	03	B1	23	80	1F	A4	34	96	E0	F0	C4	49	73	69
E0	DA	C3	09	AA	4A	51	F7	70	3E	86	66	EB	21	98	1D	B7
F0	DB	C2	BB	11	4B	50	6B	E6	9C	25	FA	7D	82	3A	A6	05

Table 16: iScream S-box, table representation.

B.1.2 Algebraic Normal Form

$y_0 = x_0 + x_1 + x_2 + x_0x_2 + x_3 + x_0x_1x_3 + x_0x_2x_3 + x_4 + x_0x_4 + x_1x_4 + x_0x_1x_2x_4 + x_0x_3x_4 + x_5 + x_0x_1x_5 + x_2x_5 + x_4x_5 + x_0x_4x_5 + x_0x_1x_4x_5 + x_2x_4x_5 + x_1x_2x_4x_5 + x_0x_3x_4x_5 + x_1x_3x_4x_5 + x_2x_3x_4x_5 + x_0x_6 + x_2x_6 + x_3x_6 + x_0x_3x_6 + x_1x_3x_6 + x_2x_3x_6 + x_0x_4x_6 + x_1x_4x_6 + x_0x_1x_4x_6 + x_2x_4x_6 + x_0x_2x_4x_6 + x_1x_2x_4x_6 + x_0x_3x_4x_6 + x_0x_5x_6 + x_1x_5x_6 + x_3x_5x_6 + x_0x_3x_5x_6 + x_0x_4x_5x_6 + x_0x_1x_4x_5x_6 + x_2x_4x_5x_6 + x_0x_2x_4x_5x_6 + x_0x_3x_4x_5x_6 + x_7 + x_0x_7 + x_0x_3x_7 + x_0x_4x_7 + x_0x_1x_4x_7 + x_0x_2x_4x_7 + x_0x_3x_4x_7 + x_5x_7 + x_4x_5x_7 + x_0x_1x_4x_5x_7 + x_0x_2x_4x_5x_7 + x_1x_6x_7 + x_0x_1x_6x_7 + x_2x_6x_7 + x_0x_2x_6x_7 + x_3x_6x_7 + x_0x_4x_6x_7 + x_1x_4x_6x_7 + x_2x_4x_6x_7 + x_3x_4x_6x_7 + x_5x_6x_7 + x_0x_5x_6x_7$

$y_1 = x_0x_1 + x_2 + x_0x_1x_2 + x_0x_3 + x_0x_1x_3 + x_1x_2x_3 + x_4 + x_3x_4 + x_1x_3x_4 + x_0x_1x_3x_4 + x_2x_3x_4 + x_0x_1x_5 + x_0x_2x_5 + x_1x_3x_5 + x_0x_1x_3x_5 + x_1x_4x_5 + x_0x_1x_4x_5 + x_0x_2x_4x_5 + x_1x_3x_4x_5 + x_2x_3x_4x_5 + x_6 + x_0x_6 + x_0x_2x_6 + x_0x_3x_6 + x_1x_3x_6 + x_0x_1x_3x_6 + x_2x_3x_6 + x_0x_2x_3x_6 + x_0x_4x_6 + x_0x_1x_4x_6 + x_2x_4x_6 + x_1x_2x_4x_6 + x_0x_1x_2x_4x_6 + x_3x_4x_6 + x_1x_3x_4x_6 + x_2x_3x_4x_6 + x_5x_6 + x_1x_5x_6 + x_2x_5x_6 + x_3x_5x_6 + x_0x_3x_5x_6 + x_1x_2x_4x_5x_6 + x_3x_4x_5x_6 + x_1x_3x_4x_5x_6 + x_2x_7 + x_1x_3x_7 + x_4x_7 + x_0x_4x_7 + x_0x_2x_4x_7 + x_3x_4x_7 + x_2x_3x_4x_7 + x_5x_7 + x_4x_5x_7 + x_1x_4x_5x_7 + x_0x_1x_4x_5x_7 + x_2x_4x_5x_7 + x_1x_2x_4x_5x_7 + x_3x_4x_5x_7 + x_0x_3x_4x_5x_7 + x_1x_6x_7 + x_2x_6x_7 + x_0x_2x_6x_7 + x_1x_2x_6x_7 + x_3x_6x_7 + x_0x_3x_6x_7 + x_1x_4x_6x_7 + x_0x_2x_4x_6x_7 + x_3x_4x_6x_7 + x_0x_3x_4x_6x_7 + x_0x_5x_6x_7 + x_1x_5x_6x_7 + x_0x_1x_5x_6x_7 + x_4x_5x_6x_7 + x_0x_1x_4x_5x_6x_7 + x_0x_2x_4x_5x_6x_7 + x_3x_4x_5x_6x_7$

$y_2 = x_0 + x_1 + x_1x_2 + x_3 + x_1x_3 + x_0x_1x_3 + x_2x_3 + x_4 + x_1x_4 + x_0x_1x_4 + x_2x_4 + x_1x_5 + x_2x_5 + x_1x_2x_5 + x_3x_5 + x_0x_3x_5 + x_4x_5 + x_1x_4x_5 + x_3x_4x_5 + x_0x_3x_4x_5 + x_1x_3x_4x_5 + x_6 + x_0x_6 + x_2x_6 + x_0x_2x_6 + x_1x_2x_6 + x_0x_1x_2x_6 + x_0x_3x_6 + x_1x_3x_6 + x_4x_6 + x_1x_4x_6 + x_0x_2x_4x_6 + x_0x_3x_4x_6 + x_1x_5x_6 + x_2x_5x_6 + x_0x_2x_5x_6 + x_4x_5x_6 + x_0x_4x_5x_6 + x_1x_2x_4x_5x_6 + x_0x_3x_4x_5x_6 + x_7 + x_1x_7 + x_3x_7 + x_0x_4x_7 + x_2x_4x_7 +$

$x_3x_4x_7+x_0x_3x_4x_7+x_5x_7+x_1x_5x_7+x_4x_5x_7+x_0x_4x_5x_7+x_0x_1x_4x_5x_7+x_3x_4x_5x_7+x_6x_7+x_1x_6x_7+$
$x_2x_6x_7+x_2x_4x_6x_7+x_0x_2x_4x_6x_7+x_3x_4x_6x_7+x_4x_5x_6x_7+x_0x_4x_5x_6x_7+x_1x_4x_5x_6x_7+x_2x_4x_5x_6x_7$

$y_3 = x_2+x_3+x_0x_3+x_0x_2x_3+x_1x_2x_3+x_0x_1x_4+x_0x_1x_2x_4+x_0x_3x_4+x_0x_1x_3x_4+x_1x_2x_3x_4+$
$x_5+x_0x_1x_5+x_0x_2x_5+x_1x_3x_5+x_0x_1x_3x_5+x_1x_4x_5+x_0x_1x_4x_5+x_2x_4x_5+x_1x_2x_4x_5+x_3x_4x_5+$
$x_0x_3x_4x_5+x_0x_1x_3x_4x_5+x_2x_3x_4x_5+x_0x_6+x_3x_6+x_0x_3x_6+x_0x_1x_3x_6+x_0x_4x_6+x_1x_4x_6+x_0x_2x_4x_6+$
$x_1x_2x_4x_6+x_3x_4x_6+x_5x_6+x_1x_5x_6+x_2x_5x_6+x_0x_4x_5x_6+x_1x_4x_5x_6+x_0x_1x_4x_5x_6+x_2x_4x_5x_6+$
$x_0x_2x_4x_5x_6+x_0x_3x_4x_5x_6+x_1x_3x_4x_5x_6+x_0x_7+x_1x_7+x_2x_7+x_0x_2x_7+x_3x_7+x_1x_3x_7+x_0x_1x_3x_7+$
$x_0x_2x_3x_7+x_4x_7+x_1x_4x_7+x_0x_1x_2x_4x_7+x_0x_3x_4x_7+x_1x_3x_4x_7+x_0x_5x_7+x_0x_1x_5x_7+x_2x_5x_7+$
$x_0x_1x_4x_5x_7+x_2x_4x_5x_7+x_1x_2x_4x_5x_7+x_0x_3x_4x_5x_7+x_1x_3x_4x_5x_7+x_2x_3x_4x_5x_7+x_6x_7+x_1x_2x_6x_7+$
$x_0x_3x_6x_7+x_1x_3x_6x_7+x_2x_3x_6x_7+x_4x_6x_7+x_2x_4x_6x_7+x_0x_2x_4x_6x_7+x_3x_4x_6x_7+x_0x_1x_5x_6x_7+$
$x_3x_5x_6x_7+x_0x_3x_5x_6x_7+x_0x_1x_4x_5x_6x_7+x_0x_2x_4x_5x_6x_7+x_3x_4x_5x_6x_7+x_0x_3x_4x_5x_6x_7$

$y_4 = x_0x_1+x_2+x_4+x_5+x_0x_5+x_0x_4x_5+x_1x_4x_5+x_6+x_1x_6+x_4x_6+x_0x_4x_6+x_0x_4x_5x_6+$
$x_7+x_0x_4x_7+x_4x_5x_7+x_4x_6x_7$

$y_5 = x_1+x_0x_1+x_0x_2+x_0x_1x_2+x_0x_3+x_0x_4+x_0x_5+x_0x_1x_5+x_0x_2x_5+x_4x_5+x_2x_4x_5+$
$x_0x_2x_4x_5+x_1x_2x_4x_5+x_3x_4x_5+x_6+x_0x_6+x_0x_1x_6+x_2x_6+x_1x_2x_6+x_3x_6+x_2x_4x_6+x_0x_2x_4x_6+$
$x_0x_5x_6+x_0x_1x_5x_6+x_2x_5x_6+x_4x_5x_6+x_0x_2x_4x_5x_6+x_0x_7+x_0x_1x_7+x_4x_7+x_0x_2x_4x_7+x_0x_5x_7+$
$x_4x_5x_7+x_0x_4x_5x_7+x_1x_4x_5x_7+x_2x_4x_5x_7+x_0x_6x_7+x_1x_6x_7+x_2x_4x_6x_7+x_5x_6x_7$

$y_6 = x_1+x_2+x_1x_2+x_3+x_4+x_5+x_1x_5+x_2x_5+x_2x_4x_5+x_1x_6+x_2x_4x_6+x_5x_6+x_1x_5x_6+$
$x_2x_4x_5x_6+x_7+x_1x_7+x_2x_4x_7+x_5x_7+x_4x_5x_7+x_6x_7$

$y_7 = x_0+x_0x_1x_3+x_2x_3+x_0x_1x_4+x_2x_4+x_3x_5+x_0x_3x_5+x_1x_4x_5+x_0x_3x_4x_5+x_1x_3x_4x_5+$
$x_6+x_3x_6+x_1x_3x_6+x_0x_4x_6+x_1x_4x_6+x_3x_4x_6+x_0x_3x_4x_6+x_0x_4x_5x_6+x_0x_3x_4x_5x_6+x_7+x_3x_7+$
$x_4x_7+x_0x_4x_7+x_0x_3x_4x_7+x_0x_4x_5x_7+x_1x_4x_5x_7+x_0x_1x_4x_5x_7+x_2x_4x_5x_7+x_3x_4x_5x_7+x_1x_6x_7+$
$x_0x_1x_6x_7+x_2x_6x_7+x_4x_6x_7+x_3x_4x_6x_7+x_5x_6x_7+x_0x_5x_6x_7$

B.1.3 Bitslice implementation

Algorithm 5 iScream S-box, bitslice implementation

Require: 8 16-bit words $(W_0, \ldots, W_7)$

First 4-bit S-box

$t_0 = W_4 \wedge W_5$
$t_0 = t_0 \oplus W_6$
$t_2 = W_6 \vee W_5$
$t_2 = t_2 \oplus W_7$
$t_3 = t_0 \wedge W_7$
$t_3 = t_0 \oplus W_4$
$t_1 = t_3 \wedge W_4$
$t_1 = t_1 \oplus W_5$
$W_0 = W_0 \oplus t_0$
$W_1 = W_1 \oplus t_1$
$W_2 = W_2 \oplus t_2$
$W_3 = W_3 \oplus t_3$

Second 4-bit S-box

$t_0 = W_0 \wedge W_1$
$t_0 = t_0 \oplus W_2$
$t_2 = W_2 \vee W_1$
$t_2 = t_2 \oplus W_3$
$t_3 = t_0 \wedge W_3$
$t_3 = t_0 \oplus W_0$
$t_1 = t_3 \wedge W_0$
$t_1 = t_1 \oplus W_1$
$W_4 = W_4 \oplus t_0$
$W_5 = W_5 \oplus t_1$
$W_6 = W_6 \oplus t_2$
$W_7 = W_7 \oplus t_3$

Third 4-bit S-box

$t_0 = W_4 \wedge W_5$
$t_0 = t_0 \oplus W_6$
$t_2 = W_6 \vee W_5$
$t_2 = t_2 \oplus W_7$
$t_3 = t_0 \wedge W_7$
$t_3 = t_0 \oplus W_4$
$t_1 = t_3 \wedge W_4$
$t_1 = t_1 \oplus W_5$
$W_0 = W_0 \oplus t_0$
$W_1 = W_1 \oplus t_1$
$W_2 = W_2 \oplus t_2$
$W_3 = W_3 \oplus t_3$

B.2 iScream L-box

B.2.1 Binary representation

$$\begin{pmatrix}
0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\
1 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 \\
1 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 \\
1 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 1 \\
1 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 1 & 1 & 0 & 0 & 1 & 1 & 0 \\
1 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 \\
1 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 \\
1 & 1 & 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
1 & 0 & 0 & 1 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 \\
1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 1 \\
1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 1 \\
1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 0 \\
1 & 0 & 0 & 1 & 0 & 1 & 1 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\
1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 1 & 0 & 0 & 0 & 0 \\
1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 & 0 & 1 & 0 & 0 & 0 & 0 \\
1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0
\end{pmatrix}$$

B.2.2 8-bit table representation

$\mathsf{L}(b_0 \,\|\, b_1) = \big(\mathsf{L}_{1,1}(b_1) \oplus \mathsf{L}_{2,1}(b_0)\big) \,\|\, \big(\mathsf{L}_{1,2}(b_1) \oplus \mathsf{L}_{2,2}(b_0)\big)$.

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
00	00	FF	CC	33	AA	55	66	99	99	66	55	AA	33	CC	FF	00
10	66	99	AA	55	CC	33	00	FF	FF	00	33	CC	55	AA	99	66
20	55	AA	99	66	FF	00	33	CC	CC	33	00	FF	66	99	AA	55
30	33	CC	FF	00	99	66	55	AA	AA	55	66	99	00	FF	CC	33
40	33	CC	FF	00	99	66	55	AA	AA	55	66	99	00	FF	CC	33
50	55	AA	99	66	FF	00	33	CC	CC	33	00	FF	66	99	AA	55
60	66	99	AA	55	CC	33	00	FF	FF	00	33	CC	55	AA	99	66
70	00	FF	CC	33	AA	55	66	99	99	66	55	AA	33	CC	FF	00
80	00	FF	CC	33	AA	55	66	99	99	66	55	AA	33	CC	FF	00
90	66	99	AA	55	CC	33	00	FF	FF	00	33	CC	55	AA	99	66
A0	55	AA	99	66	FF	00	33	CC	CC	33	00	FF	66	99	AA	55
B0	33	CC	FF	00	99	66	55	AA	AA	55	66	99	00	FF	CC	33
C0	33	CC	FF	00	99	66	55	AA	AA	55	66	99	00	FF	CC	33
D0	55	AA	99	66	FF	00	33	CC	CC	33	00	FF	66	99	AA	55
E0	66	99	AA	55	CC	33	00	FF	FF	00	33	CC	55	AA	99	66
F0	00	FF	CC	33	AA	55	66	99	99	66	55	AA	33	CC	FF	00

Table 17: $\mathsf{L}_{1,1}$.

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
00	00	FE	C1	3F	A1	5F	60	9E	91	6F	50	AE	30	CE	F1	0F
10	89	77	48	B6	28	D6	E9	17	18	E6	D9	27	B9	47	78	86
20	85	7B	44	BA	24	DA	E5	1B	14	EA	D5	2B	B5	4B	74	8A
30	0C	F2	CD	33	AD	53	6C	92	9D	63	5C	A2	3C	C2	FD	03
40	83	7D	42	BC	22	DC	E3	1D	12	EC	D3	2D	B3	4D	72	8C
50	0A	F4	CB	35	AB	55	6A	94	9B	65	5A	A4	3A	C4	FB	05
60	06	F8	C7	39	A7	59	66	98	97	69	56	A8	36	C8	F7	09
70	8F	71	4E	B0	2E	D0	EF	11	1E	E0	DF	21	BF	41	7E	80
80	7F	81	BE	40	DE	20	1F	E1	EE	10	2F	D1	4F	B1	8E	70
90	F6	08	37	C9	57	A9	96	68	67	99	A6	58	C6	38	07	F9
A0	FA	04	3B	C5	5B	A5	9A	64	6B	95	AA	54	CA	34	0B	F5
B0	73	8D	B2	4C	D2	2C	13	ED	E2	1C	23	DD	43	BD	82	7C
C0	FC	02	3D	C3	5D	A3	9C	62	6D	93	AC	52	CC	32	0D	F3
D0	75	8B	B4	4A	D4	2A	15	EB	E4	1A	25	DB	45	BB	84	7A
E0	79	87	B8	46	D8	26	19	E7	E8	16	29	D7	49	B7	88	76
F0	F0	0E	31	CF	51	AF	90	6E	61	9F	A0	5E	C0	3E	01	FF

Table 18: $\mathsf{L}_{1,2}$.

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
00	00	E0	D0	30	B0	50	60	80	70	90	A0	40	C0	20	10	F0
10	0E	EE	DE	3E	BE	5E	6E	8E	7E	9E	AE	4E	CE	2E	1E	FE
20	0D	ED	DD	3D	BD	5D	6D	8D	7D	9D	AD	4D	CD	2D	1D	FD
30	03	E3	D3	33	B3	53	63	83	73	93	A3	43	C3	23	13	F3
40	0B	EB	DB	3B	BB	5B	6B	8B	7B	9B	AB	4B	CB	2B	1B	FB
50	05	E5	D5	35	B5	55	65	85	75	95	A5	45	C5	25	15	F5
60	06	E6	D6	36	B6	56	66	86	76	96	A6	46	C6	26	16	F6
70	08	E8	D8	38	B8	58	68	88	78	98	A8	48	C8	28	18	F8
80	07	E7	D7	37	B7	57	67	87	77	97	A7	47	C7	27	17	F7
90	09	E9	D9	39	B9	59	69	89	79	99	A9	49	C9	29	19	F9
A0	0A	EA	DA	3A	BA	5A	6A	8A	7A	9A	AA	4A	CA	2A	1A	FA
B0	04	E4	D4	34	B4	54	64	84	74	94	A4	44	C4	24	14	F4
C0	0C	EC	DC	3C	BC	5C	6C	8C	7C	9C	AC	4C	CC	2C	1C	FC
D0	02	E2	D2	32	B2	52	62	82	72	92	A2	42	C2	22	12	F2
E0	01	E1	D1	31	B1	51	61	81	71	91	A1	41	C1	21	11	F1
F0	0F	EF	DF	3F	BF	5F	6F	8F	7F	9F	AF	4F	CF	2F	1F	FF

Table 19: $\mathsf{L}_{2,1}$.

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
00	00	69	55	3C	33	5A	66	0F	0F	66	5A	33	3C	55	69	00
10	69	00	3C	55	5A	33	0F	66	66	0F	33	5A	55	3C	00	69
20	55	3C	00	69	66	0F	33	5A	5A	33	0F	66	69	00	3C	55
30	3C	55	69	00	0F	66	5A	33	33	5A	66	0F	00	69	55	3C
40	33	5A	66	0F	00	69	55	3C	3C	55	69	00	0F	66	5A	33
50	5A	33	0F	66	69	00	3C	55	55	3C	00	69	66	0F	33	5A
60	66	0F	33	5A	55	3C	00	69	69	00	3C	55	5A	33	0F	66
70	0F	66	5A	33	3C	55	69	00	00	69	55	3C	33	5A	66	0F
80	0F	66	5A	33	3C	55	69	00	00	69	55	3C	33	5A	66	0F
90	66	0F	33	5A	55	3C	00	69	69	00	3C	55	5A	33	0F	66
A0	5A	33	0F	66	69	00	3C	55	55	3C	00	69	66	0F	33	5A
B0	33	5A	66	0F	00	69	55	3C	3C	55	69	00	0F	66	5A	33
C0	3C	55	69	00	0F	66	5A	33	33	5A	66	0F	00	69	55	3C
D0	55	3C	00	69	66	0F	33	5A	5A	33	0F	66	69	00	3C	55
E0	69	00	3C	55	5A	33	0F	66	66	0F	33	5A	55	3C	00	69
F0	00	69	55	3C	33	5A	66	0F	0F	66	5A	33	3C	55	69	00

Table 20: $\mathsf{L}_{2,2}$.

C Round Constants

We use simple constants in order to limit their implementation cost, but expect them to avoid any kind of slide attack of self-similarity property. The round constants in Scream and iScream are defined as: $C(\rho) = 27 \cdot \rho \bmod 256$. We consider constants of the form $C(\rho) = C \cdot \rho \bmod 256$ because they can implemented by incrementing a counter by steps of C. However, we would rather avoid the trivial choice $C = 1$ because this implies simple linear relations between the constants, such as $C(2\rho + 1) = C(2\rho) \oplus 1$. Concretely, we built 8×8 matrices with the binary representation of $C(1)$, $C(2)$, ..., $C(8)$, and computed the rank of these matrices. There are a few values that give a full rank matrix, and we decided to use the smallest value with this property: 27.

D Analysis of differential trails

In this section we give lower bounds on the number of active S-boxes for differential trails on Scream and iScream. We use a simple notation to describe trails, where '`x`' denotes an active step, and '`-`' an inactive step. With this notation, trails using the tweak to limit the number of active steps to $\lfloor \sigma/2 \rfloor$ are denoted as '`-x-x-x-`'.

D.1 Analysis of Scream

D.1.1 Single key, fixed tweak

With a fixed tweak, Scream is a slightly modified version of Fantomas, with a different L-box. Therefore, we can compute bounds on its number of active S-boxes for differential and linear trails following [11].

D.1.2 Single key, chosen tweaks

The L-box of Scream has been chosen so that trails over one step (2 rounds) with the same input and output difference pattern have at least 14 active S-boxes. Due to the structure of the tweakey scheduling, a pair of tweaks with a truncated difference δ gives tweakeys with the difference pattern δ in every round. In particular a trail '`-x-`' using a difference in the tweak leads to the same pattern of active S-boxes δ at the beginning and at the end of the second round; this implies at least 14 active S-boxes.

We also use bounds from single key trails in our analysis of related-key trails. For instance, a trail '`-xx-`' gives truncated differences $\delta \rightsquigarrow a$, $b \rightsquigarrow \delta$ assuming a difference δ in the tweakeys, and the truncated difference a must be transformed into b after a tweakey addition with difference δ. This can transformed into a single key trail over two steps: $b \rightsquigarrow \delta \rightsquigarrow a$, and we know that such a trail has at least 20 active S-boxes. Moreover, we can enumerate the trails with 20 active S-boxes, and we found than there is a single trail where the patterns a, b, and δ are compatible:

$$b = \texttt{0010011100101001} \rightsquigarrow \texttt{0100001001101000} \rightsquigarrow \texttt{1100000000000001} = \delta,$$
$$\delta = \texttt{1100000000000001} \rightsquigarrow \texttt{0010001000000111} \rightsquigarrow \texttt{1110011100101001} = a.$$

This leads to the following analysis of differential trails:

5-step trails. We can list all the possible 5-step trails with 3 active steps or less (we use symmetries to consider only half of the patterns), and compute a lower bound on the corresponding number of active S-boxes:
`-x-x-:` At least 28 active S-boxes.

`-x-xx-`: At least 30 active S-boxes $(14 + 8 + 8)$.
`-xx-x-`: At least 28 active S-boxes $(20 + 8)$.
`-xxx-`: At least 28 active S-boxes $(20 + 8$; 20 for steps 1 and 3 combined$)$.
`x-x-x-`: At least 30 active S-boxes $(8 + 14 + 8)$.
If there are 4 or more active steps, this implies at least $8 \times 4 = 32$ active S-boxes.

6-step trails. Similarly, we can list all the possible 6-step trails with 4 active steps or less:
`-x-x-x-`: At least 36 active S-boxes $(14 + 14 + 8)$.
`-x-xx-`: At least 34 active S-boxes $(14 + 20)$.
`-x-xxx`: At least 38 active S-boxes $(14 + 8 + 8 + 8)$.
`-xx-xx`: At least 36 active S-boxes $(20 + 8 + 8)$.
`-xxx-x`: At least 36 active S-boxes $(20 + 8 + 8$; 20 for steps 1 and 3 combined$)$.
`-xxxx-`: At least 36 active S-boxes $(20 + 8 + 8$; 20 for steps 1 and 4 combined$)$.
`x-x-xx`: At least 38 active S-boxes $(8 + 14 + 8 + 8)$.
`x-xx-x`: At least 36 active S-boxes $(8 + 20 + 8)$.
If there are 5 or more active steps, this implies at least $8 \times 5 = 40$ active S-boxes.

In addition, we verified that there is no valid trail with only 34 active S-boxes following `-x-xx-`: the only trail with weight 20 for steps 3 and 4 gives a tweak difference δ with no valid trails $\delta \rightsquigarrow \delta$ for step 1. This proves that 6-step trails have at least 35 active S-boxes.

D.1.3 Related keys, chosen tweaks

All tweakeys active. First, let us consider trails with a difference in all the tweakeys. Using a difference in the key and in the tweak, it could be possible to have tweakey differences with only 8 active S-boxes per step. However, such trails must still activate at least $\lfloor \sigma/2 \rfloor$ steps. Therefore, they have at least $8 \cdot \lfloor \sigma/2 \rfloor$ active S-boxes.

We can improve this bound using the property of the tweakey scheduling that the same tweakeys are used every three rounds. In particular, a trail '`-x-x-`' would have tweakey differences δ_0, δ_1, δ_2, δ_0, δ_1, δ_2, and transitions $\delta_1 \rightsquigarrow \delta_2$ and $\delta_0 \rightsquigarrow \delta_1$. This can be turned into a 2-step single key trail $\delta_0 \rightsquigarrow \delta_1 \rightsquigarrow \delta_2$ that has at least 20 active S-boxes. This implies the following bounds:

- The only 7-step trail with 3 active steps is '`-x-x-x-`'; it has at least 28 active S-boxes. Trails with 4 or more active steps have at least 32 active S-boxes.
- The only 9-step trail with 4 active steps is '`-x-x-x-x-`'; it has at least 40 active S-boxes. Trails with 5 or more active steps have at least 40 active S-boxes.

Some tweakeys inactive. The tweakey scheduling of Scream further allows to cancel some key and tweak differences. Let us now consider trails where some tweakeys are inactive.

Without loss of generality, we assume that $\delta[K] = \delta[T]$. The tweakey differences for the following rounds will be:

$$\delta[K \oplus \phi(T)] = \delta[T] \oplus \phi(\delta[T]) = \phi^2(\delta[T]),$$
$$\delta[K \oplus \phi^2(T)] = \delta[T] \oplus \phi^2(\delta[T]) = \phi(\delta[T]).$$

In particular, the truncated pattern δ will be the same for all the active tweakeys, because ϕ is computed on each state line independently.

This leads to the following analysis of trails:

2-step trails. They can have no active step if the second tweakey is inactive.

3-step trails. A 3-step trail with a single active step has at least 8 active S-boxes. This can be reached by following '`--x`'.

4-step trails. A 4-step trail with a single active step must follow '`--x-`'. The third step must have the same input and output pattern, therefore it has at least 14 active S-boxes. Trails with two or more active steps have at least 16 active S-boxes.

More generally, a trail over σ steps has at least:

- $14 \cdot \lfloor \sigma/3 \rfloor - 6$ active S-boxes if $\sigma \bmod 3 = 0$.
- $14 \cdot \lfloor \sigma/3 \rfloor$ active S-boxes otherwise

D.2 Analysis of iScream

D.2.1 Single key, fixed tweak

With a fixed tweak, iScream is a slightly modified version of Robin. Therefore, the bounds given in [11] apply: there are at least 8 active S-boxes every 2 rounds, *i.e.* every step. As in Robin, it is easy to see that this bound is tight.

D.2.2 Single key, chosen tweaks

A chosen tweak attack on iScream is similar to a related-key attack on Robin with chosen key differences. Differences in the tweak can be used to cancel differences in the state, but at least half of the steps must be active: an inactive step is always followed by an active one because the tweak difference is inserted in the state again.

In iScream, we designed the tweakey scheduling in order to avoid the simple related-key trails of Robin, and to limit the penalty of using involutive components. In particular, trails like '`-x-x-x-`' must have a transition $\delta \rightsquigarrow \delta \lll 1$ or $\delta \rightsquigarrow \delta \ggg 1$ through one step, where δ is the truncated tweakey difference. Thanks to the rotation, the best trails of this class have 12 active S-boxes, rather than 8 (as given by the branch number bound). This can be obtained easily by looking for the minimal value of $|x| + |y|$ where the transitions $x \rightsquigarrow y$ and $y \rightsquigarrow x \ggg 1$ are allowed by the L-box.

This leads to the following analysis of trails:

3-step trails. A 3-step trail with a single active step follows '`-x-`', and has at least 12 active S-boxes. If there are 2 or more active steps, this gives at least 16 active S-boxes.

4-step trails. A 4-step trail has at least 2 active steps, hence 16 active S-boxes.

5-step trails. A 5-step trail with 2 active steps must follow '`-x-x-`', and has at least 24 active S-boxes. If there are 3 or more active steps, this gives at least 24 active S-boxes.

6-step trails. We can list all the possible 6-step trails with 3 active steps or less (we use symmetries to consider only half of the patterns), and compute a lower bound on the corresponding number of active S-boxes:
`-x-x-x-`: At least 32 active S-boxes $(12 + 12 + 8)$.
`-x-xx-`: At least 28 active S-boxes $(12 + 8 + 8)$.
If there are 4 or more active steps, this implies at least $8 \times 4 = 32$ active S-boxes.

7-step trails. There is only one 7-step trail with 3 active steps or less: '`-x-x-x-`', with at least 36 active S-boxes $(12 + 12 + 12)$. If there are 4 or more active steps, this implies at least $8 \times 4 = 32$ active S-boxes. This bound can be reached using a trail '`-xx-xx-`'.

8-step trails. We can list all the possible 8-step trails with 4 active steps or less:
`-x-x-x-x-`: At least 44 active S-boxes $(12 + 12 + 12 + 8)$.

`-x-x-xx-:` At least 40 active S-boxes $(12 + 12 + 8 + 8)$.
`-x-xx-x-:` At least 40 active S-boxes $(12 + 8 + 8 + 12)$.
If there are 5 or more active steps, this implies at least $8 \times 5 = 40$ active S-boxes.

D.2.3 Related keys, chosen tweaks

Related-key trails for iScream can use a difference in the key in order to activate blocks of 2 steps. There will still be at least $16 \cdot \lfloor \sigma/4 \rfloor$ active S-boxes in this case.