Distinguishing and Key-recovery Attacks against Wheesht

Anne Canteaut, Gaëtan Leurent

▶ **To cite this version:**

Anne Canteaut, Gaëtan Leurent. Distinguishing and Key-recovery Attacks against Wheesht. 2014. hal-00966346v1

HAL Id: hal-00966346

https://inria.hal.science/hal-00966346v1

Preprint submitted on 26 Mar 2014 (v1), last revised 26 Mar 2014 (v2)

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Distinguishing and Key-recovery Attacks against Wheesht

Anne Canteaut and Gaëtan Leurent

Inria, France*

Abstract. Wheesht is one of the candidates to the CAESAR competition. In this note we present several attacks on Wheesht, showing that it is far from the advertised security level of 256 bits. In particular we describe a distinguishing attack with $2^{70.3}$ known plaintext words for any number of rounds of Wheesht, and a key-recovery attack (recovering the encryption key) for versions of Wheesht with a single finalization round with very little data and time complexity 2^{192}.

1 Introduction

Wheesht[3] is an authenticated encryption algorithm submitted to the CAESAR competition. The encryption of Wheesht is a stream cipher based on evaluating a keyed function over a counter. More precisely, the main function computes a keystream from

- An encryption key k_i;
- Some constants q_i;
- A public nonce n_i;
- A secret nonce s_i;
- A block counter b_i;
- Some extra parameters p_i;

and is described in Figure 1. We use the notation x_i, y_i, z_i for intermediate values during the computation. All values are 64-bit wide.

2 Generic attacks

We first describe generic attacks based on the high-level structure of Wheesht, as described in Figure 1. They are independent of the number of rounds t_m and t_f, and of the details of the permutation θ.

2.1 Distinguishing attack

Our first attack is a distinguishing attack with complexity around 2^{64}. The main property used in this attack is that many values are fixed for all the blocks (excepted the last one) of a given message: n, s, p are fixed, and only b is incremented from one block to the next. In Figure 1, all the values that are kept constant for a given message are shown with a dash-dotted pattern.

* SECRET project-team, {Anne.Canteaut, Gaetan.Leurent}@inria.fr

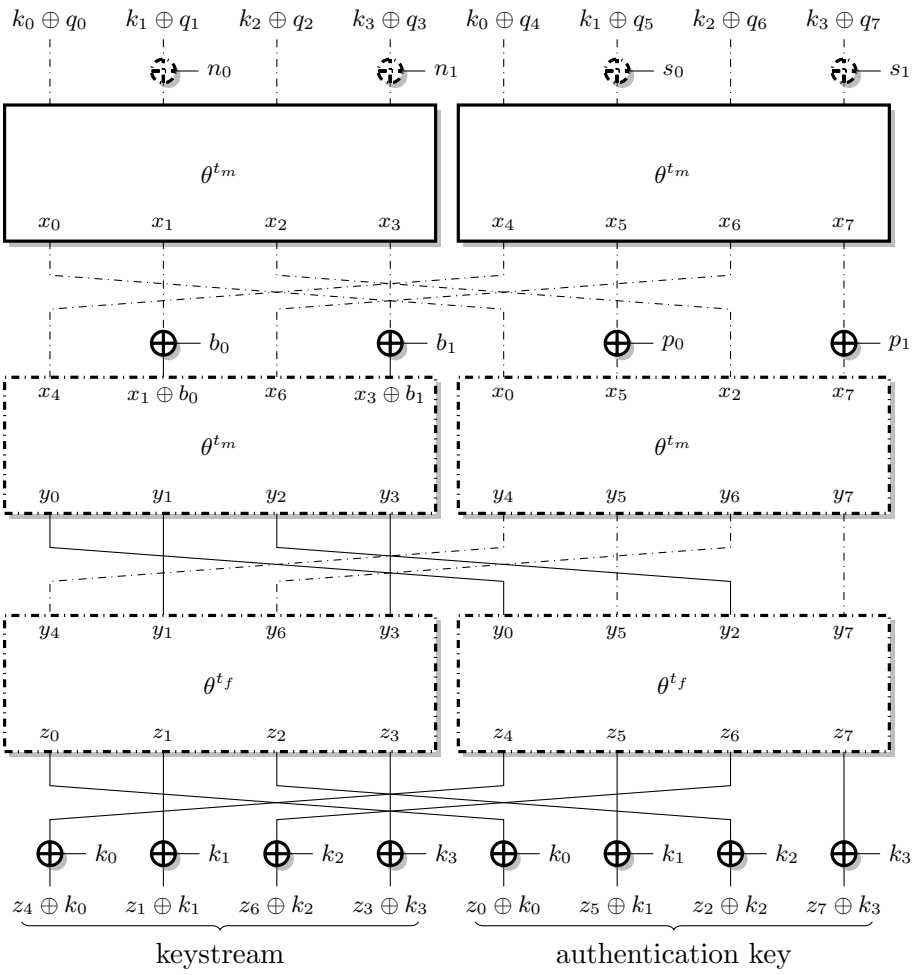

Fig. 1. Wheesht main block. Values that are fixed for all blocks (excepted the last one) of a given message are shown in a dash-dotted pattern.

More precisely, the generation of the keystream can be written as:

$$(y_1, y_3) = \lambda_{x_4,x_1,x_6,x_3}(b_0, b_1) \qquad (y_0, y_2) = \mu_{x_4,x_1,x_6,x_3}(b_0, b_1)$$
$$(z_1, z_3) = \chi_{y_4,y_6}(y_1, y_3) \qquad (z_4, z_6) = \xi_{y_4,y_6}(y_0, y_2)$$

where the functions $\lambda_{x_4,x_1,x_6,x_3}$, μ_{x_4,x_1,x_6,x_3}, χ_{y_4,y_6} and ξ_{y_4,y_6} are fixed for all the blocks of a long message, and are expected to behave like random functions (with 128-bit input and output). In particular, we expect that the image set of any of those functions is of size $(1 - 1/e) \cdot 2^{128}$. The set of possible (z_1, z_3) is the image of the composition of two random functions; following [1, Theorem 2] we expect its size to be about $(1 - \exp(-1 + 1/e)) \cdot 2^{128} \approx 0.47 \cdot 2^{128}$, rather than $(1 - 1/e) \cdot 2^{128} \approx 0.63 \cdot 2^{128}$ for a set of 2^{128} 128-bit randomly chosen blocks.

In order to distinguish Wheesht keystream efficiently, we look for collisions between the 128-bit values $(\sigma_{4i+1}, \sigma_{4i+3})$ (respectively, collisions between the values $(\sigma_{4i+0}, \sigma_{4i+2})$), *i.e.* two indexes i, j with $(\sigma_{4i+1}, \sigma_{4i+3}) = (\sigma_{4j+1}, \sigma_{4j+3})$.

In a random keystream, the first collision would be found after $\sqrt{\pi/2} \cdot 2^{64}$ blocks on average. Moreover, the time of the first collision follows a Rayleigh distribution with parameter $\sigma = 2^{64}$ [2, pages 115–116]. However, with Wheesht keystream, collisions are expected earlier, because they can occur either in λ or in χ (respectively, either in μ or in ξ). Let B' denote the random variable of the time of first collision observed in a Wheesht keystream, and B_λ and B_μ the time of the first collision in λ and μ with random inputs. We have $B' > t$ (*i.e.* the first t keystream block are distinct) if and only if the first t (y_1, y_3) values are distinct, and the corresponding outputs of χ are distinct. Since χ is evaluated on distinct random values, we can write:

$$\Pr\left[B' > t\right] = \Pr\left[B_\lambda > t\right] \times \Pr\left[B_\chi > t\right]$$

From the Rayleigh distribution, we have $\Pr\left[B_\lambda > t\right] = \Pr\left[B_\mu > t\right] \sim e^{-t^2/2\sigma^2}$, with $\sigma = 2^{64}$. This gives:

$$\Pr\left[B' > t\right] \sim e^{-t^2/2\sigma^2} \times e^{-t^2/2\sigma^2} = e^{-t^2/\sigma^2}$$

Therefore, B' follows a Rayleigh distribution with parameter $\sigma' = \sigma/\sqrt{2} = 2^{64}/\sqrt{2}$. In particular, the mean value of B' is $\sigma'\sqrt{\pi/2} = \sqrt{\pi/4} \cdot 2^{64} \approx 0.89 \cdot 2^{64}$ (rather than $\sqrt{\pi/2} \cdot 2^{64} \approx 1.25 \cdot 2^{64}$).

This allows to build a very efficient distinguisher: we capture blocks of keystream until we detect a collision between the 128-bit values $(\sigma_{4i+1}, \sigma_{4i+3})$, and we measure the time until the first collision. Since the Rayleigh distribution has a high variance ($\frac{4-\pi}{2}\sigma^2$), we repeat the measure several times, and average the results. We denote the average of n measures by B_n (respectively B'_n). We have:

$$E(B_n) = \sqrt{\pi/2} \cdot 2^{64} \qquad \mathrm{Var}(B_n) = \frac{1}{n} \cdot \frac{4-\pi}{2} \cdot 2^{128}$$
$$E(B'_n) = \sqrt{\pi/4} \cdot 2^{64} \qquad \mathrm{Var}(B'_n) = \frac{1}{n} \cdot \frac{4-\pi}{2} \cdot 2^{127}$$

In order to distinguish the distributions, we set a threshold at $T = 2^{64}\sqrt{\pi}\left(2 - \sqrt{2}\right)$. We can use Chebyshev's inequality to bound the success probability of the distinguisher. For instance, with $n = 32$ we have:

$$|T - E(B_n)| \geq 1.85 \cdot \sqrt{\mathrm{Var}(B_n)} \qquad \Pr\left[B_n < T\right] \leq 0.3$$
$$|T - E(B'_n)| \geq 1.85 \cdot \sqrt{\mathrm{Var}(B'_n)} \qquad \Pr\left[B'_n > T\right] \leq 0.3$$

The detailed attack is given as Algorithm 1. The average number of required keystream blocks for each measure of the time of the first collision is at most $\sqrt{\pi/2} \cdot 2^{64} \approx 2^{64.33}$, implying that the average data complexity of the attack corresponds to 16 known plaintexts, each of length corresponding to $2^{64.33}$ blocks (i.e., $2^{69.33}$ bytes).

If each of the 16 known plaintexts contains more than 2^{67} blocks, then the time of the first collision can be observed with overwhelming probability since $\Pr\left[B < 2^{67}\right] \approx 1 - e^{-2^{2\times 67}/2^{129}} \geq 1 - 2^{-46}$.

This attack has been verified experimentally with a reduced version of Wheesht with 16-bit words, and experimental results agree with the analysis: the first collision for Wheesht keystream is detected after 58079 blocks on average, rather than 82137 blocks on average for a random data.

Algorithm 1 Distinguishing attack on Wheesht.

Capture 16 known plaintext messages of length 2^{67} blocks.
Denote the keystream as $\left(\sigma_j^{(i)}\right), 0 \le i < 16, 0 \le j < 2^{69}$
for $0 \le i < 16$ **do**
  for $0 \le k < 2$ **do**
    $\mathcal{S} \leftarrow \varnothing$
    for $0 \le j < 2^{67}$ **do**
      if $(\sigma_{4j+k}, \sigma_{4j+2+k}) \in \mathcal{S}$ **then**
        $B[2i+k] \leftarrow j$
        break loop
      else
        $\mathcal{S} \leftarrow \mathcal{S} \cup \{(\sigma_{4j}, \sigma_{4g+2})\}$
      end if
    end for
  end for
end for
if Average$(B) < 1.038 \cdot 2^{64}$ **then**
  return 1: *keystream is from Wheesht*
else
  return 0: *keystream is random*
end if

2.2 Key-recovery attack

Another important observation is that the keystream words $(k_1 \oplus z_1, k_3 \oplus z_3)$ can be computed from only eight values: x_4, x_1, x_6, x_3, y_4, y_6, k_1, k_3, and from the block counter b. Moreover, for successive blocks of the same message, all those values are fixed, only the block counter b is modified (incremented). We denote this computation by f:

$$(z_1, z_3) = f(x_4, x_1, x_6, x_3, y_4, y_6, b), \quad (\sigma_1, \sigma_3) = (z_1, z_3) \oplus (k_1, k_3).$$

Finally, since the key k_1, k_3 is only used at the end, it can be cancelled by XORing two different outputs. In our attack we first compute the XOR of successive output blocks for $2^{3\times 64} = 2^{192}$ values of x_4, x_1, x_6, x_3, y_4, y_6; then we collect plaintext/ciphertext pairs, and we use the offline computation to detect a match for the values of x_4, x_1, x_6, x_3, y_4, y_6. We expect to find a match after about $2^{3\times 64} = 2^{192}$ output blocks, due to the birthday paradox. The full attack is described by Algorithm 2.

The attack uses only the high level structure of Wheesht, as shown in Figure 1. It is independent of the number of rounds t_m and t_f, and of the details of the permutation θ. The attack requires about 2^{192} known plaintext data, but the data can come from any number of different sessions (with different nonces),

and even with different keys (in this case, only the key of one session will be recovered). It is not clear whether this violates the designers claim (there is no clear limit on the amount of data), but this seems to violate the security level of 256 bits.

Algorithm 2 Generic key-recovery attack on Wheesht.

▷ **Offline computations:**
$(x_4, x_1, x_6) \leftarrow (0, 0, 0)$
for all x_3, y_4, y_6 **do**
  for $0 \leq i < 8$ **do**
    Evaluate $(z_1^{(i)}, z_3^{(i)}) = f(x_4, x_1, x_6, x_3, y_4, y_6, i)$
  end for
  $X \leftarrow (z_1^{(i)} \oplus z_1^{(0)}, z_3^{(i)} \oplus z_3^{(0)})_{i=1}^{7}$
  Store $X, (x_4, x_1, x_6, x_3, y_4, y_6)$ in a hash table T
end for
▷ **Online computations:**
Capture $32 \times 2^{3\times 64}$ words of known plaintext data
for all keystream words $\sigma_0, \sigma_1, \ldots \sigma_{31}$ **do**
  Compute $X = (\sigma_{4i+1} \oplus \sigma_1, \sigma_{4i+3} \oplus \sigma_3)_{i=1}^{7}$
  if X is in the table T **then**
    Recover $(x_4, x_1, x_6, x_3, y_4, y_6)$ from T
    Recover k_1, k_3 from the keystream and z_1, z_3.
    Recover k_0, k_2 by brute force
  end if
end for

3 Low data complexity Attack

We now show a low data complexity attack using properties of the components of Wheesht. In particular, we study the final transformation θ^{t_f}. We target Wheesht-3-1-256, which is supposed to give 256 bits of security with $t_f = 1$.

Following the previous observations, we know that blocks inside a fixed message have the same values y_4 and y_6. We now explain how to detect this using the outputs z_1 and z_3. Figure 2 shows the details of the finalization function θ^{t_f}, inspired by the Salsa family. In particular, we note that y_6 can be computed from z_1, z_2, z_3. We denote this function as g: $y_6 = g(z_1, z_2, z_3)$.

If we consider several blocks inside the same message, we can observe $z_1 \oplus k_1$ and $z_3 \oplus k_3$ from the keystream, and we know that the values must be coherent with a fixed y_6. If we guess k_1 and k_3, we can compute a set of possible values of y_6 as $\{g(z_1, z_2, z_3), \forall z_2 \in \mathbb{Z}_{2^{64}}\}$ for every observation of (z_1, z_3); we expect about $(1 - 1/e) \cdot 2^{64}$ distinct values in the set. If we take the intersection of the sets corresponding to several observations, we expect a single remaining value after a few hundred observations, or no value if the key guess was incorrect. More precisely, with $N = 128$ observations, a wrong key will have a non-empty intersection with probability $(1 - 1/e)^{128} \approx 2^{-84}$, therefore we expect about 2^{44} wrong candidates, and testing them is a negligible part of the attack. This leads to the attack given in Algorithm 3.

The attack has been verified experimentally with a reduced version of Wheesht with 8-bit words. Those experiments show two small differences with the theoretical analysis. First the most significant bit of z_1 cannot be recovered because flipping it only shifts the set of y_6 candidates. Second, the filtering is slightly lower than expected: after 32 blocks of keystream, bad key guess are still valid with probability about 2^{-12}, rather than the expected 2^{-20}. Therefore, we suggest to use $N = 256$ in the attack against the full Wheesht.

Algorithm 3 Low data complexity key-recovery attack on Wheesht.

Input: known keystream $\sigma_0, \sigma_1, \ldots \sigma_{4N-1}$, $N \approx 256$
  for all k_1, k_3 **do**
    $\mathcal{S} \leftarrow \{0, 1, \ldots, 2^{64} - 1\}$
    for $0 \leq i < N$ **do**
      $z_1 \leftarrow \sigma_{4N+1} \oplus k_1$
      $z_3 \leftarrow \sigma_{4N+3} \oplus k_3$
      $\mathcal{T} \leftarrow \{g(z_1, z_2, z_3), \forall z_2 \in \mathbb{Z}_{2^{64}}\}$
      $\mathcal{S} \leftarrow \mathcal{S} \cap \mathcal{T}$
    end for
    if $\mathcal{S} \neq \varnothing$ **then**
      Recover k_0, k_2 by brute force
    end if
  end for

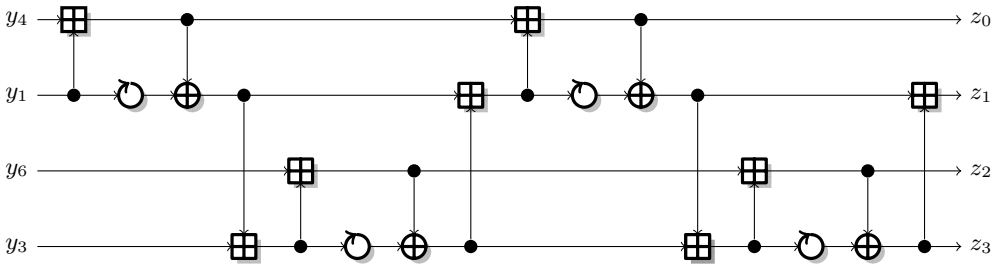

Fig. 2. θ function

Conclusion

We have described several attacks on Wheesht, proving that the security is far from the target level of 256-bit. Their respective complexities are given in Table 1.

The attacks of Section 2 are based only on the high level structure of Wheesht and can be applied with any number of rounds, while the attack of Section 3 use properties of the finalization round to reduce the complexity of a key-recovery.

Our attacks assume that the image sets of several functions are close to the size expected for a randomly chosen function. This has been checked experimentally on reduced versions on Wheesht (*e.g.* for the function g, we used a reduced version of θ operating on bytes instead of 64-bit words).

	time complexity	data complexity (words)
distinguisher	$2^{70.3}$	$2^{70.3}$
generic key-recovery	2^{192}	2^{197}
key-recovery on Wheesht-3-1-256	2^{200}	2^{10}

Table 1. Complexities of the proposed attacks. The data complexity of the distinguisher corresponds to 16 known plaintexts of average length $2^{66.3}$ words, while the data complexity of the second attack corresponds to messages of 32 words from 2^{192} possibly different sessions.

In order to avoid those attacks, a simple tweak to Wheesht would be to have more layers in the global structure. For instance, an extra finalization and swapping step seem to avoid the issues reported here.

References

1. Flajolet, P., Odlyzko, A.M.: Random mapping statistics. In: Quisquater, J.J., Vandewalle, J. (eds.) Advances in Cryptology - EUROCRYPT'89. Lecture Notes in Computer Science, vol. 434, pp. 329–354. Springer (1989)
2. Flajolet, P., Sedgewick, R.: Analytic Combinatorics. Cambridge University Press (2009)
3. Maxwell, P.: Wheesht: an AEAD stream cipher. Submission to CAESAR. Available from: `http://competitions.cr.yp.to/round1/wheeshtv03.pdf` (v1) (March 2014)