Tag Second-preimage Attack against π-cipher

Gaëtan Leurent

Tag Second-preimage Attack against π-cipher

Gaëtan Leurent

Inria, France*

Abstract. The π-cipher is one of the candidates of the CAESAR competition. One of the advertised features of the π-cipher is tag second-preimage resistance: it should be hard to generate a message with a given tag, even for the legitimate key holder (insider attack).

In this note, we show that the generalized birthday attack of Wagner gives a practical tag second-preimage attack against the π-cipher.

1 Introduction

The π-cipher [2] is an authenticated encryption algorithm submitted to the CAESAR competition. One of the extra features advertised by the designers is tag second-preimage resistance: it should be hard to produce second-preimages of a given tag, even for an adversary who knows the secret key (most authenticated encryption algorithm do not have this feature, and an insider can easily generate tag second-preimages).

As written in [2, 4.1], the tag generation of an m-block message with the π-cipher can be written as:

$$T = T'' \boxplus_8 e(1, M_1) \boxplus_8 e(2, M_2) \boxplus_8 \cdots \boxplus_8 e(m, M_m)$$

where e denotes a keyed function known to the key holder (the e-triplex), $\boxplus_8$ is a component-wise addition of vectors of 8 elements in $\mathbb{Z}_{2^\omega}$, and T'' is the associated data tag (known to the insider). The word-size ω is 16, 32, or 64, depending on the security level. In a tag second-preimage attack, an insider wants to build a message M reaching a fixed tag $\bar{T}$. Witout loss of generality, we assume $T'' = 0$ and $\bar{T} = 0$.

In the submission document of π-cipher, the tag second-preimage problem is seen as a knapsack problem, and the main attack considered is a variant of an attack by Camion and Patarin [1]. However, the generalization of this attack due to Wagner [3] can break the problem more efficiently.

2 Wagner's Generalized Birthday Attack

The generalized birthday attack of Wagner is an attack against the m-sum problem: given m lists $L_1, L_2, \ldots, L_m$ of n-bit words, one find values $l_1 \in L_1, \ldots, l_m \in L_m$ such that $\bigoplus_{i=1}^{m} l_m = 0$. If each list contains at least $2^{n/m}$ elements there is a good probability that a solution exists, but the best known algorithm is a simple birthday attack in time and memory $\widetilde{\mathcal{O}}(2^{n/2})$. One would first build two lists L_A and L_B with all the sums of elements in $L_1, ...L_{m/2}$ and $L_{m/2+1}, ...L_m$ respectively, then sort L_A and L_B, and look for a match between the two lists (L_A and L_B contain $2^{n/2}$ elements each).

* SECRET project-team, `Gaetan.Leurent@inria.fr`

Wagner's algorithm has a lower complexity, but it requires more elements in the lists. For instance, with $m = 4$, it uses lists of size $2^{n/3}$ in order to find one solution using $\widetilde{\mathcal{O}}(2^{n/3})$ time and memory. The basic operation of the algorithm is the general join $\bowtie_\tau$: $L \bowtie_\tau L'$ consists of all the elements of $L \times L'$ that agree on their τ least significant bits. More precisely, the operation can be defined over list of values with associated data:

$$L \bowtie_\tau L' = \left\{ \left(l \oplus l', (a, a') \right) \,\middle|\, (l, a) \in L, (l', a') \in L', \mathsf{low}_\tau(l \oplus l') = 0 \right\}.$$

The join operation is computed efficiently by sorting the lists L and L' according to the lower τ bits, and stepping through the lists simultaneously in order to find values that agree on their low bits. Moreover, the sorting can be done in linear time using a hash table, or a radix sort.

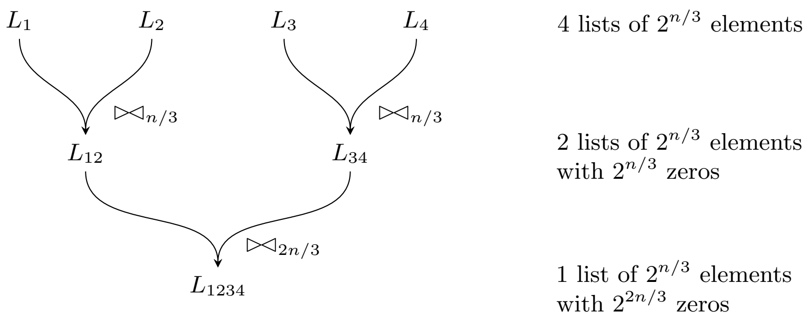

Fig. 1. Wagner's algorithm for $m = 4$

The generalized birthday algorithm for $m = 4$ is described by Figure 1. We first build the lists $L_{12} = L_1 \bowtie_{n/3} L_2$ and $L_{34} = L_3 \bowtie_{n/3} L_4$, containing about $2^{n/3}$ elements. Next, we build $L_{1234} = L_{12} \bowtie_{2n/3} L_{34}$. Since the elements of L_{12} and L_{34} already agree on their $n/3$ lower bits, we are only matching bits $n/3$ to $2n/3$, so we still expect to find $2^{n/3}$ elements. Finally, we expect one of the elements of L_{1234} to be zero. This can be generalized to any m that is a power of two, using a binary tree: if $m = 2^a$, we need m lists of $2^{n/(a+1)}$ elements and the time and memory used by the algorithm is $2^a \cdot r2^{n/(a+1)}$. The algorithm for $m = 8$ is shown by Figure 2.

3 Application to the π-cipher

In order to apply this attack to the π-cipher, we need to solve the m-sum problem for the word-wise modular addition $\boxplus_8$, instead of the exclusive-or $\oplus$. Wagner showed how to solve the generalized birthday problem with a modular addition, and his trick also works for the word-wise modular addition. More precisely, we have to modify the join operator to:

$$L \blacktriangleright\!\!\blacktriangleleft_\tau L' = \left\{ \left(l \boxplus_8 l', (a, a') \right) \,\middle|\, (l, a) \in L, (l', a') \in L', \mathsf{low}_\tau(l \boxplus_8 l') = 0 \right\}.$$

Since the word-wise modular addition $\boxplus_8$ only has carries from the low order bits to the high order bits, when x and y have their τ low-order bits set to zero, $x \boxplus_8 y$

also has τ low-order bits set to zero. Moreover, the join $\bowtie$ can still be computed efficiently. We first negate the list L and define $-L = \{(-l, a) \mid (l, a) \in L\}$, where $-l$ is the additive inverse with regard to the word-wise addition, *i.e.* $l \boxplus_8 (-l) = 0$. Then we sort $-L$ and L' according to their lower τ bits, and step through the lists in parallel. When an element of $-L$ and an element of L' agree on their low bit, the corresponding sum will have its low bits equal to zero. Therefore, this variant of Wagner's algorithm is suitable for a tag second-preimage attack on the π-cipher.

We give a full description of an attack with $\omega = 16$ in Algorithm 1; this attack uses 8 lists of size 2^{32} (illustrated by Figure 2), *i.e.* we consider an 8-block message, with 2^{32} possibilities for each block. This gives a complexity of 2^{35}. More generally, we can apply Wagner's attack to different versions of π-cipher (*i.e.* with different values of ω), and several trade-offs between the message length and the attack complexity are possible. We give some parameters in Table 1.

Table 1. Attack parameters

	Optimal parameters			Short messages		
ω	m	$\lvert L\rvert$	Complexity	m	$\lvert L\rvert$	Complexity
16	2^{11}	2^{11}	2^{22}	2^{3}	2^{32}	2^{35}
32	2^{16}	2^{15}	2^{31}	2^{7}	2^{32}	2^{39}
64	2^{22}	2^{23}	2^{45}	2^{15}	2^{32}	2^{47}

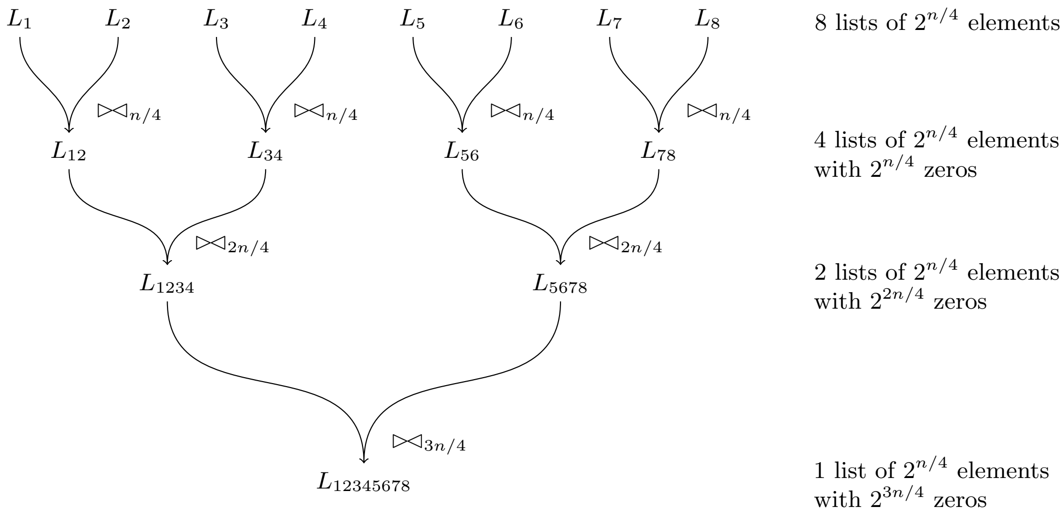

Fig. 2. Wagner's algorithm for $m = 8$

References

1. Camion, P., Patarin, J.: The knapsack hash function proposed at crypto'89 can be broken. In: Davies, D.W. (ed.) EUROCRYPT. Lecture Notes in Computer Science, vol. 547, pp. 39–53. Springer (1991)
2. Gligoroski, D., Mihajloska, H., Samardjiska, S., Jacobsen, H., El-Hadedy, M., Jensen, R.E.: π-Cipher. Submission to CAESAR. Available from: `http://competitions.cr.yp.to/round1/picipherv1.pdf` (v1) (March 2014)
3. Wagner, D.: A generalized birthday problem. In: Yung, M. (ed.) CRYPTO. Lecture Notes in Computer Science, vol. 2442, pp. 288–303. Springer (2002)

Algorithm 1 Short message attack with $\omega = 16$ and $m = 8$.

```
for 0 ≤ i < 8 do
    for 0 ≤ j < 2^32 do
        L[i][j] ← (e(i, [j]), j)
    end for
end for
L[ 8] ← Merge(L[0], L[1], 32)
L[ 9] ← Merge(L[2], L[3], 32)
L[10] ← Merge(L[4], L[5], 32)
L[11] ← Merge(L[6], L[7], 32)
L[12] ← Merge(L[8], L[9], 64)
L[13] ← Merge(L[10], L[11], 64)
L[14] ← Merge(L[12], L[13], 96)
for all (l, (((a_1, a_2), (a_3, a_4)), ((a_5, a_6), (a_7, a_8)))) ∈ L[14] do
    if l = 0 then
        return [a_1] || [a_2] || [a_3] || [a_4] || [a_5] || [a_6] || [a_7] || [a_8]
    end if
end for

function Merge(L, L', τ)
    Sort(L, −low_τ)
    Sort(L', low_τ)
    i ← 0
    j ← 0
    M ← ∅
    while i < |L| and j < |L'| do
        (l, a) ← L[i]
        (l', a') ← L'[j]
        if low_τ(−l) = low_τ(l') then
            M ← M ∪ {(l ⊞_8 l', (a, a'))}
        else if low_τ(−l) < low_τ(l') then
            i ← i + 1
        else
            j ← j + 1
        end if
    end while
    return M
end function
```