Sieve-in-the-Middle: Improved MITM Attacks (Full Version*)†

Anne Canteaut[1], María Naya-Plasencia[1], and Bastien Vayssière[2]

[1] Inria Paris-Rocquencourt, project-team SECRET
B.P. 105, 78153 Le Chesnay cedex, France
Anne.Canteaut@inria.fr, Maria.Naya_Plasencia@inria.fr
[2] Université de Versailles Saint-Quentin-en-Yvelines
45 avenue des Etats-Unis, 78035 Versailles cedex, France
bastien.vayssiere.w@gmail.com

Abstract. This paper presents a new generic technique, named sieve-in-the-middle, which improves meet-in-the-middle attacks in the sense that it provides an attack on a higher number of rounds. Instead of selecting the key candidates by searching for a collision in an intermediate state which can be computed forwards and backwards, we here look for the existence of valid transitions through some middle sbox. Combining this technique with short bicliques allows to freely add one or two more rounds with the same time complexity. Moreover, when the key size of the cipher is larger than its block size, we show how to build the bicliques by an improved technique which does not require any additional data (on the contrary to previous biclique attacks). These techniques apply to PRESENT, DES, PRINCE and AES, improving the previously known results on these four ciphers. In particular, our attack on PRINCE applies to 8 rounds (out of 12), instead of 6 in the previous cryptanalyses. Some results are also given for theoretically estimating the sieving probability provided by some subsets of the input and output bits of a given sbox.

Keywords. Meet-in-the-middle, block ciphers, bicliques, sbox, matching algorithms.

1 Introduction

Meet-in-the-middle (MITM) attacks are a widely used tool introduced by Diffie and Hellman in 1977. Through the years, they have been applied for analyzing the security of a substantial number of cryptographic primitives, including block ciphers, stream ciphers and hash functions, e.g. [36, 8, 21, 24, 23]. They exploit the fact that some internal state in the middle of the cipher can be computed both forwards from the plaintext and backwards from the ciphertext, and that none of these computations requires the knowledge of the whole master key. The attacker then only keeps the (partial) key candidates which lead to a collision in that internal state and discards all the other keys. This generic attack has drawn a lot of attention and raised many improvements, including the partial matching, where the computed internal states are not completely known, the technique of guessing some bits of the internal state [21], the all-subkeys approach [24], splice-and-cut [3, 4, 22] and bicliques [30]. The most popular application of bicliques is an accelerated exhaustive search on the full AES [6]. But, besides this degenerated application where the whole key needs to be guessed, short bicliques usually allow to increase the number of rounds attacked by MITM techniques without increasing the time complexity, but with a higher data complexity. Moreover, following [11, 12], low-data attacks have attracted a lot of attention, motivated in part by the fact that, in many concrete protocols, only a few plaintext-ciphertext pairs can be eavesdropped. MITM attacks belong

*Full version of the extended abstract published in the proceedings of CRYPTO 2013.

†Partially supported by the French Agence Nationale de la Recherche through the BLOC project under Contract ANR-11-INS-011.

to this class of attacks in most cases (with a few exceptions like bicliques): usually, 1 or 2 known plaintext-ciphertext pairs are enough for recovering the key.

Our contribution. This paper first provides a new generic improvement of MITM algorithms, named *sieve-in-the-middle*, which allows to attack a higher number of rounds. Instead of looking for collisions in the middle, we compute some input and output bits of a particular middle sbox S. The merging step of the algorithm then consists in efficiently discarding all key candidates which do not correspond to a valid transition through S. Intuitively, this technique allows to attack more rounds than classical MITM since it also covers the rounds corresponding to the middle sbox S (e.g. two middle rounds if S is a superbox). This new improvement is related to some previous results, including [3] where transitions through an ARX construction are considered; a similar idea was applied in [29] in a differential attack, and in [13] for side-channel attacks. This new generic improvement can be combined with bicliques, since short bicliques also allow to add a few rounds without increasing the time complexity. But, the price to pay is a higher data complexity. Here, we show that this increased data requirement can be avoided by constructing some improved bicliques, if the key size of the cipher is larger than its block size.

These new improvements and techniques are illustrated with 4 applications which improve previously known attacks. In Section 4, we describe a sieve-in-the-middle attack on 8 rounds of PRESENT, which provides a very illustrative and representative example of our technique. This attack applies up to 8 rounds, while the highest number of rounds reached by classical MITM is only 6. A similar analysis on DES is presented in Section 5; our attack achieves 8 rounds, while the best previous MITM attack (starting from the first one) was on 6 rounds. The cores of these two attacks have been implemented, confirming our theoretical analysis. In Section 6, the sieve-in-the-middle algorithm combined with the improved biclique construction is applied to 8 rounds (out of 12) of PRINCE, with 2 known plaintext-ciphertext pairs, while the previous best known attack was on six rounds. In Section 7, we show that we can slightly improve on some platforms the speed-up factor in the accelerated exhaustive search on the full AES performed by bicliques. The time complexity of the sieve-in-the-middle algorithm highly depends on the sieving probability of the middle sbox, i.e., on the proportion of pairs formed by a partial input and a partial output which correspond to a valid transition for S. We then give some results which allow to estimate the sieving probability of a given sbox. In particular, we show that the sieving probability is related to the branch number of the sbox, and we give a lower bound on the minimal number of known input and output bits which may provide a sieve.

2 The Sieve-in-the-Middle Attack

2.1 Basic idea

The basic idea of the attack is as follows. The attacker knows one pair of plaintext and ciphertext (P, C) (or several such pairs), and she is able to compute from the plaintext and from a part K_1 of the key candidate an m-bit vector u, which corresponds to a part of an intermediate state x. On the other hand, she is able to compute from the ciphertext and another part K_2 of the key candidate a p-bit vector v, which corresponds to a part of a second intermediate state y. Both intermediate states x and y are related by $y = S(x)$, where S is a known function from $\mathbf{F}_2^n$ into $\mathbf{F}_2^{n'}$, possibly parametrized by a part K_3 of the key. In practice, S can be a classical sbox, a superBox or some more complex function, as long as

the attacker is able to precompute and store all possible transitions between the input bits obtained by the forward computation and the output bits obtained backwards (or sometimes, these transitions can even be computed on the fly). In particular, the involved intermediate states x and y usually correspond to partial internal states of the cipher, implying that their sizes n and n' are smaller than the blocksize.

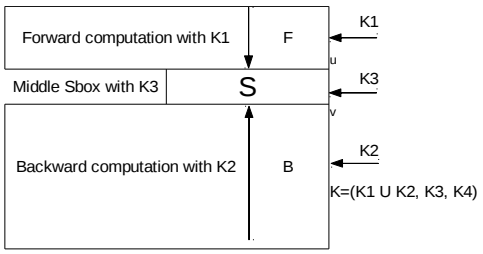


Fig. 1. Generic representation of Sieve-in-the-Middle.

Then, the attacker is able to compute some pairs (u, v) in $\mathbf{F}_2^m \times \mathbf{F}_2^p$ and she wants to determine whether those pairs can be some valid parts of a pair $(x, S(x))$ for some $x \in \mathbf{F}_2^n$ (and for some K_3 if S depends on a part of the key). If it appears that no input $x \in \mathbf{F}_2^n$ can lead to a given (u, v), then the keys (K_1, K_2) from which (u, v) has been obtained do not form a valid candidate for the key. In such a case, the $(m+p)$ positions corresponding to (u, v) can be used as a sieve. The sieving probability is then the proportion of pairs (u, v) corresponding to valid parts of $(x, S(x))$. Obviously, in classical MITM attacks, u and v correspond to the same n-bit part of an intermediate state and $S = \mathrm{Id}_n$; the sieving probability is then equal to 2^{-n}. We now define precisely when a pair (I, J) of input and output positions can be used as a sieve.

Definition 1. *Let S be a function from $\mathbf{F}_2^n$ into $\mathbf{F}_2^{n'}$. Let $I \subset \{1, \ldots, n\}$ and $J \subset \{1, \ldots, n'\}$ be two subsets with respective sizes m and p. The sieving probability of (I, J), denoted by $\pi_{I,J}$, is the proportion of all elements in $\mathbf{F}_2^{m+p}$ which can be written as $(x_i, i \in I; S_j(x), j \in J)$ for some $x \in \mathbf{F}_2^n$. The pair (I, J) is called an (m, p)-sieve for S if $\pi_{I,J} < 1$.*

The smaller $\pi_{I,J}$, the better the sieving, because more candidates will be discarded. If S depends on a k_3-bit value key K_3, the definition similarly applies but S must be seen as a function with $(k_3 + n)$ inputs.

When a large number of inputs and outputs of S can be computed by the attacker, they can be used as a sieve, as shown in the following proposition.

Proposition 1. *Any pair (I, J) of sets of size (m, p) with $m + p > n$ is a sieve for S with sieving probability $\pi_{I,J} \leq 2^{n-(m+p)}$.*

Proof. For any given u, there exists exactly 2^{n-m} values of x such that $(x_i, i \in I) = u$. Thus, $(S_j(x), j \in J)$ can take at most 2^{n-m} different values, implying that $\pi_{I,J} \leq 2^{n-(m+p)}$.

However, smaller subsets I and J may provide a sieve even when $m+p \leq n$. This issue will be extensively discussed in Section 8. More generally, u and v may consist of some information bits of x and y, i.e., of some linear combinations of the bits of x and y. We then define two linear functions $L : x \in \mathbf{F}_2^n \mapsto u \in \mathbf{F}_2^m$ and $L' : y \in \mathbf{F}_2^n \mapsto v \in \mathbf{F}_2^p$. The corresponding sieving

probability π is now the proportion of (u, v) such that there exists $x \in \mathbf{F}_2^n$ with $L(x) = u$ and $L'(S(x)) = v$. Then, π can be seen as the sieving probability of $I = \{1, \ldots, m\}$ and $J = \{1, \ldots, p\}$ for the function $L' \circ S \circ \widetilde{L}^{-1}$ where $\widetilde{L}$ is any linear permutation of $\mathbf{F}_2^n$ such that $(\widetilde{L}(x)_i, i \in I) = L(x)$.

2.2 Description of the attack

We now precisely describe the improved MITM attack and provide its complexity. The secret key K is divided into four (possibly non-disjoint) parts, K_1 to K_4. K_1 (resp. K_2) is the part of the key used in the forward (resp. backward) computation, while K_3 is involved in the middle S function only (see Fig. 1). The key bits corresponding to K_4 are not involved in the MITM step. In the following, k_i denotes the length of the key part K_i, while k is the total key length. Moreover, $K_1 \cap K_2$ denotes the bits shared by K_1 and K_2, and κ corresponds to the size of this intersection.

We denote by I (resp. J) the set of input positions of S (resp. output positions) corresponding to u (resp. v). The fact that a pair (u, v) corresponds to a valid pair of inputs and outputs of S is characterized by a Boolean relation $\mathcal{R}$ with $(m + p)$ inputs defined by

$$\mathcal{R}(u, v) = 1 \text{ if and only if } \exists x \in \mathbf{F}_2^n : (x_i, i \in I) = u \text{ and } (S(x)_j, j \in J) = v \ .$$

The attack proceeds as follows.

for all 2^κ values of $K_1 \cap K_2$ **do**
  $\mathcal{L}_f \leftarrow \emptyset$ and $\mathcal{L}_b \leftarrow \emptyset$
  // Forward computation
  for all $2^{k_1-\kappa}$ values of the remaining bits of K_1 **do**
    compute $u = F_{K_1}(P)$ and add u to $\mathcal{L}_f$
  // Backward computation
  for all $2^{k_2-\kappa}$ values of the remaining bits of K_2 **do**
    compute $v = B_{K_2}(C)$ and add v to $\mathcal{L}_b$
  // Merging step
  Merge $\mathcal{L}_f$ and $\mathcal{L}_b$ with respect to Relation $\mathcal{R}$ and return the merged list $\mathcal{L}_{sol}$.
// Testing the remaining candidates
for all K with (K_1, K_2) in $\mathcal{L}_{sol}$ **do**
  if $E_K(P) = C$ **then**
    return K

Section 2.3 details some efficient algorithms for merging the two lists $\mathcal{L}_f$ and $\mathcal{L}_b$ (i.e. for recovering all the (u, v) which satisfy $\mathcal{R}(u, v) = 1$) with complexity lower than the product of their sizes. Two representative examples of application on PRESENT and DES are provided in Section 4 and in Section 5 respectively.

With a single plaintext-ciphertext pair. Obviously, the whole secret key can be recovered only if the key length does not exceed the blocksize. Otherwise, 2^{k-b} possible keys will be returned in average where b is the blocksize. The time complexity of the attack is given by:

$$2^\kappa \left(2^{k_1-\kappa} c_F + 2^{k_2-\kappa} c_B + C_{\mathsf{merge}}\right) + \pi 2^k c_E \ ,$$

where π is the sieving probability of (I, J) as defined in Definition 1, c_E is the cost of one encryption, while c_F and c_B correspond to the costs of a partial encryption in the forward

and backward directions. In most cases, $c_F \simeq c_B \simeq c_E/2$. C_{merge} is the time complexity of the merging step, and it depends on k_3. Its value is discussed in the following section. The average time complexity of the attack needs to be compared to $2^k c_E$ which is the cost of an exhaustive search for the key. The memory complexity is mainly determined by the memory needed in the merging step. In some cases, it can be improved by storing only one among the two lists $\mathcal{L}_f$ and $\mathcal{L}_b$, when the auxiliary lists used in the merging step remain smaller.

With N plaintext-ciphertext pairs. If N plaintext-ciphertext pairs are available to the attacker, then the average number of keys returned by the attack is 2^{k-Nb}, implying that the whole key will be recovered when $N \geq k/b$. The main modification in the attack concerns the last step where all key candidates in $\mathcal{L}_{sol}$ are tested: before performing an exhaustive search over $(K_1 \cap K_2)$ and K_4 for testing all keys with $(K_1, K_2) \in \mathcal{L}_{sol}$, an additional sieving step is performed in order to reduce the size of $\mathcal{L}_{sol}$. Once a new solution $(K_1, K_2) \in \mathcal{L}_{sol}$ has been found, $(N-1)$ additional pairs (u_i, v_i) generated from the other plaintext-ciphertext pairs are considered, and only the keys for which $\mathcal{R}(u_i, v_i) = 1$ are kept in $\mathcal{L}_{sol}$ (note that, in some very particular situations, it might be more efficient to directly include in $\mathcal{L}_f$ and $\mathcal{L}_b$ the values u and v generated from several plaintext-ciphertext pairs, and then merge the lists). The average size of $\mathcal{L}_{sol}$ after this additional sieving step is then $\pi^N 2^{k_1+k_2-2\kappa}$. But this formula should be adapted to the case where S depends on a part of the secret key K_3: indeed the merging step determines a candidate for (K_1, K_2, K_3). Then, the sieving probability of the additional sieving step π' differs from π since the value of K_3 is now fixed. π' is then the sieving probability of (I, J) for S_{K_3} averaged over all K_3. Then, in the case of N plaintext-ciphertext pairs, the cost of the forward and backward computations are multiplied by N, while the cost of the testing part decreases:

$$2^\kappa \left(N 2^{k_1-\kappa} c_F + N 2^{k_2-\kappa} c_B + C_{\mathsf{merge}} \right) + \pi (\pi')^{N-1} 2^k c_E \ .$$

2.3 Merging the two lists efficiently

Very often, the middle function S can be decomposed into several smaller sboxes, and the merging step can be performed group-wise. The problem of merging two large lists with respect to a group-wise Boolean relation has been defined and addressed by Naya-Plasencia in [33, Section 2]. Here, we focus on three algorithms proposed in [33], namely instant matching, gradual matching and an improvement of the parallel matching due to [19]. We provide general and precise formulas for the average time and memory complexities of these three algorithms. Actually, in our case, the lists to be merged may be small. Then, the construction of some auxiliary tables, which had a negligible cost in [33] for large lists, must now be taken into account. It might even become the bottleneck of the algorithm. Thus, when the involved lists are small, it is harder to determine a priori which algorithm is the most efficient in a given case. Then, in each application, we need to check thoroughly which algorithm provides the best complexity. The optimal case may even sometimes correspond to the combination of two algorithms.

In the following, we consider two lists, $\mathcal{L}_A$ of size 2^{ℓ_A} and $\mathcal{L}_B$ of size 2^{ℓ_B}, whose roles are interchangeable. The elements of both lists can be decomposed into t groups: the i-th group of $a \in \mathcal{L}_A$ has size m_i, while the i-th group of $b \in \mathcal{L}_B$ has size p_i. The Boolean relation $\mathcal{R}$ can similarly be considered group-wise: $\mathcal{R}(a, b) = 1$ if and only $\mathcal{R}_i(a_i, b_i) = 1$ for all $1 \leq i \leq t$. The sieving probability π associated to $\mathcal{R}$ then corresponds to the product of the sieving

probabilities π_i associated to each $\mathcal{R}_i$. Since each $\mathcal{R}_i$ corresponds to an sbox S_i with n_i-bit inputs, a table storing all (a_i, b_i) such that $\mathcal{R}_i(a_i, b_i) = 1$ can be built with time complexity 2^{n_i}, by computing all $(x_i, S_i(x_i)), x_i \in \mathbf{F}_2^{n_i}$. The corresponding memory complexity is proportional to $\pi_i 2^{m_i+p_i}$. This cost won't be included in the cost of the merging algorithm since, in the sieve-in-the-middle process, the tables will be built once for all and not 2^κ times. As we will see, in some situations, these tables can be built "on-the-fly" with much fewer operations.

We now provide complete description of the three matching algorithms. For the sake of simplicity, we assume in the description of the algorithms that the lists are sorted, but in practice we can use standard hash tables for storage and lookup in constant time, since the keys are integers. It is worth noticing that the size of the list $\mathcal{L}_{sol}$ returned by the matching algorithm is not included in the memory complexity since each of its elements can be tested in the attack as soon as it has been found.

Instant Matching. Instant matching successively considers all elements $\mathcal{L}_B$: for each $b \in \mathcal{L}_B$, a list $\mathcal{L}_{aux}$ of all a such that $\mathcal{R}(a, b) = 1$ is built, and each element of $\mathcal{L}_{aux}$ is searched within $\mathcal{L}_A$.

Algorithm 1 Instant matching algorithm of $\mathcal{L}_A$ and $\mathcal{L}_B$ with respect to $\mathcal{R}$.

1: **for** j from 1 to t **do**
2: Build the table T_j such that $T_j[v_j]$ corresponds to all u_j with $\mathcal{R}_j(u_j, v_j) = 1$.
3: **for** each $(b_1, \ldots, b_t) \in \mathcal{L}_B$ **do**
4: $\mathcal{L}_{aux} \leftarrow \emptyset$.
5: **for** j from 1 to t **do**
6: **if** $T_j[b_j]$ is empty, **then** go to 3.
7: Add all tuples $(x_1, \ldots, x_t)$ with $x_j \in T_j[b_j]$, $\forall j$, to $\mathcal{L}_{aux}$.
8: **for** each $(x_1, \ldots, x_t)$ in $\mathcal{L}_{aux}$ **do**
9: **if** $(x_1, \ldots, x_t) \in \mathcal{L}_A$ **then**
10: Add $(x_1, \ldots, x_t, b_1, \ldots, b_t)$ to $\mathcal{L}_{sol}$.
11: Return $\mathcal{L}_{sol}$.

$$\text{Time} = \pi 2^{\ell_B+m} + \pi 2^{\ell_A+\ell_B} \text{ and Memory} = 2^{\ell_A} + 2^{\ell_B} \ .$$

Gradual Matching. Gradual matching is a recursive procedure as detailed by Algo 2. All elements are decomposed into two parts, the first t' groups and the last $(t - t')$, with $t' < t$. For each possible value β of the first t' groups, the sublist $L_B(\beta)$ is built. It consists of all elements in $\mathcal{L}_B$ whose first t' groups take the value β. Now, for each α such that $\mathcal{R}_i(\alpha_i, \beta_i) = 1, 1 \le i \le t'$, $L_B(\beta)$ is merged with the sublist $L_A(\alpha)$ which consists of all elements in $\mathcal{L}_A$ whose first t' groups take the value α. Then, we need to merge two smaller lists, of respective sizes $2^{\ell_A - \sum_{i=1}^{t'} m_i}$ and $2^{\ell_B - \sum_{i=1}^{t'} p_i}$.

$$\text{Time} = \left(\prod_{i=1}^{t'} \pi_i \right) 2^{\sum_{i=1}^{t'} m_i+p_i} C_{\mathsf{merge}} \text{ and Memory} = 2^{\ell_A} + 2^{\ell_B} \ .$$

where C_{merge} is the cost of merging the two remaining sublists.

Algorithm 2 Gradual matching algorithm of $\mathcal{L}_A$ and $\mathcal{L}_B$ with respect to $\mathcal{R}$.

for j from 1 to t **do**
  Build the table T_j such that $T_j[v_j]$ corresponds to all u_j with $\mathcal{R}_j(u_j, v_j) = 1$.
for each $\beta = (\beta_1, \ldots, \beta_{t'})$ in $(\mathbf{F}_2^{\sum_{j=1}^{t'} p_j})$ **do**
  $L_B(\beta) \leftarrow \{b \in \mathcal{L}_B \text{ with } (b_1, \ldots, b_{t'}) = \beta\}$
  $\mathcal{L}_{aux} \leftarrow \emptyset$.
  for each $(\alpha_1, \ldots, \alpha_{t'})$ with $\alpha_j \in T_j[\beta_j]$, $\forall j \leq t'$ **do**
    add $(\alpha_1, \ldots, \alpha_{t'})$ to $\mathcal{L}_{aux}$.
  for each $\alpha = (\alpha_1, \ldots, \alpha_{t'})$ in $\mathcal{L}_{aux}$ **do**
    $L_A(\alpha) \leftarrow \{a \in \mathcal{L}_A \text{ with } (a_1, \ldots, a_{t'}) = \alpha\}$
    Merge $L_A(\alpha)$ with $L_B(\beta)$ with respect to $\mathcal{R}' = \prod_{j=t'+1}^{t} \mathcal{R}_j$.
    Add the solutions to $\mathcal{L}_{sol}$.
Return $\mathcal{L}_{sol}$.

Parallel Matching without memory. We give here the first general description of the memoryless version of parallel matching. The details are provided by Algo 3. This algorithm applies an idea from [19] to the parallel matching algorithm from [33]: instead of building a big auxiliary list as in the original parallel matching, we here build small ones which do not need any additional memory. In parallel matching, the elements in both lists are decomposed into three parts: the first t_1 groups, the next t_2 groups, and the remaining $(t-t_1-t_2)$ groups. Both lists $\mathcal{L}_A$ and $\mathcal{L}_B$ are sorted in lexicographic order. Then, $\mathcal{L}_A$ can be seen as a collection of sublists $\mathcal{L}_A(\alpha)$, where $\mathcal{L}_A(\alpha)$ is composed of all elements in $\mathcal{L}_A$ whose first t groups equal α. Similarly, $\mathcal{L}_B$ is seen as a collection of $\mathcal{L}_B(\beta)$. The matching algorithm then proceeds as follows. For each possible value α for the first t groups, an auxiliary list $\mathcal{L}_{aux}$ is built, corresponding to the union of all $\mathcal{L}_B(\beta)$ where (α, β) satisfies the first t relations $\mathcal{R}_j$. The list $\mathcal{L}_{aux}$ is sorted by its next t_2 groups. Then, for each element in $\mathcal{L}_A(\alpha)$, we check if a match for its next t_2 groups exists in $\mathcal{L}_{aux}$. For each finding, the remaining $(t - t_1 - t_2)$ groups are tested and only the elements which satisfy the remaining $(t - t_1 - t_2)$ relations are returned.

Algorithm 3 Memoryless parallel matching algorithm of $\mathcal{L}_A$ and $\mathcal{L}_B$ with respect to $\mathcal{R}$.

for j from 1 to t' **do**
  Build the table T_j such that $T_j[v_j]$ corresponds to all u_j with $\mathcal{R}_j(u_j, v_j) = 1$.
for each $\alpha = (a_1, \ldots, a_{t_1})$ appearing in $\mathcal{L}_A$ **do**
  $\mathcal{L}_A(\alpha) \leftarrow \{a \in \mathcal{L}_A : (a_1, \ldots, a_{t_1}) = \alpha\}$.
  // Compute $\mathcal{L}_{aux}$
  $\mathcal{L}_1 \leftarrow \{\beta : \mathcal{R}_j(\alpha_j, \beta_j) = 1, 1 \leq j \leq t_1\}$
  $\mathcal{L}_{aux} \leftarrow \emptyset$
  for each $\beta \in \mathcal{L}_1$ **do**
    $\mathcal{L}_B(\beta) \leftarrow \{b \in \mathcal{L}_B : (b_1, \ldots, b_{t_1}) = \beta\}$.
    add all elements of $\mathcal{L}_B(\beta)$ to $\mathcal{L}_{aux}$.
  Sort $\mathcal{L}_{aux}$ by $\beta' = (b_{1+t_1}, \ldots, b_{t_1+t_2})$.
  // Merge $\mathcal{L}_A(\alpha)$ and $\mathcal{L}_{aux}$ with respect to the next t_2 groups.
  for each a in $L_A(\alpha)$ **do**
    $\mathcal{L}_2 \leftarrow \{\beta' : \mathcal{R}_j(\alpha_j, \beta'_j) = 1, t_1 < j \leq t_1 + t_2\}$
    for each $\beta' \in \mathcal{L}_2$ **do**
      if $\beta' \in \mathcal{L}_{aux}$ **then**
        for each $b \in \mathcal{L}_{aux}$ with $(b_{t_1+1}, \ldots, b_{t_1+t_2}) = \beta'$ **do**
          if $\mathcal{R}_j(a_j, b_j)$ for all $t_2 < j \leq t$ **then**
            Add (a, b) to $\mathcal{L}_{sol}$.

The time and memory complexities can be evaluated as follows. We first evaluate the average sizes of all lists involved in the algorithm. For each α, the average size of $\mathcal{L}_A(\alpha)$ is $2^{\ell_A - \sum_{i=1}^{t_1} m_i}$. Also, we have

$$|\mathcal{L}_1| = \left(\prod_{i=1}^{t_1} \pi_i\right) 2^{\sum_{i=1}^{t_1} p_i} \ , \ |\mathcal{L}_2| = \left(\prod_{i=t_1+1}^{t_1+t_2} \pi_i\right) 2^{\sum_{i=t_1+1}^{t_1+t_2} p_i}$$

and

$$|\mathcal{L}_{aux}| = \left(\prod_{i=1}^{t_1} \pi_i\right) 2^{\ell_B} \ .$$

Finally, the average number N of elements b which match with a on the first $t_1 + t_2$ groups and that should be tested at Line 18 in the algorithm is $\left(\prod_{i=1}^{t_1+t_2} \pi_i\right) 2^{\ell_B}$. Then, the average time complexity of parallel matching can be decomposed as

$$\begin{aligned} \text{Time} &= 2^{\sum_{i=1}^{t_1} m_i} \left[|\mathcal{L}_{aux}| + |\mathcal{L}_A(\alpha)| \left(|\mathcal{L}_2| + N\right)\right] \\ &= \left(\prod_{i=1}^{t_1} \pi_i\right) 2^{\ell_B + \sum_{i=1}^{t_1} m_i} + \left(\prod_{i=t_1+1}^{t_1+t_2} \pi_i\right) 2^{\ell_A + \sum_{i=t_1+1}^{t_1+t_2} p_i} + \left(\prod_{i=1}^{t_1+t_2} \pi_i\right) 2^{\ell_A + \ell_B} \ . \end{aligned}$$

It is worth noticing that the two lists $\mathcal{L}_1$ and $\mathcal{L}_2$ do not need to be stored since their elements are entirely defined by the tables T_j describing the valid transitions for S_j. The average memory required by the algorithm then corresponds to

$$\text{Memory} = |\mathcal{L}_A| + |\mathcal{L}_B| + |\mathcal{L}_{aux}| = 2^{\ell_A} + 2^{\ell_B} + \left(\prod_{i=1}^{t_1} \pi_i\right) 2^{\ell_B} \ .$$

3 Combining Sieve-in-the-Middle and Bicliques

Sieve-in-the-middle, as a generic technique, can be combined with other improvements of MITM attacks, in particular with bicliques [6, 30]. The general purpose of bicliques is to increase the number of rounds attacked by MITM techniques. Here, we briefly describe how bicliques can increase the number of rounds attacked by the previously described sieve-in-the-middle algorithm. This can be done at no computational cost, but requires a higher data complexity. In order to avoid this drawback, we then present an improvement of bicliques which applies when the key length exceeds the block size of the cipher.

3.1 Sieve-in-the-middle and classical bicliques

The combination of both techniques is depicted on Figure 2: the bottom part is covered by bicliques, while the remaining part is covered by a sieve-in-the-middle algorithm. In the following, $H_{K_8} : X \mapsto C$ denotes the function corresponding to the bottom part of the cipher, and K_8 represents the key bits involved in this part. Then, K_8 is partitioned into three disjoint subsets, K_5, K_6 and K_7. The value taken by K_i with $5 \leq i \leq 7$ will be represented by an integer in $\{0, \ldots, 2^{k_i} - 1\}$. A biclique can be built if the active bits in the computation of $H_{K_8}(X)$ when K_6 varies and the active bits in the computation of $H_{K_8}^{-1}(C)$ when K_5 varies are two disjoint sets. In this case, an exhaustive search over K_7 is performed and a biclique

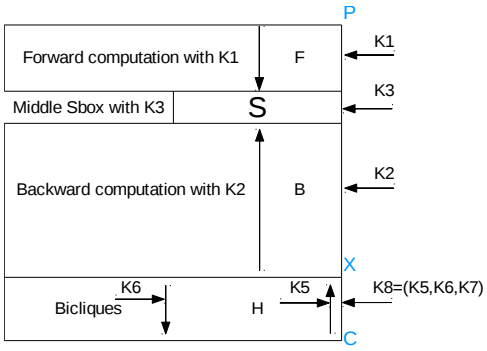


Fig. 2: Generic representation of Sieve-in-the-Middle and bicliques

is built for each value h of K_7 as follows. We start from a given ciphertext C^0 and a chosen key $K_8^0 = (0, 0, h)$ formed by the candidate for K_7 and the zero value for K_5 and K_6. We compute $X_h^0 = H_{0,0,h}^{-1}(C^0)$. Next, we compute backwards from C^0 the intermediate state $X_h^i = H_{i,0,h}^{-1}(C^0)$ for each possible value i for K_5. Similarly, we compute forwards from X_h^0 the ciphertext $C_h^j = H_{0,j,h}(X_h^0)$ for each possible value j of K_6. Since the two differential paths are independent, we deduce that $H_{i,j,h}(X_h^i) = C_h^j$ for all values (i, j) of (K_5, K_6).

Then, the sieve-in-the-middle algorithm can be applied for each K_7 and each value for $(K_1 \cap K_2)$. The list $\mathcal{L}_b$ of all output vectors v is computed backwards from X_h^i for each value i of K_5 and each value of $K_2 \setminus (K_1 \cap K_2)$. The list $\mathcal{L}_f$ of all input vectors u is computed forwards from all plaintexts P_h^j corresponding to C_h^j for each value j of K_6 and each value of $K_1 \setminus (K_1 \cap K_2)$. We then merge those two lists of respective sizes $2^{|K_2 \cup K_5|}$ and $2^{|K_1 \cup K_6|}$.

As in classical MITM with bicliques, the decomposition of K_8 should be such that the bits of K_5 do not belong to K_1, the bits of K_6 do not belong to K_2 and the bits of K_7 should lie in $(K_1 \cap K_2)$. The best strategy here seems to choose (K_5, K_6) such that the bits of K_5 belong to $K_2 \setminus (K_1 \cap K_2)$, and the bits of K_6 belong to $K_1 \setminus (K_1 \cap K_2)$. In this case, we have to add to the time complexity of the attack the cost of the construction of the bicliques, i.e., $2^{k_7}(2^{k_5} + 2^{k_6})c_H$ (very rarely the bottleneck), where c_H is the cost of the partial encryption or decryption corresponding to the rounds covered by the bicliques. The main change is that the data complexity has increased since the attack now requires the knowledge of all plaintext-ciphertext pairs (P_h^j, C_h^j) corresponding to all possible values (j, h) for (K_6, K_7). The data complexity then would correspond to $2^{k_6+k_7}$ pairs of plaintext-chosen ciphertexts, but it is usually smaller since the ciphertexts C_h^j only differ on a few positions.

3.2 Improved bicliques for some scenarios

Now, we describe a generic idea for improving bicliques in certain scenarios and reducing the data complexity to a single plaintext-ciphertext pair. Our improvement usually applies when the total key size of the cipher is larger than the block size. This occurs for instance when whitening keys are used. A detailed and successful application is demonstrated on PRINCE in Section 6. The main idea of our improvement is to gather some parts of the partial exhaustive search over K_7 into different groups such that, within a group, all obtained ciphertexts C^j are equal to C^0.

We consider a biclique repartition of keys consistent with the sieve-in-the-middle part: we choose $K_5 \subset K_2 \setminus (K_1 \cap K_2)$ as previously, and some set $K'_6 \subset K_1$ (this differs from the classical biclique construction where we had $K_6 \subset K_1 \setminus (K_1 \cap K_2)$). Let Δ_6^C be the positions of the bits of C which may be affected by K'_6 when computing forward from X, and let Δ_6^X be the positions of the bits of X which may be affected by Δ_6^C and K'_6 during the backward computation. In classical bicliques, the path generated in the backward direction by the different K_5 must be independent from the path generated in the forward direction by the different K'_6. Here, we also require this first path generated by K_5 to be independent from the backward path generated when the ciphertext bits in positions Δ_6^C vary. For instance, in the example depicted on Figure 3, H follows the Even-Mansour construction, i.e., it is composed of an unkeyed permutation H' and the addition of two whitening keys K_a and K_b. The positions of K_5 and K'_6 are represented in red and blue respectively, and it can be checked that the corresponding paths are independent.

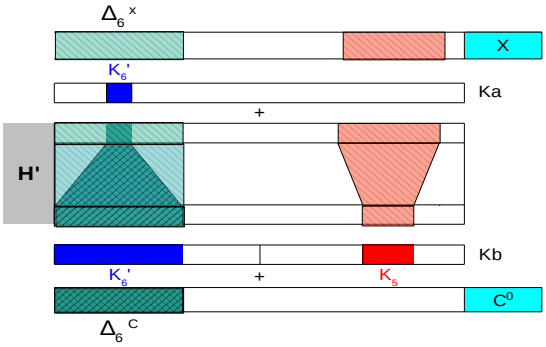
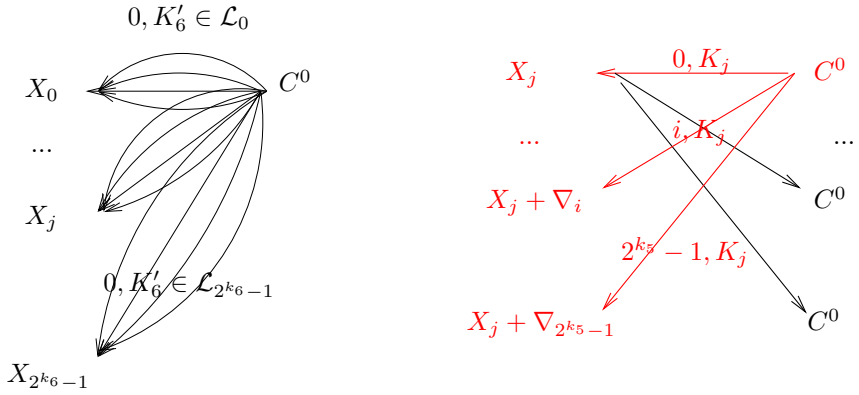


Fig. 3: Example of the improved biclique construction.

In this situation, an improved biclique without any additional data can be built if the size of Δ_6^X is smaller than k'_6. In our context, the algorithm has to be repeated for each value h for $K'_7 = K_8 \setminus (K_5 \cup K'_6)$, but the index h will be omitted in the description. First, we precompute the values obtained from a chosen C^0 when K'_6 takes all possible values. If the number of information bits in Δ_6^X is less than k'_6, all $2^{k'_6}$ transitions can be represented by several lists $\mathcal{L}_j$, each containing the different values of K'_6 which all map C^0 to the same value of the state X, X_j (see Figure 4(a)). For the sake of simplicity, we assume that all these lists have the same size 2^ℓ. In most cases, we have $\ell = k'_6 - |\Delta_6^X|$. For the example depicted on Figure 3, we assume that H' is such that the function obtained by restricting its inputs to the positions in Δ_6^X and its outputs to the positions in Δ_6^C is a permutation. Then, it clearly appears that the number of bits in Δ_6^X is equal to the number of bits of $K'_6 \cap K_b$, and thus strictly smaller than the number of bits of K'_6. More precisely, there are exactly 2^ℓ values of K'_6, with $\ell = |K'_6 \cap K_a|$, which provide the same value of $X = H'^{-1}(C^0 + K_b) + K_a$ when K'_6 varies and all other bits are fixed.

Now, for each of the $2^{k'_6-\ell}$ values of X_j, all transitions from C^0 to X_j through different values of $K'_6 \in \mathcal{L}_j$ can also be seen as the 2^ℓ biclique transitions from X_j to C^0 through some particular values of the key K'_6 (these transitions are represented in black on Figure 4(b)).



(a) Step 1

(b) Step 2: to be repeated for the $2^{k'_6-\ell}$ values of j

Fig. 4: Improved bicliques construction.

Now, the second step consists in building the bicliques in the other direction: from C^0 for each value of X_j. For each of the $2^{k'_6-\ell}$ values of j, we fix the value of K'_6 to a constant value K_j appearing in $\mathcal{L}_j$. This way, the part of X corresponding to Δ_6^X is the same for all the transitions of the bicliques, and this property holds even when K_5 is modified since both corresponding paths are independent. We then consider the 2^{k_5} possible values i for K_5 and compute the corresponding $X = X_j + \nabla_i$ (see Figure 4(b)). We then deduce the $2^{k_5+k'_6}$ transitions $H(X_j + \nabla_i)_{(i,K'_6)} = C^0$ for all $K'_6 \in \mathcal{L}_j$, from $(2^{k'_6} + 2^{k'_6-\ell+k_5})$ computations of the function. Indeed, the first term in the complexity corresponds to the precomputation phase (Step 1), and the second one to the number of lists $\mathcal{L}_j$, $2^{k'_6-\ell}$, multiplied by the cost for building the bicliques in the other direction. The main advantage of this construction is that it can be combined with the sieve-in-the-middle part as previously described, but it now requires a single plaintext-ciphertext pair, the one formed by (P^0, C^0).

Finally, we assume that the bits of K_5 belong to $K_2 \setminus (K_1 \cap K_2)$, the bits of K'_6 belong to K_1 and the bits of K'_7 are the bits from $(K_1 \cup K_2) \setminus (K_5 \cup K'_6)$, the time complexity of the attack is:

$$2^{k'_7}\left(2^{k'_6} + 2^{k'_6-\ell+k_5}\right) c_H + 2^{k_1} c_F + 2^{k_2} c_B + 2^{\kappa} C_{\mathsf{merge}} + \pi 2^k c_E$$

where C_{merge} is the cost of merging the lists of size $2^{k_1-\kappa}$ and $2^{k_2-\kappa}$ with respect to the sieving conditions.

A similar idea can also be used for choosing an appropriate K_5 which delays the propagation of the unknown bits during the forward computation. This will be shown in the case of Prince.

4 Application to PRESENT

We here discuss an application example on the block cipher PRESENT-80, which illustrates our ideas. The number of rounds reached when using the new improvement will be seven, while it can be proved that classical meet-in-the-middle attacks do not apply on more than six rounds. By using bicliques, we can directly extend the attack to 8 rounds with the same computations and a data complexity of 2^6 instead of 2. We can similarly apply our attack to 9 and 10 rounds of PRESENT-128. These results are far from reaching the number of rounds of the best known attacks, but are the first ones with (very) low data complexity. Actually, as pointed out in [11, 12], it is important to analyze the primitives when only few data are available. PRESENT could to some extend be considered as one of the most important lightweight block ciphers, and PRESENT-like functions might be used in further constructions as in [10, 15, 31]. Determining the number of rounds which can be attacked with a single (or only a few) pair of plaintext-ciphertext is then important to better understand its security.

4.1 Brief description of PRESENT

PRESENT is an ultra-lightweight block cipher proposed by Bogdanov *et al.* [7], which has been standardized by ISO in 2011. Its original structure has attracted the attention of the community, and a large number of results on reduced versions have been published [38, 16, 39, 35, 2, 34, 17, 27, 32, 5]. All these attacks need a large number of plaintext-ciphertext pairs, which in most cases reaches the full codebook.

Two versions of PRESENT have been proposed, with an 80-bit key and with a 128-bit key. Besides the key length, both versions only differ in the key schedule. PRESENT operates on 64-bit blocks. For encrypting the plaintext, 31 rounds of the following round-function are applied, followed by a last whitening subkey addition (sk_{32}).

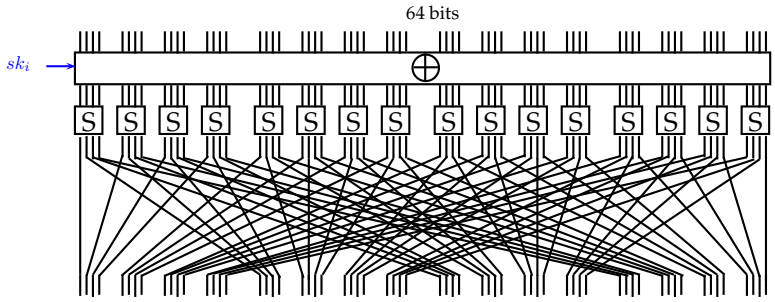


Fig. 5: One round of PRESENT.

The round function consists of 3 transformations, as depicted on Figure 5:

1. The subkey addition: at the beginning of each round i, for $i \in [1, \ldots, 31]$, the corresponding subkey sk_i of 64 bits is xored to the internal state.
2. The non-linear transformation: 16 4×4-bit sboxes S are applied in parallel to the 16 groups of 4 consecutive bits. PRESENT sbox is represented in hexadecimal notation by:

x	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
S[x]	C	5	6	B	9	0	A	D	3	E	F	8	4	7	1	2

3. A bit-wise permutation P, which operates on the 64 bits as follows:

$$P(i) = \begin{cases} 16i \bmod 63 & \text{for } 0 \leq i \leq 62 \\ 63 & \text{for } i = 63 \end{cases}$$

Key schedule for the 80-bit version. Given an 80-bit key $K = k_{79}, \ldots, k_0$, the subkeys, $(sk_1, \ldots, sk_{31}, sk_{32})$ that are xored to the internal state at each round, where sk_{32} is the final whitening key, are computed from the key bits in the following way: For i from 1 to 32

1. $sk_i = k_{79}, \ldots, k_{16}$,
2. $k_{79}, k_{78}, \ldots, k_1, k_0 = k_{18}, k_{17}, \ldots, k_0, k_{79}, \ldots, k_{20}, k_{19}$,
3. $k_{79}k_{78}k_{77}k_{76} = S(k_{79}k_{78}k_{77}k_{76})$,
4. $k_{19}k_{18}k_{17}k_{16}k_{15} = k_{19}k_{18}k_{17}k_{16}k_{15} \oplus$ roundcounter.

4.2 Sieve-in-the-Middle on 7 and 8 rounds of PRESENT-80

The attack exploits the fact that the subkeys do not involve all bits of the key, and also that the values of some bits in the "middle" can be computed without knowing the whole state. Figure 6 represents the consecutive internal states in the MITM attack on 7 rounds of PRESENT-80. One round at the end will then be added with bicliques. Each square represents a nibble (of the key or of the state). Colored bits correspond to known bits, and white bits are unknown. The colored Sboxes in the middle are those involved in the sieve-in-the middle procedure. In the forward computation, all 80 key bits are known except 9, namely bits 3, 4 and 8 to 14. Then, with the notation introduced in Section 2.2, we have $k_1 = 71$. In the backward direction, all key bits but 6 (bits 57, 60, 61, 64, 65 and 68) are known, implying $k_2 = 74$. Clearly, we have $\kappa = |K_1 \cap K_2| = 65$ and $k_3 = k_4 = 0$.

Fig. 6: MITM attack on 7 rounds of PRESENT.

The middle sieving. Figure 7 focuses on the middle part of the cipher. The size of the known input vector u is $m = 27$ and the size of the known output vector v is $p = 15$. The middle sbox S is formed by nine independent 4×4 sboxes, i.e. $t = 9$. For six of these small sboxes, $m = 3$ input bits and $p = 2$ output bits are known. For the other three sboxes, we have $m = 3$ and $p = 1$. As the configuration is the same for each of these two groups of sboxes, the total sieving probability equals $\pi = \pi_1^3 \pi_2^6$, where π_1 and π_2 are the sieving probability of the sbox corresponding to $p = 1$ and $p = 2$ respectively.

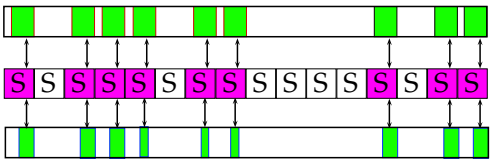


Fig. 7: Middle sieving.

An upper bound on the sieving probability π_2 can be deduced from Proposition 1 since $n = 4$, $m = 3$ and $p = 2$. Then, we have that $\pi_2 \leq 1/2$. From Prop. 4 in Section 8, we can easily deduce that equality holds in this particular case. When $p = 1$, the bound in Proposition 1 is not relevant, but Prop. 4 in Section 8 shows that $\pi_1 = 1 - \frac{1}{8} = 0.875$. Therefore, the total sieving probability is $\pi = 2^{-6.58}$. This means that in the testing phase, we only have to examine $2^{80-6.58} = 2^{73.42}$ potential key candidates and a set of $2^{80-64} = 2^{16}$ possible keys will be recovered. If two pairs of plaintext-ciphertext are known, a proportion of $\pi^2 = 2^{-13.16}$ of the keys needs to be tested, and the attack recovers the whole key, instead of a set of 2^{16} candidates.

The merging step in the attack consists in merging two lists $\mathcal{L}_A$ of size 2^6, containing elements formed by $t = 9$ groups with $m_i = 3$ bits and $\mathcal{L}_B$ of size 2^9, containing elements formed by 6 groups with $p_i = 2$ bits and 3 groups with $p_i = 1$ bits. With these parameters, the best algorithm among the ones presented in Section 2.3 is the memoryless parallel matching applied with the following parameters: $t_1 = 1$, $t_2 = 4$, where the first five groups correspond to the groups with $p = 2$. From the formula given in Section 2.3, we deduce that the time complexity of the merging step is $2^{11} + 2^{10} + 2^{10} = 2^{12}$. For each guess of the 65 bits in $K_1 \cap K_2$, the cost of merging the two lists is then 2^{12}, and we obtain in average $2^{8.42}$ solutions corresponding to the possible key candidates, among the 2^{15} initial ones.

The total time complexity of the attack is then: $2^{65}(2^6 c_F + 2^9 c_B + 2^{12}) + 2^{73.42} c_E \simeq 2^{73.42} c_E$, while the cost of the exhaustive key search is $2^{80} c_E$. The memory complexity is 2^9, and the data complexity is a single pair of known plaintext-ciphertext.

One more round with bicliques. If an 8th round is added, the differential paths in this 8th round generated by the non-common key bits are independent. Then, this last round can be covered by classical bicliques, with no additional time complexity and a data complexity of 2^6.

Experimental results. These results have been partially implemented, and we have verified the sieving part as follows: we have assumed that 48 among the 65 bits in $K_1 \cap K_2$ are

known. Then, we succeeded in recovering the 32 remaining key bits with an average predicted complexity of 2^{29} plus 2^{26} encryptions, proving that our merging phase works as predicted.

The attacks on the 128-bit version are similar to the previous ones, and the highest number of attacked rounds is 9, and 10 with bicliques, but we only gain a factor two on the exhaustive key search.

5 Application to DES

The Data Encryption Standard (DES) appeared in 1977, and was replaced as the official standard of block cipher by the AES in 2000. DES is a 16-round Feistel cipher, which encrypts 64-bit blocks with 56-bit keys. Sixteen balanced Feistel rounds are iteratively applied to the plaintext block, with a F-function composed of a layer of non-injective SBoxes followed by a bit permutation. The F-function, described in Figure 8, accommodates a 32-bit input along with

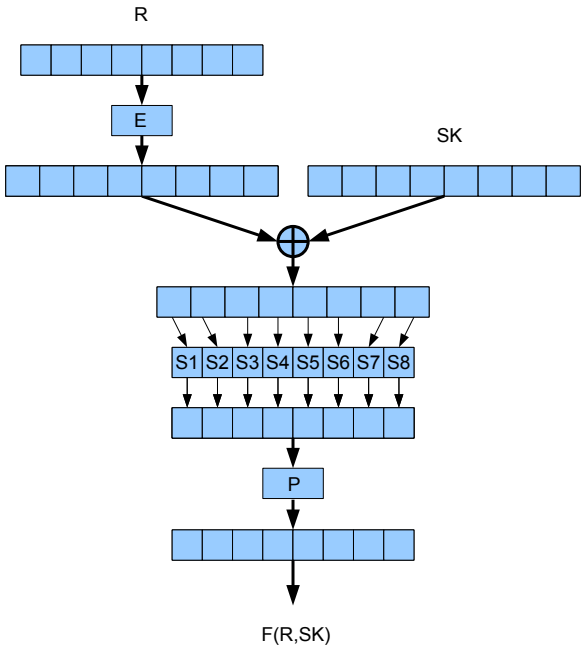
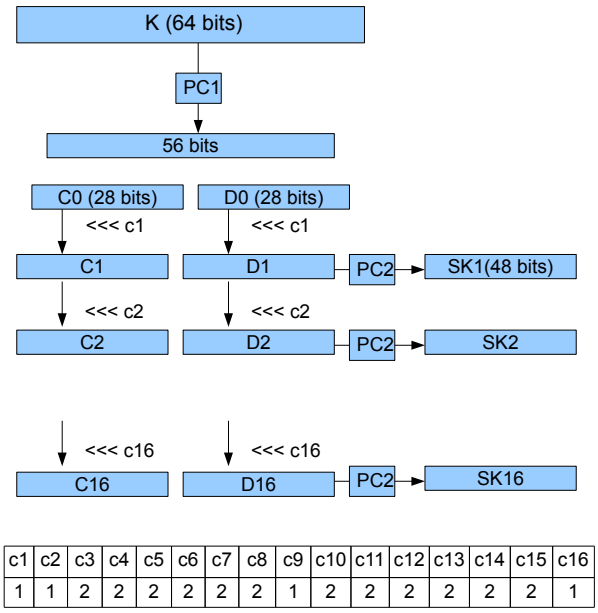


Fig. 8: F-function used in a DES round.

a 48-bit subkey. The input is expanded into 48 bits, and the expanded input is XORed with the subkey. Eight groups of 4 bits are computed by eight 6×4 S-boxes $S1$,$S2$,...,$S8$. Sixteen subkeys are derived from the key with the algorithm described in Figure 9. The effective key length is 56 bits. In the meet-in-the-middle cryptanalysis, we consider the master key as its image under $PC1$, i.e., the initial content (C_0, D_0) of the key register of Figure 9.

5.1 Previous MITM cryptanalyses of reduced DES

MITM cryptanalysis was first proposed for analyzing the extension of double DES. This cryptanalysis proved that the security of double DES would not improve on the security of



c1	c2	c3	c4	c5	c6	c7	c8	c9	c10	c11	c12	c13	c14	c15	c16
1	1	2	2	2	2	2	2	1	2	2	2	2	2	2	1

Fig. 9: Key scheduling algorithm of DES

DES. Secondly, it was used on the DES itself by Chaum and Evertse [14]. They showed that a meet-in-the-middle key recovery could be applied to six rounds and they gave an upper bound of seven DES rounds to the range of their method. Latter in [21] the efficiency of the meet-in-the-middle was improved for reduced versions of DES to 4, 5, and 6 rounds. Though there exists a meet-in-the-middle attack on 7 rounds of DES that does not start from the first round, ours is the first to reach 7 rounds starting from the beginning (and also 8).

Time complexity	4 rounds	5 rounds	6 rounds	7 rounds	8 rounds
MITM [14]	2^{35}	$2^{45.5}$	$2^{52.9}$	upper bound	
MITM partial matching [21]	2^{20}	$2^{35.5}$	$2^{51.8}$		
MITM partial matching through S-boxes (this section)				2^{53}	2^{53}

Table 1. Complexity of previous meet-in-the-middle from the first round on truncated versions of DES

5.2 Sieve-in-the-middle on 7 and 8 rounds (starting from the first one)

As this analysis is very similar to the one on PRESENT, we briefly explain here its procedure. We first determine the parameters of this attack, which uses a single plaintext-ciphertext pair. In the forward direction we guess all the key bits but the ones at positions $\{19, 26, 36, 55\}$. In the backward direction, the ones missing are $\{2, 6, 9, 21\}$. This means that $k_1 = k_2 = 52$ and $\kappa = |K_1 \cup K_2| = 48$.

With this key decomposition, we can compute 5 input bits of the sbox $S7$ in the fifth round for the forward direction, and the four bits of output of the same sbox in the backward

direction. In this case, we have $S = S7$, and so $t = 1$. Let us recall here that the DES sboxes are not bijective, but 6-bit to-4 bit sboxes. Therefore, the knowledge of all 4 output bits does not determine the input. Instead, for $m_1 = 5$ known input bits and $p_1 = 4$ known output bits, the sieving probability is $\pi = 2^{-(n'-1)} = 2^{-3}$. Indeed, Proposition 4 implies that any $(n-1, n')$-sieve has maximal probability $2^{-(n'-1)}$ if and only if there is no pair of elements at Hamming distance 1 which have both the same value under the sbox. This is the case of all DES Sboxes, which have been designed such that any two inputs which correspond to the same output always differ on at least 2 positions.

The size of both lists L_b and L_f is 2^4. We can perform an instant matching (with $L_A = L_b$ and $L_B = L_f$) for finding the $2^{4+4-3} = 2^5$ solutions with a complexity of $2^{4+1} = 2^5$. The memory complexity of this phase is determined by the size of both lists, as we do not need to store the transition tables and we can compute the output for the possible missing input bit on the fly. The total time complexity of the attack will be:

$$2^{48}(2^4 c_F + 2^4 c_B + 2^5 + 2^5 c_E) \approx 2^{53} c_E,$$

so we have won 3 bits on the exhaustive search ($2^{56} c_E$).

This attack has been implemented considering 24 of the κ bits of $(K_1 \cap K_2)$ as known and recovering the remaining 32, and we have obtained the expected complexities, verifying our theoretical approaches.

We can add one additional round with bicliques: with the previously described configuration, the paths generated by K_6 and K_5 are not independent. For that, we need to transform one of the bits of $K_1 \setminus (K_1 \cap K_2)$ into a a common bit, i.e. also included in K_2. In this case the attack with the biclique covering the 8-th round can be applied in a similar way as before, with time complexity:

$$2^{49}(2^3 c_f + 2^4 c_b + 2^4 + 2^4 c_E) \approx 2^{53} c_E,$$

and a data complexity of 2^4 plaintext-ciphertext pairs.

6 Application to PRINCE

PRINCE is a lightweight block cipher designed by Borghoff *et al.* [9]. Though being very recent, it has already waked the interest of many cryptanalysts [37, 26, 1]. The best known attacks so far on the proposed cipher, including the security analysis performed by the authors, reach 6 rounds. In particular, MITM with bicliques (without guessing the whole key) is said to reach at most 6 rounds (out of 12). In [26], a reduction of the security by one bit is presented, and in [1] an accelerated exhaustive search using bicliques is presented. Here, we describe how to build sieve-in-the-middle attacks on 8 rounds with data complexity 1 (or 2 if we want to the whole key instead of a set of candidates). In addition to the new sieve-in-the-middle technique, we use the improved method for constructing bicliques presented in Section 3.2.

6.1 Brief description of PRINCE

PRINCE operates on 64-bit blocks and uses a 128-bit key composed of two 64-bit elements, K_a and K_b. Its structure is depicted on Figure 10. PRINCE is based on the so-called FX-construction: two whitening keys $W_{in} = (K_a + K_b)$ and $W_{out} = (K'_a + K_b)$ are xored respectively to the input and to the output of a 12-round core cipher parametrized

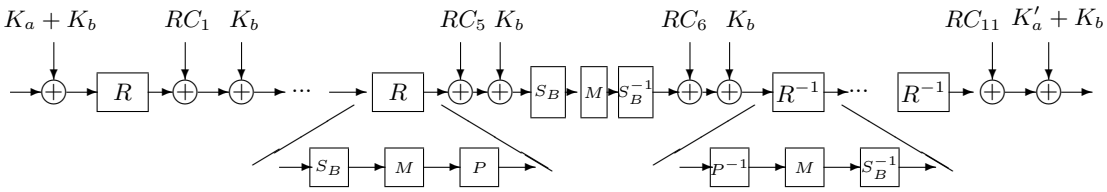


Fig. 10: Structure of PRINCE.

by K_b only. The value of K'_a involved in the post-whitening key is derived from K_a by $K'_a = (K_a \ggg 1) \oplus (K_a \gg 63)$.

The round function is composed of:

- a non-linear layer S_B corresponding to 16 parallel applications of a 4×4 sbox σ.
- a linear layer $P \circ M$, where M is the parallel application of 4 involutive mixcolumns operations on 16 bits each (defined either by $\hat{M}^{(0)}$ or by $\hat{M}^{(1)}$). This transformation is then followed by a permutation P of the 16 nibbles defined by

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

$\longrightarrow$

0	5	10	15	4	9	14	3	8	13	2	7	12	1	6	11

- the addition of a round constant RC_i and of the subkey K_b.

The first 5 rounds in PRINCE correspond to the previously described round permutation R, while the last 5 rounds are defined by the inverse permutation R^{-1}. The two middle rounds correspond to the successive applications of S_B, M and S_B^{-1}.

6.2 Sieve-in-the-middle and improved bicliques on 8 rounds

Sieve-in-the-middle on six rounds. We first describe the sieve-in-the-middle part of the attack, which covers Rounds 1 to 6 (see Figure 11). The internal state X after Round 6 is supposed to be known, as well as the plaintext. The sieving step is done with respect to a function S which covers Round 3 and the S_B level of Round 4. This middle function S can then be decomposed as four 16×16 superboxes: the colored nibbles in the middle of Figure 11 represent the nibbles belonging to the same superbox.

The 128 keybits in PRINCE are then decomposed as depicted on Figure 12:

- K_1, i.e. the keybits known in the forward direction, are represented in white and in blue in K_b and the first whitening key W_{in}. They correspond to all bits K_b and W_{in} except the 11 leftmost bits of the third 16-bit group in K_b.
- K_2, i.e. the keybits known in the backward direction, are represented in white and in red in K_b and W_{in}. They correspond to all bits of K_b and W_{in} except the leftmost nibble of K_b and the 16 bits at positions 0 and 49 to 63 in W_{in}.

It follows that the intersection $(K_1 \cap K_2)$ consists of $\kappa = 97$ information bits of (K_a, K_b): the 49 white bits in K_b and the 48 white bits in W_{in}.

The algorithm is described on Figure 11, where each nibble which contains 'K' is known in the backward computation, each nibble which contains 'k' is known in the forward computation and '1' means that there is a known bit in the nibble. The right part of the figure represents the key. We will exploit the fact that, for each 16×16 mixcolumns operation, there

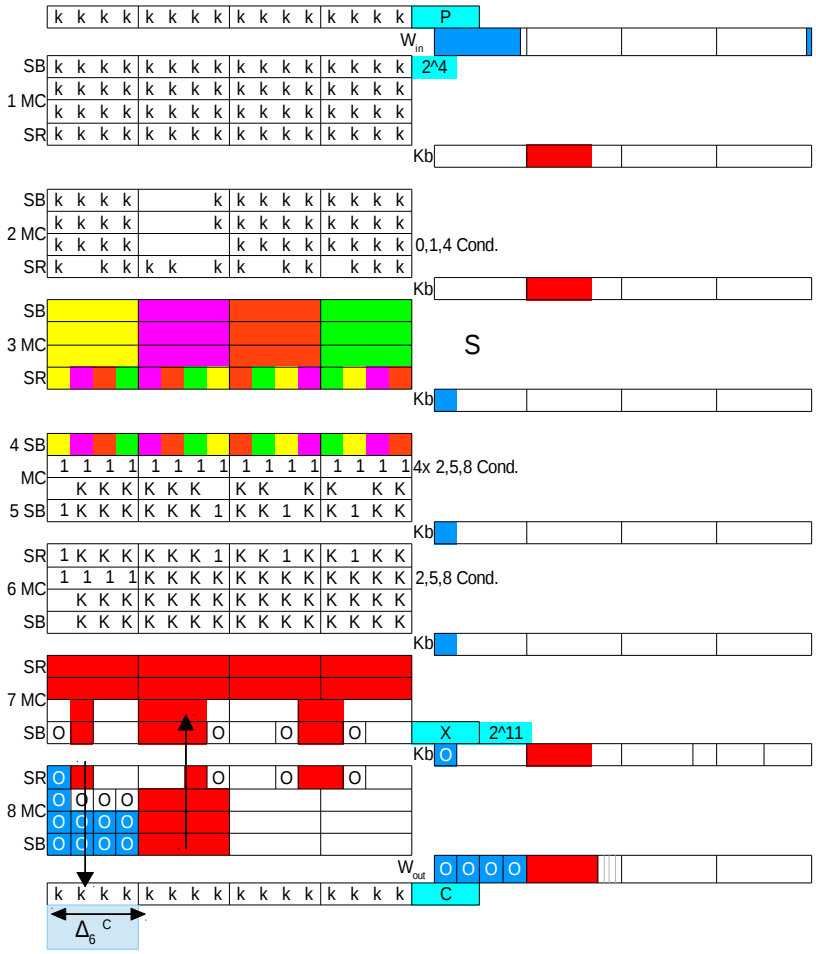


Fig. 11: Sieve-in-the-middle attack on 8 rounds of PRINCE with data complexity of 1.

exist 4 output bits (one per nibble), as well as 8 information bits of the output, which do not depend on a given input nibble. Each of these 8 information bits corresponds to the sum of two output bits. Indeed, the 16×16 transformation $\hat{M}_0$ is defined by

$$\begin{pmatrix} a_0 & a_1 & a_2 & a_3 \\ b_0 & b_1 & b_2 & b_3 \\ c_0 & c_1 & c_2 & c_3 \\ d_0 & d_1 & d_2 & d_3 \end{pmatrix} \longmapsto \begin{pmatrix} b_0+c_0+d_0 & a_1+c_1+d_1 & a_2+b_2+d_2 & a_3+b_3+c_3 \\ a_0+b_0+c_0 & b_1+c_1+d_1 & a_2+c_2+d_2 & a_3+b_3+d_3 \\ a_0+b_0+d_0 & a_1+b_1+c_1 & b_2+c_2+d_2 & a_3+c_3+d_3 \\ a_0+c_0+d_0 & a_1+b_1+d_1 & a_2+b_2+c_2 & b_3+c_3+d_3 \end{pmatrix},$$

where a, b, c and d represent the four input nibbles. Then, it can be checked for instance that the four output bits a'_0, b'_1, c'_2 and d'_3 do not depend on nibble a, as well as the eight information bits $b'_0 + c'_0$, $b'_0 + d'_0$, $a'_1 + c'_1$, $a'_1 + d'_1$, $a'_2 + b'_2$, $a'_2 + d'_2$, $a'_3 + b'_3$ and $a'_3 + c'_3$. The same situation holds for every input nibble, as also for the other mixcolumns transformation $\hat{M}_1$.

In the backward computation, from State X and K_2, we can compute 3 nibbles of each input of the mixcolumns operations at Round 5. Then, we deduce one bit in each nibble of

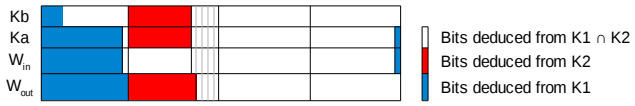


Fig. 12: Decomposition of the key in the attack on 8 rounds of PRINCE. $W_{in} = K_a \oplus K_b$ and $W_{out} = (K_a \ggg 1) \oplus (K_a \gg 63) \oplus K_b$.

the output of the middle function S, as well as 32 information bits which involve the outputs of two different superboxes. When considering $s < 4$ superboxes together, the number of information bits known is reduced to 8 if $s = 2$, and to 20 if $s = 3$.

In the forward computation, from the plaintext P and K_1, we compute three input nibbles of each superbox. From the mixcolumns operation in Round 2 whose input is partially known, we can also have 4 additional information bits on the input of the middle function S. When considering $s < 4$ superboxes together, the number of information bits known is reduced to 0 if $s = 2$ and to 1 if $s = 3$.

Then, we need to merge the two lists $\mathcal{L}_f$ and $\mathcal{L}_b$ of respective sizes 2^4 and 2^{11}. Since $m = 4 \times 12 + 4 = 52$ input bits and $p = 4 \times 4 + 32 = 48$ output bits are known, the total sieving probability π is at most $2^{64-(52+48)} = 2^{-36}$. In the following, the tables T_j providing all transitions for the four superboxes S_j are supposed to be known[1].

We are going to first apply the instant matching on the first two blocks (orange and green), i.e., we apply Algorithm 1 described on Page 6 with parameters $n_1 = n_2 = 16$ and $m_1 = m_2 = 12$ and $p_1 + p_2 = 8 + 8 = 16$. The sieving probability of these two superboxes together is then $\pi_{1,2} = 2^{32-(24+16)} = 2^{-8}$. We consider $\mathcal{L}_A = \mathcal{L}_b$ and $\mathcal{L}_B = \mathcal{L}_f$. From the corresponding formula in Section 2.3, we get that the time complexity of this step is $2^{-8}2^{4+16} + 2^{-8}2^{15} \approx 2^{12}$. With this complexity we have found $2^{15}\pi_{1,2} = 2^7$ input-output pairs of S which are valid for the first two superboxes. We can now check whether each of these pairs is also valid for the two remaining superboxes. Now, the sieving probability for the remaining part is at most $2^{-36} \times 2^{+8} = 2^{-28}$ as the total sieving probability is at most 2^{-36}.

Therefore, at the end of the merging step, for each guess of the $\kappa = 113$ bits of $(K_1 \cap K_2)$, we have a probability of $2^{7-28} = 2^{-21}$ of finding a correct configuration for the 15 remaining bits of (K_1, K_2). This means that the testing step will consider $2^{113-21} = 2^{92}$ keys, and it will recover 2^{64} possible candidates for the whole key. If two plaintext-ciphertext pairs are available, the testing step will consider $2^{92-36} = 2^{56}$ keys instead of 2^{92}, leading to performing a test over 2^{56} candidates for recovering the correct key.

Improved bicliques part. Our attack combines the previous sieve-in-the-middle algorithm with bicliques built as described in Section 3.2, without increasing the data complexity. We define K'_6 as the five nibbles corresponding to the union of the leftmost nibble of K_b and the four leftmost nibbles of the whitening key $W_{out} = (K'_a + K_b)$. Then, Δ_6^C is represented on Figure 11 by the four 'O' symbols in the line before C. Also, Δ_6^X then corresponds to the 'O'

[1]The orange and green superboxes that involve common key bits only can be computed on the fly and will be used first for the instant matching. For each pair we obtain, the whole key is already known, so we can repeat the on-the-fly procedure.

symbols in X. Then, $|\Delta_6^C| = 16$ and $|\Delta_6^X| = 16$. The remaining 'O' show the path from Δ_6^C to Δ_6^X. All 2^{20} transitions obtained when K_6' varies correspond, for each one of the 2^{16} possible values of j, to 2^4 biclique transitions from X_j to C. Then, K_5 is defined as the 11 leftmost bits of the third 16-bit group of K_b, implying that K_5 is equal to $K_2 \setminus (K_1 \cap K_2)$. The path generated in the backward direction, represented in red, is then independent from the blue path generated by K_6', and also from the path with 'O' symbols from Δ_6^C to Δ_6^X.

The complete algorithm then consists in performing an exhaustive search over the $\kappa = 97$ common bits corresponding to the white bits of K_b and W_{in} in Figure 12. The previously described bicliques determine 2^{16} states X_j, and 2^4 transitions from each X_j to C. Then, for each X_j, we examine the corresponding 2^4 values of K_6'. For those K_6', we compute forwards from the plaintext P the list of all 2^4 vectors u. It is worth noticing that even if the red bits of K_a and K_b are unknown in the forward direction, their sum is known (see Fig. 12). Similarly, the list $\mathcal{L}_b$ of all vectors v is computed backwards from the 2^{11} X^i and their associated value i for K_5. From the formula given in Section 3.2, we deduce that, for one plaintext-ciphertext pair, the time complexity is

$$\text{Time} = 2^{97}\left(2^{20} + 2^{16+11}\right) c_H + 2^{117} c_F + 2^{113} c_B + 2^{97} \times 2^{12} + 2^{-36} \times 2^{128} c_E \simeq 2^{124} c_H \ .$$

We have then gained more than four bits over the exhaustive search ($2^{128} c_E$). The memory complexity is of 2^{20}, corresponding to the precomputed table in the construction of the improved bicliques, since the transition tables for the superboxes can be computed on the fly.

7 Application to AES Biclique

In the analysis on the full-round AES-128 proposed by Bogdanov *et al.* in [6], though the whole key needs to be guessed, the authors count the number of sboxes which need to be recomputed for each guess, among all sboxes involved in one encryption. Then, despite a loop of size 2^{128}, a speed-up factor of 0.27 is obtained, leading to a complexity of $2^{126.14}$ encryptions.

Our sieve-in-the-middle technique can be applied and it may improve the complexity on some platforms. We can consider that the middle function S in our algorithm is defined by a 32×32 superbox. By precomputing and storing the possible transitions for the superbox in a table of size 2^{32}, we do not need to recompute the five sboxes included in S each time. Instead, we determine by a table-lookup whether a transition from one known input byte to 4 known output bytes is possible. If the attack is performed on a platform on which a look-up in a table of size 2^{32} is faster than five evaluations of the sbox, this variant is slightly faster than the original attack. Because of the branch number of MixColumns, the corresponding sieving probability is 2^{-8}, implying that the sieving performed is the same as in the original analysis (where a collision on 8 bits is considered). Our technique seems similar to the concept of MITM through linear relations, introduced in [25, 28]. However, in our case, we do not check linear relations, but possible transitions for a nonlinear function, since the function which provides the sieving is a superbox and involves sboxes. For this reason, this technique allows us to decrease the number of sbox evaluations: compared to an exhaustive search for the key, the proportion of sbox evaluations is now 0.203.

It is worth noticing that the precomputation of the tables has a negligible cost, as they only need to be computed once for each guess of the 2^{32} values of key bits involved in S. Also, we are able to choose an S involving a part of the key which does not need to be recomputed.

Though the improvement is very tiny, we believe that our technique can also be useful for future biclique attacks which aim at an accelerated exhaustive search for the key.

8 Sieving Probability and Related Properties of the Sbox

8.1 General properties

In this section, we focus on the general problem of theoretically estimating the sieving property provided by two subsets $I \subset \{1,\dots,n\}$ and $J \subset \{1,\dots,n'\}$, with respective sizes m and p, for a given function S from $\mathbf{F}_2^n$ into $\mathbf{F}_2^{n'}$. In particular, we provide some results on the minimal value of $(m+p)$ for which a sieve exists. In the following, S_J denotes the function from $\mathbf{F}_2^n$ into $\mathbf{F}_2^p$ corresponding to the p coordinates of S defined by J. Also, for any affine subspace W, $S_{|W}$ denotes the restriction of S to W, i.e., the function defined on W by $S_{|W}(x) = S(x)$. Obviously, $S_{|W}$ can be identified with a function of $\dim W$ input variables.

For a given input set I, V denotes the linear subspace $V = \{x \in \mathbf{F}_2^n \ : x_i = 0, i \in I\}$. Then, the sieving probability of (I,J) can be expressed in terms of the sizes of all $\mathsf{Range}(S_J)_{|u+V}$ when u varies.

Proposition 2. *Let A_k, $1 \le k \le 2^{\min(p,n-m)}$, be the number of cosets $u+V$ such that $\#\mathsf{Range}(S_J)_{|u+V} = k$ when u varies in $\mathbf{F}_2^m$. Then, the sieving probability of (I,J) is equal to*

$$\pi_{I,J} = 2^{-p} + 2^{-(p+m)} \sum_{k=2}^{2^{\min(p,n-m)}} (k-1)A_k \ .$$

Proof. The value of $\pi_{I,J}$ is deduced from the number of valid pairs (u,v), which equals

$$\sum_{k=1}^{\min(2^p,2^{n-m})} kA_k = 2^m + \sum_{k=2}^{\min(2^p,2^{n-m})} (k-1)A_k$$

where the last equality comes from the fact that the sum of all A_k equals 2^m. □

Then, we deduce the following corollary by using that $\pi_{I,J} = 2^{-p}$ if and only if $A_1 = 2^m$. This means that S_J is constant on all cosets of V.

Corollary 1. *The sieving probability of (I,J) satisfies $\pi_{I,J} \ge 2^{-p}$, with equality if and only if S_J does not depend on its inputs at positions in $\{1,\dots,n\} \setminus I$.*

Link with the branch number of S. We associate to S the (nonlinear) code $\mathcal{C}_S$ of length $(n+n')$ and of size 2^n defined by $\mathcal{C}_S = \{(x, S(x)), x \in \mathbf{F}_2^n\}$. The minimum distance of $\mathcal{C}_S$ is the lowest value of $wt(x+y) + wt(S(x)+S(y))$ for distinct x, y. It corresponds to the *branch number* of S. Obviously, when $m+p > n$, the sieving probability of any (I,J) of size (m,p) is at most $2^{n-(m+p)}$ (see Proposition 1). Now, the following proposition shows that this upper bound is tight when $(m+p)$ exceeds some bound depending on the branch number of S.

Proposition 3. *Let $I \subset \{1,\dots,n\}$ and $J \subset \{1,\dots,n'\}$ be two subsets with respective sizes m and p with $m+p \ge n$. Then, the following three statements are equivalent:*

(i) $\pi_{I,J} < 2^{n-(p+m)}$

(ii) *there exist two distinct elements x and y in $\mathbf{F}_2^n$ such that*

$$\mathsf{Supp}(x+y) \subseteq \overline{I} \text{ and } \mathsf{Supp}(S(x)+S(y)) \subseteq \overline{J}$$

(iii) *there exist some input difference of the form $a = (0_I, \alpha)$ and some output difference $b = (0_J, \beta)$ such that the entry of index (a, b) in the difference table of S is non-zero.*

Most notably, all (m,p)-sieves have probability $2^{n-(m+p)}$ if and only if $m+p > n+n'-d_{\min}$ where $d_{\min}$ is the branch number of S (i.e., the minimal distance of $\mathcal{C}_S$).

Proof. The last two statements are clearly equivalent. Then, we will prove the equivalence between the first two. For any $u \in \mathbf{F}_2^m$, the restriction of S_J to $u+V$ can take at most 2^{n-m} values. Then, $\pi_{I,J} = 2^{n-(m+p)}$ if and only if, for any $u \in \mathbf{F}_2^m$, all values of $S_J(x)$ are distinct when x varies in $u+V$. This equivalently means that there is no pair of inputs x_1 and x_2 which coincide on I (i.e., which have the same u) such that $S(x_1)$ and $S(x_2)$ coincide on all positions in J. Thus, $\pi_{I,J} < 2^{n-(m+p)}$ if and only if there exists x_1 and x_2 such that $\mathsf{Supp}(x_1+x_2) \subset \overline{I}$ and $\mathsf{Supp}(S(x_1)+S(x_2)) \subset \overline{J}$. Then, the Hamming distance between $(x_1, S(x_1))$ and $(x_2, S(x_2))$ equals $n - +n' - (m+p)$, implying that such a pair of elements exists if and only if $n+n'-(m+p) < d_{\min}$. □

For instance, the branch number of the 4×4 PRESENT sbox is equal to 3. It follows that any (m,p) sieve with $m+p \geq 6$ has probability $2^{n-(m+p)}$.

Lower bound on the minimal value of $(m+p)$. Even if the code $\mathcal{C}_S$ is a nonlinear code, its dual distance can be defined as follows (if $\mathcal{C}_S$ is linear, this definition coincides with the minimum distance of the dual code $\mathcal{C}_S^{\perp}$).

Definition 2. *Let $\mathcal{C}$ be a code of length N and size M over $\mathbf{F}_q$ and $A = (A_0, \ldots, A_N)$ be its distance distribution, i.e., $A_i = \frac{1}{M}\#\{(x,y) \in \mathcal{C}\times\mathcal{C} \ : d_H(x,y) = i\}$.*

Let $A' = (A'_0, \ldots, A'_N)$ be the image of A under the MacWilliams transform, $A'(X,Y) = A(X+(q-1)Y, X-Y)$ where $A(X,Y) = \sum_{i=0}^{N} A_i X^{N-i}Y^i$ and $A'(X,Y) = \sum_{i=0}^{N} A'_i X^{N-i}Y^i$. The dual distance *of $\mathcal{C}$ is the smallest nonzero index i such that $A'_i \neq 0$.*

The dual distance of $\mathcal{C}_S$ is a lower bound on the lowest $(m+p)$ for which an (m,p)-sieve exists. Indeed, we can use the following theorem due to Delsarte.

Theorem 1. *[18] Let $\mathcal{C}$ be a code of length N and size M over $\mathbf{F}_q$. Then, the words of $\mathcal{C}$ restricted to any t positions take all the q^t possible values exactly M/q^t times if and only if $t < d^{\perp}$ where $d^{\perp}$ is the dual distance of $\mathcal{C}$.*

Then, we derive the following result.

Theorem 2. *Let $d^{\perp}$ be the dual distance of the code $\mathcal{C}_S$. Then, for any (m,p) such that $m+p < d^{\perp}$, there is no (m,p)-sieve for S. Moreover, there exists no (m,p)-sieve for S with $m+p \leq n$ if and only if $\mathcal{C}_S$ is an MDS code, which cannot occur if S is defined over $\mathbf{F}_2$.*

Proof. The first part of the theorem is a direct consequence of Delsarte's theorem (Theorem 1).

The second part comes from the fact that, for $m+p = n$, (I,J) is not an (m,p)-sieve if and only if $(x_i, i \in I; S_j(x), j \in J)$ takes all possible values in $\mathbf{F}_q^n$ exactly once. From Delsarte's theorem, this situation occurs for all (I,J) with $m+p = n$ if and only if the dual distance

of $\mathcal{C}$ is greater than or equal to $(n+1)$. But, as noted in [18, Page 426], $d^{\perp} = n+1$ implies that the minimum distance of $\mathcal{C}$ is also maximal, i.e., $d_{\min} = n'+1$ (or equivalently that $\mathcal{C}$ is MDS). In this case, we deduce from Prop 3 that all (m,p) sieves with $m+p \geq n$ have efficiency $2^{n-(m+p)}$. □

In some scenarios, S is defined over a larger alphabet, and I and J may be defined as two sets of byte (or nibble) positions. Then, the previous theorem proves that, if the corresponding code $\mathcal{C}_S$ is an MDS code, there is no (m,p)-sieve for $m+p \leq n$, and we deduce also from Proposition 3 that all (m,p)-sieve with $m+p > n$ have probability $2^{n-(m+p)}$.

8.2 Sieving probability for some particular values of (m,p)

$(m,1)$-sieves and nonlinearity. When $p=1$, a pair $(I,\{j\})$ of size $(m,1)$ is a sieve if and only if S_j is constant on some coset $u+V$. Therefore, if $(I,\{j\})$ is a sieve, then S_j is $(n-m)$-normal, i.e. constant on an affine subspace of dimension $(n-m)$. In particular, it can be approximated by an affine function with a probability at least $\frac{1}{2}(1+2^{-m})$ [20]. It follows that, if S provides the best resistance to linear cryptanalysis for even n, then it has no sieve $(I,\{j\})$ with $|I| < \frac{n}{2}-1$. As an example, the AES Sbox does not have any $(2,1)$-sieve.

$(n-1,p)$-sieves. When $m = n-1$, the sieving probability can be easily determined by the difference table of S.

Proposition 4. *Let $I = \{1,\ldots,n\} \setminus \{\ell\}$ and let $J \subset \{1,\ldots,n'\}$ with $|J| = p$. Then,*

$$\pi_{I,J} = 2^{-(p-1)} - 2^{-(p+n)} \sum_{\beta \in \mathbf{F}_2^{n'-p}} \delta(e_\ell, (0_J, \beta)) \ ,$$

where $\delta(a,b) = |\{x \in \mathbf{F}_2^n : S(x+a)+S(x) = b\}|$ is the element of index (a,b) in the difference table of S, and e_ℓ is the input vector with a 1 at position ℓ. Thus, $(I,\{j\})$ is a sieve except if S_j is linear in x_ℓ.

Proof. From Prop. 2, we have

$$\pi_{I,J} = 2^{-p} + 2^{-(p+n-1)} A_2$$

where A_2 is the number of u such that $S_J(x)$ takes two values when x varies in $\{u, u+e_\ell\}$. We can compute A_2 from the difference table of S:

$$\begin{aligned} A_2 &= \frac{1}{2}\#\{x \in \mathbf{F}_2^n : S(x+e_\ell)+S(x) = (\alpha_J, \beta), \text{ with } \alpha_J \neq 0\} \\ &= \frac{1}{2}(2^n - \#\{x \in \mathbf{F}_2^n : S(x+e_\ell)+S(x) = (0_J, \beta)\} \\ &= \frac{1}{2}(2^n - \sum_{\beta \in \mathbf{F}_2^{n'-p}} \delta_S(e_\ell, (0_J, \beta)) \ . \end{aligned}$$

It follows that $(I,\{j\})$ is not a sieve if and only if the function $x \mapsto S_j(x+e_\ell)+S_j(x)$ is the all-one function. This equivalently means that S_j is linear in x_ℓ. □

For instance, since the branch number of the PRESENT sbox is 3, Prop. 3 implies that (m,p)-sieves with $m+p=5$ exist for this sbox. Indeed, by considering its difference table, we get that all (I,J) of size $(3,2)$ correspond to a sieving probability $\pi_{I,J} \in \{\frac{1}{2}, \frac{1}{2}-\frac{1}{32}, \frac{1}{2}-\frac{1}{16}\}$. It is worth noticing that the sieve used in the attack presented in Section 4, $I = \{0,1,2\}$ and $J = \{0,1\}$ has probability $\frac{1}{2}$. We also derive from Prop. 4 the exact sieving probability involved in the attack on the DES presented in Section 5.

9 Conclusions

The main contributions of this paper are a generic improvement of MITM attacks, the sieve-in-the-middle technique, which allows to attack more rounds, and an improved biclique construction which avoids the need of additional data. These two methods have been applied to PRESENT, DES, AES and PRINCE. Moreover, some general results on the sieving probability of an sbox are given, which allow to theoretically estimate the complexity of the attack.

A future possible line of work is to investigate some possible combinations with other existing MITM improvements: with the guess of intermediate state bits [21], or with the all-subkeys approach [24]. A promising direction would be to try to make a first selection within each of the two lists before the merging step, by keeping only the input values (resp. output values) which have the lowest probability of corresponding to a valid transition. This introduces some non-detection probability, since some correct candidates would be discarded, but the sieving would be improved. Such an approach does not seem easy, but it would surely be a big step forward for further improving MITM attacks.

Acknowledgements

We thank Dmitry Khovratovich for his valuable comments, and all CryptoExperts members for their kindness and hospitality.

References

1. Farzaneh Abed, Eik List, and Stefan Lucks. On the Security of the Core of PRINCE Against Biclique and Differential Cryptanalysis. Cryptology ePrint Archive, Report 2012/712, 2012. `http://eprint.iacr.org/2012/712`.
2. Martin R. Albrecht and Carlos Cid. Algebraic Techniques in Differential Cryptanalysis. In *FSE 2009*, volume 5665 of *Lecture Notes in Computer Science*, pages 193–208. Springer, 2009.
3. Kazumaro Aoki and Yu Sasaki. Preimage Attacks on One-Block MD4, 63-Step MD5 and More. In *Selected Areas in Cryptography - SAC 2008*, volume 5381 of *Lecture Notes in Computer Science*, pages 103–119. Springer, 2008.
4. Kazumaro Aoki and Yu Sasaki. Meet-in-the-Middle Preimage Attacks Against Reduced SHA-0 and SHA-1. In *CRYPTO 2009*, volume 5677 of *Lecture Notes in Computer Science*, pages 70–89. Springer, 2009.
5. Céline Blondeau and Benoît Gérard. Multiple Differential Cryptanalysis: Theory and Practice. In *FSE 2011*, volume 6733 of *Lecture Notes in Computer Science*, pages 35–54. Springer, 2011.
6. Andrey Bogdanov, Dmitry Khovratovich, and Christian Rechberger. Biclique Cryptanalysis of the Full AES. In *ASIACRYPT 2011*, volume 7073 of *Lecture Notes in Computer Science*, pages 344–371. Springer, 2011.
7. Andrey Bogdanov, Lars R. Knudsen, Gregor Leander, Christof Paar, Axel Poschmann, Matthew J. B. Robshaw, Yannick Seurin, and C. Vikkelsoe. PRESENT: An Ultra-Lightweight Block Cipher. In *CHES 2007*, volume 4727 of *Lecture Notes in Computer Science*, pages 450–466. Springer, 2007.
8. Andrey Bogdanov and Christian Rechberger. A 3-Subset Meet-in-the-Middle Attack: Cryptanalysis of the Lightweight Block Cipher KTANTAN. In *Selected Areas in Cryptography - SAC 2010*, volume 6544 of *Lecture Notes in Computer Science*, pages 229–240. Springer, 2010.
9. Julia Borghoff, Anne Canteaut, Tim Güneysu, Elif B. Kavun, Miroslav Knezevic, Lars R. Knudsen, Gregor Leander, Ventzislav Nikov, Christof Paar, Christian Rechberger, Peter Rombouts, Søren S. Thomsen, and Tolga Yalçin. PRINCE - A Low-Latency Block Cipher for Pervasive Computing Applications. In *ASIACRYPT 2012*, volume 7658 of *Lecture Notes in Computer Science*, pages 208–225. Springer, 2012.
10. Julia Borghoff, Lars R. Knudsen, Gregor Leander, and Søren S. Thomsen. Cryptanalysis of PRESENT-Like Ciphers with Secret S-Boxes. In *FSE 2011*, volume 6733 of *Lecture Notes in Computer Science*, pages 270–289. Springer, 2011.
11. Charles Bouillaguet, Patrick Derbez, Orr Dunkelman, Pierre-Alain Fouque, Nathan Keller, and Vincent Rijmen. Low-data complexity attacks on AES. *IEEE Transactions on Information Theory*, 58(11):7002–7017, 2012.

12. Charles Bouillaguet, Patrick Derbez, and Pierre-Alain Fouque. Automatic Search of Attacks on Round-Reduced AES and Applications. In *CRYPTO 2011*, volume 6841 of *Lecture Notes in Computer Science*, pages 169–187. Springer, 2011.
13. Billy Bob Brumley, Risto M. Hakala, Kaisa Nyberg, and Sampo Sovio. Consecutive S-box Lookups: A Timing Attack on SNOW 3G. In *Information and Communications Security - ICICS 2010*, volume 6476 of *Lecture Notes in Computer Science*. Springer, 2010.
14. David Chaum and Jan-Hendrik Evertse. Crytanalysis of DES with a Reduced Number of Rounds: Sequences of Linear Factors in Block Ciphers. In *CRYPTO'85*, volume 218 of *Lecture Notes in Computer Science*, pages 192–211. Springer, 1985.
15. Huiju Cheng, Howard M. Heys, and Cheng Wang. PUFFIN: A Novel Compact Block Cipher Targeted to Embedded Digital Systems. In *DSD*, pages 383–390. IEEE, 2008.
16. Joo Yeon Cho. Linear Cryptanalysis of Reduced-Round PRESENT. In *CT-RSA 2010*, volume 5985 of *Lecture Notes in Computer Science*, pages 302–317. Springer, 2010.
17. Baudoin Collard and François-Xavier Standaert. A Statistical Saturation Attack against the Block Cipher PRESENT. In *CT-RSA 2009*, volume 5473 of *Lecture Notes in Computer Science*. Springer, 2009.
18. Philippe Delsarte. Four fundamental parameters of a code and their combinatorial signifiance. *Information and Control*, 23(5):407–438, December 1973.
19. Itai Dinur, Orr Dunkelman, Nathan Keller, and Adi Shamir. Efficient Dissection of Composite Problems, with Applications to Cryptanalysis, Knapsacks, and Combinatorial Search Problems. In *CRYPTO 2012*, volume 7417 of *Lecture Notes in Computer Science*, pages 719–740. Springer, 2012.
20. Hans Dobbertin. Construction of Bent Functions and Balanced Boolean Functions with High Nonlinearity. In *FSE'94*, volume 1008 of *Lecture Notes in Computer Science*, pages 61–74. Springer, 1994.
21. Orr Dunkelman, Gautham Sekar, and Bart Preneel. Improved Meet-in-the-Middle Attacks on Reduced-Round DES. In *INDOCRYPT 2007*, volume 4859 of *Lecture Notes in Computer Science*, pages 86–100. Springer, 2007.
22. Jian Guo, San Ling, Christian Rechberger, and Huaxiong Wang. Advanced Meet-in-the-Middle Preimage Attacks: First Results on Full Tiger, and Improved Results on MD4 and SHA-2. In *ASIACRYPT 2010*, volume 6477 of *Lecture Notes in Computer Science*, pages 56–75. Springer, 2010.
23. Takanori Isobe. A Single-Key Attack on the Full GOST Block Cipher. In *FSE 2011*, volume 6733 of *Lecture Notes in Computer Science*, pages 290–305. Springer, 2011.
24. Takanori Isobe and Kyoji Shibutani. All Subkeys Recovery Attack on Block Ciphers: Extending Meet-in-the-Middle Approach. In *Selected Areas in Cryptography - SAC 2012*, volume 7707 of *Lecture Notes in Computer Science*, pages 202–221. Springer, 2012.
25. Takanori Isobe and Kyoji Shibutani. Security Analysis of the Lightweight Block Ciphers XTEA, LED and Piccolo. In *Australasian Conference on Information Security and Privacy - ACISP 2012*, volume 7372 of *Lecture Notes in Computer Science*, pages 71–86. Springer, 2012.
26. Jérémy Jean, Ivica Nikolic, Thomas Peyrin, Lei Wang, and Shuang Wu. Security Analysis of PRINCE. In *FSE 2013*, Lecture Notes in Computer Science. Springer, 2013. To appear.
27. Stéphanie Kerckhof, Baudoin Collard, and François-Xavier Standaert. FPGA Implementation of a Statistical Saturation Attack against PRESENT. In *AFRICACRYPT 2011*, volume 6737 of *Lecture Notes in Computer Science*, pages 100–116. Springer, 2011.
28. Dmitry Khovratovich, Gaëtan Leurent, and Christian Rechberger. Narrow-Bicliques: Cryptanalysis of Full IDEA. In *EUROCRYPT 2012*, volume 7237 of *Lecture Notes in Computer Science*, pages 392–410. Springer, 2012.
29. Dmitry Khovratovich, María Naya-Plasencia, Andrea Röck, and Martin Schläffer. Cryptanalysis of *Luffa* v2 Components. In *Selected Areas in Cryptography - SAC 2012*, volume 6544 of *Lecture Notes in Computer Science*, pages 388–409. Springer, 2010.
30. Dmitry Khovratovich, Christian Rechberger, and Alexandra Savelieva. Bicliques for Preimages: Attacks on Skein-512 and the SHA-2 Family. In *FSE 2012*, volume 7549 of *Lecture Notes in Computer Science*, pages 244–263. Springer, 2012.
31. Lars R. Knudsen, Gregor Leander, Axel Poschmann, and Matthew J. B. Robshaw. PRINTcipher: A Block Cipher for IC-Printing. In *CHES 2010*, volume 6225 of *Lecture Notes in Computer Science*, pages 16–32. Springer, 2010.
32. Jorge Nakahara, Pouyan Sepehrdad, Bingsheng Zhang, and Meiqin Wang. Linear (Hull) and Algebraic Cryptanalysis of the Block Cipher PRESENT. In *Cryptology and Network Security - CANS 2009*, volume 5888 of *Lecture Notes in Computer Science*. Springer, 2009.
33. María Naya-Plasencia. How to Improve Rebound Attacks. In *CRYPTO 2011*, volume 6841 of *Lecture Notes in Computer Science*, pages 188–205. Springer, 2011.

34. Kenji Ohkuma. Weak Keys of Reduced-Round PRESENT for Linear Cryptanalysis. In *Selected Areas in Cryptography - SAC 2009*, volume 5867 of *Lecture Notes in Computer Science*, pages 249–265. Springer, 2009.
35. Onur Özen, Kerem Varici, Cihangir Tezcan, and Çelebi Kocair. Lightweight Block Ciphers Revisited: Cryptanalysis of Reduced Round PRESENT and HIGHT. In *Australasian Conference on Information Security and Privacy - ACISP 2009*, volume 5594 of *Lecture Notes in Computer Science*, pages 90–107. Springer, 2009.
36. Yu Sasaki. Meet-in-the-Middle Preimage Attacks on AES Hashing Modes and an Application to Whirlpool. *IEICE Transactions*, 96-A(1):121–130, 2013.
37. Hadi Soleimany, Céline Blondeau, Xiaoli Yu, Wenling Wu, Kaisa Nyberg, Huiling Zhang, Lei Zhang, and Yanfeng Wang. Reflection Cryptanalysis of PRINCE-like Ciphers. In *FSE 2013*, Lecture Notes in Computer Science. Springer, 2013. To appear.
38. Meiqin Wang. Differential Cryptanalysis of Reduced-Round PRESENT. In *AFRICACRYPT 2008*, volume 5023 of *Lecture Notes in Computer Science*, pages 40–49. Springer, 2008.
39. Muhammad Reza Z'aba, Håvard Raddum, Matthew Henricksen, and Ed Dawson. Bit-Pattern Based Integral Attack. In *FSE 2008*, volume 5086 of *Lecture Notes in Computer Science*, pages 363–381. Springer, 2008.