An Analysis of the Hermes8 Stream Ciphers

Steve Babbage[1], Carlos Cid[2], Norbert Pramstaller[3] and Håvard Raddum[4]

[1] Vodafone Group R&D,
Newbury, United Kingdom
steve.babbage@vodafone.com
[2] Information Security Group,
Royal Holloway, University of London
Egham, United Kingdom
carlos.cid@rhul.ac.uk
[3] IAIK, Graz University of Technology
Graz, Austria
norbert.pramstaller@iaik.tugraz.at
[4] Dept. of Informatics, The University of Bergen,
Bergen, Norway
haavardr@ii.uib.no

Abstract Hermes8 [6,7] is one of the stream ciphers submitted to the ECRYPT Stream Cipher Project (eSTREAM [3]). In this paper we present an analysis of the Hermes8 stream ciphers. In particular, we show an attack on the latest version of the cipher (Hermes8F), which requires very few known keystream bytes and recovers the cipher secret key in less than a second on a normal PC. Furthermore, we make some remarks on the cipher's key schedule and discuss some properties of ciphers with similar algebraic structure to Hermes8.
Keywords: Hermes8, Stream Cipher, Cryptanalysis.

1 Introduction

Hermes8 is one of the 34 stream ciphers submitted to eSTREAM, the ECRYPT Stream Cipher Project [3]. The cipher has a simple byte-oriented design, consisting of substitutions and shifts of the state register bytes. Two versions of the cipher have been proposed. Originally, the cipher Hermes8 [6] was submitted as candidate to eSTREAM. Although no weaknesses of Hermes8 were found during the first phase of evaluation, the cipher did not seem to present satisfactory performance in either software or hardware [4]. As a result, a slightly modified version of the cipher, named Hermes8F [7], was submitted for consideration during the second phase of eSTREAM. In this paper we present an analysis of the Hermes8 stream ciphers. In Section 2 we present an alternative description of the Hermes8 ciphers. Section 3 describes an attack against the latest version of Hermes8. Section 4 contains some remarks on the key schedule of Hermes8, while we discuss some algebraic properties of the ciphers in Section 5.

2 Description of Hermes8F

According to [7], Hermes8F is a stream cipher based on the Substitution– Permutation network principle. Hermes8F is defined for two different key lengths: Hermes8F-80 uses 80-bit keys, while Hermes8F-128 uses 128-bit keys. The cipher uses two byte-oriented registers: a 17-byte state register and a 10-byte key register (16 bytes for Hermes8F-128). Additionally, there is a single byte register *Accu*, which seems to have the use of a memory register (Figure 1). The diffusion is provided by moving pointers through both registers, while non-linearity is provided by the AES S-Box [2].

The main operation of the cipher consists of the following steps:

1. XOR the value stored at *Accu* with a byte from the state register and a byte from the key register;
2. Use the previous result as input for the AES S-Box;
3. Replace the state register value used in step 1. by the output of the S-Box;
4. Store the output of the S-Box also in *Accu*;
5. Increment both the state and key register pointers (denoted by $p1$ and $p2$, respectively).

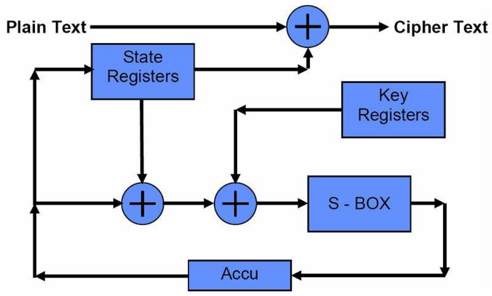

Figure1. Hermes8F stream cipher [7].

The steps above are performed at each clocking. A round of the cipher consists of 17 clockings. At every 7 clockings, two bytes of the key register are updated. The updating function is also based on the AES S-Box (Section 4). In the cipher's initialization, the encryption key is loaded into the key register, and the IV is loaded into the state register. The register *Accu* starts with the zero byte as content[1]. The initialization process consists of five rounds (i.e. 85 clockings),

[1] In Hermes8, the initial value of *Accu* is key-dependent; see Section 4.

and so all the state registers are updated five times before the cipher enters the normal mode of operation. The first bytes of the keystream are produced after two further rounds. The output consists of 8 bytes from the state register, taken from alternating positions of the register. Further bytes of the output are produced at every two rounds. More details of the algorithm can be found in [7].

2.1 Alternative Description of Hermes8F

We note that it follows from the description above that during the cipher operation, the contents of the registers $Accu$ and state$[p1-1]$ are always the same. Thus a more natural description of Hermes8F is given in Figure 2. It consists of the state register R, which is represented as a feedback shift register of length 17, defined as

$$s_i^t = \text{state}[p1+i] \quad , \quad 0 \le i \le 16,$$

where state$[p1]$ is the byte addressed by pointer $p1$ at time t. This FSR is updated according to the following relations:

$$\begin{aligned} s_i^{t+1} &= s_{i+1}^t \quad , \quad 0 \le i \le 15, \\ s_{16}^{t+1} &= S(s_0^t \oplus s_{16}^t \oplus k^t), \end{aligned}$$

where the byte k^t is the output of the key register K at time t (that is, $k[p2]$), and S represents the AES S-Box.

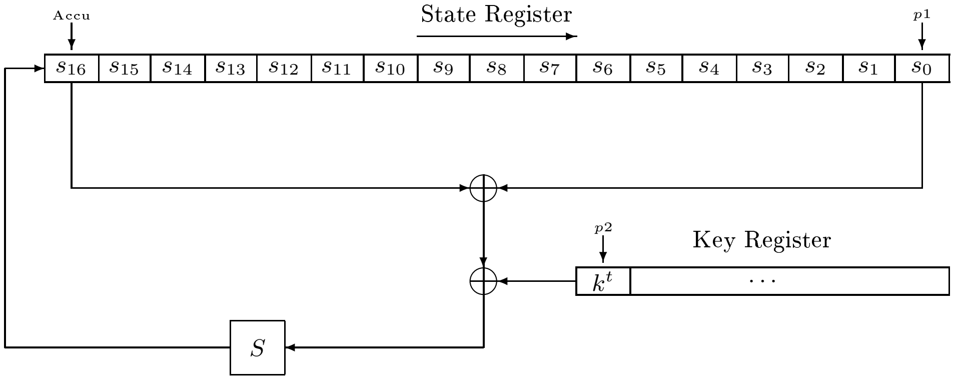

Figure2. Hermes8F as a feedback shift register.

In our attack, we need to consider the reverse cipher (clocking the generator backwards, and so generating the keystream blocks in reverse order[2]). The

[2] As pointed out by one of the anonymous referees, the backward keystream was also used in the attack described in [5].

relation of the feedback register of the reverse cipher is given by

$$\begin{aligned} s_0^t &= S^{-1}(s_{16}^{t+1}) \oplus s_{16}^t \oplus k^t \\ &= S^{-1}(s_{16}^{t+1}) \oplus s_{15}^{t+1} \oplus k^t. \end{aligned}$$

The inverse cipher is depicted in Figure 3.

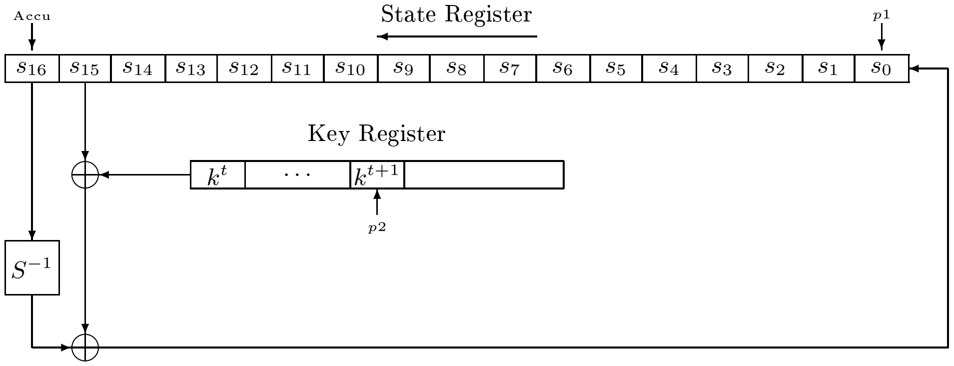

Figure3. The inverse of Hermes8F.

3 Cryptanalysis of Hermes8F

The attack we describe exploits two features of Hermes8F:

1. In contrast to the forward cipher, the reverse cipher has slow diffusion. (In the forward cipher, the new byte s_{16} contributes to the feedback in the very next clock. But in the reverse cipher, the new byte s_0 has no influence on the feedback until it has shifted all the way along to the s_{15} position.)

2. The IV does not affect the key register.

Let us consider the keystream produced by Hermes8F under a secret key and a random IV, and let B_j be the j^{th} set of 8 bytes output by the cipher. Thus, if we define $T = 34 \cdot j + 85$, we have

$$B_j = [s_0^T, s_2^T, s_4^T, s_6^T, s_8^T, s_{10}^T, s_{12}^T, s_{14}^T].$$

Consider the first two sets of B_1 and B_2, for which T is equal to $7 \times 17 = 119$ and $9 \times 17 = 153$ respectively. Suppose that in addition to the last two bytes of B_2 (that is, s_{12}^{153} and s_{14}^{153}), we also know the values of s_{13}^{153}, k^{150} and k^{149}. Then we have

$$S^{-1}(s_{14}^{153}) \oplus s_{13}^{153} \oplus k^{150} = S^{-1}(s_{16}^{151}) \oplus s_{15}^{151} \oplus k^{150} = s_0^{150}.$$

Likewise, we have that

$$S^{-1}(s_{13}^{153}) \oplus s_{12}^{153} \oplus k^{149} = S^{-1}(s_{16}^{150}) \oplus s_{15}^{150} \oplus k^{149} = s_0^{149}.$$

Now, assuming that we also know k^{133}, we have

$$S^{-1}(s_0^{150}) \oplus s_0^{149} \oplus k^{133} = S^{-1}(s_{16}^{134}) \oplus s_{15}^{134} \oplus k^{133} = s_0^{133} = s_{14}^{119}.$$

We note however that s_{14}^{119} is the last byte of B_1.

Thus consider an attack where we guess on the values of k^{133}, k^{149} and k^{150} and verify against the known byte s_{14}^{119}. The equation we have is

$$S^{-1}(S^{-1}(s_{14}^{153}) \oplus s_{13}^{153} \oplus k^{150}) \oplus S^{-1}(s_{13}^{153}) \oplus s_{12}^{153} \oplus k^{149} \oplus k^{133} = s_{14}^{119}, \qquad (1)$$

where the key bytes and s_{13}^{153} are unknown. By setting $c_1 = S^{-1}(s_{14}^{153}) \oplus k^{150}$ and $c_2 = s_{12}^{153} \oplus s_{14}^{119} \oplus k^{149} \oplus k^{133}$ the equation can be more simply written as

$$S^{-1}(s_{13}^{153} \oplus c_1) \oplus S^{-1}(s_{13}^{153}) = c_2. \qquad (2)$$

That is, a particular guess of the three key bytes is possible if and only if an input difference of c_1 to S^{-1} can lead to an output difference of c_2. We know that S^{-1} is affinely equivalent to the inverse mapping in GF(2^8), and thus it is rather close to being APN [9]. This means that just under one half of all (c_1, c_2)-values are possible, or equivalently that one half of the guesses of the three key bytes remains as possible after checking them against (1).

Note that since c_2 depends on the sum $k^{149} \oplus k^{133}$ we can never learn the individual values of k^{149} and k^{133} this way, only the sum of them. Hence we are not guessing on 3-byte values but only on 2-byte values, and the complexity of guessing once is 2^{16} and not 2^{24}. By repeating the guessing for several IVs we can remove all wrong guesses, and find two bytes of information - the values of k^{150} and $k^{149} \oplus k^{133}$.

The process above can be repeated using the output bytes s_{12}^{153} and s_{10}^{153} to obtain k^{148} and $k^{147} \oplus k^{131}$, and so on, until we have 14 bytes of information about the key register at times $121 \le t \le 150$. For Hermes8F-80 it is then not too hard to find the contents of the key register at a specific time t, and we can run the key register back to obtain the original encryption key.

The key register in Hermes8F-128 is 16 bytes long, and so after finding the 14 bytes of information we will need to guess the values of the two remaining bytes. For each of these guesses we run the key register back to the point where the encryption key was loaded to obtain a candidate for the encryption key. This key is then used for generating a keystream with a known IV, and we can verify our guess against the corresponding known keystream.

The attack requires no more than 16 bytes of output under a few (about 16) distinct IVs. In general, the complexity of the attack is of the order of $7 \times 16 \times 2^{16} < 2^{23}$ very simple operations for Hermes8F-80 and $7 \times 16 \times 2^{16} + 2^{16} < 2^{23}$ for Hermes8F-128. The attack for Hermes8F-80 has been implemented on a normal workstation and succeeds in recovering the key in less than a second.

3.1 Analysis of Hermes8

We have considered extending the attack presented above to the original Hermes8 cipher. The main differences between Hermes8 and Hermes8F are the length of the state register (23 bytes and 37 bytes for Hermes8-80 and Hermes8-128, respectively, against 17 bytes for Hermes8F), and the number of rounds between each output of the cipher (three rounds for Hermes8 against two rounds for Hermes8F). Some of the features that we have exploited in our attack, such as the simpler representation of the generator as a shift register, slow diffusion of the reverse clocking cipher, and the fact that the key register is not IV-dependent, apply also to Hermes8. The main difficulty in extending the attack to Hermes8 is the number of rounds between output of the cipher. With three full rounds in Hermes8 between each output, the relations obtained contain a larger number of unknown key and state register bytes. As the state register values are expected to be different for each IV used, we have not been able to obtain a simple equation such as (2) to derive key bits. Therefore a simple extension of the attack does not seem to work against Hermes8. We note however that the increase in the length of the state register alone would in no way have strengthened the cipher against our attack.

4 Equivalent Keys in Hermes8

The key schedule for Hermes8 is described in detail in [6] and is illustrated in Figure 4 (Hermes8F features a similar key scheduling method [7]). The cipher's designer presents a brief analysis of the key schedule and remarks the existence of weak keys for Hermes8. More precisely, keys with equal byte patterns lead to a repetition of byte values in the output of the key scheduling method [6]. In an extreme case, the key defined as $k_i = 63_{\text{hex}}$, for $0 \leq i \leq 9$, is invariant by the key schedule, and it therefore always outputs the byte value 63_{hex} (this follows from the fact that $S(00_{\text{hex}}) = 63_{\text{hex}}$).

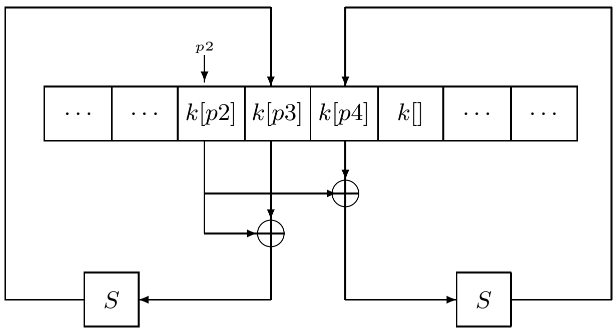

Figure4. Hermes8 key schedule.

A further property of the Hermes8 key schedule that seemed to have been overlooked by the designer is the existence of equivalent keys. These are keys that for a given IV result in the same keystream. This is an immediate consequence of the structure of the key scheduling method and the key-dependent initialization of the pointers *p1*, *p2*, *src*, and the *Accu* register [6].

Consider a key k^*, which results from the byte-wise rotation of the key k. In order to get the same keystream we have to ensure that for both keys, the pointers *p1*, *src*, and the register *Accu* have the same value, that is $p1_k = p1_{k^*}$, $src_k = src_{k^*}$, and $Accu_k = Accu_{k^*}$. Additionally, we require that the pointers $p2_k$ and $p2_{k^*}$ address the key register in such a way that the key scheduling method produces the same output for both keys. For instance, consider the 80-bit version of Hermes8 and assume the 10-byte cipher key is given by $k = k_0, \ldots, k_9$. The rotated key $k^* = k_9, k_0, \ldots, k_8$ is *equivalent* to k if the following conditions are satisfied:

$$\begin{aligned}
\text{cond. } p1 : &\quad (k_0 \oplus k_1 \oplus k_2) \bmod 23 = (k_0 \oplus k_1 \oplus k_9) \bmod 23 &\quad (3)\\
\text{cond. } src : &\quad (k_0 \oplus k_3 \oplus k_9) \bmod 7 \ = (k_2 \oplus k_8 \oplus k_9) \bmod 7 &\quad (4)\\
\text{cond. } Accu : &\quad k_6 \oplus k_7 \oplus k_8 = k_5 \oplus k_6 \oplus k_7 &\quad (5)\\
\text{cond. } p2 : &\quad (k_2 \oplus k_3 \oplus k_4) \bmod 10 = ((k_3 \oplus k_4 \oplus k_5) \bmod 10) + 1 &\quad (6)
\end{aligned}$$

Condition (6) ensures that the output of the key schedule is the same for k and k^*. If, in addition, the remaining conditions (3)-(5) are satisfied, then the key stream generation is equivalent for both keys k and k^*. There are approximately

$$2^{80-(8-log_2(\lceil\frac{256}{23}\rceil))-(8-log_2(\lceil\frac{256}{7}\rceil))-8-(8-log_2(\lceil\frac{256}{10}\rceil)+log_2(1.109))} \approx 2^{61}$$

keys k satisfying the conditions above, which are therefore essentially equivalent to the key k^* obtained by a simple cyclic shift of its bytes. A similar analysis can be done for other rotation values of the key k, giving us approximately $5 \times 2^{61} \approx 2^{63}$ pairs of equivalent keys. Although this represents a very small fraction of an 80-bit key space, the above argument shows however that Hermes8-80 does not reach the theoretically expected entire 80-bit key space. In fact, if we assume that 80-bit encryption keys are randomly generated, we have that approximately 2^{63} keys effectively occur with twice the expected probability, while 2^{63} keys do not occur at all.

5 Algebraic Structure

Given the highly algebraic structure of Hermes8, it is natural to consider the feasibility of algebraic attacks against the cipher. The only two operations in Hermes8 are the S-Box operation (which is based on the inversion over GF(2^8)) and XOR. Thus at each clocking, we can express the resulting register updated through a relation over GF(2^8) (which in turn can be described as a set of multivariate quadratic equations over GF(2)). After a number of rounds we should have enough equations to solve the system of equations and therefore recover the secret key. In our estimates however the size of the resulting system appears to be too large to be solved in practice. This is due to the large number of clockings between the cipher output. However it may be possible that one can simplify some of the relations, or exploit this rich algebraic structure in some other way.

We note that the attack presented in section 3 can also be mounted using a more algebraic approach. Due to the algebraic structure of the S-Box, the expressions considered when describing the attack can also be written as a simple system of multivariate equations. If we solve the system (e.g. by computing the corresponding Gröbner basis under the appropriate monomial ordering), requiring that the equations have solutions in $\mathrm{GF}(2^8)$, we obtain relations between the key bytes. This corresponds to the bit of information we derived from the relation (2). If we repeat this procedure for a number of IVs, we should obtain enough such relations to allow us to solve the resulting system and recover the respective key bytes. Again, this approach does not seem to work with Hermes8, as we have not been able to obtain relations on the key bytes alone (they always involve at least one unknown register value, which as noted in section 3.1, should change with each different IV). Moreover, this algebraic approach does not seem to be more efficient than the attack described early in this paper.

5.1 Algebraic Structure of a Variant of Hermes8

In this section we consider a slightly modified version of Hermes8, to illustrate how its highly algebraic structure may be exploited. In this modified version, we remove the final affine transformation from the Sbox, so that the variant uses as S-Box the modified *inversion* in the Rijndael field only, that is $S : x \mapsto x^{254}$. We note that the only two operations of the cipher (SBox and XOR) correspond to the exponentiation and addition in the Rijndael field $\mathbb{F} \cong \mathrm{GF}(2^8)$, respectively. We also know that the original AES S-Box is affinely equivalent to the inversion, and so this variant of Hermes8 should share much of the security properties with the original Hermes8 cipher.

However the new cipher presents a very interesting property. Let $\tau : \mathbb{F} \to \mathbb{K}$ be any isomorphism from $\mathbb{F}$ to a field $\mathbb{K} \cong \mathrm{GF}(2^8)$ (in particular, we may have $\mathbb{K} = \mathbb{F}$ so that τ is an automorphism of $\mathbb{F}$). Then we have

$$S(\tau(x)) = \tau(S(x)) \text{ and } \tau(x \oplus y) = \tau(x) \oplus \tau(y), \ \forall x, y \in \mathbb{F}.$$

If we assume the simplified version of initialization of the cipher's pointers (as with Hermes8F), we can then use these relations to construct a very simple chosen-key algebraic distinguisher against the cipher. Let $KS = \mathcal{E}(k, IV)$ represent the keystream (of length m) generated by the cipher using initialisation vector IV and encryption key k. Then we have

$$\mathcal{E}(\tau(k), \tau(IV)) = \tau(KS),$$

where $\tau(k)$ denotes the application of τ on each byte of the encryption key k (similar for $\tau(IV)$ and $\tau(KS)$).

This property is called *self-duality* [1], and is similar to the complementation property of DES [8]. In particular, it allows us to construct a simple method that reduces the key space when performing exhaustive key search, as following.

Let k be the secret encryption key to be searched, so that an attacker has access to the encryption operation $\mathcal{E}(k, \cdot)$, and can generate the keystream for any IV. Let τ be an automorphism of $\mathbb{F}$.

Prior to performing the exhaustive search, the attacker partitions the key space into equivalence classes

$$k_1 \equiv k_2 \iff k_2 = \tau^r(k_1),$$

and given an IV, computes the set of initialisation vectors

$$\{IV, \tau(IV), \tau^2(IV), \ldots, \tau^{n-1}(IV)\},$$

where n is the order of τ. It can now compute the set of keystreams of length m (for m long enough)

$$KS_i = \tau^{-i}(\mathcal{E}(\tau^i(IV), k)) = \mathcal{E}(IV, \tau^{-i}(k))$$

for $i = 0, \ldots, n-1$.

To perform the exhaustive key search, for each equivalence class of encryption keys, the attacker selects a key k' and computes the keystream of length m $K = \mathcal{E}(IV, k')$. If $K = KS_i$ for some i, then $\tau^i(k')$ is a candidate for the encryption key k. Otherwise k is not in the equivalence class of k'. This method should reduce the complexity of exhaustive key search by a factor of about n, and is similar to the method that exploits the complementation property of DES (which uses the complementation map of order 2).

For a concrete example, let us consider the Frobenius automorphism defined as $\tau : x \mapsto x^2$. Since the order of τ is 8, this method should reduce the complexity of exhaustive key search to the order of 2^{77} operations (enabling key recovery on average in the order of 2^{76} operations). From the many isomorphisms of fields of order 2^8 [10], this map seems to provide the best reduction for the key space search.

We note however that this property and method of attack does not apply to the original Hermes8 cipher, since the affine operation in the SBox does not commute with the field isomorphisms.

6 Conclusion

We presented in this paper an analysis of the Hermes8 [6] stream cipher, and some of its variants. In particular, we showed how to mount an attack to recover the secret key for the latest version of the cipher (Hermes8F-80) with complexity of around the order of 2^{23} operations, requiring a very small number of known keystream bytes. Although we have not been able to extend the method of attack used to the original version of Hermes8, we note that many of the features that we have exploited - the simpler representation of the generator as a shift register, slow diffusion of the reverse clocking cipher, and the fact that the key register is not IV-dependent - apply also to Hermes8. An interesting topic for further research is whether there are other stream ciphers that may have their security compromised by analysis of the reverse cipher, as with Hermer8F.

Acknowledgments

The work described in this paper has been supported in part by the European Commission through the IST Programme under Contract IST-2002-507932 ECRYPT. We would also like to thank Vincent Rijmen for his suggestion to consider the existence of equivalent keys for the Hermes8 stream ciphers.

References

1. E. Barkan and E. Biham. In How Many Ways Can You Write Rijndael?. *Cryptology ePrint Archive* 2002/157, 2002. `http://eprint.iacr.org/2002/157/`.
2. J. Daemen and V. Rijmen. *The Design of Rijndael.* Springer–Verlag, 2002.
3. eSTREAM, the ECRYPT Stream Cipher Project. `http://www.ecrypt.eu.org/stream/`.
4. C. De Cannière. eSTREAM testing framework. `http://www.ecrypt.eu.org/stream/perf/`.
5. J. Golic. Iterative Probabilistic Cryptanalysis of RC4 Keystream Generator. In E. Dawson, A. Clark and C. Boyd, editors, *Information Security and Privacy, 5th Australasian Conference, ACISP 2000*, volume 1841 of *LNCS*, pages 220–233. Springer–Verlag, 2000.
6. U. Kaiser. Hermes8 : A Low-Complexity Low-Power Stream Cipher. Cryptology ePrint Archive, Report 2006/019. `http://eprint.iacr.org/2006/019.pdf`.
7. U. Kaiser. Hermes8F : A Low-Complexity Low-Power Stream Cipher. eSTREAM, the ECRYPT Stream Cipher Project, Second Phase Ciphers. `http://www.ecrypt.eu.org/stream/p2ciphers/hermes8/hermes8f_p2.pdf`.
8. A.J. Menezes, P.C. Van Oorschot, and S.A. Vanstone. *Handbook of Applied Cryptography.* CRC Press, 1996.
9. K. Nyberg. Diferentially uniform mappings for cryptography, *Advances in Cryptography, EUROCRYPT'93*, LNCS 765, pp. 55–64, Springer–verlag, 1994.
10. H. Raddum. More Dual Rijndaels. In H. Dobbertin, V. Rijmen, and A. Sowa, editors, *Advanced Encryption Standard - AES, Fourth International Conference*, volume 3373 of *LNCS*, pages 142–147. Springer–Verlag, 2005.