Differential Meet-In-The-Middle Cryptanalysis

Abstract. In this paper we introduce the differential-meet-in-the-middle framework, a new cryptanalysis technique against symmetric primitives. The idea of this new cryptanalysis method consists in combining into one attack techniques from both meet-in-the-middle and differential cryptanalysis. The introduced technique can be seen as a way of extending meet-in-the-middle attacks and their variants but also as a new way to perform the key recovery part in differential attacks. We provide a simple tool to search, given a differential, for efficient applications of this new attack and apply our approach, in combination with some additional techniques, to `SKINNY-128-384`. Our attack on `SKINNY-128-384` permits to break 25 out of the 56 rounds of this variant and improves by two rounds the previous best known attacks in the single key model.

Keywords: new cryptanalysis family, differential cryptanalysis, meet-in-the-middle cryptanalysis, `SKINNY`

1 Preliminaries

We start by recalling the basic frameworks of both differential and meet-in-the-middle attacks, the two techniques that are directly linked with the new cryptanalysis framework introduced in this work.

1.1 Differential Cryptanalysis

Differential cryptanalysis was introduced in 1990 by Biham and Shamir who used this method to break the Data Encryption Standard (DES) [3]. This technique applied to block ciphers consists in exploiting a *differential distinguisher*: input differences that propagate through the cipher to output differences with a probability significantly higher than what is expected for a random permutation. Adding rounds around the distinguisher allows to perform a *key recovery*, this step consists in guessing the elements of the key that allow the pairs of plaintexts or ciphertexts to propagate through the differential distinguisher.

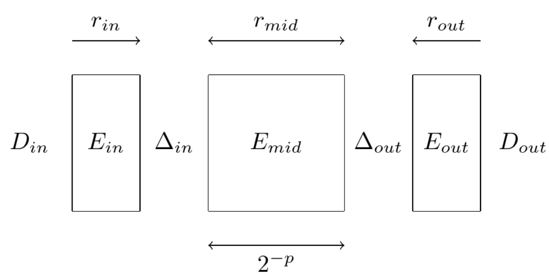

Fig. 1. Differential Cryptanalysis framework

More precisely, let E be an n-bit block cipher with r rounds. We write $E = E_{out} \circ E_{mid} \circ E_{in}$, as in Figure 1, where E_{out}, E_{mid} and E_{in} have r_{out}, r_{mid} and r_{in} rounds respectively ($r = r_{in} + r_{mid} + r_{out}$). A differential attack is based on a differential distinguisher, that is a tuple $(\Delta_{in}, \Delta_{out}, p)$ such that the probability that Δ_{in} propagates to Δ_{out} after r_{mid} rounds is 2^{-p}.
Next, we define the two sets of differences D_{in} and D_{out} such that Δ_{in} maps backwards to D_{in} through E_{in} and Δ_{out} maps forwards to D_{out} through E_{out}. We define the quantities d_{in} and d_{out} such that $2^{d_{in}} = |D_{in}|$ and $2^{d_{out}} = |D_{out}|$.

A differential attack first consists in generating data by using *structures*, i.e. set of plaintexts that differ only on the d_{in} bits that correspond to the active part of D_{in}. Note that it is possible to build $2^{2d_{in}-1}$ pairs for each structure. Since the probability for a random pair of D_{in} to propagate to Δ_{in} usually is $2^{-d_{in}}$, we can expect to have a pair that satisfies the differential if we generate $2^{p+d_{in}}$ pairs. By considering 2^s structures with $2^s 2^{2d_{in}-1} = 2^{p+d_{in}}$, that is $s = p - d_{in} + 1$, it is possible to generate enough data. Then, we filter the data generated thanks to the output, i.e. we discard all pairs that have a difference outside D_{out}. This results in a $n - d_{out}$-bit sieve, hence we are left with $2^{p+d_{in}+d_{out}-n}$ pairs. Lastly, we use the remaining pairs to perform a key recovery. For this, we guess for each pair the needed key bits and check whether the pair propagates to the input/output differences of the differential distinguisher.

1.2 Meet-In-The-Middle.

Meet-In-The-Middle attacks were introduced by Diffie and Hellman in 1977 [6]. This widely used technique consists in computing some internal state in the middle of the cipher from both the input and the output without needing the whole key to perform either of these computations. To recover key bits, the attacker keeps tracks of the keys leading to a collision in the middle state.

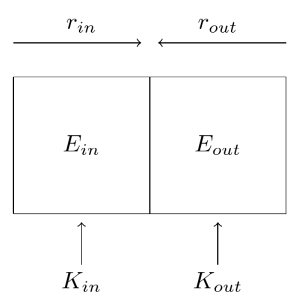

Fig. 2. Meet-In-The-Middle framework

More precisely, let E be an n-bit block cipher with r rounds and let (P, C) denote a pair of plaintext/ciphertexts. We write $E = E_{out} \circ E_{in}$, as in Figure 2, where E_{out} and E_{in} have r_{out} and r_{in} rounds respectively ($r = r_{in} + r_{out}$). We denote by K_{in} (respectively K_{out}) the set of key bits required in the computation of E_{in} (respectively E_{out}).

Algorithm 1 describes the Meet-In-The-Middle procedure. Since this attack can be performed in the encryption or the decryption direction, without loss of generality, we can suppose that $|K_{in}| \leq |K_{out}|$. The size of the table is set to $|T| = |K_{in}|$, therefore the data complexity is $\mathcal{D} = |K_{in}|$. Since the algorithm consists in a first loop over K_1 and a second loop over K_2, the time complexity is $\mathcal{T} = |K_{in}| + |K_{out}|$.

The cryptographic community raised many improvements to this generic attack, such as the technique of guessing some bits of the internal state [7], the all-subkeys approach [9], splice-and-cut [1, 2, 8] and bicliques [10]. In the following, we will present the improvements proposed in [4] and [5] : *partial matching, sieve-in-the-middle.*

Algorithm 1 Meet-In-The-Middle attack

```
T ← Empty table                              ▷ Table Initialisation
S ← ∅                                        ▷ Solution Set Initialisation
for k_in ∈ K_in do
    M_in ← E_in(P, k_in)
    T[hash(M_in)] ← k_in
end for
for k_out ∈ K_out do
    M_out ← E_out(P, k_out)
    if T[hash(M_out)] is not empty then
        S ← S ∪ {k_in, k_out}
    end if
end for
return S
```

Partial Matching. Partial matching is a technique that generalises the meet-in-the-middle technique. Indeed, instead of matching on the whole state in the middle state, we will now allow to match on a substate composed of fewer bits (see Figure 3). Computing this subspace might involve fewer key bits, hence reducing the size of K_1 and K_2, therefore reducing the overall complexity of the meet-in-the-middle procedure.

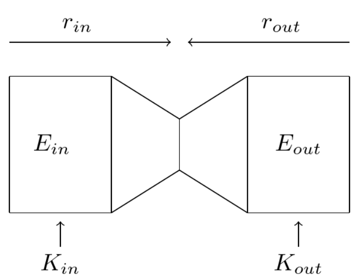

Fig. 3. Partial Matching framework

Sieve-in-the-middle Sieve-In-The-Middle is a technique that generalises partial matching (and also meet-in-the-middle) that consists in discarding the middle states computed form the input and output that can not match.

As represented in Figure 4, this technique consists first in splitting the cipher into three parts : *input part, middle part and output part*. The *input part* (respectively *output part*) consists in a subcipher E_{in} (respectively E_{out}). We denote by $\mathcal{U}$ (respectively $\mathcal{V}$) a subspace of $E_{in}(K, \{0,1\}^n)$ (respectively $E_{in}(K, \{0,1\}^n)$) and K_{in} (respectively K_{out}) the key bits involved in the computation of the restricted function $E_{in}|_{\mathcal{U}}$ (respectively $E_{out}|_{\mathcal{V}}$). The *middle part* consists of a keyed function S such that $S : K_{mid} \times \mathcal{U} \to \mathcal{V}$.

Now, it is possible to discard every guess (k_{in}, k_{out}) that produces a pair (u, v) that can not match through S. Hence, by precomputing the relation R as follows

$$R(u, v) \Leftrightarrow \exists k_{mid} \in K_{mid} \text{ s.t. } S(k_{mid}, u) = v$$

we can sieve the pairs (u, v), therefore the pairs (k_{in}, k_{out}). The Algorithm 2 summurizes the procedure of a Sieve-In-The-Middle attack.

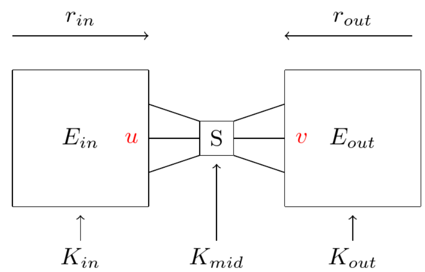

Fig. 4. Sieve-in-the-middle framework

Algorithm 2 Sieve-In-The-Middle attack

$L_{in} \leftarrow \emptyset$ ▷ Input Table Initialisation
$L_{out} \leftarrow \emptyset$ ▷ Output Table Initialisation
$L_{sol} \leftarrow \emptyset$ ▷ Soltion Set Initialisation
for $k_{in} \in K_{in}$ **do** ▷ Forward computation
    $u \leftarrow E_{in}|_{\mathcal{U}}(k_{in}, P)$
    $L_{in} \leftarrow L_{in} \cup \{(u, k_{in})\}$
end for
for $k_{out} \in K_{out}$ **do** ▷ Backward computation
    $v \leftarrow E_{out}|_{\mathcal{V}}(k_{out}, C)$
    $L_{out} \leftarrow L_{out} \cup \{(v, k_{out})\}$
end for
Merge L_{in} and L_{out} with respect to R and return the merged list L_{sol} ▷ Merge step
for k such that $(k_{in}, k_{out}) \in L_{sol}$ **do**
    if $E_k(C) = P$ **then**
        return k
    end if
end for

References

1. Aoki, K., Sasaki, Y.: Preimage attacks on one-block MD4, 63-step MD5 and more. In: Avanzi, R.M., Keliher, L., Sica, F. (eds.) SAC 2008. Lecture Notes in Computer Science, vol. 5381, pp. 103–119. Springer (2008)
2. Aoki, K., Sasaki, Y.: Meet-in-the-middle preimage attacks against reduced SHA-0 and SHA-1. In: Halevi, S. (ed.) CRYPTO 2009. Lecture Notes in Computer Science, vol. 5677, pp. 70–89. Springer (2009)
3. Biham, E., Shamir, A.: Differential cryptanalysis of DES-like cryptosystems. In: Menezes, A., Vanstone, S.A. (eds.) CRYPTO '90. Lecture Notes in Computer Science, vol. 537, pp. 2–21. Springer (1990)
4. Bogdanov, A., Rechberger, C.: A 3-subset meet-in-the-middle attack: Cryptanalysis of the lightweight block cipher KTANTAN. In: Biryukov, A., Gong, G., Stinson, D.R. (eds.) SAC 2010. Lecture Notes in Computer Science, vol. 6544, pp. 229–240. Springer (2010)
5. Canteaut, A., Naya-Plasencia, M., Vayssière, B.: Sieve-in-the-middle: Improved MITM attacks. In: Canetti, R., Garay, J.A. (eds.) CRYPTO 2013, Part I. Lecture Notes in Computer Science, vol. 8042, pp. 222–240. Springer (2013)
6. Diffie, W., Hellman, M.: Special feature exhaustive cryptanalysis of the NBS Data Encryption Standard. Computer **10**(6), 74–84 (1977)
7. Dunkelman, O., Sekar, G., Preneel, B.: Improved meet-in-the-middle attacks on reduced-round DES. In: Srinathan, K., Rangan, C.P., Yung, M. (eds.) INDOCRYPT 2007. Lecture Notes in Computer Science, vol. 4859, pp. 86–100. Springer (2007)
8. Guo, J., Ling, S., Rechberger, C., Wang, H.: Advanced meet-in-the-middle preimage attacks: First results on full Tiger, and improved results on MD4 and SHA-2. In: Abe, M. (ed.) ASIACRYPT 2010. Lecture Notes in Computer Science, vol. 6477, pp. 56–75. Springer (2010)
9. Isobe, T.: A single-key attack on the full GOST block cipher. In: Joux, A. (ed.) FSE 2011. Lecture Notes in Computer Science, vol. 6733, pp. 290–305. Springer (2011)
10. Khovratovich, D., Rechberger, C., Savelieva, A.: Bicliques for preimages: Attacks on Skein-512 and the SHA-2 family. In: Canteaut, A. (ed.) FSE 2012. Lecture Notes in Computer Science, vol. 7549, pp. 244–263. Springer (2012)