Université Libre de Bruxelles

Biclique Attack on AES-128

Students:
AMANOR Deborah
HASSANI Mortaza

Supervisor:
Prof. VAN ASSCHE Gilles

December 15, 2024

Contents

$$P \xrightarrow[g_1]{K[i,\cdot]} v_i \tag{2}$$

2. **Backward Computation** The adversary decrypts the known ciphertext using all possible values of the second key subset 2^d. The Intermediate values V_j are computed and stored alongside their respective keys

$$g_2 = DEC_{k_2}(C) \tag{3}$$

$$v_j \xleftarrow[g_2]{K[\cdot,j]} C \tag{4}$$

3. **Matching** The adversary compares the intermediate values, V_i and V_j to find a match. If $V_i = V_j$, then $K[i,j]$ are the candidate key pairs

The basic meet-in-the-middle attack has clear limitations in AES cipher cryptanalysis since an intermediate value can be found for a very small number of rounds only. Biclique cryptanalysis overcomes these limitations by introducing a more advanced structure, enabling attacks on the full AES.

3 The Biclique Attack

A **biclique** is a mathematical structure used in cryptanalysis of block ciphers to efficiently explore relationships between keys, plaintexts, and ciphertexts in a cipher. It connects 2^d intermediate states $\{S_j\}$ to 2^d Ciphertexts $\{C_i\}$ using 2^{2d} keys represented as $K[i,j]$. Each key $K[i,j]$ maps an intermediate state $\{S_j\}$ to a ciphertext $\{C_i\}$ through a subcipher f, mathematically expressed as [1]:

$$\forall i,j : S_j \xrightarrow[f]{K[i,j]} C_i \tag{5}$$

$$C_i = f_{K[i,j]}(S_j), \forall i,j \in \{0,\dots,2^d-1\} \tag{6}$$

A biclique is then said to be d-dimensional if it connects 2^d intermediate states to 2^d ciphertexts. The dimension d determines the size and complexity of the structure.

3.1 Preliminary Steps

To conduct the attack, the adversary performs two preparatory steps; Key partitioning and Cipher splitting.

1. **Key Partitioning:** The adversary firstly partitions the key space of the cipher into subsets or groups of size 2^{2d} for some d, where the keys are represented as $K[i,j]$ in a matrix $2^d \times 2^d$ similar to that of the MITM [1].

2. **Cipher splitting** The adversary then splits the cipher into two subciphers, f and g such that the encryption process e can be expressed as:

$$e = f \circ g \tag{7}$$

Where g maps the plaintext P to an intermediate state S, and f maps the intermediate state S to the ciphertext C [1].

3.2 Constructing bicliques

After completing the preliminary steps, the next phase involves constructing the biclique. Bogdanov et al. proposed two primary methods for biclique construction: Independent Related-Key Differentials and Interleaving Related-Key Differentials.

However, for the purpose of this study, we will focus only on the Independent Related-Key Differentials approach.

3.2.1 Naïve Approach or Bruteforce

A straightforward way to construct the biclique is to use the naive approach or brute-force method. This can be achieved when the adversary maps 2^d intermediate states to 2^d ciphertexts and then derives a key $K[i,j]$ for each intermediate state-ciphertext pair shown in equation (5). However, this involves evaluating the cipher for all 2^{2d} possible key pairs thus increasing the computational complexity because it takes a lot of time. A much more efficient way is for the adversary to choose the keys in advance and require them to conform to specific differentials [1]:

3.2.2 Independent Related Key Differentials

This approach exploits differences in keys and how they propagate through the cipher. In this case, two types of differentials are used over the subcipher f, Δ_i-differentials and ∇_j-differentials.

- Δ_i-differentials: shows a difference in the output Δi under a key difference ΔK_i when there is no difference in the starting state. This means that a specific difference in the key (ΔK_i) produces a predictable difference (Δ) in the ciphertext:

$$0 \xrightarrow[f]{\Delta_i^K} \Delta_i \tag{8}$$

- ∇_j-differentials: A difference in the input ∇j and the key ∇K_j reveals no difference in the output ciphertext:

$$\nabla_j \xrightarrow[f]{\nabla_j^K} 0 \tag{9}$$

To construct the biclique, a base key $K[0,0]$ is chosen which maps the intermediate states S_0 to the ciphertext C_0 over the subcipher f.

$$C_0 = f_{K[0,0]}(S_0) \tag{10}$$

The two sets of related-key differentials ΔK_i and ∇K_j are combined by an XOR operation only if the trails of Δ_i-differentials do not share active non-linear components such as S-boxes with the trails of ∇_j-differentials. Thus we obtain a $2^d \times 2^d$ matrix of the keys i.e (Δ_i, ∇_j):

$$\nabla_j \xrightarrow[f]{\Delta_i^K \oplus \nabla_j^K} \Delta_i \text{ for } i,j \in \{0,\dots,2^d-1\}. \tag{11}$$

If the trails of the differentials do not share any active non-linear components, the differentials are completely independent and can be directly combined. The result is combined with the base key, $K[0,0]$ and initial intermediate state S_0, ciphertext pair C_0.

$$S_0 \oplus \nabla_j \xrightarrow[f]{K[0,0]\oplus\Delta_i^K\oplus\nabla_j^K} C_0 \oplus \Delta_i. \tag{12}$$

The result is a biclique where the subsequent intermediate states, ciphertext and keys can be obtained by:

$$\begin{aligned} S_j &= S_0 \oplus \nabla_j, \\ C_i &= C_0 \oplus \Delta_i, \text{ and} \\ K[i,j] &= K[0,0] \oplus \Delta_i^K \oplus \nabla_j^K. \end{aligned} \tag{13}$$

This construction satisfies the biclique condition where every key $K[i,j]$ maps an intermediate state S_j to a ciphertext C_i and get a d-dimensional biclique. The independence of the related-key differentials ensures that the biclique can be efficiently constructed with computational complexity of $2 \cdot 2^d$ evaluations of the subcipher f, instead of 2^{2d} that the naive approach provides.

3.3 Steps of the General Biclique Attack [1]

After successful preparation, the adversary moves on to implement the attack in the following steps;

(Step1): Constructing the Biclique For each group of keys, $K[i,j]$, the adversary constructs a biclique structure which maps 2^d intermediate states $\{S_j\}$ to 2^d ciphertexts $\{C_i\}$. This structure is based on what was discussed in the section *Constructing bicliques.*

$$\begin{matrix} S_0 & K[0,0] & C_0 \\ \vdots & \vdots & \vdots \\ S_{2^d-1} & K[2^d-1, 2^d-1] & C_{2^d-1} \end{matrix} \tag{14}$$

$$\forall i,j : S_j \xrightarrow[f]{K[i,j]} C_i \tag{15}$$

(Step2): Obtain the data The adversary takes the possible ciphertexts C_i and passes it through the decryption oracle. With the secret key K_{secret} unknown to the adversary, the oracle decrypts the ciphertext C_i and returns the corresponding set of 2^d plaintexts.

$$C_i \xrightarrow[\epsilon^{-1}]{\text{decryption oracle}} P_i. \tag{16}$$

(Step3): Meet-In-The-Middle For each key $K[i,j]$ in the group, the adversary maps the plaintexts obtained in step 2 to their respective intermediate state S_j using the first subcipher g. Simultaneously, the adversary computes the **ciphertexts** backward to their intermediate states S_j using the second subcipher f. Using the MITM approach, the adversary tries to match the forward and backward computations at the intermediate state S_j .

$$\exists i,j : P_i \xrightarrow[g]{K[i,j]} S_j. \tag{17}$$

$$\exists i, j : S_j \xleftarrow[f]{K[i,j]} C_i. \tag{18}$$

(Step4): Matching with Precomputations For each key candidate, $K[i, j]$, the adversary evaluates the cipher directly to check if the computed intermediate states and ciphertexts match the expected results for the given plaintext-ciphertext pair. A valid pair proposes $K[i, j]$ as a key candidate.

3.3.1 Improvement with Precomputation

The matching process in step 4 can be significantly improved with precomputations.The adversary precomputes and stores in memory the partial results for the forward and backward computations:

- **Forward computation:** Compute the intermediate states S_j for all possible plaintexts P_i using a fixed key $K[i, 0]$.
- **Backward computation:** Compute the intermediate states S_j for all ciphertexts C_i using a fixed key $K[0, j]$.

$$\text{for all } i \quad P_i \xrightarrow{K[i,0]} \vec{v} \quad \text{and} \quad \text{for all } j \quad \vec{v} \xleftarrow{K[0,j]} S_j \tag{19}$$

Instead of recalculating all intermediate states for each key $K[i, j]$ from scratch, the adversary **recomputes only the parts of the cipher that differ** from the precomputed results. The amount of recalculation depends on the diffusion properties of both internal rounds and the key schedule of the cipher. This approach significantly reduces the number of operations compared to what was presented in (step 4).

4 Biclique Attack on the Full AES-128 Cipher

The cipher we chose to assess the biclique attack on is the AES-128. As presented earlier, in 2011 Bogdanov et al [1] attacked the full AES with computational operation of $2^{126.18}$ slightly efficient than 2^{128} operations of the exhaustive key search. The attack uses the independent-biclique approach which combines related key-differentials to optimize key recovery.

4.1 Brief Description of AES-128

The AES-128 block cipher consists of 128 bit-internal states and uses a 128-bit key, each represented by a 4×4 byte matrix. The cipher performs 10 rounds of encryption, where the plaintext is xored with the keys. The subkeys for each round are derived from the master key through key scheduling.

Each round consists of four transformations:

- **SubBytes**: Uses an S-box to provide non-linear transformation.
- **ShiftRows**: Provides diffusion by rotating bytes in each row of the matrix to the left.

- **MixColumns**: Combines bytes within each column to produce new values for further diffusion.
- **AddRoundKey**: Adds a subkey derived from the master key.

In the last round, the **MixColumns** operation is omitted. To conduct the biclique attack on AES 128, the authors [1] focused on two internal states in each round: the state before the **SubBytes** and the state after **MixColumns**.

4.1.1 Paradigms of Key Recovery

In the developing the attack, the authors [1] presented two paradigms for key recovery after successful construction of the bicliques. Long Biclique and Independent Biclique. The optimal choice depends on the cryptographic primitive, its diffusion properties and the key schedule. The AES attack uses the Independent Biclique but we will briefly describe both paradigms below.

4.1.2 Independent Biclique [1]

This technique exploits the diffusion properties of the cipher rather than the differential properties. It does not aim to construct the longest biclique. Instead, it proposes constructing shorter bicliques with high dimensions using independent related-key differentials. The independent biclique provides the following advantages:

1. Low Data Complexity: Since the biclique covers a small portion of the cipher, the adversary gains more flexibility to impose constraints on the ciphertext. This allows the adversary to restrict the ciphertext to a smaller, specific set, reducing the data complexity of the attack.
2. Efficient Computation: The approach reduces the complexity of constructing the biclique itself since it leverages precomputations to match intermediate states.

4.1.3 Long Biclique [1]

Assuming there are r rounds of the primitive, the adversary applies the MITM attack to m number of rounds to recover m partial keys. The long biclique approach aims to construct a biclique over the remaining $r - m$ rounds, to recover the full keys. However, the disadvantage of this paradigm is that the construction of bicliques over many rounds is difficult due to diffusion and non-linear properties of the cipher, limiting the total number of rounds that can be attacked.

4.2 Steps of the Key Recovery for the Full AES-128 using Independent Bicliques

The goal of the adversary is to recover the full AES-128 key. In this attack, rounds 1 to 7 is attacked using the MITM and rounds 8 to 10 using the independent biclique. Details of the attack are outlined in the steps below;

(Step1): Key Partitioning [1]

The adversary partitions the 2^{128} key space into smaller groups. The focus is on subkey 8 (K_8) because the biclique attack starts from round 8. The AES master key maps bijectively (one-to-one) to all the subkeys including K_8. This means that the subkey 8 can directly map back to the master key K.

The key partitioning step is as follows:

(a) The adversary fixes 2 bytes (2^{16}) of the key space, K_8 (2^{128}) to 0. The remaining 14 bytes are allowed to take on possible values, i.e 2^{112} possible values. This defines 2^{112} base keys each representing a group of keys.

In the matrix below, the two fixed bytes are represented with 0 and the remaining 14 bytes are marked with $*$.

$$K_8 = \begin{pmatrix} * & * & * & 0 \\ 0 & * & * & * \\ * & * & * & * \\ * & * & * & * \end{pmatrix}$$

(b) Each of the 2^{112} possible values defines a unique base key $K[0,0]$, creating 2^{112} groups.

$$\text{Base key } K[0,0] = \begin{pmatrix} * & * & * & 0 \\ 0 & * & * & * \\ * & * & * & * \\ * & * & * & * \end{pmatrix}$$

$$\text{Base key } K[1,0] = \begin{pmatrix} * & * & * & 0 \\ 0 & * & * & * \\ * & * & * & * \\ * & * & * & * \end{pmatrix}$$

$$\vdots$$

$$\text{Base key } K[2^{112}-1, 2^{112}-1] = \begin{pmatrix} * & * & * & 0 \\ 0 & * & * & * \\ * & * & * & * \\ * & * & * & * \end{pmatrix}$$

(c) Starting from each base key $K[0,0]$, the adversary introduces small differences i and j in the two fixed bytes. Each byte difference i or j can take 2^8 values. Since there are 2 bytes, the total combinations are: $2^8 K[i,0]$ and $2^8 K[0,j]$ which gives 2^{16} keys $K[i,j]$ per group. Meaning for each base key, there are a group of 2^{16} keys.

(Step2): Biclique Construction [1]

The adversary now constructs a 3-round biclique using the combined related key differentials discussed in **Constructing bicliques**. The requirement for using this technique is that the forward- and backward-differential trails that need to be combined, do not share any active non-linear elements. This is achieved through the key partitioning step where the differential trails Δ_i using the keys $K[i,0]$ never share any active S-boxes, with the differential trails ∇_j using the key $K[0, j]$ [2]. The biclique is constructed as follows:

(a) With the chosen base key $K[0, 0]$ from step 1, the adversary fixes the ciphertext C_0 to 0 and derives the intermediate state, S_0 by performing an inverse round function on C_0:

$$C_0 = 0 \tag{20}$$

$$S_0 = f^{-1}_{K[0,0]}(C_0) \tag{21}$$

(b) The adversary combines the two differentials Δ_i and ∇_j to generate an 8×8 biclique i.e an 8-dimension biclique. The adversary observes that the Δ_i-differentials only affect 12 bytes of the ciphertext, leaving some bytes unchanged. For example Bytes C_0, C_1, C_4, and C_{13} remain constant. Similarly, the bytes C_{10} and C_{14} are always equal due to specific properties of the key differences, ΔK_i.

(Step3): Forward and Backward Computations [1]

(a) **Forward Computation:** For each plaintext P_i, the adversary computes the intermediate state v under different keys $K[i, j]$. The influence of the key difference $K[i, j]$ and $K[i, 0]$ determines the changes to the intermediate states.

$$P_i \xrightarrow{K[i,j]} v \qquad P_i \xrightarrow{K[i,0]} v_i' \tag{22}$$

(b) **Backward Computation:** For the Ciphertext S_j, the adversary computes the intermediate state v. The backward computation is determined by the influence of the difference between keys $K[i, j]$ and $K[0, j]$.

$$\overleftarrow{v} \xleftarrow{K[i,j]} S_j \qquad \overleftarrow{v}_j \xleftarrow{K[0,j]} S_j \tag{23}$$

Due to **low diffusion** in the AES key schedule, only a small number of bytes i.e. 9 bytes differ, which minimizes the recomputation cost.

(Step4): Matching over 7 rounds [1]

The adversary checks whether the secret key K_{secret} belongs to the key group $K[i, j]$ using a **partial matching approach**:

(a) Precompute and store intermediate states v.

(b) For each candidate key $K[i, j]$, recompute only the parts of the cipher that differ from the precomputed results.

To attain a candidate key, the adversary matches the **forward** intermediate states from plaintexts with the **backward** intermediate states from ciphertexts at the same position v.

4.3 Results of the attack

The Biclique attack on the full AES-128 achieves a key recovery with computational complexity about $2^{126.18}$, data complexity 2^{88}, memory complexity 2^8, and success probability of 1. [1]

5 Reduced Implementation of the Attack on AES

In this section, we present a reduced implementation of the biclique attack on AES-128. The implementation is designed to demonstrate the key concepts and steps involved in the biclique attack, as described in the previous sections. This implementation demonstrates a simplified version of the biclique attack on AES-128. The implementation is written in C and consists of several key components that work together to perform the attack. The source code is available in the Github repository.

5.1 Implementation Structure

The implementation is organized into three main files:

- `AES.h/c`: Contains the core AES operations including encryption, decryption, and key scheduling
- `Biclique.h/c`: Implements the biclique attack functionality
- `Bicliquemain.c`: Orchestrates the attack execution and testing

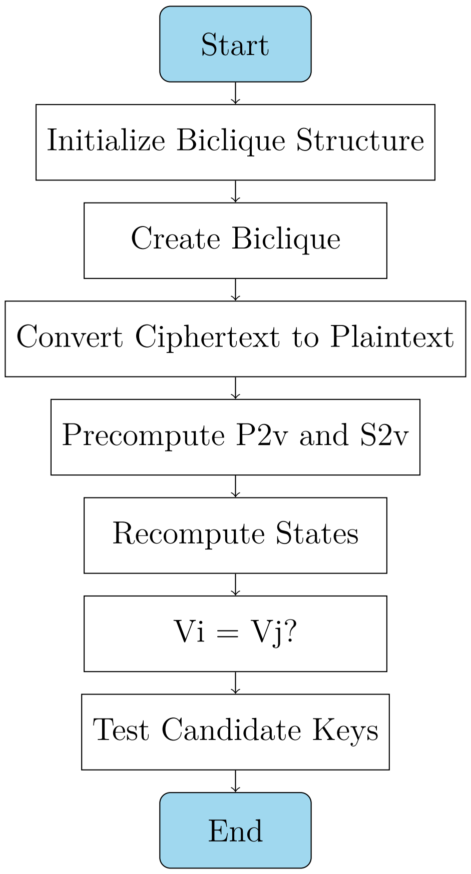

Figure 2: Biclique Structure

5.2 Key Components

5.2.1 Biclique Structure

The biclique structure is represented by the `BICL` struct, which maintains:

- Current state (`S[SSize]`)
- Ciphertext (`C[CSize]`)
- Plaintext (`P[PSize]`)
- Biclique key (`BicliqueKey[KeySize]`)
- Differential transitions (`Delta_i[CSize]` and `Nabra_j[SSize]`)
- Forward and backward states (`f_state` and `b_state`)

5.2.2 Biclique Construction

The biclique construction is implemented in the `createBiclique` function, which:

1. Generates a base key using `KeyCreate`
2. Constructs differential transitions using `Delta_i_Key` and `Nabra_j_Key`
3. Maps intermediate states to ciphertexts using the constructed differentials

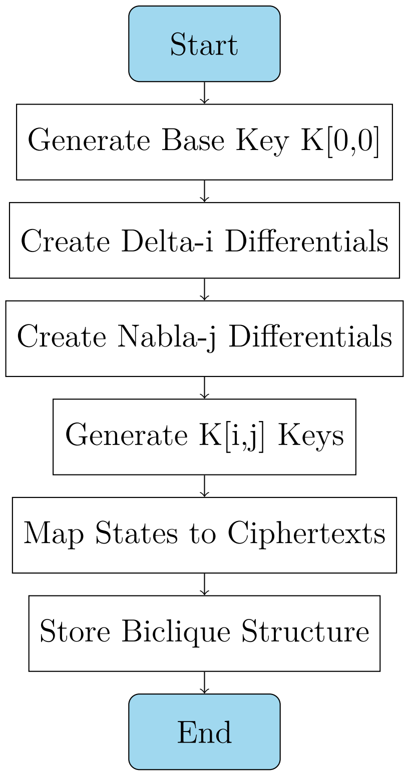
Start
Generate Base Key K[0,0]
Create Delta-i Differentials
Create Nabla-j Differentials
Generate K[i,j] Keys
Map States to Ciphertexts
Store Biclique Structure
End

Figure 3: Biclique Attack

5.3 Attack Implementation

The attack follows these main steps:

5.3.1 1. Precomputation Phase

Two main precomputation functions are implemented:

- `precompute_P2v`: Computes forward transitions from plaintexts to intermediate state
- `precompute_S2v`: Computes backward transitions from ciphertext to intermediate state

The precomputation reduces the computational complexity by storing intermediate results that can be reused during the attack.

5.3.2 2. Recomputation Phase

The `recompute` function implements the recomputation step, which:

- Recomputes key schedules using `KeyRecompute`
- Updates forward states using `RecomputeF`
- Updates backward states using `RecomputeB`

5.3.3 3. Key Testing

The `testCandidateKeys` function verifies potential key candidates by:

1. Decrypting ciphertexts using candidate keys
2. Comparing results with known plaintexts
3. Identifying matching keys as potential candidate keys

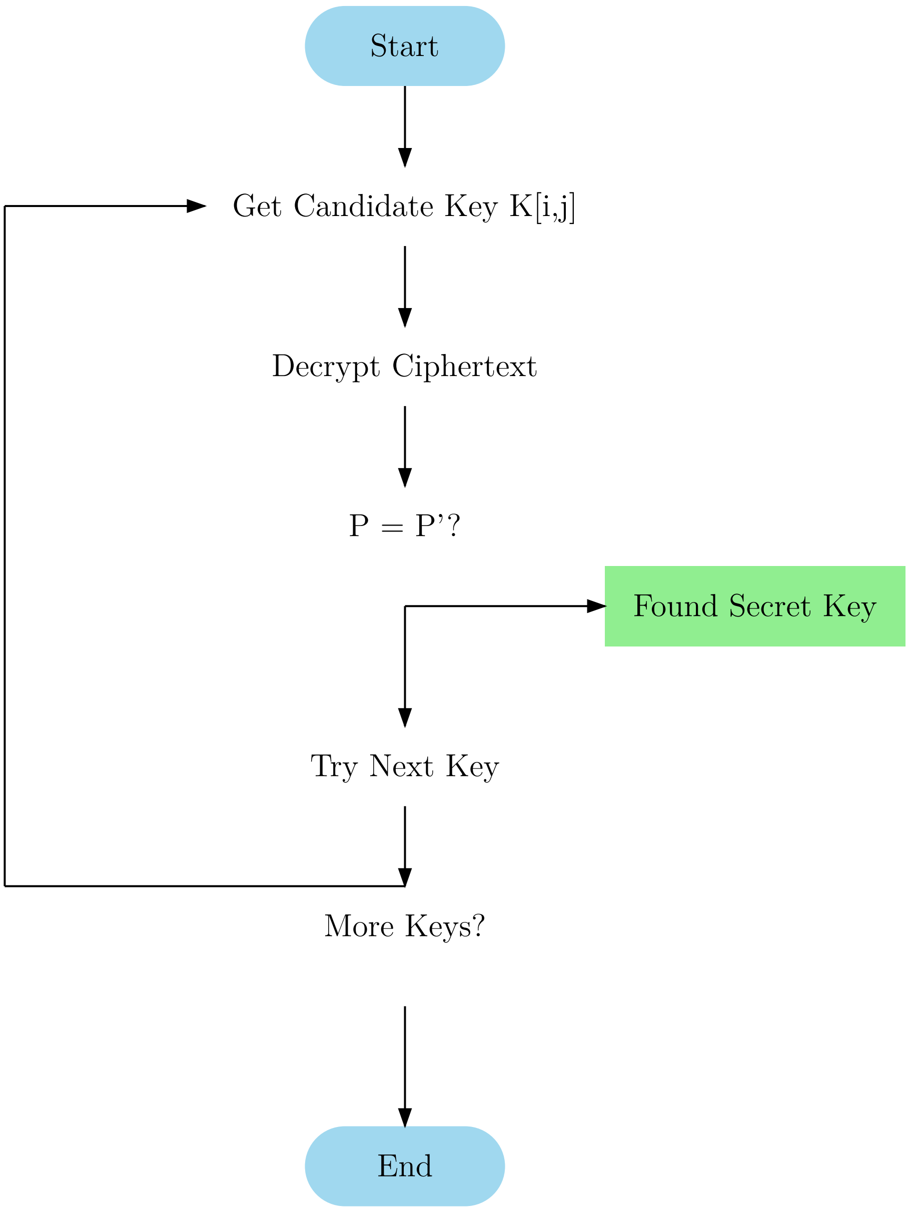

Figure 4: Key Recovery Flow

```
%% Output sample: Vi=Vj
---
Possible key here: K[81][36]
9a,de,ca,2,fa,37,9b,bc,2b,d2,65,7b,5f,7e,bb,39
Vi is : b8 Vj is : b8

%% Output Sample: P=P'?
Known plaintext for K[81][36]: 3f 25 3f 1a af 90 28 8c 79 12 4 eb d4 e0 b 27
Testing candidate key K[81][36]: 9a de ca 2 fa 37 9b bc 2b d2 65 7b 5f 7e bb 39
Decrypted plaintext: ee cc e2 27 3c b1 60 d5 15 7e 55 56 8e 22 59 43
```

5.4 Performance Considerations

Our implementation includes several optimizations:

- Efficient key schedule computation
- Minimized state transitions
- Optimized memory usage for state storage

5.5 Limitations

The current implementation has several limitations:

- Focuses only on rounds 8-10 of AES
- Uses a fixed dimension ($d = 8$) for the biclique
- Requires significant memory for state storage
- Does not implement all optimizations from the theoretical attack

5.6 Testing and Validation

The implementation includes a validation function `validateAESImplementation` that:

- Verifies correct AES operation
- Tests biclique construction
- Validates key recovery functionality

Test results demonstrate successful attack for key recovery for the implemented rounds, though with higher computational complexity than theoretical bounds due to implementation simplifications. Although there has not been any successful recovery of key achieved by this code, several factors impacts this including computation limitations and possible mistakes during implementation.

6 Conclustion

Conclusion This report explored the biclique attack on the full AES-128, demonstrating how advanced cryptanalysis techniques can marginally improve key recovery efficiency to $2^1 26.18$. By leveraging independent-biclique structures and related-key differentials, the attack systematically reduces computational complexities compared to brute force. However, this attack remains theoretical due to constraints in computational resources. We attempted to implement a reduced version of the attack in code; however, the implementation did not succeed in recovering the key candidates due to possible mistakes in implementation and the high computational complexity of the attack. This study provided valuable insights into the complexities of the biclique attack and its application to AES-128.

References

[1] Andrey Bogdanov, Dmitry Khovratovich, and Christian Rechberger. Biclique cryptanalysis of the full AES. Cryptology ePrint Archive, Paper 2011/449, 2011.

[2] Wikipedia. Biclique attack. `https://en.wikipedia.org/wiki/Biclique_attack`.

[3] Wikipedia. Bipartite graph. `https://en.wikipedia.org/wiki/Bipartite_graph`.

[4] Wikipedia. Meet-in-the-middle attack. `https://en.wikipedia.org/wiki/Meet-in-the-middle_attack`.