Building an Evict and Time Simulator to Assess the Security of Memory Cache Designs

Research Review 2017

Author: Thomas Foster

Supervisor: Dr Daniel Page

Acknowledgements

I would like to thank my supervisor Dr Daniel Page for his patience and his support during researching this complicated but fascinating area of cryptography.

I would also like to thank Professor Elisabeth Oswald for meeting with me to discuss about the area of side channel attacks.

Executive Summary

AES (Advanced Encryption Standard) is the now one of the industry standard ways in ciphering data due to its strong mathematical underpinning. Yet memory caches can leak information about the key if an attacker could send and observe encryption operations.

This project aims to evaluate the weakness of specific cache designs when it comes to information leakage in one to two controlled side-channel attacks. The focus will be on the impact of time-based attacks, and whether different cache designs can resist information leakage more successfully than other designs.

Software development and investigatory research will be the two prominent focuses divided 80|20 respectively. The project aims to build an operating simulator with the ability to execute an evict and time side channel attack on a memory cache design.

Objectives:

- Design and build a basic memory cache architecture, take this as the primary design to test the side channel attack. I will also look in developing a second design of a memory cache to compare weakness of information leakage.
- Subject these one to two cache designs with evict and time attack.
- Deliver a tool that can evaluate cache designs based on previous cache designs as validation.

Deliverables

- Working simulator along with at least two cache designs with one clear method to execute at least one side-channel attack.
- Analysing the data of the time-based attack to offer a series of preliminary conclusions.
- Simulator that will provide a general illustrative overview of how cache designs are vulnerable to time-based attacks along with recommendations of future research and how to expand the capacity of the simulator.

Added value

- This project will aim to deliver a flexible working cache simulator to be utilised in further research within the cryptographic side-channel attack field.
- This research will thus aid those both new and currently in this area of research to gain a greater understanding of information leakage in side channel attacks concerning memory cache design.
- The simulator will provide the foundation for expansion of a more sophisticated and comprehensive side channel attack simulator to test memory caches.
- Criteria of which to base the success of the resilience of cache designs in evict and time attacks based on the evidence outputted from the simulator.

Contents

Table of Figures

1 Introduction

1.1 Background and Motivations

Encryption relates to the secrecy (or confidentiality) of messages between two parties through the process of converting plaintext M into ciphertext C, which can then be de-encrypted by a user with an appropriate key.

The area of focus here is side channel attacks. This is where there is 'accidental leakage' from software and hardware devices [1], [2]. Accidental leakage is where a device unknowingly is leaking information to an attacker. This information is then used by the attacker to decode parts of the key used to encrypt messages. There are several types of side channel attacks, for this research I am concerned with timing attacks, where timing of instruction execution can lead to accidental leakage.

There are various reasons why side channel attacks exist, such as spatial and temporal sharing of processor units within the computer processing unit (CPU) and improvements in processor architecture, this has resulted in optimisations creating faster execution times and increased computational power [1, pp. 1–2]. With these increased performance capabilities, a new field to security researchers exploiting these capabilities to attack the key through information leakage has been created.

Timing attacks are when the machine is performing a cryptographic function and the attacker can measure how long it takes to perform these functions, a prerequisite of these attacks is often that the attacker have intimate knowledge of the processor (but not necessarily) [2, p. 2]. Thus, if the attacker has access to the inputs, and by carefully selecting the inputs and timing the processing speed, they can decrypt the key. As Szefer states, "The more the software uses functional units that have most impact on performance, the more it is susceptible to attacks"[1, p. 1]. This is because whilst building new software and hardware there is often a trade-off between security and performance.

Memory caches have been targeted specifically by many researchers [3]–[5] investigating this area, as the purpose of caches is to speed up processing times, and software developers have adapted their encryption software to utilise the cache. For example, AES an industry standard of encryption [6] uses look up tables to speed up encryption, this is where multiple functions are pre-computed and stored in a table to be matched with its plaintext pair depending on the algorithm.

Models used to uncover side channel leakage are more involved than the usual methodologies of software testing, as it often has a physical measurements of hardware that has to be replicated in the software [3, p. 29]. Theoretical models of cache designs [8], [9] even if in theory are resistant to side channel attacks, can still be susceptible in practise. Having a simulator to test the cache design in a software stage, will provide a means to make improvements before building the design in hardware.

Information leakage in caches introduces a new challenge in cache design. In addition to performance, power efficiency, reliability etc. cache designs now must also take security into account, which typically introduces even more restrictions in cache designs and compromises other design goals [9, p. 84]. Protected cache lines or randomised cache line algorithms [3] are often a great security capability, but can be detrimental in performance capabilities.

Cache designers are thus stuck in a juxtaposition of being a prime target for side channel attacks, but also has the capabilities to make modern CPUs highly resistant to these attacks [9], [10].

Therefore, my project will be focusing on aiding cache designers in the software design stage by building a simulator to assess the resilience of these designs to evict and time attacks. In doing so I will have a working simulator and be able to offer preliminary conclusions on the security of two different cache designs. One cache design will be of a basic layout without security in mind, whilst the other will have some sort of security addition. This should provide substantive results in evaluating these designs.

The scope of this project is limited to the security aspects of cache design rather than performance, and any performance metrics in this research review will only be analysed from a cryptanalysis point of view.

1.2 Chapter Breakdown

The literature review will follow this format:

AES Encryption – An overview of AES encryption with explanation of the functions and the most relevant parts of AES encryption to this project. I will also explain briefly which AES encryption I will be using in the simulator.

Memory Cache – An introduction to memory caches as well as motivations for choosing the 8-way set associative memory cache for the basic design. Then I explain in detail the secondary cache design, Newcache, which has important security features to resist side channel attacks.

Side Channel Attacks – Explanation of a time-based side channel attacks as well as examples of side channel attacks used to expose parts of the AES key from memory caches. Also, how AES look-up tables are used by attackers to reduce the complexity of the key algorithm.

Simulator – This will introduce the inspiration for this project, Dinero IV, and discuss some of its limitations of Dinero IV. Furthermore, it will highlight important design principles when building the simulator, as well as general objectives from the research for this project.

1.3 Research Methodology

During the project, I have primarily utilised three sources when applying a research methodology to find relevant academic papers:

Google Scholar – I put relevant key terms into Google Scholar and collected the top 10 papers for this field, after looking over their abstracts and author keywords to determine their relevance to this project I narrow it down to about five papers to read. Furthermore, I used the Google Scholar metrics to find out the most popular journals within the area of cryptography.

Library – Whilst researching I emailed the library services for the Engineering and Computer Science department for advice on how to search for top journals within the area. Using Scopus, they could send me the top journals, which I have then cross referenced with Google Scholar's results to determine credibility.

Academic staff and colleagues: Due to the well-established Cryptography Research Group at Bristol University, many of the lecturers and academic staff in the department are knowledgeable of the sector. Dr Daniel Page and Professor Elisabeth Oswald have been invaluable in providing their expertise whilst researching and writing up this research review.

2 Literature Review

2.1 AES Encryption

2.1.1 Overview of AES Encryption

AES is an industry standard[6] and widely used, thus I will be implementing an AES-128 encryption algorithm in the simulator. AES-128 encryption is a symmetric iterated block cipher algorithm which utilises a substitution-permutation network. A sub-permutation network is a network of substitution boxes (S-Boxes) and permutation boxes (P-boxes) going through a series of rounds.

Equation 1 – AES Encryption

$$Enc: \{0,1\}n_k \; X \; \{0,1\}n_b \rightarrow \{0,1\}n_b$$

$$Dec: \{0,1\}n_k \; X \; \{0,1\}n_b \rightarrow \{0,1\}n_b$$

Equation 1 is a simple representation of the encryption and decryption of AES, where the encryption is taking some set of key bits and set of block bits as inputs, and outputting the ciphered block set. The decryption is the reverse, taking some key bit and some block bit and outputting a plaintext bit. As a symmetric block cipher, it must be inversible for the encryption/decryption (and vice versa) to work.

Roberto Avanzi stated that "A block cipher is a pair of deterministic algorithms, called encryption and decryption, that respectively apply and undo a permutation, selected by a key, of the set of all blocks (bit strings) of a fixed length" [11, p. 21]. Thus, AES has 10 rounds which applies to four different actions after the user has inputted a 16-byte (or 128 bit[1]) key:

`SubBytes` - Replacing selected input bits from the S-box. Creating a matrix of 4x4.

`ShiftRows` – Then the matrix depending on the row is shifted to the left by some number from 0-3 depending on row number.

`MixColumns` – The columns of the current matrix bytes are taken in as inputs and a new matrix is outputted. This step is not repeated in the last round. This along with the `ShiftRows,` are the permutation stages.

`keyAddition` - The matrix bits are $\oplus$ (XOR) to the 128 bits of the round key. In the last round, the output is cipher-text, every other round the resulting 128 bits are interpreted as 16 bytes making a sub-key.

[1] User input is translated into ASCII value to comprise the key

For a 128 AES key the block size is represented as a 4x4 matrix of bytes of elements of Galois Field $\mathcal{F}_{2^8}$, which appeases closure, associativity, identity and inverse together with a binary operator[2]. Figure 1 provides a visual representation of the matrix.

$$\begin{pmatrix} a_{0,0} & a_{0,1} & a_{0,2} & a_{0,3} \\ a_{1,0} & a_{1,1} & a_{1,2} & a_{1,3} \\ a_{2,0} & a_{2,1} & a_{2,2} & a_{2,3} \\ a_{3,0} & a_{3,1} & a_{3,2} & a_{3,3} \end{pmatrix}$$

Figure 1 4x4 Matrix [11, p. 180]

The field is a binary field where $\mathcal{F}n \in \{0,1\}$ using a polynomial basis representation, where the degree of the polynomial is no more than n-1. As this is a binary field all coefficients are in the fields $\mathcal{F}^2$. Arithmetic is done through a modulo p (In this case p = 2) whilst multiplication is achieved via an irreducible polynomial $X^8 + X^4 + X^3 + X + 1$[3].

Block ciphers operate by diffusion and confusion to provide confidentiality of messages between parties. Diffusion means the statistical structure of the plaintext is altered, whilst confusion means that the key is used so even when the attacker knows the plaintext, or has many plaintext/ciphertext pairs, they cannot easily deduce the key [11, p. 28].

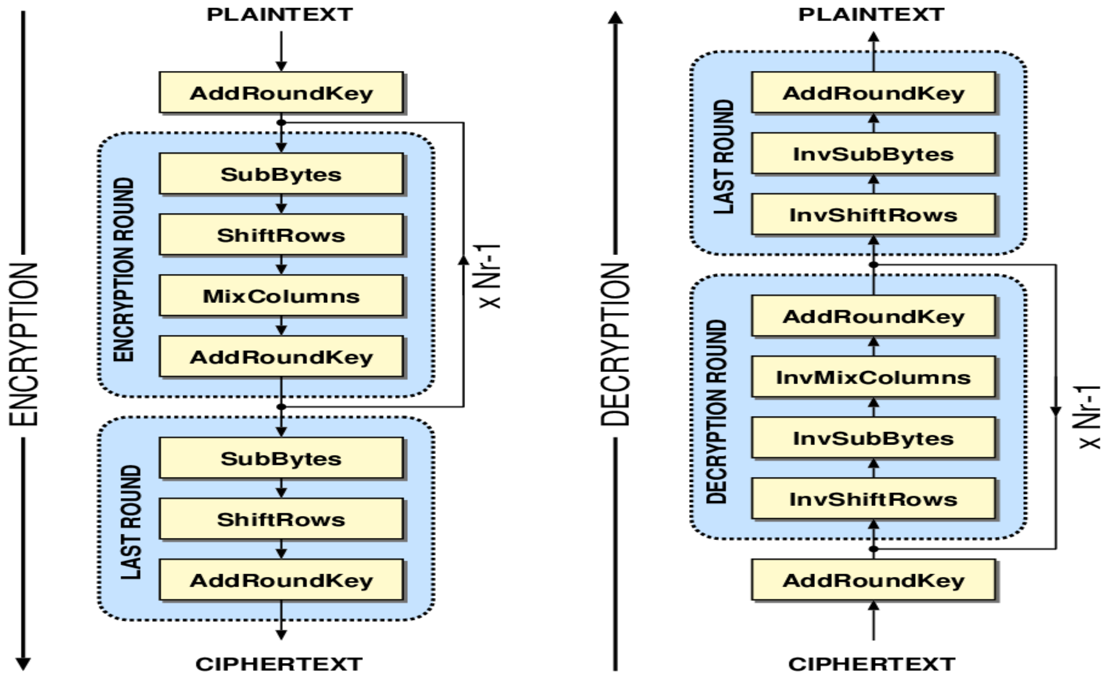


Figure 2 AES Layout [12]

AES uses both confusion and diffusion to achieve confidentiality of messages, refer to Figure 2 for an overview of AES encryption as I go through each step starting with the `KeyAddition` step.

---

[2] Explanation of Galois Field and its application in AES is beyond the scope of this project, please refer to Introduction to Finite Fields and their Applications [27]

[3] I will be using bytes to explain AES thus this is the field representation of AES I will be using, for more information on the fields used in AES refer to [16]

Placing substitution and permutation layers at the very beginning or end of a cipher does not increase security, because they are fixed modifications of the plaintext or ciphertext thus an attacker can simply invert the operations to get the message. This is why the round key function is added at the beginning or end of the encryption [13, p. 30].

The actual `KeyAddition` function is implemented by a simple XOR operator with the key state (the intermediate cipher result in the rounds[13, p. 8]) being equal to the block length of the inputted plaintext (in this is case 4x4 of 128 bits).

After the first `KeyAddition`, the round begins, which starts with the `SubByte` stage. This is a non-linear invertible function substituting each part of the input matrix for the S-box.

The S-box itself is a composition of two functions:

Equation 2 – S-box composition of functions

$$Sbox(a) = (f \circ g)(a) = f(g(a))$$

Where g is the field inversion and f is the affine transformation. The function g is taking the multiplicative inverse of $\mathcal{F}^8$[13, p. 11]. In the field two numbers are co-prime if they share no factors other than 1. Therefore, if the greatest common divisor is $(\gcd)(a, b) = 1$, then b has a multiplicative inverse modulo a. Therefore, for any positive integer $b < a$, there exists, $b \wedge \{-1\} < a$ such that $b \cdot b \wedge \{-1\} = 1 \; mod \; a$ [14, p. 4117]. These multiplicative inverses are then represented as a hexadecimal or polynomial in the S-box. The second step is to apply an invertible affine transformation on each value over $\mathcal{F}^2$.

The maths involved in the `SubByte` stage is complicated, however, you can visually represent it as a simple substitution as shown in Figure 3.

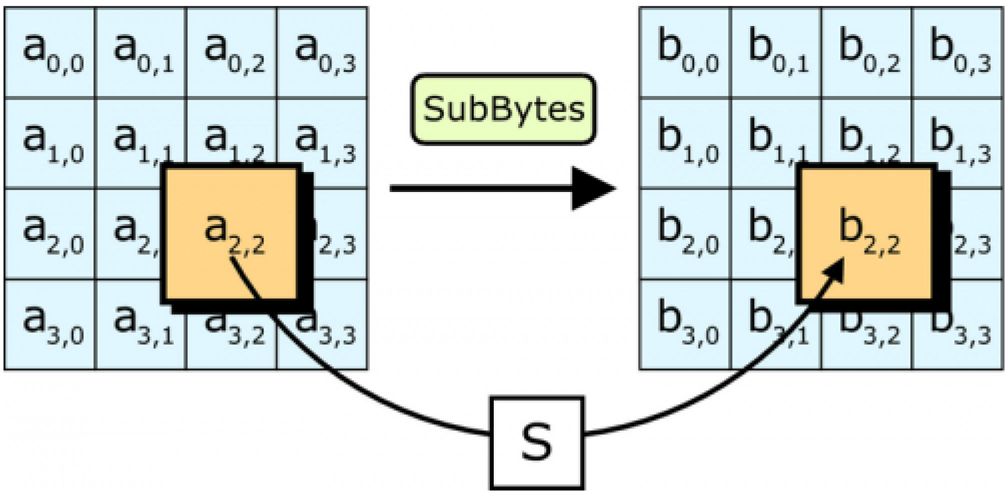


Figure 3 `SubByte` Stage [15]

The next step of the AES encryption process is `ShiftRows`. This step is simply shifting the row n by n-1, so the first row remains unchanged whilst the second, third and fourth row are moved by 1, 2 and 3 respectively.

The next step in the round is `MixColumns` where each column of the matrix is mixed. To do this the column is treated as a $(4x1)$ column vector of entries in $\mathcal{F}_{2^8}$, and pre-multiplied by column vector by the $(4\ x\ 4)\ \mathcal{F}_{2^8}$ (matrix M (Figure 4)), which replaces the column being processed [16, pp. 39–40].

$$M = \begin{bmatrix} 2 & 3 & 1 & 1 \\ 1 & 2 & 3 & 1 \\ 1 & 1 & 2 & 3 \\ 3 & 1 & 1 & 2 \end{bmatrix}$$

Figure 4 Matrix M [17, p. 9]

As you can see from Figure 4 the matrix is composed entirely of 1, 2 and 3s. Therefore, any multiplication in the table will either leave the byte unchanged if it is a dot product with 1, and any output of 2 or 3 can be stored in a lookup table.

For each round to be secure there needs to be a `RoundKey` to provide round specific constants. This is provided by a key schedule which is split into:

- circular rotate on a 32-bit word
- `SubBytes`
- rcon operation
- Key schedule routine

The key from the user input are split into columns into a matrix of 4x4. The key schedule is denoted by $W = NB\ \cdot (NR + 1)$[13, p. 16], where W is word (one of the four columns of 4 bytes), NB is number of bytes and NR is number of rounds. Therefore, for a block length of 128 bits and 10 rounds, 1408 `RoundKey` bits or 176 bytes are needed.

Circular rotation moves each byte to the left by one. If we had the input 1d2c3a4f then the output after the circular rotation would be 2c3a4f1d.

The rcon operation is the exponentiation of 2 to a byte in $\mathcal{F}_{2^8}$, the rcon operation is denoted by rcon(i) = b^{i-1} where the exponent marks where in the field to swap the value [13, pp. 14–15]. An example would be rcon(3) = 4 = b^3 = 00000100. This step is done just to the MSB (most significant bit) byte of the word outputted from the S-box.

This is then repeated for all four columns till there is a new key to be used in the round. This is done for each round till we reach 176 bytes of the key or completed the 10 rounds. The overview of AES is necessary to be able to understand how the AES lookup tables will be put into the cache, and provides context in why pre-computed AES tables as a trade-off between performance and security. Due to the complicated mathematical operations, without AES lookup tables, the CPU performance can be impacted greatly.

2.1.2 AES Encryption in the Simulator

After examining two different source codes for AES encryption [18], [19] I decided to work with mbed TLS due to it already having pre-computed search tables and clear documentation on implementation.

2.2 Memory Cache

2.2.1 Memory Cache Overview

Most modern computers have a data and instruction cache, however for this overview and project I will be describing and implementing a unified cache to narrow down the scope of inquiry[4]. A memory cache is often an essential part of the CPU allowing the CPU to avoid sending the data bus to the main memory, slowing down processing times, instead the data bus can just retrieve information from the memory cache. Memory caches are thus designed to be efficient in the retrieval of information [20], [21], but I will be looking at purely how well it can resist in leaking information about these retrievals in evict and time attacks. I will first introduce how a simple memory cache works in theory and then go onto explain how this design can be improved (in a security aspect) by explaining the Newcache design.

One of the simplest introductory concepts of cache memory addressing is direct-mapping, when an address determines which cache line a block should get stored into. The address translation mechanism computes two components: word address A_{word} and line address A_{line}.

Equation 3 [4, p. 41]

$$A_{word} = A \bmod 2^{\varphi}$$

$$A_{line} = [\left\lfloor \frac{A}{2^{\varphi}} \right\rfloor / 2^{b}]$$

Word is figured out by block capacity whilst line is by lines in cache. Therefore, the word and line are modified by different bit sizes (φ or b) as shown in equation 3. Yet in direct mapped caches there is a many-to-one mapping issue, where one block of the cache is appropriate for multiple blocks of memory [4, p. 41]. The CPU needs to know where the memory block is supposed to be in the cache, to accurately determine whether it is present or in the main block of memory (this termed as either a hit if in the cache or a miss if it needs to fetch the block from the main memory) [4, p. 41].

This done through a tag, which identifies each line in the cache with a unique identifier, this allows the CPU to compare the tag with the tag address it has received to determine whether the memory block is in the cache or not. Equation 4 is how a tag is determined for the address in a direct mapped cache.

Equation 4 [4, p. 41]

$$A_{tag} = [\left\lfloor \frac{A}{2^{\varphi}} \right\rfloor / 2^{b}]$$

[4] I will assume the reader to have a basic understanding of CPU design and implementation, if not I encourage the reader to refer to Chapter 2 of Structured Computer Organization by Andrew S.Tanenbaum[2006]

Direct-mapped memory caches are notorious for being inefficient and being a barrier to optimal processing times [4, p. 42], [9]. Thus, designers introduced an address scheme which used sets to divide the cache.

A set groups by $W = 2^b/2^s$, W denotes a specific translation scheme of a way set associative cache, whilst s is the number of sets [4, p. 42]. With the introduction of the set the scheme is changed as follows:

Equation 5 [4, p. 42]

$$A_{word} = A \; mod \; 2^{\varphi}$$

$$A_{set} = \left\lfloor \frac{A}{2^{\varphi}} \right\rfloor mod \; 2^{s}$$

$$A_{tag} = [[\frac{A}{2^{\varphi}}]/2^{s}]]$$

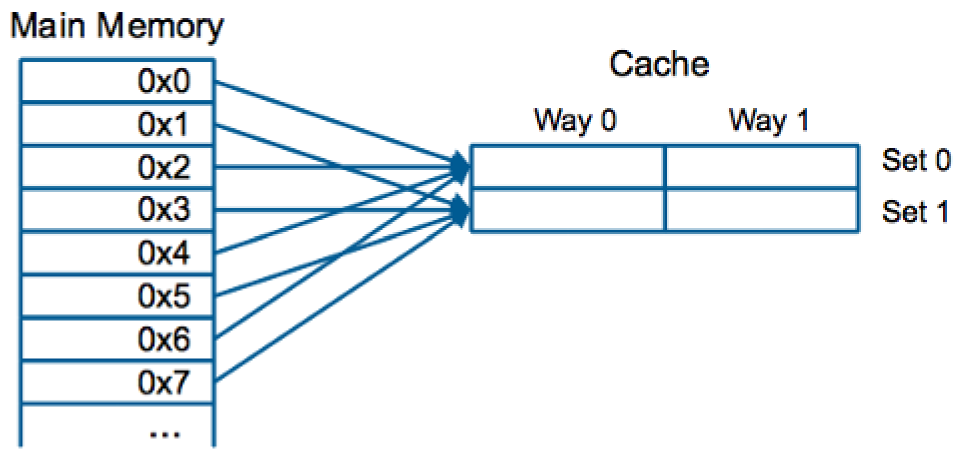


Figure 5 2-Way Set Associative Cache [1, p. 6]

Figure 5 displays how main memory can map onto a 2-way set associative cache. Thus, if memory addresses 0x0, 0x2 and 0x4 are accessed in that order, then when 0x4 is accessed, it will remove 0x0 from the two-way set associative cache as 0x0 was least recently used from set 0. As cache lines holds one aligned, power of two block of adjacent bytes loaded from memory, if any byte needs to be evicted, the entire line is reloaded [2, p. 3]. As set associative cache designs are widely implemented as a design that incorporates the best of direct and fully associative caches [2, p. 4], this is why I have chosen this as the basic design to represent a standard base level design of a modern cache.

2.2.2 Basic Memory Cache Design & Secondary Secure Design

The proposed designs of the two memory caches to test in the simulator were chosen purely on their security credentials rather than power efficiency, to narrow the scope of the project. The basic design has no security additions to it, to highlight the weaknesses of the design in evict and time attacks, as well as provide a basis of which to judge the security of the second design.

As stated previously the basic memory cache design will be very similar to the one discussed in section 2.2.1, however, it will be an 8-way set associative design so as to attempt to replicate the results of Liu and Lee (2013, p.5), when comparing that cache design to Newcache design in an evict and time attack.

The Newcache architecture adopts the direct-mapped architecture, and incorporates a dynamic memory-to-cache remapping and a longer cache index. With a new security-aware cache replacement algorithm [9, p. p.84].

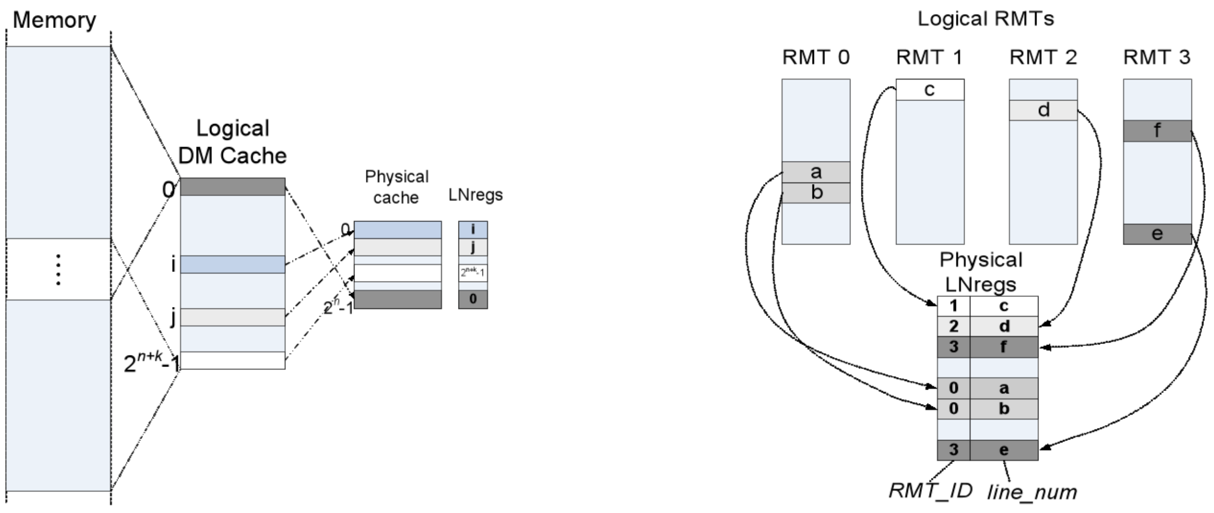


Figure 6 Newcache Design [9, p. 85]

Dynamic remapping mechanism enables the proposed cache to store the most used cache lines at run time, rather than holding a fixed set of cache lines (as in direct mapped caches). Dynamic random mapping is when a set is selected by hashing the main memory address dynamically (at run time)[20, pp. 121–2]. So instead of a traditional 2^n cache lines, as in direct mapped caches, it uses $n + k$ bits to index. The authors liken this to a larger logical direct-mapped [LDM] cache with 2^{n+k} lines that maps to the physical cache, whilst this is not true in implementation the concept is useful in understanding the design of the Newcache [9, pp. 84–85].

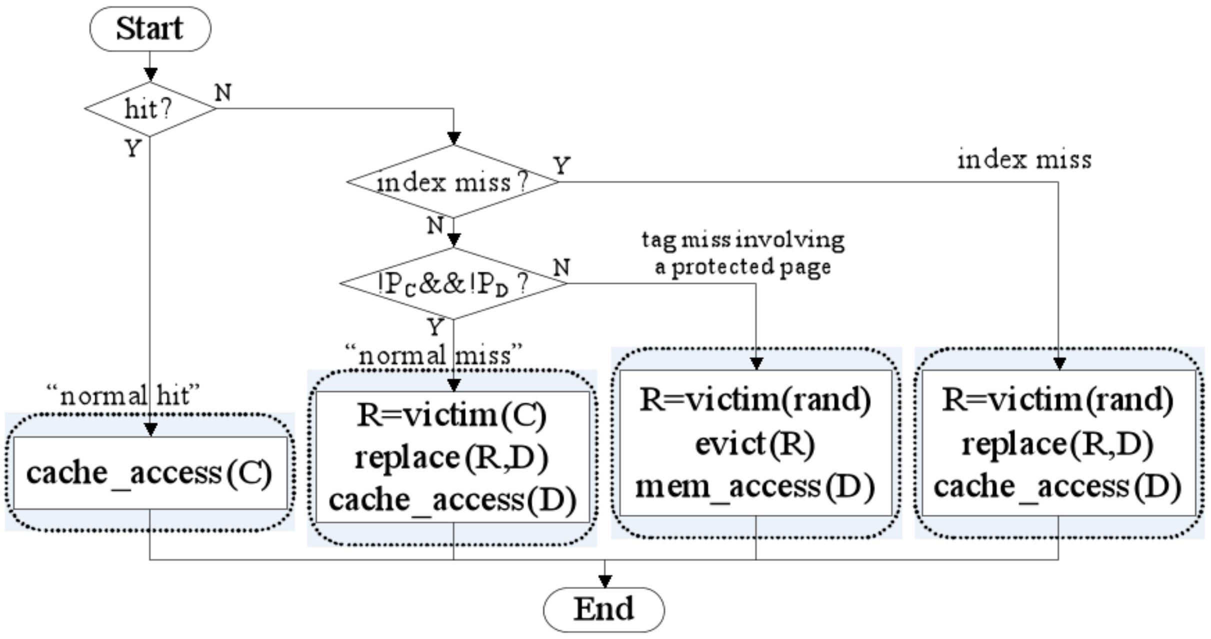
To remember which lines in the LDM cache are stored in the real cache, every physical cache line is associated with a Line Number register (LNreg), which stores the (n+k)-bit line number of the corresponding logical cache line (cache lines in LDM) as seen in Figure 6 [9, p. 85].

The dynamic remapping requires multiple RMTs (Remapping tables) due to the complexity of the algorithm, but from Figure 6 you can see the physical cache only matches to one set of LNregs. Only the RMT associated to the logical cache line needs to be stored in the LNreg, the rest are disregarded. Furthermore, there is a RMT_ID field in each LNreg in addition to the line number field to allocate RMT values [9, p. 85].

Each process is attached to a context RMT ID which specifies the RMT it will use. Different processes can have different memory-to-cache mappings if attached to different context RMT IDs, which introduces the dynamic aspect of the remapping mechanism [9, pp. 85–86].

Each cache line also has a P flag bit, indicating protected cache lines. Adding in this protected line means the memory block in that cache cannot be evicted, and the attacker cannot observe the access pattern of this cache [3, p. 29].

The Newcache also implements a security conscious random cache replacement algorithm [9, pp. 85–87]. A random mapping function between memory and cache has been proposed by many researchers as a way to challenge side channel attacks [1, p. 20].



Start
hit?
N
Y
index miss ?
Y
index miss
N
tag miss involving a protected page
!P_C&&!P_D ?
N
Y
"normal miss"
"normal hit"
cache_access(C)
R=victim(C) replace(R,D) cache_access(D)
R=victim(rand) evict(R) mem_access(D)
R=victim(rand) replace(R,D) cache_access(D)
End

Figure 7 Newcache Security Algorithm [9, p. 87]

From what has been discussed we can see there can be two types of misses in the Newcache [9, p. 86]:

- Index miss occurs if none of the LNregs matches the given RMT_ID and index. None of the cache lines is selected in an index miss.
- Tag miss occurs if the index hits in one LNreg, but the tag of the selected cache lines does not match the address tag.

Figure 7 charts the Newcache security algorithm, when a cache miss occurs, if the LNreg of a cache line C matches the Context RMT_ID and index of memory block D, then this is a tag miss. If either C or D are protected, the algorithm does not replace C with D, and D is sent back to the CPU main memory block. If it is not protected then it simply replaces the R (the cache line selected for replacement). An index miss means D can replace any cache line, thus a cache line is randomly selected and evicted.

To summarise the key design specifications of the Newcache are [22, p. 3]:

- There is fixed mapping from the memory to the ephemeral cache and a full-associative mapping from the LDM to the physical cache.

- The same set of LNregs can be shared by multiple remapping tables (identified by RMT_ID) so the attacker observes a different memory-cache mapping than the victim.

- Internal cache interferences may still exist through tag misses so to protect security critical information, the Newcache associates a protection bit in the tag array for each cache line.

During development, I will have to explore whether I look to implement and adapt only key specifications of the Newcache in secondary design (LDM or protecting bits in the tag array) due to the time frame, but this will be discussed further in the risk and analysis part of the review. Furthermore, whilst Wang and Lee (2008) do not utilise a content-address memory array due to power concerns, the ability to implement this design in the time allocated without the use of switches makes it more preferable for this project.

2.3 Side Channel Attacks

2.3.1 Evict and Time Attack

For a side channel attack to be successful it needs to fulfil three requirements [4, p. 4]:

Perturbation: Occurs when the secret that the attacker wants to reveal alters the behaviour of the system or state.

Manifestation: Most often the perturbation in the system cannot be directly accessed by the attacker. The information of the perturbation is manifested via some channel.

Observation: An attacker would have to observe the side channel to obtain the required information.

Here is a layout of a time-based side channel attack. There is some look-up table T and a function called AccessT, which takes two parameters X^1 and X^2 [4, pp. 53–54]. The function accesses the table T twice - at locations $K^1 \oplus X^1$ and $K^2 \oplus X^2$, where K^1 and K^2 are secret data (we can think of them as data relating to the key).

Suppose an attacker can invoke AccessT almost as a user. Since no direct channels are present, the information can be sourced from an indirect channel. In this case, we are looking at the time taken to execute instructions.

If there was no table in the memory cache the first instruction will result in a miss and load the table into the cache. A cache hit will arise if the data searched by the attacker is in the cache, and the access of $K^1 \oplus X^1$ and $K^2 \oplus X^2$ are in the same memory block (also called a collision [4, p. 54]. The cache hit implies $K^1 \oplus X^1$ and $K^2 \oplus X^2$ (if a cache line held 32 bytes) that these two locations differ by at most 32, in other words by 5 significant bits as $\log^2 32 = 5$ [4, p. 54].

If X^1 and X^2 represented addresses 0x45 and 0xc4 respective then we do the following calculation:

Equation 6 [4, p. 54]

$$K^1 \oplus K^2 = (X^1 \oplus X^2) \gg \log\left(\frac{32}{1}\right) = (0x45) \oplus 0xc4) \gg 5 = 4$$

Another way to illustrate this point is to see the binary notation of $K^1 \oplus K^2$ is 10000001 and right shift this by 5 we get 101, which is 4 in decimal. This limits the candidates of K^1 and K^2 to 0x80 to 0x9f as any higher will raise the MSB to 5 and any lower the MSB to 3-0 with equation 6 [4, p. 55]. Thus, from this simple calculation we can reduce the complexity of the key. A miss can also expose information about the key, if $X^1 \oplus K^1 \neq X^2 \oplus K^2$ it means they are not in the same cache line, thus eliminating 32 values for K or that $X^1 \oplus K^1$ has been evicted from the cache at run time [4, p. 55].

To determine whether it was a hit or miss, we need to calculate the time taken to execute instructions. As a miss will incur a time penalty as it must fetch a memory block from the main memory. This example shows how through perturbation, manifestation and observation a key's complexity can be broken down, and in the process, reveal parts of the key.

Measurement of time based side channel attacks of microarchitectural events, such as cache hit and misses, require precise clocks and a time stamp counter. The attacker must first get a base time level to determine whether it was a hit or miss during execution, setting this baseline or threshold is achieved by comparing time stamps over multiple executions of instructions. Usually a mixture of hardware factors can interfere with the clock leading to longer execution times during processing, but for this project I will just be running the simulator in an idealised setting.

For a system with a single level of cache memory, the baseline is calculated by:

Equation 7 [4, p. 43]

$$Average\ Memory\ Access\ Time = Hit\ Time + Miss\ Rate \times Miss\ Penalty$$

Abstractly we can see the timing channels as a pair of clocks, if the relative rate of these clocks varies significantly in some manner then it can expose a supposedly protected algorithm [2, p. 6].

For this project, I will be focusing solely on evict and time attacks, which follows this format [10, p. 9]:

1. Trigger an encryption of some plaintext (p)
2. Evict a memory address or cache line and replace with attacker's data
3. Trigger another encryption of p and time it

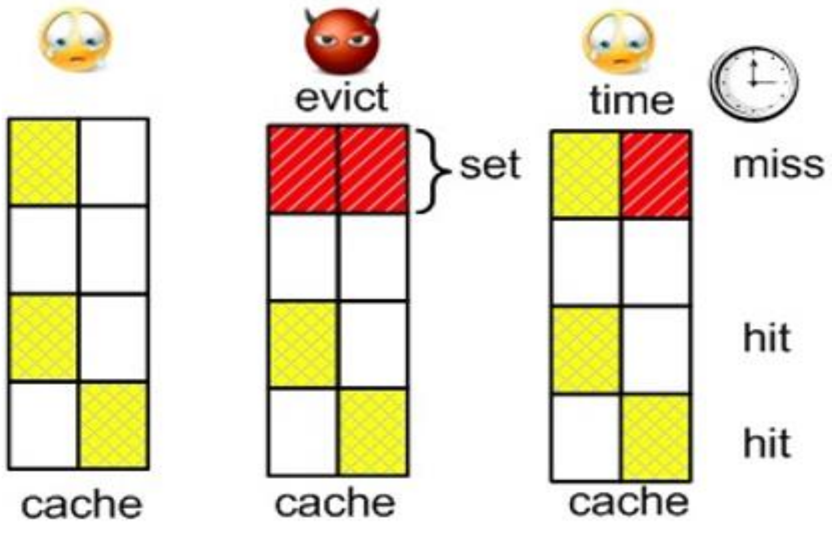


Figure 8 Evict and Time Attack [22, p. 2]

The aim of evict and time attacks are to learn whether a specific memory line is accessed during encryption. Imagine we had a table T that holds the key algorithm. If the attacker has an array as large as the cache and load their memory blocks to the cache, whilst evicting specific cache lines, they can determine which memory blocks were accessed. If the data of the attacker is evicted it will take longer as it will induce a miss [4, pp. 66–68], [22, p. 2]. Let us now examine how this relates to AES encryption.

2.3.2 AES Look Up Table and Estimated Cache Resistance

In simulated attacks on AES [5], [10], [22] the encryption tables, which will be loaded into the cache, will be a eight 256 byte look-up tables $T^{0^0} - T^{4^0}$ and $T^{0^{10}} - T^{3^{10}}$. $T^{0^0} - T^{3^0}$ are used in the first nine rounds, to implement the `SubBytes`, `ShiftRows` and `MixColumns` operations, $T^{0^{10}} - T^{3^{10}}$ is exclusively used in the tenth round and just implements the `SubBytes` and `ShiftRows` operations (mimicking the round implementation of AES) [10, p. 4]. In `KeyAddition` the user inputs a 16-byte key, which is then expanded into 10 round keys. For a secret key $k = K_0 - K_{15}$ with the round key $K^r = K_1^r - K_{10}^r$, the 0th round is the original K^0 [10, p. 4]. The initial state x^0 is computed by $x_i^0 = p_i \oplus k_i$ $(i = 0 \dots 15)$ and the subsequent rounds are computed as laid out in Figure 9, in the last round $T^{0^0} - T^{4^0}$ is replaced with $T^{0^{10}} - T^{3^{10}}$. x^r in figure 9 are the intermediate states of 16 bytes for each round.

$$
\begin{aligned}
(x_0^{(r+1)}, x_1^{(r+1)}, x_2^{(r+1)}, x_3^{(r+1)}) &\leftarrow T_0[x_0^{(r)}] \oplus T_1[x_5^{(r)}] \oplus T_2[x_{10}^{(r)}] \oplus T_3[x_{15}^{(r)}] \oplus K_0^{(r+1)} \\
(x_4^{(r+1)}, x_5^{(r+1)}, x_6^{(r+1)}, x_7^{(r+1)}) &\leftarrow T_0[x_4^{(r)}] \oplus T_1[x_9^{(r)}] \oplus T_2[x_{14}^{(r)}] \oplus T_3[x_3^{(r)}] \oplus K_1^{(r+1)} \\
(x_8^{(r+1)}, x_9^{(r+1)}, x_{10}^{(r+1)}, x_{11}^{(r+1)}) &\leftarrow T_0[x_8^{(r)}] \oplus T_1[x_{13}^{(r)}] \oplus T_2[x_2^{(r)}] \oplus T_3[x_7^{(r)}] \oplus K_2^{(r+1)} \\
(x_{12}^{(r+1)}, x_{13}^{(r+1)}, x_{14}^{(r+1)}, x_{15}^{(r+1)}) &\leftarrow T_0[x_{12}^{(r)}] \oplus T_1[x_1^{(r)}] \oplus T_2[x_6^{(r)}] \oplus T_3[x_{11}^{(r)}] \oplus K_3^{(r+1)}
\end{aligned}
$$

Figure 9 AES Look-up Table [10, p. 4]

Liu and Lee [22] performed a first-round evict and time attack against AES on an 8-way set associative cache and the Newcache, the first round was chosen as the attacker can exploit the fact that in the first round of AES encryption with the AES look-up table is achieved by $x_i^0 = p_i \oplus k_i$. This means that if the plaintext byte and the accessed table index are known to the attacker, they can retrieve the whole key byte.

Figure 10 illustrates the encryption times of a plaintext byte under evict and time attack, (a) = 8-way set associative cache and (b) = Newcache. It is clear from the results of Liu and Lee's [22] evict and time attack, that the Newcache did not expose any significant part of the key, whilst the other cache shows when the plaintext byte takes values 128-135 due to significantly higher encryption times.

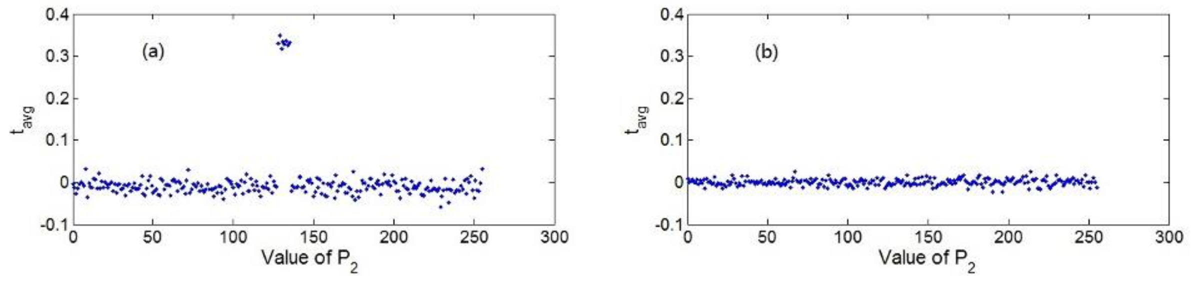


Figure 10 8-Way Set Associative vs Newcache [22, p. 5]

Figure 11 also supports the evidence that the Newcache should be significantly stronger in resisting evict and time attacks due to its low miss rates.

	4KB	8KB	16KB	32KB	64KB
DM	0.133	0.093	0.068	0.055	0.048
SA-2way, LRU	0.101	0.075	0.057	0.045	0.041
SA-4way, LRU	0.096	0.068	**0.053 (1)**	**0.042 (1)**	**0.040 (1)**
SA-8way, RAND	0.095	0.071	0.054	0.044	0.041
FA, RAND	**0.090 (1)**	**0.067 (1)**	**0.053 (1)**	0.044	**0.040 (1)**
Newcache k=4, SecRAND	0.093 (1.033)	0.068 (1.015)	0.054 (1.019)	0.044 (1.048)	0.041 (1.024)
Newcache k=6, SecRAND	**0.090 (1)**	**0.067 (1)**	**0.053 (1)**	0.044 (1.048)	**0.040 (1)**

Figure 11 Cache Miss Rates [5, p. 90]

Due to the strength of the attack I will be focusing on implementing a first round evict and time attack to target AES lookup tables. I also expect from the results from the simulator that the Newcache will outperform basic design cache significantly in resisting exposing key bytes.

2.4 Simulator

2.4.1 Dinero Simulator

An inspiration for this project is the Dinero IV simulator [23], a simulator which reads from a specifically formatted text file (similar to Pixie notation[24, pp. 69–70]) to assess the hit and misses of a cache[5].

It is not a timing simulator but runs through the memory reference traces to assess the hit and miss ratio of the cache. It is not a functional simulator and thus data or instructions are not actually existent in the simulator. Rather than trying to simulate multi-threading, Dinero IV simulates a memory hierarchy of various caches connected to one or more trees, with reference sources at the leaves and a memory at each root [23].

For my proposed simulator, it will differentiate to Dinero IV in its inclusion of a timing element, but the simulator utilises a key equation to average memory access in the output, which I will look to adapt as well as its output format:

Equation 8 – Average Memory Access (AMAT) [23]

$$AMAT = HT + MR \cdot MP$$

Where HT is hit time, MR is miss rate and MP is the access time taken if it is a miss, this equate AMAT to the cache access time, thus when a miss occurs AMAT is increased by the access time.

2.4.2 Design Considerations for Simulator

Researchers have looked into designing a flexible and adaptable model to provide specific design specifications for models testing for security concerns [7], [25], [26]. Whilst these papers do not directly relate to evict and time models, they provide general design principles, which will guide me throughout the building process.

The instruction set simulator (ISS) of Breier (2016) takes assembly code as the input and checks all the possible fault models that may occur in the device. Whilst there will be significant differences concerning inputs and the nature of the attack, there are design specifics that are applicable to my proposed simulator.

[5] Due to page limits I cannot go through the notation of Dinero IV, please visit ftp://ftp.cs.wisc.edu/markhill/DineroIV[28] for instructions on how to download and run a test program.

Brier laid out a series of requirements for his ISS [25, p. 475], the two which are applicable to my simulator are:

- Simulate execution of assembly code for microcontrollers and the ability to execute different instruction sets and register sizes.
- Provide an easy-to-understand output that can be used for improving the code.

These design principles mean that I will endeavour to make the simulator adaptable in the sense it can be used to test multiple cache designs in software, and the output should clearly highlight the degree of resilience of the design to evict and time attacks. As well as these general requirements, my simulator should be able to simulate a microarchitectural clock during the evict and time attacks.

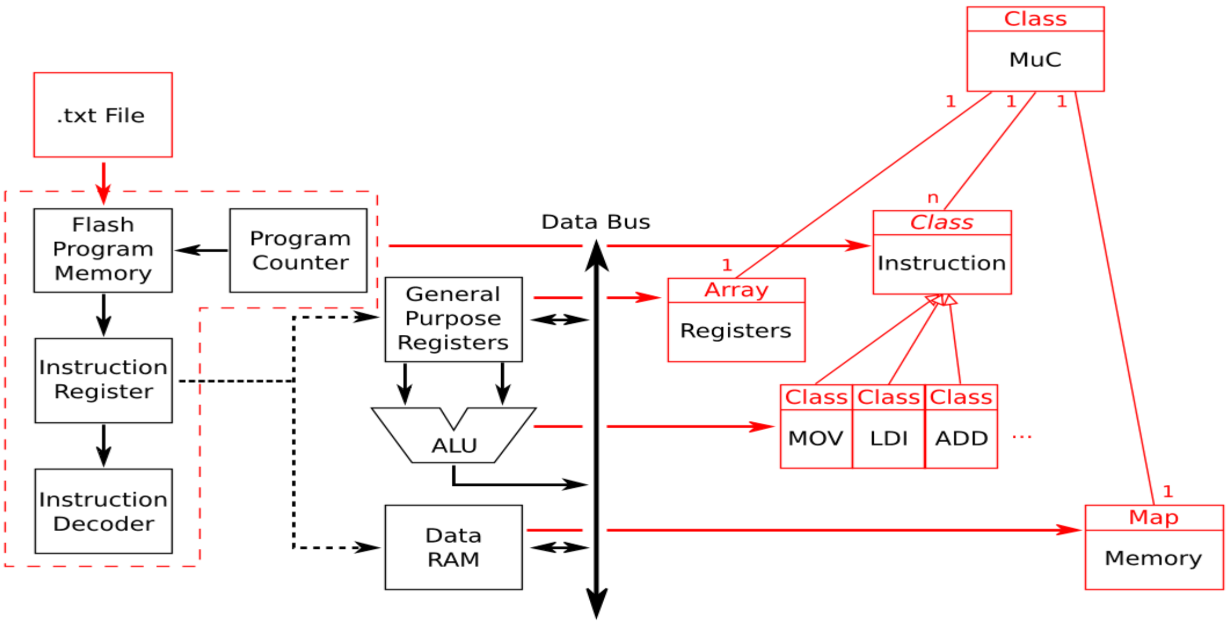


Figure 12 Microcontroller architecture mapped to an object-oriented computer program[25, p. 476].

Figure 12 is useful in visually representing how hardware can be mapped to object-oriented programming classes. These papers discuss model specifications for a CPU or microcontroller architecture, in my simulator design I will just be having it so there is the main memory, memory cache with input and output text files and address instruction method of the attacker. Yet I can use Figure 12 as a top-level design representation to map out my simulator. As I will be using Figure 12Figure 12 as a model to layout my simulator, I will also endeavour to write the software in object-oriented programming in Java.

The output of the simulator or 'comparator', will give the user two different layouts of the data [25, p. 477]:

Overview - The total number of faults and this number is then broken down into key statistics such as total number of misses and hits.

Detailed view - Provides insight on all the successful faults, such as whether it was a hit or miss per iteration of the loop.

The overview should be understandable from a cache designer point of view, and thus laid out in a manner, which whilst requires a knowledge of side channel attacks, not necessarily an expertise in the area [7, p. 30].

3 Project Outline

3.1 Estimated Timeline

The estimated timeline for this project is explained in detail here and an overview provided by a Gnatt-chat. Work on this project will commence 29th May, with an intended finishing date on the 26th August. Each date given is the start date with the preceding milestone date signalling that the original milestone has been completed.

Milestone 1 (29th May): Implement a working AES encryption model using mbed TLS AES.

Task	Deliverable
1.1 To build an AES encryption script using an AES look-up table that intakes plaintext and outputs cipher text.	A working AES model that takes plaintext files and outputs cipher text.

Milestone 2 (1st June): A basic cache design that stores an AES look-up table

Task	Deliverable
2.1 To build a L1 (level 1) 8-way set associative cache design with capacity enough to store AES look-up tables	A software implementation of an 8-way set associative cache.
2.2 To have it so the memory cache is used when the encryption process takes place. The AES look-up tables will be stored in the cache.	Integration between cache and encryption process to unify the model.

Milestone 3 (15th June): Build an attacker that can evict cache lines and a main memory block

Task	Deliverable
3.1 Build a main memory block that is connected to the cache, so if the cache line requested is not in the cache it can retrieve it from the main memory.	A main memory class associated with the cache.
3.2 Layout a simple processor syntax that enables the user to load and evict data into the cache during the first round.	The ability to load and evict cache lines forcing it to retrieve memory lines from the main memory.
3.3 Implement an attacker in the simulation that can evict cache lines forcing the cache to retrieve information from the main memory.	A simulation that for i^{th} round will evict and load data into the cache.

Milestone 4 (3rd July): Integrate a clock with time stamp into the simulator to record processing times and output results

Task	**Deliverable**
4.1 To integrate a timing element within the simulator that times processing times, and can measure cache hit and misses	A clock in the simulator with an accurate time stamp
4.2 Calculate hit and misses and look at breaking down this data to highlight how much key complexity was narrowed down by	An output file with key information about hit and miss ratio, and estimates of the breakdown of key complexity

Milestone 5 (1stth August): Build a second memory cache and test it with the simulator

Task	**Deliverable**
5.1 To build a Newcache style memory cache that has the capacity to hold the AES look-up tables and connect to main memory	A security conscious secondary cache design that can be used during encryption of plaintext files
5.2 To test the second cache design in the simulator	Successful output of results from testing the seconded design in the simulator

Milestone 6 (20th August): Compare/contrast initial data with final evaluation

Task	**Deliverable**
6.1 To gather all results so far from the simulator after running the simulator for a series of times, to get an accurate predication of average hit and misses per cache design	A preliminary set of results which assess the security of each design in evict and time attacks
6.2 To look over the output and the simulator in general to critically assess its capabilities and future work	A report on the current capabilities with suggestions of future work of the simulator

Milestone 7 (26th August): Written report

Task	Deliverable
7.1 Design and finish poster	a poster summing up main deliverables and evaluation
7.2 Write the thesis	An MSC Computer Science Conversion thesis

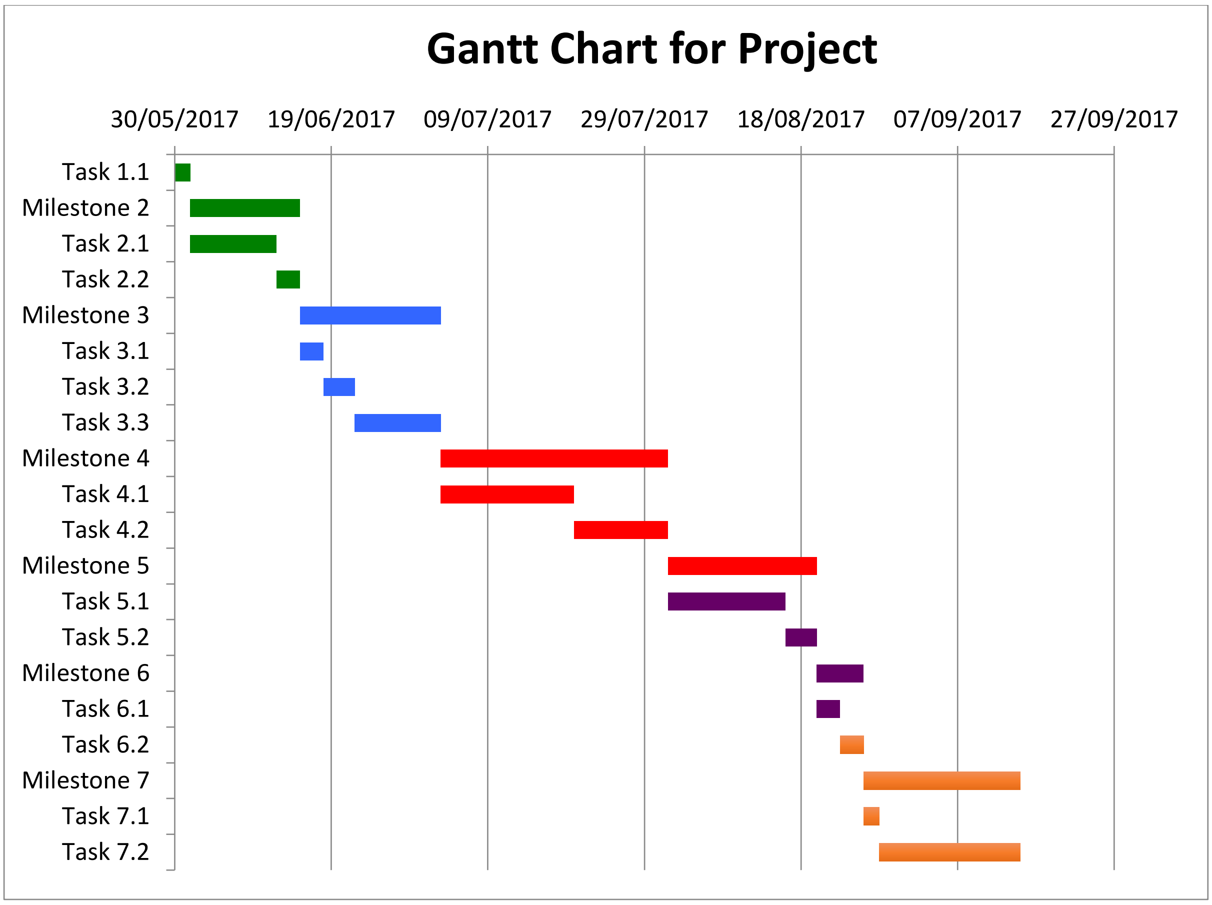


Figure 13 Gnatt Chart - Project Outline

3.2 Risk Analysis

The risks proposed here are ranked on a likelihood 1 to 5 with 1 being the lowest and 5 the highest probability. The severity of the risk is also marked in a comparable manner.

Risk	Likelihood	Severity	Prevention	Contingency
Failure for cache to integrate to produce accurate hit and misses	2	5	To ensure that timeline is kept to so any software issues may be raised with supervisor	To at least have a replication of hit and misses produced by the simulator, or implement a trace syntax (similar to Dinero IV)
Newcache design is too complicated to implement fully	4	2	To plan carefully software implementation of Newcache and keeping to time allocation	To implement part of the Newcache security algorithm such as protected cache lines
Data is inconclusive or not adequate for analysis	1	3	To ensure that throughout the process data is quality reviewed	To evaluate why data is not adequate for analysis, and highlight areas for improvement

Delay due to underestimation of time frame or unexpected circumstances	2	3	To regularly keep in contact with supervisor and bring up any issues early on	To re-assess time frame and look at limiting deliverables
Slow performance of simulator leading to timeline disruptions – inability to execute multiple encryptions efficiently	3	2	To look in parallel computer resources such as using multiple processors or at least split operations so it can be done by more than one computer.	To re-assess time frame and look the initial data to draw conclusions.
Technical faults and data loss	1	4	To regularly back up data and to have it so I can access code/data on other machines other than laptop. Furthermore, if technical issues arise immediately notify supervisor or appropriate member of the university.	To modified project deliverables to fit in a narrow scope of time.

Figure 14 Risk Table

4 Conclusion

The research I have done for this project has laid out the theoretical framework to start with the practical application of the project. Evict and time attacks are a sub-category of the wider range of side channel attacks, but by testing cache designs in the software design stage we can gain vital insight into the security of the cache and adapt the cache design accordingly.

The outcome of my simulator should clearly demonstrate that the Newcache has strong resistance to the evict and time attack whilst the 8-way set associative design should during the attack at least reveal the MSBs of the key.

The analysis part of my program will be stored in a separate array of the program (attacker) which stores the hit or misses from the simulator, and then can be used to produce the output file with the results. This output file as discussed previously will split into a general overview with summarised calculations from the data, whilst the detailed section will be a complete breakdown of the attack.

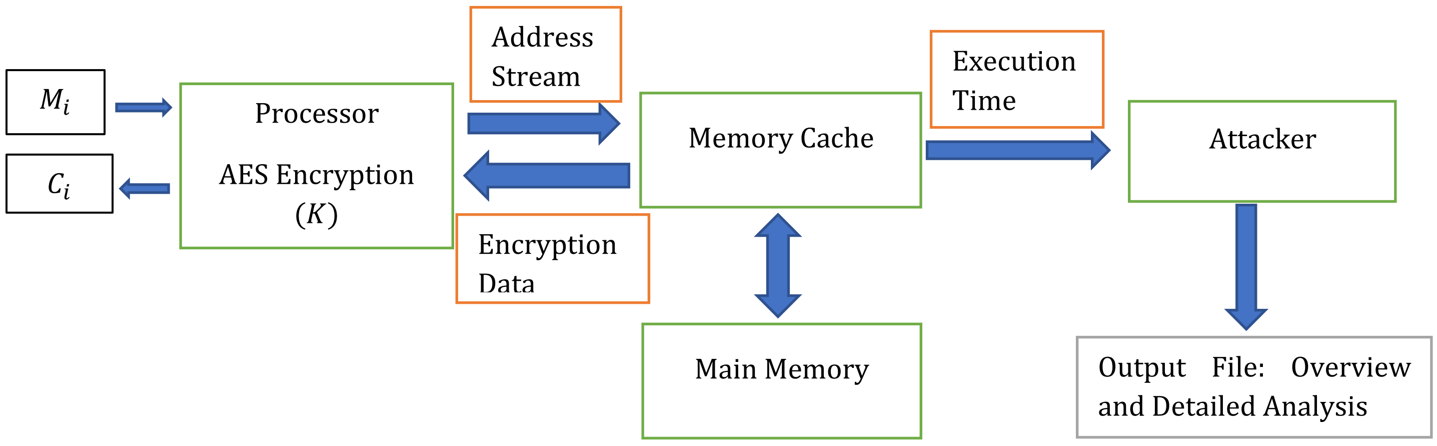


Figure 15 Simulator Architecture

As you can see from Figure 15 the input is a series of plaintext files M_i which is encrypted by K and outputting cipher text $C_i = Enc(K, M_i)$. The processor encrypts M_i by retrieving encryption data from the memory cache. The memory cache either has the data or must retrieve it from the main memory. For this project, I will be focusing on implementing just a L1 cache design for both the basic and secondary cache due to time constraints. Whilst these plaintext files are fed into the simulator the attacker gathers the execution time from the first round storing the data till an average execution time is established, and enough data has been collected to analysis the effectiveness of the evict and time attack.

As in Figure 12, this can also be thought of mapping onto hardware via object-oriented classes. Each green box can be interpreted as a Java class, this will allow incremental development during the process and errors to be identified more easily.

Equation 9 – Execution Time

$$Execution\,Time \sim= F + Ch \times H + Cm \times M$$

F = offset; Ch = cost of a cache-hit; H = number of cache-hits; Cm = cost of a cache-miss; M = number of cache-misses

Equation 9 is the basic model execution time I will utilise to test that the simulator is able to calculate execution time to a degree of accuracy enough to evaluate the cache design. Whilst the equation does ignore quite a few factors, it will be adequate to provide a model of which to formulate the output of the simulator; as it provides the attacker with the most information possible from the attack. Then I will compare and contrast these with the results of similar attacks as laid out in previous papers [1], [5], [9], [22], evaluating whether the simulator is able to produce similar results than these hardware/software simulations of side channel attacks.

As stated in the introduction the aims of the project will be to provide a flexible simulator to test memory cache designs in the software stage. I will endeavour to have two cache designs be tested by the simulator, providing a means to both test the simulator accuracy and to offer a critique of each cache design. From this I will write a thesis on the development process, conclusions from the work done so far and suggestions for future work of the simulator or security cache design research.

5 Bibliography

[1] J. Szefer, “Survey of Microarchitectural Side and Covert Channels , Attacks , and Defenses,” *Cryptol. ePrint Arch. Rep. 2016/47*, pp. 1–27, 2016.

[2] Q. Ge, Y. Yarom, D. Cock, and G. Heiser, “A Survey of Microarchitectural Timing Attacks and Countermeasures on Contemporary Hardware,” *Cryptol. ePrint Arch. Rep. 2016/613*, pp. 1–37, 2016.

[3] J. Kong, O. Acıicmez, J. Seifert, and H. Zhou, “Deconstructing New Cache Designs for Thwarting Software Cache-based Side Channel Attacks,” *2nd ACM Work. Comput. Secur. Archit.*, pp. 25–33, 2008.

[4] C. Rebeiro, M. Debdeep, and B. Sarani, *Timing Channels in Cryptography*. New York: Springer International Publishing, 2015.

[5] D. J. Bernstein, “Cache-timing attacks on AES,” 2005. [Online]. Available: http://cr.yp.to/papers.html#cachetiming.

[6] NIST, “Announcing the ADVANCED ENCRYPTION STANDARD (AES),” 2001.

[7] D. Shumow and P. Montgomery, “Side Channel Leakage Profiling in Software,” *Cosade 2010*, pp. 29–35, 2010.

[8] Z. Wang and R. B. Lee, “New Cache Designs for Thwarting Software Cache-based Side Channel Attacks,” *ISCA ’07 Proc. 34th Annu. Int. Symp. Comput. Archit.*, pp. 494–505, 2007.

[9] Z. Wang and R. B. Lee, “A Novel Cache Architecture with Enhanced Performance and Security,” *2008 41st IEEE/ACM Int. Symp. Microarchitecture*, pp. 83–93, 2008.

[10] D. A. Osvik, A. Shamir, and E. Tromer, “Cache attacks and counter-measures: The case of AES,” *Lect. Notes Comput. Sci. (including Subser. Lect. Notes Artif. Intell. Lect. Notes Bioinformatics)*, vol. 3960 LNCS, pp. 1–20, 2006.

[11] R. Avanzi, “A Salad of Block Ciphers,” *Cryptology ePrint Archive, Report 2016/1171*, 2016. [Online]. Available: http://eprint.iacr.org/2016/1171.

[12] “AES Layout,” 2017. [Online]. Available: https://iis-people.ee.ethz.ch/~kgf/acacia/fig/aes.png.

[13] J. Daemen and V. Rijmen, *The Rijndael Block Cipher: AES Proposal*. 2003.

[14] M. Saeed and M. Saleem Mian, “Methods of finding multiplicative inverses in GF(28),” *Comput. Commun.*, vol. 31, no. 17, pp. 4117–4123, 2008.

[15] “SubByte Stage,” 2017. [Online]. Available: http://www.everystockphoto.com/photo.php?imageId=1658292.

[16] M. J. . Knudsen, Lars R., Robshaw, *The Block Cipher Companion*. New York: Springer International Publishing, 2011.

[17] C. J. Benvenuto, “Galois Field in Cryptography,” 2012. [Online]. Available: https://sites.math.washington.edu/~morrow/336_12/papers/juan.pdf.

[18] “mbed TLS,” 2017. [Online]. Available: https://tls.mbed.org/aes-source-code.

[19] "BrianGladman," 2017. [Online]. Available: https://github.com/BrianGladman/aes.

[20] A. J. A. Y. Smith, "Smith - 1978 - A Comparative Study of Set Associative Memory Mapping Algorithms and Their Use for Cache and Main Memory," *IEEE Trans. Softw. Eng. VOL. SE-4*, no. 2, pp. 121–30, 1978.

[21] M. Zhang and K. Asanovic, "Highly-Associative Caches for Low-Power Processors," *Kool Chips Work. 33rd Int. Symp. Microarchitecture, MICRO 2000*, no. December, pp. 1–6, 2000.

[22] F. Liu and R. B. Lee, "Security testing of a secure cache design," *Proc. 2nd Int. Work. Hardw. Archit. Support Secur. Priv. - HASP '13*, pp. 1–8, 2013.

[23] J. Edler and M. D.Hill, "Dinero IV," 1998. [Online]. Available: http://pages.cs.wisc.edu/~markhill/DineroIV/. [Accessed: 04-Feb-2017].

[24] G. C. E. Conte, Thomas M., *Fast Simulation of Computer Architectures*, 1st Editio. New York: Springer US, 1995.

[25] J. Breier, "On Analyzing Program Behavior Under Fault Injection Attacks," *2016 11th Int. Conf. Availability, Reliab. Secur.*, pp. 474–479, 2016.

[26] D. Mccann, C. Whitnall, and E. Oswald, "ELMO : Emulating Leaks for the ARM Cortex-M0 without Access to a Side Channel Lab," *Cryptol. ePrint Arch. Rep. 2016/517*, pp. 1–21, 2016.

[27] R. Lidl, R. Lidl, H. Niederreiter, and H. Niederreiter, *Introduction to finite fields and their applications*. 1986.

[28] M. D.Hill, "Index of /markhill/DineroIV," 1999. [Online]. Available: ftp://ftp.cs.wisc.edu/markhill/DineroIV. [Accessed: 22-Apr-2017].