A Cache Trace Attack on CAMELLIA

Rishabh Poddar, Amit Datta, and Chester Rebeiro

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

Abstract. CAMELLIA is a 128 bit block cipher certified for its security by NESSIE and CRYPTREC. Yet an implementation of CAMELLIA can easily fall prey to cache attacks. In this paper we present an attack on CAMELLIA, which utilizes cache access patterns along with the differential properties of CAMELLIA's s-boxes. The attack, when implemented on a PowerPC microprocessor having a 32 byte cache line size requires power traces from 2^{16} different encryptions. Further, the work shows that this trace requirement reduces to 2^{11} if a 64 byte cache line is used.

1 Introduction

With the development of newer and better encryption schemes, it has become increasingly difficult to find flaws in the algorithm and therefore the schemes are more secure. However, implementations of the encryption algorithms are highly susceptible to being attacked. Attacks that target implementations are known as *side channel attacks*, and were discovered by Paul Kocher in 1996 [10]. These attacks take advantage of the information that gets leaked during the cipher's execution. The channels for leakage are generally power consumption, timing for execution, and electro-magnetic radiation.

Cache attacks are a class of side-channel attacks that glean secret information from the behavior of the processor's cache memory. These attacks utilize the fact that a cache miss has a different power and timing profile compared to a cache hit. Cache attacks were first prophesied by Kelsey et al. in [9]. A theoretical model of a cache attack was then constructed by Page in [12]. In [17] and [18], the first cache attacks were successfully demonstrated. The ciphers targeted were MISTY1, DES, and 3-DES. Since the arrival of AES, it has been the favorite choice among side-channel attackers. There were several variants of cache attacks that were demonstrated on AES [1,3,4,5,6,7,8,11,16]. All attacks on AES can be classified into three depending on the channel used to collect information. These channels are power consumption traces, spy processes, and timing information. In scenarios where the attacker has direct access to the encryption device, monitoring power consumption traces is the best strategy in order to minimize the interactions with the device. These attacks, which came to be known as *cache trace attacks*, was the method used to attack AES in [1,4,7,8].

M. Joye et al. (Eds.): InfoSecHiComNet 2011, LNCS 7011, pp. 144–156, 2011.

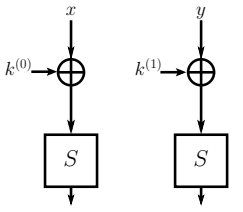

Fig. 1. S-box Table Accesses

One strategy common to all cache attacks is to split the large secret key of the cipher into a number of small key parts (for example the 128 bit key of AES is split into 16 bytes). During the attack, each of these key parts is obtained independently and then combined to obtain the entire key. In a Substitution-Permutation Network (SPN) structure like AES, all 16 bytes of the key are used in the first round itself. Attacking just the first round is simple because it is easy for the adversary to have control of the round inputs. In Feistel ciphers however, retrieving the entire 128 bit key often requires attacking more than one round. This is much more difficult because as the depth of the attack (in terms of the round being attacked) increases, it becomes increasingly difficult for the attacker to control the round inputs. In [13] for example, an attack was demonstrated on the generalized Feistel cipher CLEFIA [15], where obtaining the entire 128 byte key required attacking three rounds of the cipher. As seen in [13], the attack on the first round is very simple compared to the second round, while the third is the most complex.

In this paper we propose a cache trace attack on the 128-bit block cipher CAMELLIA [2]. CAMELLIA like CLEFIA is based on the Feistel structure. Therefore our attack follows a similar strategy as the attack on CLEFIA described in [13]. CAMELLIA however has the classical Feistel structure, while CLEFIA uses a type-2 generalized Feistel structure [19]. In a type-2 generalized Feistel structure, the adversary can control round inputs up to the 4^{th} round. In the classical Feistel structure, however, only the second round inputs can be controlled. Further, retrieving the 128 bit CLEFIA key required attacking only 3 rounds. On the other hand, for CAMELLIA, 4 rounds need to be attacked. These reasons make an attack on CAMELLIA a bigger challenge and hence motivates the work in this paper.

The outline of the paper is as follows: the next section introduces cache attacks and gives a brief summary of CAMELLIA. Section 3 present the attack procedure. Section 4 discusses how the attack was practically mounted. The final section has the conclusions and future directions.

2 Preliminaries

In this section we first present the principle behind cache attacks and then give a brief description of the CAMELLIA structure.

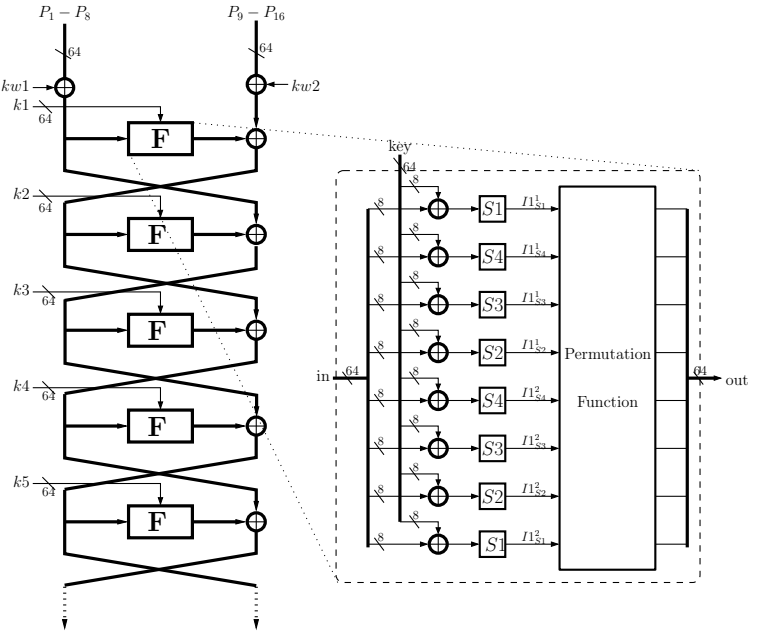

Fig. 2. Partial Structure of CAMELLIA

2.1 Principle of Cache Attacks

All cache attacks target structures in the block cipher such as in Figure 1. The figure shows two accesses to table S with indices $(x \oplus k^{(0)})$ and $(y \oplus k^{(1)})$. When a cache hit occurs the following relation holds, leading to leakage of information about the ex-or of the keys.

$$\langle k^{(0)} \oplus k^{(1)} \rangle = \langle x \oplus y \rangle \tag{1}$$

We note that due to the affects 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 $k^{(0)}$ and $k^{(1)}$ 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.

$$\langle k^{(0)} \oplus k^{(1)} \rangle \neq \langle x \oplus y \rangle \tag{2}$$

2.2 The CAMELLIA Structure

CAMELLIA is the 128-bit block cipher that was jointly developed by Mitsubishi and NTT in 2000. Since this cipher has been made available under a royalty-free license, it has been certified for use by the European Union and Japan. It has also become part of the OpenSSL Project, and incorporated in Mozilla's Network Security Services (NSS modules). Support for CAMELLIA has been added to several security libraries as well as Mozilla's popular web-browser, Firefox 3.

CAMELLIA has been so designed that an encryption can be done using either a 128-bit, a 192-bit or a 256-bit key. We have tested our attack on a 128-bit implementation. However the techniques described can be extended to other key lengths.

The 128−bit block cipher CAMELLIA [2] has a Feistel structure as shown in Figure 2. The 16 bytes plaintext input $P_1 \cdots P_{16}$ is grouped in two words of 8 bytes each. There are 18 rounds in all, broken up into groups of 6 each. After the 6^{th} and the 12^{th} rounds, there are two FL/FL^{-1} function layers. In each round, there is an F-function, which is a combination of key addition, substitution and diffusion. The substitution is done by using four s-boxes, whereas the diffusion is implemented as follows:

$$\begin{pmatrix} z_1 \\ z_2 \\ . \\ . \\ z_8 \end{pmatrix} \longmapsto \begin{pmatrix} z'_1 \\ z'_2 \\ . \\ . \\ z'_8 \end{pmatrix} = M \cdot \begin{pmatrix} z_1 \\ z_2 \\ . \\ . \\ z_8 \end{pmatrix}$$

where,

$$M = \begin{pmatrix} 1\,0\,1\,1\,0\,1\,1\,1 \\ 1\,1\,0\,1\,1\,0\,1\,1 \\ 1\,1\,1\,0\,1\,1\,0\,1 \\ 0\,1\,1\,1\,1\,1\,1\,0 \\ 1\,1\,0\,0\,0\,1\,1\,1 \\ 0\,1\,1\,0\,1\,0\,1\,1 \\ 0\,0\,1\,1\,1\,1\,0\,1 \\ 1\,0\,0\,1\,1\,1\,1\,0 \end{pmatrix}$$

Each round has an addition of a round key. The i^{th} round uses the round key k_i. Each of these round keys are of 64 bits. Additionally, whitening keys kw_1 and kw_2 are applied at the start of encryption, while kw_3 and kw_4 are applied at the end of encryption.

The implementation of CAMELLIA attacked in this paper consists of one 256 byte table which implements each s-box. The next part of the section discusses the basic principle behind cache attacks.

3 The Attack on CAMELLIA

We depict the first two rounds of CAMELLIA's Feistel structure to describe the principle behind the proposed attack (Figure 3).

The input $\mathbf{x}$ consists of 8 concatenated bytes $(x_1|x_2|x_3|x_4|x_5|x_6|x_7|x_8)$ and is known as the *differential introducing input*. The input $\mathbf{y}$ consists of the bytes $(y_1|y_2|y_3|y_4|y_5|y_6|y_7|x_8)$ and is known as the *restoring input*. The F in the figure is CAMELLIA's F function (see Figure 2). For a particular fixed value of $\mathbf{x}$, we vary the bytes of $\mathbf{y}$ until we obtain cache hits in all s-box tables in the second

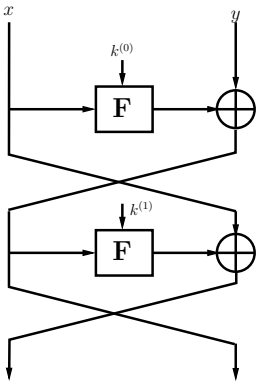

Fig. 3. First two rounds of CAMELLIA

round F function. We call this the *collision setup* phase. At the end of the setup, the following equations holds for $1 \leq i \leq 8$ if a cache hit is obtained.

$$\langle x_i \oplus k_i^{(0)} \rangle = \langle y_i \oplus k_i^{(1)} \oplus F(x, k^{(0)})_i \rangle \tag{3}$$

Similarly, the following inequalities hold if a cache miss is obtained.

$$\langle x_i \oplus k_i^{(0)} \rangle \neq \langle y_i \oplus k_i^{(1)} \oplus F(x, k^{(0)})_i \rangle \tag{4}$$

We now displace the input byte x_1 by $d_{x_1} \neq 0$, keeping the rest of the bytes of $\mathbf{x}$ unchanged. After the s-box operation, the displacement is diffused to the output bytes of the F function. As a result, some or all of the cache hits in the second round are lost. We now modify the bytes of $\mathbf{y}$ to restore cache hits in the second round, and once again obtain the collision state. This is called the *restoring phase*. Let $\mathbf{y}' = (y_1'|y_2'|y_3'|y_4'|y_5'|y_6'|y_7'|y_8')$ be the new value of $\mathbf{y}$ after the modification. Therefore, the differences in the output of the F function are $d_{y_i} = y_i \oplus y_i'$.

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_1}. Let this set be called $\mathcal{D}$. For every output differential $d_o \in \mathcal{D}$, we compute the matrix product $M \cdot (d_o\,,\,0\,,\,0\,,\,0\,,\,0\,,\,0\,,\,0\,,\,0)^T$. This is used to obtain the differentials $(d_{z_1}\,,\,d_{z_1}\,,\,d_{z_2}\,,\,d_{z_3}\,,\,d_{z_4}\,,\,d_{z_5}\,,\,d_{z_6}\,,\,d_{z_7}\,,\,d_{z_8})^T$. For the correct s-box output differential d_o, $\langle d_{z_i} \rangle = \langle d_{y_i} \rangle$ for $1 \leq i \leq 8$. We exploit this to obtain the set of possible keys (S) for the key byte $k_1^{(0)}$. The number of candidate keys can be reduced by repeating the attack several times and taking the intersection between the sets. If r repetitions are done, then,

$$\text{Expected number of candidate keys after } r \text{ repetitions} = \frac{|S|^r}{256^{r-1}} \tag{5}$$

In a similar way, displacements introduced at x_2, x_3,..., x_8, would lead to leakages in $k_2^{(0)}$,$k_3^{(0)}$, ..., $k_8^{(0)}$ respectively. The technique used to generate the candidate keys is given by Algorithm 1, provided the collisions have been set up. The

next part of the section describes the full attack on CAMELLIA. The proposed attack first determines $k_1 \oplus kw_1$, then $k_2 \oplus kw_2$, followed by $k_3 \oplus kw_1$, and finally $k_4 \oplus kw_2$ (see Figure 2). This information is used to reverse the key scheduling algorithm to obtain the entire key.

Algorithm 1. find : Finding Key Byte $k_i^{(0)}$ assuming collisions have been setup

Input: $i \in \{1, 2, \ldots, 8\}$, the differential introducing input x and restoring input y
Output: $S(k_i^{(0)})$: Candidate Key Set for $k_i^{(0)}$

```
1  begin
2      S(k_i^(0)) ← {}
3      x_i ← x_i ⊕ d_{x_i}
4      Restore collisions : Find y'_1, y'_2, ..., y'_8 which causes collisions in the 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_z1, d_z2, d_z3, d_z4, d_z5, d_z6, d_z7, d_z8)^T ← M · (d_o, 0, 0, 0, 0, 0, 0, 0)^T
8          if (⟨d_z1⟩ = ⟨y_1 ⊕ y'_1⟩ and ⟨d_z2⟩ = ⟨y_2 ⊕ y'_2⟩ and ⟨d_z3⟩ = ⟨y_3 ⊕ y'_3⟩ and
           ⟨d_z4⟩ = ⟨y_4 ⊕ y'_4⟩ and ⟨d_z5⟩ = ⟨y_5 ⊕ y'_5⟩ and ⟨d_z6⟩ = ⟨y_6 ⊕ y'_6⟩ and
           ⟨d_z7⟩ = ⟨y_7 ⊕ y'_7⟩ and ⟨d_z8⟩ = ⟨y_8 ⊕ y'_8⟩) then
9              S(k_i^(0)) ← S(k_i^(0)) ∪ {d_o}
10         end
11     end
12 end
```

3.1 Determining $k_1 \oplus kw_1$

Let the 16 bytes of the input plaintext be $(P_1|P_2|\cdots|P_{16})$. We consider the memory accesses to the first and second round F functions. The structure is similar to Figure 3 with the 8 leftmost bytes $P_1, P_2, \cdots P_8$ as the differential introducing input, and the 8 rightmost bytes $P_9, P_{10}, \cdots P_{16}$ as the restoring input. In CAMELLIA, each s-box is used twice per round. Therefore, a resulting cache hit in the second round would be due to collisions with either of these accesses. Let $I\alpha^i_{S\beta}$ be the index to the i^{th} access to table $S\beta$ in round α (see Figure 2). Thus, a collision in $I2^1_{S1}$ could be with either or both of $I1^1_{S1}$ and $I1^2_{S1}$. To eliminate this ambiguity, we ensure that all accesses in the first round are themselves colliding. That is for $S1$,

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

Similarly, for $S2$, $S3$ and $S4$ we have

$$\langle I1^1_{S2} \rangle = \langle I1^2_{S2} \rangle$$

$$\langle I1^1_{S3} \rangle = \langle I1^2_{S3} \rangle$$

and

$$\langle I1^1_{S4} \rangle = \langle I1^2_{S4} \rangle$$

We call such a state the $1-$*round collision state*. Algorithm 2 shows how a $1-$round collision state can be obtained in the cipher. Algorithm 3 is then used to determine $k_1 \oplus kw_1$.

Algorithm 2. 1−round collision

```
   Input: P_1, P_2, P_3 and P_4
   Output: P_5, P_6, P_7 and P_8
1  begin
2      Find P_5 causing a collision in I1^2_{S1}
3      Find P_6 causing a collision in I1^2_{S2}
4      Find P_7 causing a collision in I1^2_{S3}
5      Find P_8 causing a collision in I1^2_{S4}
6  end
```

Analysis : From side-channel analysis, it is possible to determine whether a memory access resulted in a cache hit or a miss. Testing for a collision at a particular memory access requires $\frac{256}{C}$ encryptions, where C is the cache line size. However, by using the inequality in Equation 4, it is sufficient to have $\frac{256}{C} - 1$ encryptions in order to find the collision. Algorithm 2 therefore requires $4(\frac{256}{C} - 1)$ encryptions. Setting up the 8 collisions in line 4 of Algorithm 3 requires $8(\frac{256}{C} - 1)$ encryptions. Further, each invocation of the *find* function requires $8(\frac{256}{C} - 1)$ encryptions. Thus in total, finding all the 8 bytes of $k_1 \oplus kw_1$ requires $60(\frac{256}{C} - 1)$ encryptions.

Algorithm 3. Finding candidate keys for $\kappa = k_1 \oplus kw_1$

```
   Output: S(κ_i) : Candidate Keys for κ = k_1 ⊕ kw_1, where i ∈ {1, 2, ..., 8}
1  begin
2      Randomly select P_1, P_2, P_3 and P_4
3      (P_5, P_6, P_7, P_8) ← 1−RoundCollision(P_1, P_2, P_3, P_4)
4      Set up collisions : Find P_9, P_10, ··· , P_16 causing collisions in I2^1_{S1}, I2^1_{S2}, ..., I2^2_{S4}
       respectively
5      S(κ_1) ← find(1, P_1 ··· P_8, P_9 ··· P_16)
6      S(κ_2) ← find(2, P_1 ··· P_8, P_9 ··· P_16)
7      S(κ_3) ← find(3, P_1 ··· P_8, P_9 ··· P_16)
8      S(κ_4) ← find(4, P_1 ··· P_8, P_9 ··· P_16)
9      S(κ_5) ← find(5, P_1 ··· P_8, P_9 ··· P_16)
10     S(κ_6) ← find(6, P_1 ··· P_8, P_9 ··· P_16)
11     S(κ_7) ← find(7, P_1 ··· P_8, P_9 ··· P_16)
12     S(κ_8) ← find(8, P_1 ··· P_8, P_9 ··· P_16)
13 end
```

For a 64 byte cache line, 228 encryptions are required to obtain a set of 64 candidates per key on average. On average repeating the attack 4 times would result in a single key. Therefore in total 912 encryptions are required. Similarly for a 32 byte cache line, 532 encryptions are required to obtain a candidate key set. Filtering out the wrong keys would require a total of 1596 encryptions.

3.2 Determining $k_2 \oplus kw_2$

We consider the F functions in the second and third rounds to determine candidates for $k_2 \oplus kw_2$, with $P_9 \cdots P_{16}$ as the differential introducing inputs, and $P_1 \cdots P_8$ as the restoring inputs. As in the first stage of the attack, ambiguities about collisions may arise when cache hits are forced in the third round F function. Therefore, the cipher is put in a 2−*round colliding state*. In a 2−*round colliding state*, all accesses except the first are in collision for each table, i.e.,

$$\langle I1^1_{S1}\rangle = \langle I1^2_{S1}\rangle = \langle I2^1_{S1}\rangle = \langle I2^2_{S1}\rangle$$
$$\langle I1^1_{S2}\rangle = \langle I1^2_{S2}\rangle = \langle I2^1_{S2}\rangle = \langle I2^2_{S2}\rangle$$
$$\langle I1^1_{S3}\rangle = \langle I1^2_{S3}\rangle = \langle I2^1_{S3}\rangle = \langle I2^2_{S3}\rangle$$
$$\langle I1^1_{S4}\rangle = \langle I1^2_{S4}\rangle = \langle I2^1_{S4}\rangle = \langle I2^2_{S4}\rangle$$

We note that the results obtained in the previous stage of the attack (i.e. determining $k_1 \oplus kw_1$) may be used to put the cipher in the 2−*round colliding state*. Hence, no additional encryptions are required.

To prevent further ambiguities in the third round accesses to a table, the inputs must be controlled in a manner that causes no hits to be lost in all previous accesses made to that table, i.e. the 2−*round colliding state* remains intact. This is accomplished as follows.

The restoring inputs for this stage of the attack are formed by $P_1 \cdots P_8$ ex-ored with the outputs of the second round F (see Figure 2). These values may be controlled by displacements made in the values of $P_1 \cdots P_8$ ensuring that collisions are not lost in the first two rounds. For example, suppose a hit is desired in $I3^1_{S1}$. This is done by controlling P_1. However changing P_1 may result in loss of collision in $I1^2_{S1}$. Further, the change in P_1 would affect several outputs of the first round F function due to the diffusion. This might disturb the collisions in the second round. To avoid this loss in the 2−round colliding state, the previously determined value of $k_1 \oplus kw_1$ is used. Using this, P_8 can be set to ensure collision in $I1^2_{S1}$ persists. The determined key is also used to compute the outputs of the first round F. The values of $P_9 \cdots P_{16}$ are now ex-ored with these outputs, so that the effect of the disturbance on the second round accesses is annulled. Thus, the 2−round colliding state persists.

This principle can similarly be applied to determine candidates for all the bytes of $k_2 \oplus kw_2$. Algorithm 4 describes the steps for the attack. An important point to note is the modification that needs to be made in the *restore collisions* phase of Algorithm *find*. This step must now be performed in keeping with the discussion in the previous paragraph.

Algorithm 4. Finding candidate keys for $\kappa = k_2 \oplus kw_2$

Input: $P_1 \cdots P_{16}$, so that a 2−*round colliding state* has been set up
Output: $S(\kappa_i)$: Candidate Keys for $\kappa = k_2 \oplus kw_2$, where $i \in \{1, 2, \ldots, 8\}$

```
1  begin
2      Set up collisions : Find P_1, P_2, ..., P_8 causing collisions in I3^1_S1, I3^1_S2, ..., I3^2_S4
       respectively
3      S(κ_1) ← find(1, P_9 ··· P_16, P_1 ··· P_8)
4      S(κ_2) ← find(2, P_9 ··· P_16, P_1 ··· P_8)
5      S(κ_3) ← find(3, P_9 ··· P_16, P_1 ··· P_8)
6      S(κ_4) ← find(4, P_9 ··· P_16, P_1 ··· P_8)
7      S(κ_5) ← find(5, P_9 ··· P_16, P_1 ··· P_8)
8      S(κ_6) ← find(6, P_9 ··· P_16, P_1 ··· P_8)
9      S(κ_7) ← find(7, P_9 ··· P_16, P_1 ··· P_8)
10     S(κ_8) ← find(8, P_9 ··· P_16, P_1 ··· P_8)
11 end
```

Analysis : Obtaining collisions in all eight accesses in the third round requires $8(\frac{256}{C}-1)$ encryptions. Once this is done, deducing each key byte requires $8(\frac{256}{C}-1)$ encryptions. Since there are 8 unknown key bytes, a total of $65(\frac{256}{C}-1)$ encryptions is needed to obtain the candidate key set. With a 64−byte cache line, this attack step should be repeated 4 times to isolate a single key. This requires 768 encryptions. Similarly with a 32−byte cache line, and 3 iterations of the attack step, a single key is isolated in 1365 encryptions.

3.3 Determining $k_3 \oplus kw_1$ and $k_4 \oplus kw_2$

To determine candidates for $k_3 \oplus kw_1$, we consider the F functions in the third and fourth rounds, with $P_1 \cdots P_8$ as the differential introducing inputs, and $P_9 \cdots P_{16}$ as the restoring inputs. The cipher should be ideally put in a 3−*round colliding state* before mounting the attack. However, achieving such a state is more difficult than obtaining a 2−*round colliding state*. To circumvent this difficulty, we initially put the cipher in a *partial* 3−*round colliding state* using the following technique, for the third stage of the attack.

We first obtain a 2−*round colliding state*. The values of $P_9 \cdots P_{16}$ are subsequently ex-ored by small amounts less than the cache line size. Since these displacements are small, accesses in the second round continue to collide with the same cache lines, and no hits are lost. These small displacements, however, would become random changes affecting all outputs of the second round F. Thus, for suitable values of these displacements, collisions may be established in particular accesses of our choice in the third round (creating a *partial* 3−*round colliding state*), without losing any hits in the first 2 rounds. The state is thus a *partial* colliding state because not all third round accesses are hits. The colliding accesses are chosen as follows.

Suppose the byte of the key to be determined is ex-ored with a byte of the differential introducing input ($P_1 \cdots P_8$) that accesses table $S\beta$ in the third round F (see Figure 2). Then both accesses to $S\beta$ must be established as hits. Evidently, once this state has been obtained, for $\beta = 1$,

$$\langle I1^1_{S1}\rangle = \langle I1^2_{S1}\rangle = \langle I2^1_{S1}\rangle = \langle I2^2_{S1}\rangle = \langle I3^1_{S1}\rangle = \langle I3^2_{S1}\rangle$$

while for $\beta \neq 1$, $\langle I3^1_{S\beta}\rangle$ and $\langle I3^2_{S\beta}\rangle$ may or may not be equal to $\langle I1^1_{S\beta}\rangle = \langle I1^2_{S\beta}\rangle = \langle I2^1_{S\beta}\rangle = \langle I2^2_{S\beta}\rangle$. This does not affect the results of the attack in any way.

The restoring inputs are controlled following the principle described in section 3.2 using bytes $P_9 \cdots P_{16}$. However, this might lead to ambiguities in the fourth round memory accesses. For example, a cache hit in $I4^1_{S1}$ can be forced by small displacements in one or more of $P_9 \cdots P_{16}$. Since these displacements are small, no collisions will be lost in the second round accesses to $S1$. However, changing these bytes may cause hits to be lost in the third round $S1$ accesses. Thus a collision in $I4^1_{S1}$ may be a result of the desirable cache hit with $I1^1_{S1}$, $I1^2_{S1}$, $I2^1_{S1}$ and $I2^2_{S1}$, or due to undesirable cache hits with $I3^1_{S1}$ and $I3^2_{S1}$.

Thus, for a table $S\beta$, a collision in the fourth round is the result of a desirable cache hit with a probability $\frac{1}{n+1}$, where n is the number of misses in the third

round accesses to the table. Since 2 accesses are made to each table per round, this probability is at least $\frac{1}{3}$, and sufficient confidence in the correctness of the collision is obtained if the test is repeated 3 times.

In the fourth stage of the attack, candidates for $k_4 \oplus kw_2$ are obtained by considering the F functions in the fourth and fifth rounds, with $P_9 \cdots P_{16}$ as the differential introducing inputs, and $P_1 \cdots P_8$ as the restoring inputs. Ideally, the cipher should be initially put in at least a *partial* $4-$*round colliding state* before mounting the attack on the fourth round. However, such a state cannot be easily obtained without losing collisions in the accesses of the previous rounds. Moreover, attempting small displacements in the values of $P_9 \cdots P_{16}$ will entail a lot more encryptions to obtain a *partial* $4-$*round colliding state*, as opposed to a *partial* $3-$*round colliding state*. Therefore, we proceed to mount the attack with the cipher in a *partial* $3-$*round colliding state* as obtained earlier.

Collisions in the fourth round may be established at the expense of hits in the third round. For a table $S\beta$, this implies that hits obtained in the fourth round may either be due to desirable collisions with accesses in the first 2 rounds, or due to undesirable collisions with any access resulting in a miss in the third round. That is, for table $S1$,

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

are desirable collisions. Accesses $\langle I3^1_{S1} \rangle$ and $\langle I3^2_{S1} \rangle$ may or may not be desirable. Thus, for a table $S\beta$, a collision in the fourth round is the result of a desirable cache hit with a probability $\frac{1}{m+1}$, where m is the number of misses in the third round accesses to the table.

Similarly, we may obtain hits in the fifth round with the restoring inputs by small displacements of $P_9 \cdots P_{16}$ at the expense of the third and fourth round hits. The probability of a correct hit is thus $\frac{1}{n+1}$, where n is the total number of misses in the third and fourth round accesses. Therefore, the entire method results in a success with probability $\frac{1}{(m+1)(n+1)}$. Since in the worst case, $m = 2$ and $n = 4$, this probability is at least $\frac{1}{(2+1)(4+1)} = \frac{1}{15}$, and the test should be repeated 15 times on an average, to obtain sufficient confidence in the result.

Analysis : Once a $2-$*round colliding state* has been obtained, we need to enforce hits in 2 accesses in the third round to get a *partial* $3-$*round colliding state*. Since an access to a table may occupy any of 4 cache lines for $C = 64$ and 8 cache lines for $C = 32$, we get the required cache hit with a probability $\frac{1}{4}$ on 64 byte cache lines and $\frac{1}{8}$ on 32 byte cache lines after one encryption.

For the third stage of the attack, once a *partial* $3-$*round colliding state* has been created, collisions are obtained in the fourth round with a minimum probability of $\frac{1}{3}$. Thus, an attack in the third stage is successful with a probability $\frac{1}{3\times4} = \frac{1}{12}$ for $C = 64$ and $\frac{1}{3\times8} = \frac{1}{24}$ for $C = 32$. Thus for $C = 64$, 12 encryptions are required to extract key candidates for one byte of the key. Since the key comprises 8 bytes, the expected number of encryptions is $12 \times 8 = 96$. This has to be repeated at-least 8 times to isolate a unique key, making a total requirement of 768 encryptions. Similarly, for $C = 32$, a total of 1536 encryptions are required.

For the fourth stage, the attack is successful with a minimum probability $\frac{1}{15}$. Since we need to obtain a *partial* $3-$*round colliding state* before mounting the attack, the minimum probability of success is $\frac{1}{15\times 4} = \frac{1}{60}$ for $C = 64$. Thus, 60 encryptions are required on an average to extract one byte of the key. The expected number of encryptions for extracting the entire key is $60 \times 8 \times 8 = 3480$ when $C = 64$ and 7680 encryptions are needed when $C = 32$.

3.4 Obtaining the Secret Key

The third and fourth stages of the attack on the cipher leak the values of $k_3 \oplus kw_1$ and $k_4 \oplus kw_2$ respectively. From the CAMELLIA key schedule for a 128-bit key, we know that

$$k_3 \oplus kw_1 = (K_L \oplus (K_L \lll 15))_L$$

$$k_4 \oplus kw_2 = (K_L \oplus (K_L \lll 15))_R$$

Thus, from the attack on the first 4 rounds, we can obtain the value $C = (K_L \oplus (K_L \lll 15))$. This information is sufficient to derive the value of K_L, which is the required secret 128-bit key. Algorithm 5 shows how the candidates for the secret key can be extracted. Using the algorithm, we get at most 2 candidates for K_L. For the sake of convenience, the 128-bit values have been represented as arrays of dimension 128.

Algorithm 5. Finding candidates for the secret key K_L

```
   Input: C[] = (K_L ⊕ (K_L ⋘ 15))
   Output: S(K) : Candidate key set for the secret key K_L
 1 begin
 2     S ← {}
 3     K[] ← {0}
 4     for lsb ∈ {0, 1} do
 5         K[0] ← lsb
 6         j ← 0
 7         for i ∈ {0, 1, ..., 127} do
 8             if j ≥ 128 then
 9                 j ← j%128
10             end
11             K[(j − 15)%128] ← K[j] ⊕ C[j]
12         end
13         S ← S ∪ {K}
14     end
15     return S
16 end
```

4 Practically Mounting the Attack

To test the attack we used a similar setup as in [13] with CAMELLIA's reference code[1] modified to use 4 tables. The attack consists of two phases, the online and the offline phase, which are repeated for each step of the attack. The online phase dealt with obtaining the required power traces from the board. The attack

[1] http://info.isl.ntt.co.jp/crypt/eng/camellia/source.html

targeted the cache of the PowerPC present in the Xilinx FPGA in the SASEBO board [14]. The PowerPC cache has a 32 byte cache line and a size of $16KB$. During the offline phase, the traces are first analyzed in Matlab to retrieve the cache access patterns. This is then fed to the analysis program, which guesses the secret key. For the 32 byte cache line, the number of encryptions required to be made is 12177.

5 Conclusions and Future Directions

The paper presents a cache trace attack on the 128 bit block cipher CAMELLIA. The attack first finds the keys used in rounds 1 to 4 and then uses the key scheduling algorithm of CAMELLIA to retrieve the entire secret key. On a PowerPC processor, having a 32−byte cache line size, this requires monitoring of around 2^{16} power traces. On a processor using the more standard 64 byte cache line, the number of power traces required are 2^{11}.

A comparison of the proposed attack on CAMELLIA with cache trace attacks on AES exemplifies the difficulty in attacking Feistel ciphers. As a future work various cipher structures can be analyzed for their robustness against cache attacks. This analysis would play a pivotal role in constructing a cipher inherently secure against cache attacks.

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. Aoki, K., Ichikawa, T., Kanda, M., Matsui, M., Moriai, S., Nakajima, J., Tokita, T.: Specifications of Camellia – a 128-bit Block Cipher (2001)
3. Bernstein, D.J.: Cache-timing Attacks on AES. Tech. rep. (2005)
4. 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, Los Alamitos (2005)
5. Bonneau, J., Mironov, I.: Cache-Collision Timing Attacks Against AES. In: Goubin, L., Matsui, M. (eds.) CHES 2006. LNCS, vol. 4249, pp. 201–215. Springer, Heidelberg (2006)
6. Canteaut, A., Lauradoux, C., Seznec, A.: Understanding Cache Attacks. Research Report RR-5881, INRIA (2006), `http://hal.inria.fr/inria-00071387/en/`
7. Fournier, J.J.A., Tunstall, M.: Cache Based Power Analysis Attacks on AES. In: Batten, L.M., Safavi-Naini, R. (eds.) ACISP 2006. LNCS, vol. 4058, pp. 17–28. Springer, Heidelberg (2006)
8. Gallais, J.-F., Kizhvatov, I., Tunstall, M.: Improved Trace-Driven Cache-Collision Attacks against Embedded AES Implementations. In: Chung, Y., Yung, M. (eds.) WISA 2010. LNCS, vol. 6513, pp. 243–257. Springer, Heidelberg (2011)
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. Kocher, P.C.: Timing Attacks on Implementations of Diffie-Hellman, RSA, DSS, and Other Systems. In: Koblitz, N. (ed.) CRYPTO 1996. LNCS, vol. 1109, pp. 104–113. Springer, Heidelberg (1996)

11. 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)
12. Page, D.: Theoretical Use of Cache Memory as a Cryptanalytic Side-Channel (2002)
13. 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)
14. Research Center for Information Security National Institute of Advanced Industrial Science and Technology: Side-channel Attack Standard Evaluation Board Specification (Version 1.0) (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. 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)