The MAC function Pelican 2.0

Joan Daemen[1] and Vincent Rijmen[2]

[1] STMicroelectronics Belgium
joan.daemen@st.com
[2] KU Leuven & iMinds (Belgium)
vincent.rijmen@esat.kuleuven.be

Abstract. We present an update of the Pelican MAC function, called Pelican 2.0. Both versions have the Alred construction [5] and are based on Rijndael. they are a factor 2.5 more efficient than CBC-MAC with Rijndael, while providing a comparable claimed security level.
The difference between Pelican 2.0 and the original version is that the initial value changes from the all-zero string to another constant. The reason for this is the negative impact on security if key check values are available computed with a certain standard key check value algorithm that applies the block cipher to the zero string and takes as key check value its truncated output. The security impact of this on a number of standard MACs is studied in [11], and the analysis carries over for Pelican.

1 Introduction

Message Authentication Codes (MAC functions) are symmetric primitives, used to ensure authenticity of messages. They take as input a secret key and the message, and produce as output a short tag. The basic property of a MAC function is that it provides an unpredictable mapping between messages and the tag for someone who does not know, or only partially knows the key.

We propose here a new version of the Pelican MAC function [6], called Pelican 2.0. We will indicate the first version of Pelican with the suffix 1.0 and use Pelican without suffix in statements that apply to both versions.

The design of Pelican is based on the Alred construction, which was presented in [5]. Pelican 1.0 can be seen as a simplification of Alpha-MAC, the concrete design presented in [5]. In a follow-up paper [7], we refined the security claims for iterative MAC functions that we proposed in [5] and did some additional analysis on internal collisions in the Alred structure.

Pelican is based on Rijndael [4]. We have chosen for Rijndael mainly because it is widely available thanks to its status as the AES standard. Additionally, Rijndael is efficient in hardware and software and it has withstood intense public scrutiny very well since its publication [8].

We specify Pelican 2.0 in Section 2. In Section 3, we briefly repeat the security claims introduced in [5], and in Section 5 we recall the provable security properties of the Alred construction, that apply to Pelican. In Section 4 we motivate the change between PELICAN 1.0 and 2.0 and in Section 6 we briefly summarize the state-of-the-art of cryptanalysis of PELICAN. We discuss performance in Section 7 and conclude in Section 8.

2 Specification

PELICAN is an ALRED construction with RIJNDAEL [4], restricted to a block length of 128 bits, as underlying block cipher. As RIJNDAEL, PELICAN supports keys of 16, 20, 24, 28 and 32 bytes. For the key lengths 16, 24 and 32 bytes, RIJNDAEL coincides with AES [12]. PELICAN can take a message m of any length and generates tags with length up to 128 bits.

In the PELICAN 2.0 algorithm we can distinguish a number of stages. First the message is padded and split in message words. These message words are applied to a *state* that is initialized using the key and that afterwards undergoes a final step again using the key. More exactly:

Message padding and splitting: pad the message by appending a single 1 bit followed by the minimum number of 0 bits so that the resulting length is a multiple of 128 bits (padding method 2 in [10]). Split the result in 128-bit *message words* $x_1, x_2, \dots x_q$.

State initialization: fill the 128-bit *state* with the initialisation vector IV and subsequently apply the Rijndael block cipher to it using the key.

$$\mathrm{IV} = \begin{bmatrix} \text{0x00 0x00 0x00 0x00} \\ \text{0x01 0x01 0x00 0x01} \\ \text{0x01 0x00 0x01 0x00} \\ \text{0x01 0x01 0x01 0x00} \end{bmatrix}$$

Chaining: XOR the first message word x_1 to the state. For each additional message word x_i, apply to the state the iteration function and then XOR the message word x_i to the state. The iteration function consists of four RIJNDAEL rounds with round keys set to 0.

Finalization: Apply RIJNDAEL encryption to the state using the key and form the tag by taking the first ℓ_m bits of the state.

PELICAN 2.0 is illustrated in Figure 1.

3 Security claims

PELICAN is an iterative MAC function. Iterative MAC functions have the disadvantage that different messages may be found that lead to the same value of the state during the MAC generation process. This is called an *internal collision* [13]. We have proposed in [5] a set of three orthogonal security claims for iterative MAC functions that reflect this limitation and used these to express the security claim for PELICAN 1.0. Later we found a problem with the claim related to internal collisions that was expressed in terms of number of messages.

In our paper [7] we addressed this by proposing a claim expressed in terms of total number of blocks in the messages. We base the security claim of PELICAN 2.0 on this new formulation. For clarity, we repeat the claims here.

The claims are formulated in terms of three dimension parameters: the tag length ℓ_m, the key length ℓ_k and the *capacity* ℓ_c. The capacity is the size of the internal memory of a random map with the same probability for internal collisions as the MAC function and the designer must fix its value in the security claim.

For PELICAN 1.0 we based the value of the capacity in our security claims on a preliminary analysis of generating internal collisions, based on the theory presented in [5]. We made our rationale more explicit in our paper [7]. This rationale remains valid for PELICAN 2.0.

Claim 1 *The probability of success of any forgery attack not involving key recovery or internal collisions is* $2^{-\ell_m}$.

Claim 2 *There are no key recovery attacks faster than exhaustive key search, i.e. with an expected complexity less than* $2^{\ell_k - 1}$ *MAC function executions.*

Claim 3 *The probability that an internal collision occurs in a set of adaptively chosen messages presented to the MAC function is not above* $1 - \exp(-A^2/2^{\ell_c+1})$, *with* A *the total number of blocks in the messages.*

PELICAN 2.0 should satisfy these three claims for the following range of values of the dimension parameters:

– Tag length ℓ_m: any value up to 128 bits.
– Key length ℓ_k: 128, 160, 192, 224 and 256 bits.
– Capacity ℓ_c: 120 bits

4 Motivation of the change of initial value

ANSI X9.24-1:2009 [1] specifies the manual and automated management of keying material used for financial services. This includes the suggestion to compute a *key check value* (KCV) on a block cipher key K. The method proposed is to apply the associated block cipher to an all-zero block under the key K and take the *s most significant bits* of the result as KCV, with s typically 24. The KCV can then be used to verify the integrity of the key and is treated as a public value. Unfortunately, this method has been widely applied in a.o. hardware security modules (HSM).

In [11], the authors study the impact on the security of MAC algorithms specified in ISO/IEC 9797-1 [10] if the KCV of the used key is known. It turns out that for the majority of them there is a significant impact on the security.

Clearly, for PELICAN 1.0, the KCV simply gives away s bits of the chaining value, the secrecy of which is critical for its security. Changing the IV from the all-zero block to any other value solves this problem and so this is what we did for PELICAN 2.0.

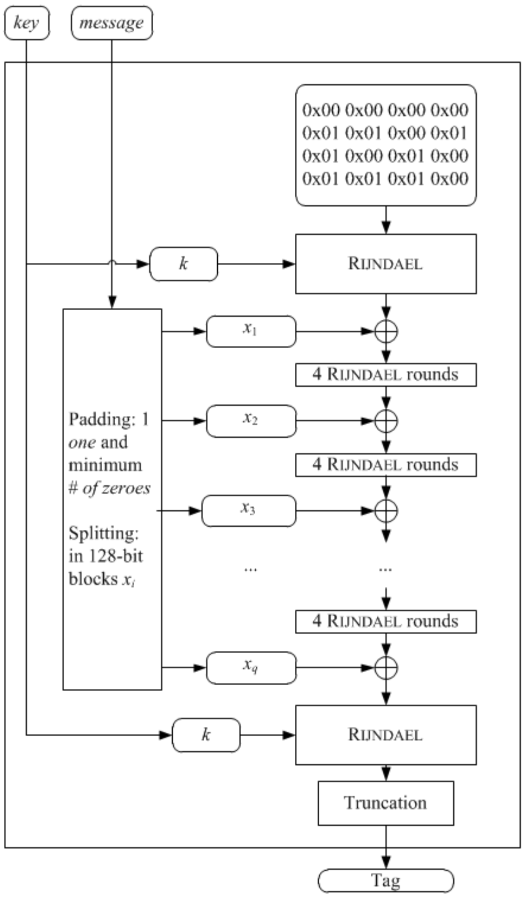

Fig. 1. Block scheme of PELICAN 2.0.

5 Provability

PELICAN has the ALRED structure [5]. For this structure, the security with respect to forgery in the absence of collisions and with respect to key recovery can be provably reduced to security properties of the used block cipher. This results in the following theorems for PELICAN.

Theorem 1. *Every key recovery attack on* PELICAN, *requiring t (adaptively) chosen messages, can be converted to a key recovery attack on* RIJNDAEL, *requiring $t + 1$ adaptively chosen plaintexts.*

Theorem 2. *Every forgery attack on* PELICAN *not involving internal collisions, requiring t (adaptively) chosen messages, can be converted to a ciphertext guessing attack on* RIJNDAEL, *requiring $t + 1$ adaptively chosen plaintexts.*

For the proofs of these theorems we refer to [5].

6 State of cryptanalysis of PELICAN

PELICAN has received quite some cryptanalysis since it was first published [2,3, 9,14,15]. All this cryptanalysis concentrates on retrieving the state value during the chaining stage.

Recovery of the value of the state during the chaining stage is not equivalent to key recovery. Still, as observed in a.o. [2], its secrecy is essential for the security of PELICAN. An attacker who can find out its value for a given key and input message, can perform forgeries at the expense of a single *correcting block* of size 128 bits, where she can even choose the position of this block in the message.

At the time of writing, all known cryptanalytic methods for recovering this state value require internal collisions. They make use of the specific properties of the RIJNDAEL round function, including its symmetry properties in the absence of round keys (or constants). Upon inspection, it turns out that none of these attacks violate the PELICAN security claims. In fact, none of them allows generating inner collisions in PELICAN with a higher success probability than generic methods. In other words, with respect to the published attacks, even a claim with capacity $\ell_c = 128$ would hold up and with respect to the claimed capacity there is still a safety margin of 8 bits.

Some of the published attacks can be countered by applying round constants or sub-keys in the chaining phase. However, as the attacks do not compromise the security claims, we have decided to honor the motto “if it ain’t broke, don’t fix it”. Implementations *shall* take special precautions to protect the secrecy of the internal state against applicable side channel and fault attacks.

7 Performance

In this section we express the performance of PELICAN in terms of RIJNDAEL operations, more particularly, the RIJNDAEL key schedule and the RIJNDAEL encryption operation. This allows to use Rijndael (or AES) benchmarks for software implementations on many platform and hardware implementations to get a pretty good idea on the performance of Pelican. We restrict ourselves to the case of 128-bit keys.

One iteration of Pelican corresponds roughly to 4 rounds of Rijndael, hence 2/5 of a Rijndael encryption. It is actually better because the XORs with 0 in the round key addition don't have to be implemented. Some implementations of Rijndael recompute the round keys for every encryption. This overhead is not present in Pelican. As the first message word is simply XORed without additional rounds, the message processing of the first message word must be subtracted from the message processing. Using these rough approximations, we can state that MACing a message requires:

setup: 1 Rijndael key schedule and 1 Rijndael encryption,
message processing: 0.4 Rijndael encryptions per 16-byte message word,
finalization: 0.6 Rijndael encryptions.

Hence, for long messages the computational workload of Pelican is only 40 % of Rijndael encryption. For short messages, the minimum cost is 1 Rijndael encryption for generating a tag and 1 additional Rijndael encryption and key schedule at key setup.

On many platforms this can speed up MAC computation by a factor up to 2.5. Like any other serial mode of operation, Pelican gets only a limited performance improvement from the AES-NI instructions present on modern processors. This is due to the long latency of these instructions. Exploiting the pipelining on such platforms is possible only with parallelizable MAC constructions. Note however that Pelican is still 2.5 times faster than CBC-MAC with Rijndael.

8 Conclusions

We presented Pelican 2.0, an update of the Pelican MAC function that is based on the Alred construction. The difference with its predecessor is the change in initial value to avoid negative interference with a popular key check value method. The Alred construction comes with some security proofs, which are also applicable to this primitive.

References

1. ANSI, *X9.24: Retail financial services symmetric key management part 1: Using symmetric techniques*, 2009.
2. Alex Biryukov, Andrey Bogdanov, Dmitry Khovratovich, and Timo Kasper, *Collision attacks on AES-based MAC: Alpha-MAC*, CHES (Pascal Paillier and Ingrid Verbauwhede, eds.), Lecture Notes in Computer Science, vol. 4727, Springer, 2007, pp. 166–180.
3. Charles Bouillaguet, Patrick Derbez, and Pierre-Alain Fouque, *Automatic search of attacks on round-reduced aes and applications*, CRYPTO (Phillip Rogaway, ed.), Lecture Notes in Computer Science, vol. 6841, Springer, 2011, pp. 169–187.

4. J. Daemen and V. Rijmen, *The design of Rijndael — AES, the advanced encryption standard*, Springer-Verlag, 2002.
5. ______, *A new MAC construction ALRED and a specific instance ALPHA-MAC*, Fast Software Encryption (Henri Gilbert and Helena Handschuh, eds.), Lecture Notes in Computer Science, vol. 3557, Springer, 2005, pp. 1–17.
6. ______, *The Pelican MAC function*, IACR Cryptology ePrint Archive **2005** (2005), 8 p.
7. ______, *Refinements of the ALRED construction and MAC security claims*, IET information security **4** (2010), 149–157.
8. Joan Daemen and Vincent Rijmen, *AES proposal:* RIJNDAEL, AES submission, 1999, http://jda.noekeon.org/.
9. Orr Dunkelman, Nathan Keller, and Adi Shamir, *ALRED blues: New attacks on AES-based MAC's*, Cryptology ePrint Archive, Report 2011/095, 2011, http://eprint.iacr.org/.
10. ISO/IEC, *9797-1: Information technology - security techniques - message authentication codes (macs) - part 1: Mechanisms using a block cipher*, 2011.
11. Tetsu Iwata and Lei Wang, *Impact of ANSI X9.24-1:2009 key check value on ISO/IEC 9797-1:2011 MACs*, Cryptology ePrint Archive, Report 2014/183, 2014, http://eprint.iacr.org/.
12. NIST, *Federal information processing standard 197, advanced encryption standard (AES)*, November 2001.
13. Bart Preneel and Paul C. van Oorschot, *On the security of iterated message authentication codes*, IEEE Transactions on Information Theory **45** (1999), no. 1, 188–199.
14. Wei Wang, Xiaoyun Wang, and Guangwu Xu, *Impossible differential cryptanalysis of Pelican, MT-MAC-AES and PC-MAC-AES*, IACR Cryptology ePrint Archive **2009** (2009), 5.
15. Zheng Yuan, Keting Jia, Wei Wang, and Xiaoyun Wang, *Distinguishing and forgery attacks on Alred and its AES-based instance Alpha-MAC*, Cryptology ePrint Archive, Report 2008/516, 2008, http://eprint.iacr.org/.