An Enhanced Differential Cache Attack on CLEFIA for Large Cache Lines

Chester Rebeiro, Rishabh Poddar, Amit Datta, and Debdeep Mukhopadhyay

Department of Computer Science and Engineering
Indian Institute of Technology Kharagpur, India
{chester,rishavp,adatta,debdeep}@cse.iitkgp.ernet.in

Abstract. Reported results on cache trace attacks on CLEFIA do not work with increased cache line size. In this paper we present an enhanced cache trace attack on CLEFIA using the differential property of the s-boxes of the cipher and the diffusion properties of the linear transformations of the underlying Feistel structures. The attack requires 3 round keys, which are obtained by monitoring cache access patterns of 4 rounds of the cipher. A theoretical analysis is made on the complexity of the attack, while experimental results are presented to show the effectiveness of power and timing side-channels in deducing cache access patterns. The efficacy of the attack is theoretically justified by showing the effect of cache line size on the time and space complexity of the attack. Finally countermeasures that guarantee security against cache-attacks are compared for their efficiency on large cache lines.

1 Introduction

In 2000, John Kelsey, Bruce Schneier, David Wagner, and Chris Hall prophesied that cryptographic ciphers implemented on systems with cache memory are vulnerable to side-channel attacks [9]. These attacks, which came to be known as *cache attacks*, exploited the non-uniform behavior between a cache hit and a cache miss. The differential behavior between a cache hit and a miss is manifested through side-channels such as timing, power, and electro-magnetic radiation. Over the decade, several forms of cache attacks have been discovered [1,4,5,7,10,11,12,18,17], which provides different strategies to extract information from the side-channels. Cache attacks have been found to be a serious threat to the security of modern crypto-systems due to the small number of encryptions that need to be monitored and the possibility of remote attacks [2,8].

Most of the cache attacks developed target AES and all of them follow a *divide-and-conquer* approach. Some of these attacks are [1,4,5,7,10]. The attacks on AES split the 128 bit key into 16 bytes and then recovers each byte independently. Obtaining all 16 bytes can easily be done by targeting just the first round of the cipher. In Feistel ciphers such as CAMELLIA [3] and CLEFIA [14] however, the first round provides only 64 bits of the key. Obtaining the remaining 64 bits requires more rounds to be attacked. This is difficult due to two reasons. First, for inner rounds it becomes more difficult to have control of the inputs to that

D.J. Bernstein and S. Chatterjee (Eds.): INDOCRYPT 2011, LNCS 7107, pp. 58–75, 2011.

round; an essential requirement in cache-attacks (except for a variant in [16], which requires knowledge of neither the plaintext nor ciphertext and depends on a spy process to reveal the cache access patterns). Second, the correctness of the round key recovered depends on the correctness of the previous round keys obtained, this adds to the unpredictability.

In [12] and [19], a cache attack on CLEFIA was described, which required attacking 3 rounds of the cipher to completely recover the 128 bit secret key. However, the attack in [19] by Zhao and Wang used a strong assumption of misaligned tables. Misalignment of tables does not happen unless forced by the programmer; for example to optimize for space. The attack in [12] used a combination of differential techniques with cache access patterns to attack CLEFIA in 2^{14} encryptions. The attack however is restricted to cache memories with a cache line size of 32 bytes. This is not the conventional cache line size on modern day microprocessors. In this paper we present and critically analyze a new attack on CLEFIA that can be mounted on standard microprocessors. The contributions of this paper are as follows:

- We enhance the attack on CLEFIA in [12] to eliminate the constraint of the cache line size. On a microprocessor with a 64-byte cache line, the attack has a complexity of 2^{11} encryptions, while on a system with a 32-byte cache line, the attack has an estimated complexity of 2^{12} encryptions. This is lesser than the 2^{14} encryptions reported in [12].
- We critically analyze the difficulty in attacking more than one round of a Feistel cipher using the proposed attack on CLEFIA.
- A study is made on the effect of cache-line size on the complexity of the attack. Based on this analysis we show a peculiar property that the attack is better suited for microprocessors with large cache lines, provided the entire table does not completely fit into a single line in the cache.
- A discussion on countermeasures is done and their overhead on the performance compared.

The outline of the paper is as follows: Section 2 provides a brief introduction to CLEFIA and the differential cache attack proposed in [12]. The drawbacks of the existing attack and the new attack's principle are presented in Section 3. This section also presents a theoretical analysis on the complexity of attacking different rounds of a cipher. Section 4 presents the complete attack on CLEFIA. Section 5 compares the capabilities of a power adversary and a timing adversary with respect to the number of measurements required to distinguish between a cache hit and a miss. The effect of the cache line size and number of encryptions required is analyzed in Section 6, while countermeasures for the attack are discussed in Section 7. The final section has the conclusion of the paper.

2 Preliminaries

In this section we first give a brief description of the CLEFIA structure, then present the attack in [12].

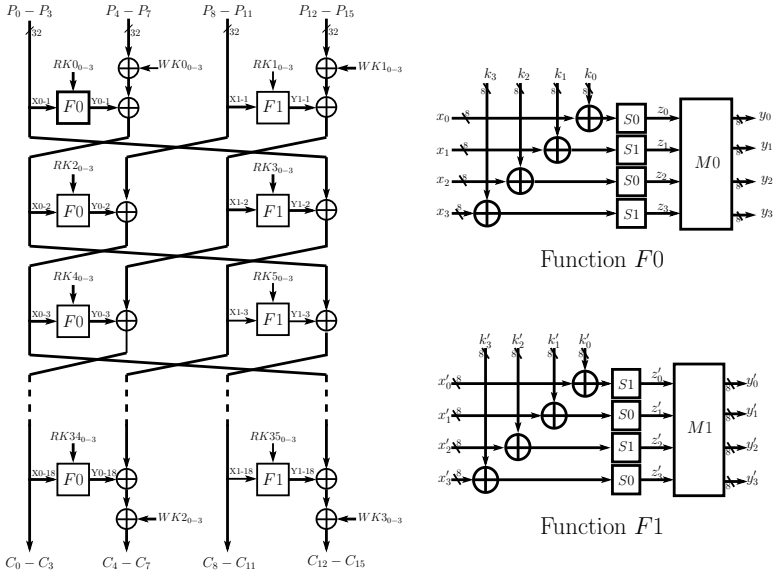


Fig. 1. CLEFIA Block Diagram

2.1 The CLEFIA Structure

The 128-bit block cipher CLEFIA [15] has a type-2 generalized Feistel structure [20] as shown in Figure 1. The 16 bytes plaintext input P_0 to P_{15} is grouped in 4 byte words. There are 18 rounds and in each round, the key addition, substitution and diffusion is provided by the application of two F functions, $F0$ and $F1$. The substitution in the F functions are done by two 256 element s-boxes $S0$ and $S1$, while the diffusion is done by the self-inverting matrices $M0$ and $M1$, which are defined as follows.

$$M0 = \begin{pmatrix} 1 & 2 & 4 & 6 \\ 2 & 1 & 6 & 4 \\ 4 & 6 & 1 & 2 \\ 6 & 4 & 2 & 1 \end{pmatrix} \qquad M1 = \begin{pmatrix} 1 & 8 & 2 & A \\ 8 & 1 & A & 2 \\ 2 & A & 1 & 8 \\ A & 2 & 8 & 1 \end{pmatrix} \tag{1}$$

Each round has an addition of round keys. The i^{th} round uses the round keys RKi and $RKi{+}1$. Each of these round keys are of 32 bits. Additionally, whitening keys $WK0$ to $WK3$ are applied at the start and end of encryption as seen in Figure 1.

2.2 Cache Attacks on CLEFIA

All cache attacks target structures in the block cipher such as in Figure 2. The figure shows two accesses to table S with indices $(x \oplus k1)$ and $(y \oplus k2)$. When

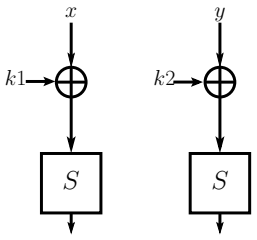


Fig. 2. Look-up Structure Targeted in Cache Attacks

a cache hit occurs the following relation holds leading to leakage of information about the ex-or of the keys; assuming that the inputs x and y are known.

$$\langle k1 \oplus k2 \rangle = \langle x \oplus y \rangle \tag{2}$$

We note that due to the effects of the cache line, only the most significant bits can be equated, therefore $\langle \cdot \rangle$ refers to only these most significant bits. If the size of $k1$ and $k2$ is l bits, and there are 2^δ elements that share a cache line, then only the most significant $b = l - \delta$ bits satisfy the above equation. Similarly, when a cache miss occurs, the following inequality holds.

$$\langle k1 \oplus k2 \rangle \neq \langle x \oplus y \rangle \tag{3}$$

Given any set of 4 round keys $(RK4i, RK4i+1, RK4i+2, RK4i+3)$, where $i \mod 2 = 0$, CLEFIA's key expansion algorithm can be reversed to obtain 121 out of the 128 bit secret key. For cache-attacks, determining the first set of round keys $RK0$, $RK1$, $RK2$, and $RK3$ is most suited. However, the presence of the whitening keys $WK0$ and $WK1$ makes the determination of these round keys not straight forward. In [12] this difficulty is circumvented by having 3 stages in the attack. First $RK0$ and $RK1$ are discovered, then $WK0 \oplus RK2$ and $WK1 \oplus RK3$, and finally $RK4$ and $RK5$. Determining each of these keys requires knowing the inputs to the respective F functions. For example, determining $WK0 \oplus RK2$ requires knowledge of $X0$-2 (see Figure 1), which in turn can be only computed if $RK0$ is known. Thus, an error in determining $RK0$ would also cause an error in $WK0 \oplus RK2$.

3 Enhancing the Differential Cache Attack

In the first part of this section, we discuss the reason why the attack proposed in [12] fails on systems with large cache lines. We then present a general technique to enhance the attack that would work with larger cache lines.

3.1 Why the Attack in [12] Fails for Large Cache Lines?

Figure 3 shows the application of three rounds of an F function (either $F0$ or $F1$). The differential attack in [12], used the fact that if at-least 3 bits of the

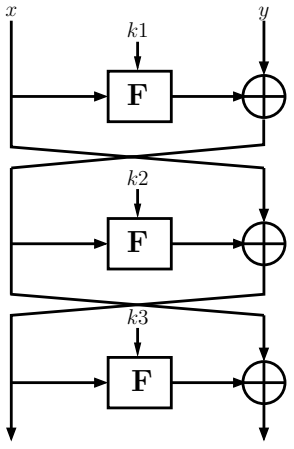


Fig. 3. Three Round Feistel Structure

difference of each output byte of F is known, then properties of the matrix ($M0$ for $F0$ and $M1$ for $F1$) can be used to obtain few output bits of the s-boxes. The difference distribution tables of the s-boxes can then be used to obtain a set of candidate keys. The crucial part of obtaining 3 bits of the F function output is done by forcing cache hits in the s-box tables of the second round. Since the CLEFIA implementation[1] attacked used 256 byte tables, and elements within a cache line cannot be distinguished, the table should occupy at-least 8 cache lines in order to obtain the required 3 bits of information. This means that each cache line can be of at-most $\frac{256}{8} = 32$ bytes. If the cache has the more conventional 64 byte cache lines, then only 2 bits of information can be obtained. This is insufficient to apply the properties of the matrices in order to derive the output difference of the s-boxes.

3.2 The Proposed Differential Cache Attack

CLEFIA is based on the generalized Feistel structure therefore we use Figure 3 to explain the principle of the new attack. The input x consists of 4 concatenated bytes $(x_0|x_1|x_2|x_3)$ and is known as the *differential introducing input*, while y comprising of the bytes $(y_0|y_1|y_2|y_3)$ is the *restoring input*. The F in the figure is either CLEFIA's $F0$ or $F1$ function (see Figure 1). For any fixed value of x, each byte of y is varied until cache hits are obtained in all s-box tables in the second round F function. This is called the *collision setup* phase. In this state, the following equation holds for $0 \leq i \leq 3$ if a cache hit is obtained.

$$\langle x_i \oplus k1_i \rangle = \langle y_i \oplus k2_i \oplus F(x, k1)_i \rangle \tag{4}$$

[1] http://www.sony.net/Products/cryptography/clefia/download/data/clefia_ref.c

Similarly, the following inequality holds if a cache miss is obtained.

$$\langle x_i \oplus k1_i \rangle \neq \langle y_i \oplus k2_i \oplus F(x, k1)_i \rangle \tag{5}$$

Now the input byte x_0 is displaced by $d_{x_0} \neq 0$, while all other bytes of x are unchanged. After the s-box operation, the displacement is diffused to all output bytes of the F function (since the branch number for the F functions in CLEFIA are 5 [14]), due to which, some of the cache hits in the second round are lost. The cache hits that remain are due to the tables in the second round being accessed in the same cache lines as before. The collision state is restored by reinforcing cache hits in the second round by modifying y. This is called the *restoring phase*. Let $y' = (y'_0|y'_1|y'_2|y'_3)$ be the new value of y. The differences at the output of the F function is $d_{y_i} = y_i \oplus y'_i$.

From the difference distribution table for the s-box, one can derive the set of possible output differentials corresponding to the input differential d_{x_0}. Let this set be called $\mathcal{D}$. For every differential $d_o \in \mathcal{D}$, the matrix product $M \cdot (d_o\,,\,0\,,\,0\,,\,0)^T$ is computed to obtain the differentials $(d_{z_0}\,,\,d_{z_1}\,,\,d_{z_2}\,,\,d_{z_3})^T$ (M is either $M0$ or $M1$ depending on the F function used). The property that is exploited in the attack is that for the correct s-box output differential d_o, $\langle d_{z_i} \rangle = \langle d_{y_i} \rangle$ for $0 \leq i \leq 3$. Each equality leads to some information about the key $k1_0$. In both $M0$ and $M1$ matrices, 3 out of the 4 equalities reveal unique information, therefore it is sufficient to have only 3 equality tests instead of 4. In a similar way, displacements introduced at x_1, x_2, and x_3 would lead to leakages in $k1_1$,$k1_2$, and $k1_3$ respectively. The collision set up phase is common for all keys. However the restoring phases differ. Algorithm 1 shows the technique used to generate the candidate keys given that the collisions have been set up.

Algorithm 1. find: Finding Key Byte $k1_i$ assuming collisions have been setup

Input: $i \in \{0, 1, 2, 3\}$, the differential introducing input x and restoring input y
Output: $S^1_{k1_i}$: Candidate Key Set for $k1_i$

```
1  begin
2      S^1_{k1_i} ← {}
3      x_i ← x_i ⊕ d_{x_i}
4      Restore collisions: Find y'_0, y'_1, y'_2 which causes collision in the first three accesses of
       the 2^nd round
5      D ← output difference set corresponding to the input difference d_{x_i}
6      foreach d_o ∈ D do
7          (d_z0 , d_z1 , d_z2 , d_z3)^T ← (d_o , 0 , 0 , 0)^T · M0
8          if (⟨d_z0⟩ = ⟨d_y0⟩ and ⟨d_z1⟩ = ⟨d_y1⟩ and ⟨d_z2⟩ = ⟨d_y2⟩) then
9              S^1_{k1_i} ← S^1_{k1_i} ∪ {d_o}
10         end
11     end
12 end
```

The Amount of Leakage: Let the number of bits revealed due to Equation 4 be b. This means that b bits in each of d_{y_i} (where $0 \leq i \leq 3$) is revealed. Since each

Table 1. Expected Number of Candidate Key Bytes with Varying Cache Line sizes

CacheLine Size → Matrix, S-box ↓	128 ($b = 1$)	64 ($b = 2$)	32 ($b = 3$)	16 ($b = 4$)	8 ($b = 5$)
$M0$, $S0$	81.9	40.9	20.4	10.2	5.1
$M0$, $S1$	64.4	32.2	16.1	8.0	4.0
$M1$, $S0$	81.9	20.4	10.2	5.1	2.5
$M1$, $S1$	64.4	16.1	8.0	4.0	2.0

d_{y_i} is a linear transformation of the difference output of the s-box, a few bits of this output difference is obtained. Further, a valid input-output difference pair in $S0$ has on average 1.28 input possibilities, while $S1$ has 1.007. Table 4 shows the expected number of candidate keys obtained for a key byte (the size of the set $S^1_{k1_i}$ in Algorithm 1) for different cache line sizes (assuming each element in the table occupies one byte). As seen in the table, this depends on the matrix as well as the s-box. The number of candidate keys can be reduced by repeating the attack several times and taking the intersection between the sets. If q repetitions are done then the expected number of candidate keys for $k1_i$ at the end of the q repetitions is:

$$|S^q_{k1_i}| = \frac{(|S^1_{k1_i}|)^q}{256^{q-1}} \tag{6}$$

3.3 Attacking a Feistel Structure from Cache Traces

Attack on Feistel ciphers such as CLEFIA requires keys from more than one round to be obtained. However two problems are likely to arise while recovering keys for rounds other than the first round. This part of the section discusses these problems.

Increase in the Number of Candidate Keys: For a key to be recovered, the input and output difference of the F function is required. Consider the task of recovering $k2$ in Figure 3. The input difference to the second round F function is $(y \oplus F(x, k1)) \oplus (y' \oplus F(x', k1))$, while the output difference obtained from the cache access patterns is $x \oplus x'$. Thus it is seen that determining any $k2_i$, for $0 \leq i \leq 3$, depends on the value of $k1$. Let the number of candidate keys for each byte of $k1$ be n_1, then the number of possible values for an output byte of $F(x, k1)$ has an upper bound of n_1^4. When $n_1 = 1$ and after q repetitions, each key byte $k2_i$ would have the same number of candidates as $k1_i$, (ie. $|S^q_{k1}|$). For $n_1 > 1$, each output byte of $F(x, k1)$ produces a different set of keys for $k2_i$. The union of these sets should be considered while determining $k2_i$. The estimated size of this union is less than $256(1 - (1 - \frac{|S^q_{k1_i}|}{256})^{n_1^4})$. In general, for round $r > 1$,

$$|S^q_{kr_i}| \leq 256\left(1 - \left(1 - \frac{|S^q_{k1_i}|}{256}\right)^{n^4_{r-1}}\right) \tag{7}$$

We see that the expected number of candidates increases exponentially with the round number. There are two ways to reduce this set.

- Increasing q, the number of repetitions in r^{th} round, would reduce the size of the set $|S^q_{k1_1}|$, thus reducing the number of candidates.
- A more effective approach is to increase the number of repetitions of the previous rounds, thus reducing the value of n_{r-1}. We see that the number of keys in the first round most influences the size of the candidate key set for kr_i and this influence reduces as the round number increases. The best strategy would therefore be to have q for a round much larger than the q for the following rounds. This would result in $n_1 \ll n_2 \cdots \ll n_{r-1}$.

Control of the Round Inputs: Determining the key bytes for round r requires collisions between the s-box accesses in rounds r and $r+1$. For the Equations (4) and (5) to be used we need to know the inputs (ie. x and y) to round r. This becomes increasingly difficult as r increases. Secondly, even if we were able to control these inputs, a cache hit in a table access in round $r+1$ could be due to a collision with any of the accesses in the previous rounds, not necessarily with the access in round r. Methods to improve controllability and reduce ambiguity about collisions are influenced by the cipher's structure. In the next section we present several such strategies for CLEFIA.

4 The New Differential Cache Attack against CLEFIA

Just as in [12], the proposed attack first finds $RK0$ and $RK1$, then $RK2 \oplus WK0$ and $RK3 \oplus WK1$, and finally $RK4$ and $RK5$. With this information, the key expansion algorithm can be exploited (shown in [12]) to determine 121 bits out of 128 bits of the secret key.

4.1 Determining $RK0$ and $RK1$

To find $RK0$, the memory accesses in the first and second round $F0$ functions (Figure 1) are considered. This is similar to a two round Feistel structure (Figure 3) with the differential introducing input being $P_0 \cdots P_3$, while the restoring input being $P_4 \cdots P_7$. Since in CLEFIA, each s-box is used 4 times per round, a resulting cache hit in the second round would be due to collisions with any of these four accesses. Let $I\alpha^i_{S\beta}$ be the index to the i^{th} access to table $S\beta$ in round α. Thus a collision in $I2^1_{S0}$ could be with one or more of the accesses $I1^1_{S0}$, $I1^2_{S0}$, $I1^3_{S0}$, and $I1^4_{S0}$. In order to apply Equations 4 and 5, we need to know which of the four accesses has caused the collision. To prevent this ambiguity, we ensure that all accesses in the first round are themselves colliding. That is for $S0$, $\langle I1^1_{S0}\rangle = \langle I1^2_{S0}\rangle = \langle I1^3_{S0}\rangle = \langle I1^4_{S0}\rangle$ and for $S1$, $\langle I1^1_{S1}\rangle = \langle I1^2_{S1}\rangle = \langle I1^3_{S1}\rangle = \langle I1^4_{S1}\rangle$. We call such a state the 1-*round collision state*. Obtaining the 1-round collision state requires finding the appropriate values for P_2, P_3, P_8, P_9, P_{10}, and P_{11}.

To find $RK1$, $F1$ of the first two rounds are considered with differential introducing inputs being $P_8 \cdots P_{11}$ and restoring inputs $P_{12} \cdots P_{15}$. Algorithm 2 uses the *find* procedure described in Algorithm 1 to determine $RK0$ and $RK1$.

Algorithm 2. Finding Candidate Keys for $RK0$ and $RK1$

Output: S_{RK0_i} and S_{RK1_i}: Respective candidate key sets for $RK0$ and $RK1$, where $i \in \{0, 1, 2, 3\}$

1 **begin**
2 Randomly select P_0 and P_1
3 $(P_2, P_3, P_8 \cdots P_{11}) \leftarrow$ One Round Collision State
4 *Set up collisions:* Find P_4, P_5, P_6 causing collisions in $I2^1_{S0}, I2^1_{S1}, I2^2_{S0}$ respectively
5 $S_{RK0_0} \leftarrow find(0, P_0 \cdots P_3, P_4 \cdots P_7)$
6 $S_{RK0_1} \leftarrow find(1, P_0 \cdots P_3, P_4 \cdots P_7)$
7 $S_{RK0_2} \leftarrow find(2, P_0 \cdots P_3, P_4 \cdots P_7)$
8 $S_{RK0_3} \leftarrow find(3, P_0 \cdots P_3, P_4 \cdots P_7)$
9 Find collision in $I2^2_{S1}$ using P_7
10 *Set up collisions:* Find P_{12}, P_{13}, P_{15} causing collisions in $I2^3_{S1}, I2^3_{S0}, I2^4_{S1}$ respectively
11 $S_{RK1_0} \leftarrow find(0, P_8 \cdots P_{11}, P_{12} \cdots P_{15})$
12 $S_{RK1_1} \leftarrow find(1, P_8 \cdots P_{11}, P_{12} \cdots P_{15})$
13 $S_{RK1_2} \leftarrow find(2, P_8 \cdots P_{11}, P_{12} \cdots P_{15})$
14 $S_{RK1_3} \leftarrow find(3, P_8 \cdots P_{11}, P_{12} \cdots P_{15})$
15 **end**

Analysis: Let ρ be the number of encryptions required to determine (from the side-channel information) if a memory access resulted in a cache hit or a cache miss. Let b be the number of bits that are revealed in Equation 4 or 5. Obtaining a collision requires $\rho(2^b - 1)$ encryptions, because b bits can have 2^b possibilities, and if $(2^b - 1)$ choices fail to give a collision, then the final choice is the correct one. Thus obtaining a 1-round collision state requires $6\rho(2^b - 1)$ encryptions. Algorithm 2 requires $3\rho(2^b - 1)$ encryptions each to set up collisions in lines 4 and 10. Moreover, each call to $find$ requires $3\rho(2^b - 1)$ encryptions and line 9 requires $\rho(2^b - 1)$ encryptions. Thus in total, finding both $RK0$ and $RK1$ requires $37\rho(2^b - 1)$ encryptions.

4.2 Determining $RK2 \oplus WK0$ and $RK3 \oplus WK1$

To obtain the candidate keys of $RK2 \oplus WK0$, the $F0$ functions in the second and third rounds are considered with $P_4 \cdots P_7$ used as differential introducing inputs, while $P_8 \cdots P_{11}$ used as restoring inputs. Just as in the first stage of the attack, ambiguities about collisions may arise when cache hits are forced in the tables in the third round $F0$. Therefore, before forcing hits in the third round, the cipher is put in a 2-*round colliding state*. Besides the first access to each table, a 2-round colliding state has all remaining accesses in collision.

$$\langle I1^1_{S0} \rangle = \langle I1^2_{S0} \rangle = \langle I1^3_{S0} \rangle = \langle I1^4_{S0} \rangle = \langle I2^1_{S0} \rangle = \langle I2^2_{S0} \rangle = \langle I2^3_{S0} \rangle = \langle I2^4_{S0} \rangle$$

for $S0$ and for $S1$,

$$\langle I1^1_{S1} \rangle = \langle I1^2_{S1} \rangle = \langle I1^3_{S1} \rangle = \langle I1^4_{S1} \rangle = \langle I2^1_{S1} \rangle = \langle I2^2_{S1} \rangle = \langle I2^3_{S1} \rangle = \langle I2^4_{S1} \rangle$$

In spite of the 2-round colliding state, ambiguities about the collisions still occur in the third round $F0$ memory accesses. For example, a cache hit in $I3^1_{S0}$ can be forced by the plaintext byte P_8. However, changing P_8, may loose the cache

Algorithm 3. Elimination Method

Input: I_a: memory access where a collision is required, I_{b1} and I_{b2}: accesses which cause undesirable collisions, C_1 and C_2: controlling plaintexts for I_{b1} and I_{b2} respectively
Output: Collision at I_a is desirable or undesirable

```
1  begin
2      c1 ← C1
3      C1 ← (C1 + cache line size)mod table size
4      if (Ia not in collision) then
5          return "Undesirable Collision"
6      end
7      C1 ← c1
8      C2 ← (C2 + cache line size)mod table size
9      if (Ia not in collision) then
10         return "Undesirable Collision"
11     end
12     C1 ← (C1 + cache line size)mod table size
13     if (Ia not in collision) then
14         return "Undesirable Collision"
15     end
16     return "Desirable Collision"
17 end
```

hits in the two $S0$ accesses in $F1$ of the second round (Note that the cache hits in the first round are not altered). Thus a collision in $I3^1_{S0}$ may be due to three reasons: the desirable cache hit with $I2^1_{S0}$ and undesirable cache hits due to collisions with $I2^3_{S0}$ and $I2^4_{S0}$. There are two ways to identify the ambiguous cache hits:

- *Elimination Method:* This method uses a set of controlling plaintext bytes C_1 and C_2. These inputs can control locations causing ambiguous cache hits ($I2^3_{S0}$ or/and $I2^4_{S0}$). In the elimination method C_1 is first varied and checked if a cache hit persists. Then C_2 varied and finally both C_1 and C_2 are varied. A cache hit is desirable only if all three tests result in cache hits. Algorithm 3 presents this method. For the example above, I_a is $I3^1_{S0}$, I_{b1} is $I2^3_{S0}$ and I_{b2} is $I2^4_{S0}$. The controlling plaintexts C_1 and C_2 are P_{12} and P_{14} respectively.
- *Probabilistic Method:* In the example described above, assume that P_8 is ex-ored by a non-zero value d, which is less than the size of the cache line. The small displacement of P_8 ensures that if $I3^1_{S0}$ was the desirable collision, then the collision would remain with probability 1. However the small displacement of P_8 would become a random change after the s-box, affecting all four outputs of the first round $F1$. Thus the accesses to $I2^3_{S0}$ and $I2^4_{S0}$ would also change randomly. Therefore, if the collision at $I3^1_{S0}$ is undesirable, the collision would remain with probability $(1-(1-\frac{1}{2^b})^2)$. Sufficient confidence in the correctness of the collision is obtained if this test is repeated $1/(1-\frac{1}{2^b})^2$ times as shown in Algorithm 4.

The elimination method, which gives a deterministic result, is the suited technique. However application of this method is not always feasible (as in the case of determining $RK3 \oplus WK1$), in which case the probabilistic method will need to be applied. Algorithm 5 gives the procedure for determining the byte ($RK2 \oplus WK0$).

Algorithm 4. Probabilistic Method

Input: I_a: access where a cache hit is required, P_a: plaintext byte used to force cache hits in I_a
Output: Cache hit at I_a is desirable or not

```
1 begin
2     for i ∈ {0, ··· , 2^{2b} / (2^b − 1)^2} do
3         P_a ← P_a ⊕ i
4         if I_a not in collision then
5             return "Undesirable Collision"
6         end
7     end
8     return "Desirable Collision";
9 end
```

Algorithm 5. Finding Candidate Keys for $RK2 \oplus WK0$

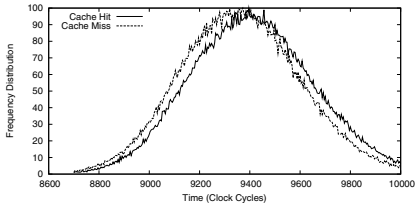
Output: $S_{(RK0 \oplus WK0)_i}$): Candidate Key Set for $(RK0 \oplus WK0)_i$, where $i \in \{0, 1, 2, 3\}$

```
1 begin
2     Put cipher in Two round colliding state
3     Set up collisions: Find P_8, P_9, P_10 causing collisions in I3^1_{S0}, I3^1_{S1}, I3^2_{S0} respectively
      (using elimination method)
4     S(RK0 ⊕ WK0)_0 ← find(0, P_4 ··· P_7, P_8 ··· P_11)
5     S(RK0 ⊕ WK0)_1 ← find(1, P_4 ··· P_7, P_8 ··· P_11)
6     S(RK0 ⊕ WK0)_2 ← find(2, P_4 ··· P_7, P_8 ··· P_11)
7     S(RK0 ⊕ WK0)_3 ← find(3, P_4 ··· P_7, P_8 ··· P_11)
8 end
```

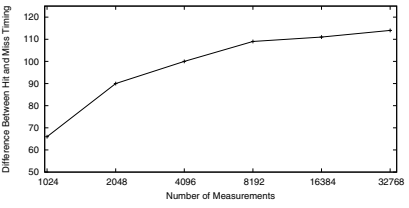
To determine $RK3 \oplus WK1$, the second and third round $F1$ functions are considered with $P_{12} \cdots P_{15}$ as the difference introducing inputs, while $P_0 \cdots P_3$ the restoring inputs. Starting from the 2-round colliding state, ambiguities in collisions in the third round $F1$ are due to collisions with the tables of $F0$ of the third round. Using the elimination method would imply P_8 to P_{15} be used as controlling inputs. But changing P_8 would also alter the cache hit in the third round $F1$. Therefore, the elimination method is not feasible and probabilistic method needs to be applied.

Analysis: For the elimination method, in addition to the $\rho(2^b - 1)$ encryptions that are required to find a collision, 3ρ additional encryptions are necessary to eliminate each of the undesirable collisions. Therefore in all $\rho(2^b + 5)$ encryptions are required to correctly identify a collision. In Algorithm 2, setting up collisions will therefore require $3\rho(2^b + 5)$ and each execution of *find* would need an additional $3\rho(2^b + 5)$. In total, there would be $15\rho(2^b + 5)$ encryptions required to find $RK0 \oplus WK0$.

For $RK3 \oplus WK1$, the probabilistic method is used. To determine the correct collision with significant probability, $\rho \frac{2^{b+1}}{(2^b-1)^2}$ encryptions are required in addition to the $\rho(2^b - 1)$ encryptions. Thus obtaining the whole of $RK3 \oplus WK1$ can be done in $16\rho((2^b - 1) + \frac{2^{2b+1}}{(2^b-1)^2})$.



(a) Timing Distributions for $I1^2_{S0}$ for a Cache Miss and Cache Hit



(b) Difference between Expected Values for a Cache Miss and Cache Hit Distributions

Fig. 4. Analysis of Timing Side Channel on Intel Xeon (E5606)

4.3 Determining $RK4$ and $RK5$

To compute the candidate keys for $RK4$, we use the 3^{rd} and 4^{th} round $F0$ functions with $P_8 \cdots P_{11}$ as the difference introducing inputs and $P_{12} \cdots P_{15}$ the restoring inputs. The initial state is a 2-round colliding state and all tables in $F0$ of the third round are also in collision. The undesirable collisions occurring with $F1$ of round three can be detected by the probabilistic method. The number of encryptions required for $RK4$ is be found to be $16\rho((2^b-1)+\frac{2^{2b+1}}{(2^b-1)^2})$. Candidate keys for $RK5$ is similarly determined from the 3^{rd} and 4^{th} round $F1$ functions with $P_0 \cdots P_3$ as the difference introducing inputs and $P_4 \cdots P_7$ the restoring inputs. In this case however, more undesirable collisions can occur due to $F0$ in rounds three and four. Due to this, the number of encryptions required is increased to $16\rho((2^b-1)+\frac{2^{4b+2}}{(2^b-1)^4})$.

5 Distinguishing between a Cache Hit and Miss

The attack on CLEFIA presented in the previous section relies on distinguishing between a cache hit and a miss. The number of encryptions (ρ) required to make this distinction depends on the form of side-channel used. In this section we determine ρ for two side-channels namely power consumption and timing.

Cache Access Patterns from Power Consumption: In [12], the attack was mounted on an embedded PowerPC in the Xilinx XC2VP30 FPGA present on the SASEBO side channel attack evaluation board [13]. Cache access patterns were identified by correlating the power trace obtained with templates. For a match, a correlation value of 0.997 was obtained, compared to 0.8 for a mismatched power trace. Thus a single trace ($\rho = 1$) is sufficient to obtain cache access patterns.

Cache Access Patterns from Timing: Obtaining side channel distributions from timing requires several encryptions to be done. The distinguisher is a shift in

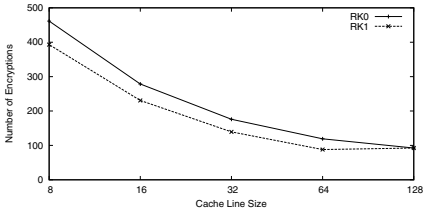


Fig. 5. Cache Line Size vs Encryptions Required to Uniquely Identify a Key

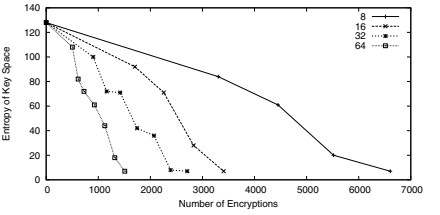


Fig. 6. Encryptions vs Key Space Remaining for Various Cache Lines

the timing distribution for a cache hit compared to the cache miss distribution. For example, the timing distributions for a hit and miss at $I1^2_{S0}$ is shown in Figure 4(a). The distributions are Gaussian and a noticeable difference between the distributions is observable. On an average, when there is a cache hit in $I1^2_{S0}$, CLEFIA takes around 100 clock cycles longer than when a miss is present in $I1^2_{S0}$, assuming that all encryptions start with a clean cache. The distance between the cache hit and miss distributions represents our confidence in the measurement. A small distance implies low confidence about the correctness of the measurements (and vice-versa). Figure 4(b) shows the distance between a cache hit and a cache miss distribution as the number of measurements increase. We find that the difference is around 60 clock cycles when 2^{10} encryptions are made. This difference increases as the number of measurements increase until it saturates at around 114 clock cycles. A ρ of 2^{10} ensures with certainly the correct detection of the cache hit.

6 Effect of Cache Line Size on the Number of Encryptions

In the cipher's implementation, if a table occupies 2^b cache lines, then $2^b - 1$ encryptions are required to identify a collision. Further, if 2^δ elements share a common cache line then there still remains an uncertainty of δ bits.

With increase in the size of the cache line, the table (which we assume to have the same size) occupies fewer lines in memory (b reduces), consequently fewer number of encryptions are required to identify a collision. On the other hand, an increase in the cache line size would increase δ causing an increase in the size of the candidate key set (as seen in Table 4). The attack reduces this uncertainty by repeating the experiment and taking intersections between the candidate key sets obtained. Thus we see that the cache line size has opposite effects on the number of encryptions required to find a collision and the number of encryptions required to reduce the candidate key set. Figure 5 shows the effect of cache line size on the number of encryptions required on average to identify $RK0$ and $RK1$ uniquely. This is assuming a power side-channel adversary where $\rho = 1$. As seen from the graph large cache lines are more prone to being attacked as they require fewer encryptions to be monitored. This can also be seen in Figure 6, which estimates the entropy of the remaining key space with increasing number

of encryptions for the entire attack (assuming $\rho = 1$). This being said, if the cache line was large enough to fit the entire table then every access to the table (except the first) would result in a cache hit, and therefore the attack would fail.

7 Countermeasures Suited for Large Cache Lines

Although there have been several countermeasures proposed against cache attacks (such as in [6,11,16]), not all of them guarantee protection. Countermeasures that do guarantee protection either disable the cache, do not use tables, or implement the cipher to fit tables within a single cache line. Disabling the cache is known to slow down an execution 100 times [16] and affects the entire system, therefore is not an option. The other countermeasures too have significant performance overheads. In this section we compare these overheads and also suggest splitting of tables as a countermeasure for large cache lines.

Using Single Cache Lines: In [12], techniques were presented to implement CLEFIA's s-boxes using small table of 32 bytes. This table is small enough to fit into a single cache line, thus except for the first access to the table, all other memory accesses results in cache hits. However, the small tables implied substantial number of logical operations were required in addition to the table lookups; adding to the overhead. Naturally, when a cache line of larger dimensions is available, one would attempt to increase the number of tables to fit in the entire cache line. To the countermeasure in [12], we have added more tables, to adapt it for a 64 byte cache line (See Appendix-D).

Split Tables: Consider a table T having t elements with each element occupying u bits. Let the size of the cache line be c bytes. Then the table occupies $s = \frac{tu}{8c}$ lines in cache. Each of the t elements in the table is split into s parts and each part is put into a different table. There are therefore s tables, each of c bytes. To access an s-box element, all tables require to be read in order to obtain the element, therefore memory accesses are no longer dependent on the key protecting the implementation against cache attacks.

We found that on standard Intel platforms, the number of cache misses has a dominating influence on the execution time of a block cipher. In fact on an Intel Core 2 Duo platform, a cache miss has an average overhead of 10 clock cycles compared to a cache hit. If the size of the cache line (c) is large with respect to the table size (that is, the table occupies few cache lines), then with high probability the entire table is loaded into memory during the execution.

In the countermeasure proposed, the cumulative sum of the sizes of the s tables equals that of T. If we assume a large cache line exists, then the countermeasure has the same number of cache misses as the non-protected implementation. The only overhead of the countermeasure is in the non cache miss executions. This is also kept minimum if s is small, as is the case.

Table 7 shows the overheads of various countermeasures applied on CLEFIA implementation[2] for a cache memory with 64 byte cache line.

[2] http://www.sony.net/Products/cryptography/clefia/download/data/clefia_ref.c

Table 2. Execution Time for Different Implementations of CLEFIA on an Intel Core 2 Duo Processor

Implementation	Clock Cycles	Overhead
Reference Code (RC)	13452	-
RC without tables	47032	3.5
RC with countermeasure from [12] for 32 byte cache lines	24446	1.81
RC with countermeasure from [12] for 64 byte cache line	22356	1.66
Split tables	16235	1.2

8 Conclusion

In this paper we presented an attack on CLEFIA, which is not restricted to systems having small cache lines. The work shows that while existing attacks fail on systems with large cache lines, better exploitation of the differential properties of the non-linear layers and the diffusion properties of the linear layer inside the rounds, still lead to attacks. The work analyzes the effect of cache line size on the attack complexity, showing that systems with large cache lines are more prone to being attacked. This is supported by theoretical analysis of the time and space complexity of the attack. Further, techniques are suggested for CLEFIA to reduce these difficulties. The techniques presented in the paper can easily be adopted to break other ciphers such as CAMELLIA that have Feistel structures. Finally countermeasures for the attack are analyzed and their overheads compared on a standard computing platform.

References

1. Acıiçmez, O., Koç, Ç.K.: Trace-Driven Cache Attacks on AES (Short Paper). In: Ning, P., Qing, S., Li, N. (eds.) ICICS 2006. LNCS, vol. 4307, pp. 112–121. Springer, Heidelberg (2006)
2. Acıiçmez, O., Schindler, W., Koç, Ç.K.: Cache Based Remote Timing Attack on the AES. In: Abe, M. (ed.) CT-RSA 2007. LNCS, vol. 4377, pp. 271–286. Springer, Heidelberg (2006)
3. Aoki, K., Ichikawa, T., Kanda, M., Matsui, M., Moriai, S., Nakajima, J., Tokita, T.: *Camellia*: A 128-Bit Block Cipher Suitable for Multiple Platforms - Design and Analysis. In: Stinson, D.R., Tavares, S. (eds.) SAC 2000. LNCS, vol. 2012, pp. 39–56. Springer, Heidelberg (2001)
4. Bernstein, D.J.: Cache-timing Attacks on AES. Tech. rep. (2005)
5. Bertoni, G., Zaccaria, V., Breveglieri, L., Monchiero, M., Palermo, G.: AES Power Attack Based on Induced Cache Miss and Countermeasure. In: ITCC (1), pp. 586–591. IEEE Computer Society (2005)
6. Brickell, E., Graunke, G., Neve, M., Seifert, J.P.: Software Mitigations to Hedge AES Against Cache-based Software Side Channel Vulnerabilities. Cryptology ePrint Archive, Report 2006/052 (2006), `http://eprint.iacr.org/`
7. Canteaut, A., Lauradoux, C., Seznec, A.: Understanding Cache Attacks. Research Report RR-5881, INRIA (2006), `http://hal.inria.fr/inria-00071387/en/`

8. Crosby, S.A., Wallach, D.S., Riedi, R.H.: Opportunities and Limits of Remote Timing Attacks. ACM Trans. Inf. Syst. Secur. 12(3) (2009)
9. Kelsey, J., Schneier, B., Wagner, D., Hall, C.: Side Channel Cryptanalysis of Product Ciphers. J. Comput. Secur. 8(2,3), 141–158 (2000)
10. Osvik, D.A., Shamir, A., Tromer, E.: Cache Attacks and Countermeasures: The Case of AES. In: Pointcheval, D. (ed.) CT-RSA 2006. LNCS, vol. 3860, pp. 1–20. Springer, Heidelberg (2006)
11. Page, D.: Theoretical Use of Cache Memory as a Cryptanalytic Side-Channel (2002)
12. Rebeiro, C., Mukhopadhyay, D.: Cryptanalysis of CLEFIA Using Differential Methods with Cache Trace Patterns. In: Kiayias, A. (ed.) CT-RSA 2011. LNCS, vol. 6558, pp. 89–103. Springer, Heidelberg (2011)
13. Research Center for Information Security National Institute of Advanced Industrial Science and Technology: Side-channel Attack Standard Evaluation Board Specification, Version 1.0 (2007)
14. Shirai, T., Shibutani, K., Akishita, T., Moriai, S., Iwata, T.: The 128-Bit Blockcipher CLEFIA (Extended Abstract). In: Biryukov, A. (ed.) FSE 2007. LNCS, vol. 4593, pp. 181–195. Springer, Heidelberg (2007)
15. Sony Corporation: The 128-bit Blockcipher CLEFIA: Algorithm Specification (2007)
16. Tromer, E., Osvik, D.A., Shamir, A.: Efficient Cache Attacks on AES, and Countermeasures. Journal of Cryptology 23(2), 37–71 (2010)
17. Tsunoo, Y., Saito, T., Suzaki, T., Shigeri, M., Miyauchi, H.: Cryptanalysis of DES Implemented on Computers with Cache. In: Walter, C.D., Koç, Ç.K., Paar, C. (eds.) CHES 2003. LNCS, vol. 2779, pp. 62–76. Springer, Heidelberg (2003)
18. Tsunoo, Y., Tsujihara, E., Minematsu, K., Miyauchi, H.: Cryptanalysis of Block Ciphers Implemented on Computers with Cache. In: International Symposium on Information Theory and Its Applications, pp. 803–806 (2002)
19. Zhao, X., Wang, T.: Improved Cache Trace Attack on AES and CLEFIA by Considering Cache Miss and S-box Misalignment. Cryptology ePrint Archive, Report 2010/056 (2010), `http://eprint.iacr.org/`
20. Zheng, Y., Matsumoto, T., Imai, H.: On the Construction of Block Ciphers Provably Secure and Not Relying on Any Unproved Hypotheses. In: Brassard, G. (ed.) CRYPTO 1989. LNCS, vol. 435, pp. 461–480. Springer, Heidelberg (1990)

Appendix A: Number of Encryptions for Key Recovery

Table 3 summarizes the number of encryptions required to perform one repetition for a key ($q = 1$). In the table b is the number of bits revealed from the cache access profiles and $l = 2^b - 1$.

Appendix B: Plotting Figure 5

After q repetitions, the expected size of $RK0_i$ for $0 \leq i \leq 3$ is $|S^q_{RK0_i}|/256^{q-1}$. Therefore for the entire $RK0$ the set of candidate keys is

$$\frac{|S^q_{RK0_0}|}{256^{q-1}} \times \frac{|S^q_{RK0_1}|}{256^{q-1}} \times \frac{|S^q_{RK0_2}|}{256^{q-1}} \times \frac{|S^q_{RK0_3}|}{256^{q-1}}$$

To obtain a unique $RK0$, q should be found so that the above expression reduces to 1. In a similar way the trend for $RK1$ is computed.

Table 3. Number of Encryption required for $q = 1$

Key	Encryptions	Key	Encryptions
$RK0$	$21\rho l$	$RK1$	$16\rho l$
$RK2 \oplus WK0$	$15\rho(2^b + 5)$	$RK2 \oplus WK1$	$16\rho\left(l + \frac{2^{2b+1}}{l^2}\right)$
$RK4$	$16\rho\left(l + \frac{2^{2b+1}}{l^2}\right)$	$RK5$	$16\rho\left(l + \frac{2^{4b+2}}{l^4}\right)$

Appendix C: Plotting Figure 6

Figure 6 shows the estimated number of keys remaining versus the number of encryptions required. This appendix presents how the graph is obtained.

Estimating the Key Space Reduction: Table 4 gives the average number of candidate keys per byte for the F functions when $q = 1$. For example, when $b = 2$, each output byte for $F0$ in the first round has $(40.9 \times 32.2)^2$ number of options. However each output byte for $F0$ can take only 256 values. Therefore in this case $n_1^4 = 256$ in Equation 7 resulting in no key space reduction for the following round. The key space can be reduced by increasing q, which would then reduce the value of n_1^4. In a similar way, the number of outputs for $F0$ in the second round is used to compute the possible keys for the third round. We therefore obtain the key sets S^q_{RK0}, $S^q_{RK2 \oplus WK0}$, and S^q_{RK4}. Similarly $F1$ is analyzed to obtain S^q_{RK1}, $S^q_{RK3 \oplus WK1}$, and S^q_{RK5}.

Every possible combination formed by the keys in S^q_{RK0}, S^q_{RK1}, S^q_{RK4}, and S^q_{RK5} produces a candidate CLEFIA key, therefore the product of these sets have to be considered.

Encryptions per Round : This appendix gives details about the break up of the number of encryptions per round and the corresponding number of keys remaining.

Appendix D: Adapting the Countermeasure in [12] for 64 byte Cache Lines

The sboxes in CLEFIA are designed differently. $S0$ is composed of four sboxes $SS0$, $SS1$, $SS2$, and $SS3$; each of 16 bytes. The output of $S0$ is given by:

$$\begin{aligned} \beta_l &= SS2[SS0[\alpha_l] \oplus 2 \cdot SS1[\alpha_h]] \\ \beta_h &= SS3[SS1[\alpha_h] \oplus 2 \cdot SS0[\alpha_l]] \end{aligned} \tag{8}$$

,where $\beta = (\beta_h|\beta_l)$, $\alpha = (\alpha_h|\alpha_l)$, and $\beta = S0[\alpha]$. In [12], a pair of the 16 byte sboxes share a 16 byte table. So all 4 sboxes require 32 bytes. For 64 byte cache lines, the sharing is not required and each sbox can exclusively use 16 bytes of memory. This produces a small speedup.

For the sbox $S1$, the output corresponding to the input byte α is given by $g((f(\alpha))^{-1})$, where g and f are affine transforms and the inverse is found in the

Table 4. Expected Number of Candidate Keys with Varying Cache Line sizes

Cache Line Size	Number of Repetitions			Encryptions Required	Key Space Remaining
	Round 1	Round 2	Round 3		
8	1	1	1	3303	2^{84}
	2	1	1	4450	2^{61}
	2	2	1	5518	2^{20}
	2	2	2	6606	2^{7}
16	1	1	1	1703	2^{92}
	2	1	1	2258	2^{71}
	2	2	1	2830	2^{28}
	2	2	2	3406	2^{7}
32	1	1	1	903	2^{100}
	2	1	1	1162	2^{72}
	2	2	1	1486	2^{53}
	2	2	2	1806	2^{38}
	3	2	2	2065	2^{14}
	3	3	2	2389	2^{8}
	3	3	3	2709	2^{7}
64	1	1	1	503	2^{108}
	2	1	1	614	2^{82}
	3	1	1	725	2^{72}
	3	2	1	925	2^{61}
	3	3	1	1125	2^{44}
	3	3	2	1317	2^{18}
	3	3	3	1509	2^{7}

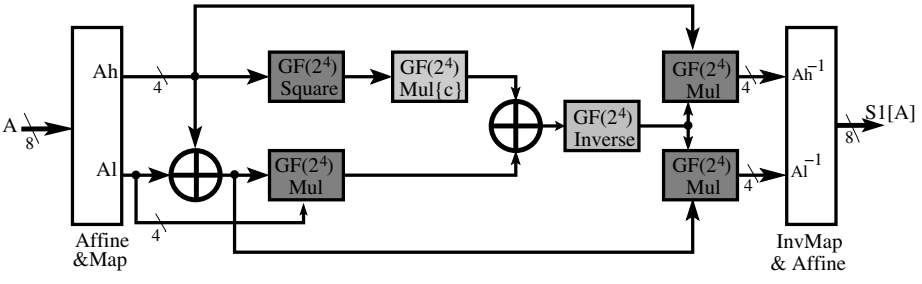


Fig. 7. Composite Field Implementation of $S1$ for 64 byte Tables

field $GF(2^8)$. To fit this into a 32 byte cache line, [12] used composite fields, and had some of the composite field operations such as inversion and multiplication by a constant stored in tables. However due to the 32 byte constraint, many operations were still performed by logical equations, resulting in significant overheads. Since in our case, the cache line is larger so more operations can be done using tables. Figure 7 shows our modified composite field implementation for $S1$. The lighter shaded boxes are tables which were also implemented in [12], while the darker boxes are new tables for 64 byte cache lines. The squaring is done using a table of 16 bytes and part of the $GF(2^4)$ multiplication is done using a single 32 byte table. This table stores the product of a 4 bit element and a 2 bit element. Thus a $GF(2^4)$ multiplication is done with only two table accesses.