Speculative Parallelization For Security Encryption

Prashant Solanki
NC State University
psolank@ncsu.edu

ABSTRACT

Encryption is present in many modern applications. Its importance is undoubtedly higher for security in distributed computing systems and its many communication networks. Cypher Block Chaining mode is one the most widely used block chaining modes in encryption due to its simple and elegant design. But the design is essentially sequential as every stage carries a dependence on the previous stage so. it can utilize only 1 core in a machine. Speculating on the value(s) carried by dependencies is one way to break such critical dependencies. We look at the feasibility of principled speculation for parallelizing the CBC mode.

Keywords

CBC, AES, Principled Speculation, LaTeX, text tagging

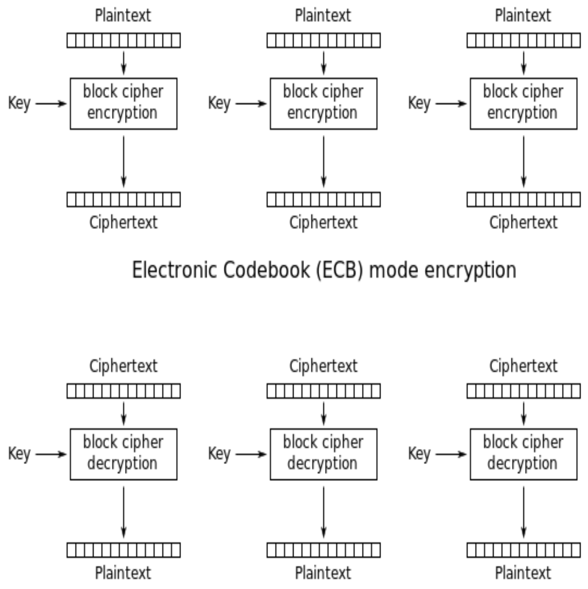
1. INTRODUCTION

The growing popularity of Graphic Processing Units for general purpose computing has increased the pressure on developers to parallelize traditionally sequential applications. Here traditionally sequential means that applications which do not inherently have a parallel design. There are two main bottlenecks faced by such applications, data dependence and control dependence which prevent parallelism. Data dependence is the case where the same memory location would be read and written (RAW/WAR) or written (WAW) by two separate threads creating a data race condition. Control dependence is when the result of a previous iteration is needed to take a control decision for the current iteration. Where data dependence is an easier problem to solve with well-established techniques such as privatization, control dependence is relatively challenging. Speculation is one of the major research topics for solving control dependence. The CBC mode is an example of control dependence preventing parallelization.[3][2]
In this project we look at techniques for speculating the control dependence in CBC mode.

2. BACKGROUND

This section provides a brief primer on CBC encryption and introduces terminology used in the rest of the paper. It then describes speculative parallelization as a general technique.

Figure 1: ECB Mode



2.1 CBC AES mode

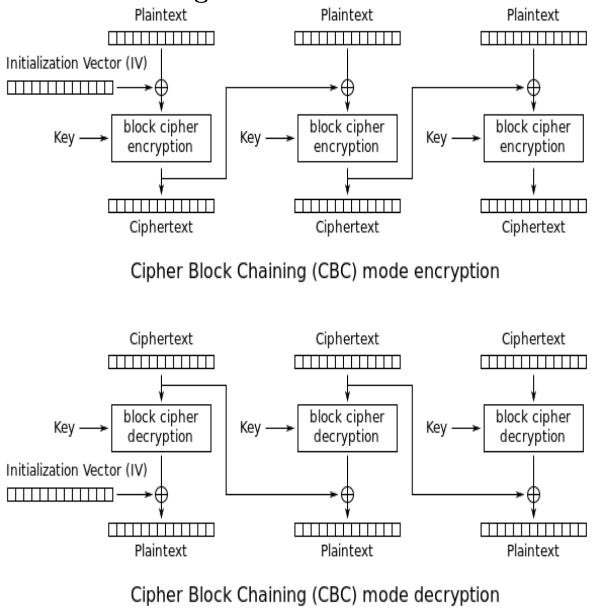
In cryptography, a mode of operation is an algorithm that uses a block cipher to provide an information service such as confidentiality or authenticity. A block cipher by itself is only suitable for the secure cryptographic transformation (encryption or decryption) of one fixed-length group of bits called a block. A mode of operation describes how to repeatedly apply a cipher's single-block operation to securely transform amounts of data larger than a block.
One of the most important requirements of modes is efficient hiding of repeated data patterns. Most modes require a unique binary sequence called an initialization vector (IV), for each encryption or decryption operation. The initialization vector is used to ensure distinct cipher texts are produced even when the same plaintext is encrypted multiple times independently with the same key. The block cyphers can be configured to use many different block sizes. The larger the block size the better the security. Block cipher modes operate on whole blocks and require that the last

part of the data be padded to a full block if it is smaller than the current block size.

The CBC mode evolved from the ECB mode which did not satisfy the repeated pattern requirement. The ECB mode can be described through the figure 1

The Key and plain text are input to the block cypher and cypher text is obtained. Decryption is the reverse operation. The CBC mode augments the ECB mode by chaining the blocks to produce hide repeated patterns. The algorithm overview can be seen in figure 2. The first plain text block is XORed with an initialization vector IV and given as input to the block cypher. The output of the first block is now used as IV for the second block. For decryption the first cypher text is decrypted by the block cypher and then XORed with the IV. But here the first cypher block is used as IV for the second block. It is now clear that encryption carries a dependence on the previous block but the decryption can be easily parallelized.

Figure 2: CBC Mode



2.2 Speculative Parallelization

In this project I will focus on the speculative parallelization technique as described in Principled Speculation paper by Zhijia Zhao, Bo Wu and Xipeng Shen[4][5]. The paper focuses on parallelizing general FSMs. I will try to model the CBC mode as a FSM in an attempt to solve the problem.
The technique describes a make-span formula for computing the benefit of speculation during runtime. It take as arguments properties and characteristics of the FSM as described below:

- **Size:** number of states in the FSM
- **State Feasibility:** The probability that a state will be reached during execution.
- **Character Probability:** The probability for a state to be the next state given the current state.
- **Expected Convergent Length:** The length of input data which needs to be processed to converge to a common state irrespective of the starting state. Take an example of two FSM instances in states S1 and S2. Then the convergent length will be defined by the length after which both FSMs rest at the same state S3. It is a function of the initial states as well as the structure of the FSM.

2.3 Objectives

In this project I have tried to model the CBC algorithm as a FSM and utilize the techniques of the FSM paper with the following steps:

- Implement CBC mode for 2-bit and 8-bit mode.
- Model the algorithm as a FSM
- Extract properties of the FSM and establish a probabilistic model for input vs. output.
- Compute average convergent length for the FSM.
- Speculate and try to parallelize the CBC algorithm

3. CHALLENGES

The standard implementation of AES CBC mode is for 128 bits. A 2bit and 8 bit mode needed to be implemented.
Calculating the FSM such as convergent length and character probability.

4. MODELING THE CBC MODE AS A FSM

In the project I have considered each possible input block and output block as a state for the FSM. This gives 2^n states. Since this is too large a problem to solve at standard minimum block lengths of 128 bits for AES cypher, I have analyzed the problem at two levels 2-bit and 8-bit giving 4 and 256 states respectively.
Since AES is not implemented for less than 128 bits in the industry standard libCrypto library I have implemented an 8-bit AES block cypher in C++ with the help of reference JAVA implementation. For 2-bit mode no block cypher mode is used as it is too small to implement, instead I have swapped the 2-bits to emulate some form of encryption.
The CBC mode is relatively easier to model for 2-bit as well as 8-bit. The code for 2 bit is as shown in figure 3.

Each byte of the input data broken down and is re-packaged into 2-bit blocks and encrypted using the block cypher. Similarly figure 4 shows the implementation for 8 bit cbc mode.

5. METHODOLOGY

Figure 5 shows algorithm for feasibility analysis in 2 bit cbc mode. The code hashed the blocks into an histogram array and finally divides by the total count to produce the probability distribution of each state.
Figure 6 shows the algorithm for analysing prediction accuracy with loopback. We start from a state and compute the result upto the loopback length and check if we arrived at the correct stage.

Figure 3: 2-bit CBC Mode

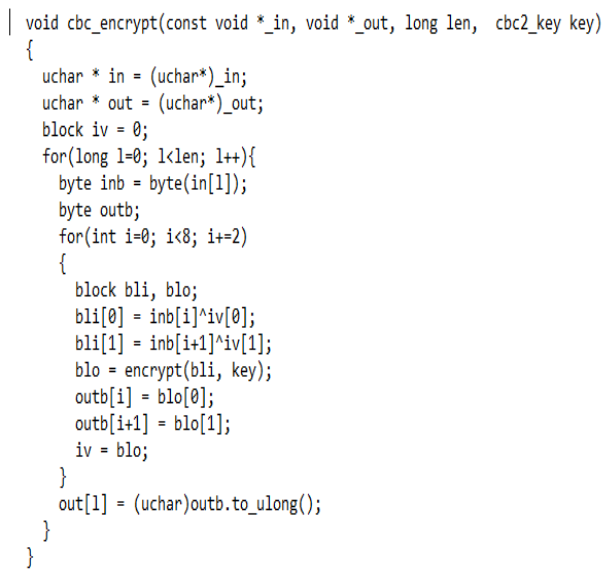
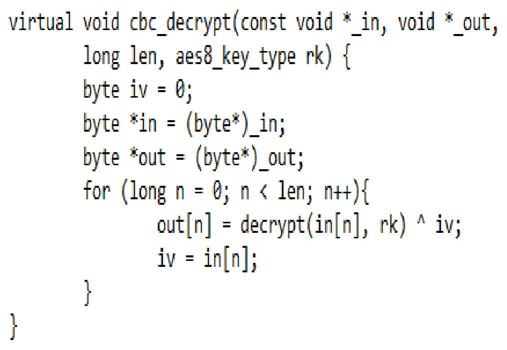

```
void cbc_encrypt(const void *_in, void *_out, long len,  cbc2_key key)
{
  uchar * in = (uchar*)_in;
  uchar * out = (uchar*)_out;
  block iv = 0;
  for(long l=0; l<len; l++){
    byte inb = byte(in[l]);
    byte outb;
    for(int i=0; i<8; i+=2)
    {
      block bli, blo;
      bli[0] = inb[i]^iv[0];
      bli[1] = inb[i+1]^iv[1];
      blo = encrypt(bli, key);
      outb[i] = blo[0];
      outb[i+1] = blo[1];
      iv = blo;
    }
    out[l] = (uchar)outb.to_ulong();
  }
}
```

Figure 4: 8-bit CBC Mode

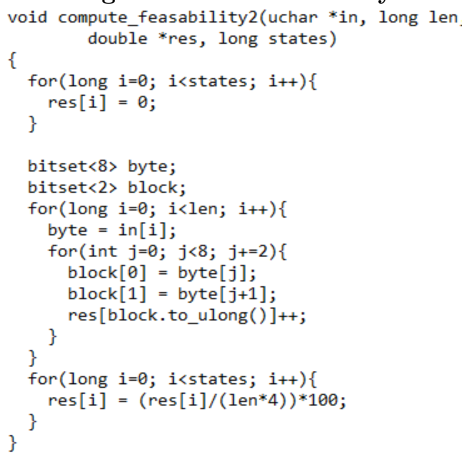

```
virtual void cbc_decrypt(const void *_in, void *_out,
        long len, aes8_key_type rk) {
        byte iv = 0;
        byte *in = (byte*)_in;
        byte *out = (byte*)_out;
        for (long n = 0; n < len; n++){
                out[n] = decrypt(in[n], rk) ^ iv;
                iv = in[n];
        }
}
```

6. PROPERTIES OF THE CBC FSM

After implementation of the CBC code for a reduced number of bits the following methodologies were used to test the properties of the FSM. I have kept the key constant and used a real text input to analyse the algorithm.

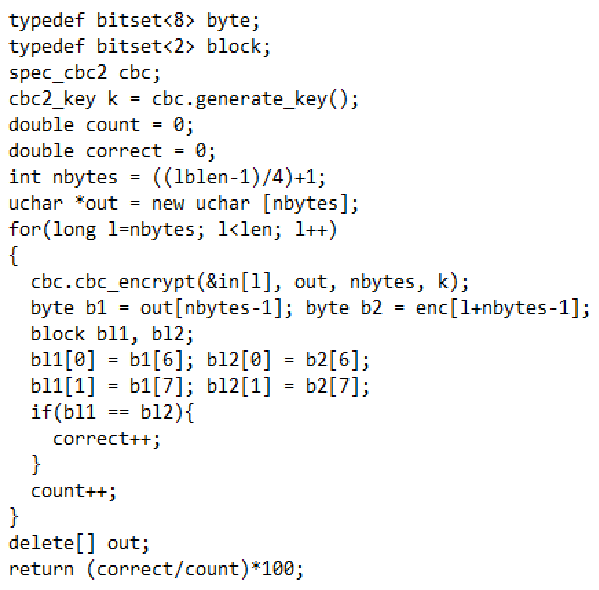
- **Size:** the number of possible outputs of a cypher block which is equal to 2^n bits i.e. 4 states for 2-bit mode and 256 states for 8-bit mode.
- **State Feasibility:** I have measured the state feasibility by encrypting a large text string and hashing and counting all outcomes to measure probability of each state.
- **Character Probability:** The character probability is measured by selecting am input state and measuring frequency of all output states during the profile runs.
- **Expected Convergent Length:** Average expected convergent length and probabilistic prediction of starting state with lookback as described in the FSM paper can be measured by comparing the encrypted results of two runs, one which starts from the beginning of the input and other which starts from an arbitrary point in input. Whenever it is found that both outputs are equal the length is noted and averaged.

Figure 5: 2-bit Feasibility


```
void compute_feasability2(uchar *in, long len,
        double *res, long states)
{
  for(long i=0; i<states; i++){
    res[i] = 0;
  }

  bitset<8> byte;
  bitset<2> block;
  for(long i=0; i<len; i++){
    byte = in[i];
    for(int j=0; j<8; j+=2){
      block[0] = byte[j];
      block[1] = byte[j+1];
      res[block.to_ulong()]++;
    }
  }
  for(long i=0; i<states; i++){
    res[i] = (res[i]/(len*4))*100;
  }
}
```

Figure 6: 2-bit predict with loopback


```
typedef bitset<8> byte;
typedef bitset<2> block;
spec_cbc2 cbc;
cbc2_key k = cbc.generate_key();
double count = 0;
double correct = 0;
int nbytes = ((lblen-1)/4)+1;
uchar *out = new uchar [nbytes];
for(long l=nbytes; l<len; l++)
{
  cbc.cbc_encrypt(&in[l], out, nbytes, k);
  byte b1 = out[nbytes-1]; byte b2 = enc[l+nbytes-1];
  block bl1, bl2;
  bl1[0] = b1[6]; bl2[0] = b2[6];
  bl1[1] = b1[7]; bl2[1] = b2[7];
  if(bl1 == bl2){
    correct++;
  }
  count++;
}
delete[] out;
return (correct/count)*100;
```

7. RESULTS AND LESSONS

State	0	1	2	3
0	0	∞	∞	∞
1	∞	0	∞	∞
2	∞	∞	0	∞
3	∞	∞	∞	0

Table 1: Convergent Lengths

The state feasibility analysis resulted in equal probability across all possible states as shown in 2. This would imply that all states of the CBC FSM are reached during execution with equal probability. Hence we would have a poor

State	0	1	2	3
P(s)%	24.89	25.11	24.93	25.05

Table 2: 2-Bit Feasibility Analysis

starting point for the state speculation function.

The character probability analysis also resulted in equal probability across all states. This would imply that the CBC FSM is an all-state to all-state dense FSM with no probabilistic relation between state transitions and input data.

The Expected convergent Length analysis also concluded that there is no average convergent length.**??**

Through my analysis I was able to conclude that the CBC algorithm is not a suitable candidate for speculation. The following reasons attribute to this cause:

- The input is XORd with the output of the previous block making it unpredictable.
- Due to the randomizing properties of the AES cypher block and the CBC mode there is no finite convergent length.
- The CBC algorithm has uniform state probabilities and infinite merging lengths leaving little potential for speculative parallelization. We can process the blocks in parallel by processing all the states at each stage. This however will have limited applicability as a secure form of CBC encryption will have a block size of atleast 128 bits making the number of states far too many to process.

8. ISSUES AND FUTURE WORK

Although the CBC mode is not a good candidate for speculative execution, the novelty and benefits of speculation parallelization cannot be undermined.
For parallelizing encryption of large text file we can explore CTR mode which has a parallelism friendly design. There has been some research in implementing AES CTR on GPUs with focus on memory access pattern optimization which can be further researched[1].

9. ACKNOWLEDGEMENT

I would like to acknowledge the guidance and support from Dr. Xipeng Shen.

References

[1] John Black and Phillip Rogaway. “A block-cipher mode of operation for parallelizable message authentication”. In: *Advances in CryptologyâĂŤEUROCRYPT 2002*. Springer. 2002, pp. 384–397.

[2] Yaobin Wang et al. “Accelerating Block Cryptography Algorithms in Procedure Level Speculation”. In: *Computational Intelligence and Security (CIS), 2011 Seventh International Conference on*. 2011, pp. 874–877. DOI: 10.1109/CIS.2011.197.

[3] WM Nunan Zola and LCE De Bona. “Parallel speculative encryption of multiple AES contexts on GPUs”. In: *Innovative Parallel Computing (InPar), 2012*. IEEE. 2012, pp. 1–9.

[4] Zhijia Zhao, Bo Wu, and Xipeng Shen. “Challenging the embarrassingly sequential: parallelizing finite state machine-based computations through principled speculation”. In: *Proceedings of the 19th international conference on Architectural support for programming languages and operating systems*. ACM. 2014, pp. 543–558.

[5] Zhijia Zhao and Xipeng Shen. “On-the-Fly Principled Speculation for FSM Parallelization”. In: *Proceedings of the Twentieth International Conference on Architectural Support for Programming Languages and Operating Systems*. ACM. 2015, pp. 619–630.