High Efficiency Counter Mode Security Architecture via Prediction and Precomputation

Weidong Shi Hsien-Hsin S. Lee Mrinmoy Ghosh Chenghuai Lu Alexandra Boldyreva

School of Electrical and Computer Engineering
College of Computing
Georgia Institute of Technology, Atlanta, GA 30332
{shi, lulu, aboldyre}@cc.gatech.edu *{leehs, mrinmoy}@ece.gatech.edu*

ABSTRACT

Encrypting data in unprotected memory has gained much interest lately for digital rights protection and security reasons. *Counter Mode* is a well-known encryption scheme. It is a symmetric-key encryption scheme based on any block cipher, e.g. AES. The scheme's encryption algorithm uses a block cipher, a secret key and a counter (or a sequence number) to generate an encryption pad which is XORed with the data stored in memory. Like other memory encryption schemes, this method suffers from the inherent latency of decrypting encrypted data when loading them into the on-chip cache. One solution that parallelizes data fetching and encryption pad generation requires the sequence numbers of evicted cache lines to be cached on-chip. On-chip sequence number caching can be successful in reducing the latency at the cost of a large area overhead.

In this paper, we present a novel technique to hide the latency overhead of decrypting counter mode encrypted memory by predicting the sequence number and pre-computing the encryption pad that we call *one-time-pad* or OTP. In contrast to the prior techniques of sequence number caching, our mechanism solves the latency issue by using idle decryption engine cycles to speculatively predict and pre-compute OTPs before the corresponding sequence number is loaded. This technique incurs very little area overhead. In addition, a novel adaptive OTP prediction technique is also presented to further improve our regular OTP prediction and precomputation mechanism. This adaptive scheme is not only able to predict encryption pads associated with static and infrequently updated cache lines but also those frequently updated ones as well. Experimental results using SPEC2000 benchmark show an 82% prediction rate.

Moreover, we also explore several optimization techniques for improving the prediction accuracy. Two specific techniques, *Two-level prediction* and *Context-based prediction* are presented and evaluated. For the two-level prediction, the prediction rate was improved from 82% to 96%. With the context-based prediction, the prediction rate approaches 99%. *Context-based OTP prediction* outperforms a very large 512KB sequence number cache for many memory-bound SPEC programs. IPC results show an overall 15% to 40% performance improvement using our prediction and precomputation, and another 7% improvement when context-based prediction techniques is used.

1. INTRODUCTION

There is a growing interest in creating secure software execution environment that combines the strengths of cryptography and secure operating systems to fight against attacks on software copyrights [12, 13, 20, 21, 25, 26] or to counter malicious buffer overflow and code injection exploits [6]. One critical design component of these security systems is memory encryption that provides protection for program and data privacy. For such security systems, memory encryption often lies at the center of protection. It is primarily used to create a security platform to obstruct software-based exploits or to construct a network security system to avoid code injection attacks. For example, a security system can feature a security co-processor to provide protection on data privacy. Important information and dynamic data can be encrypted or sealed by the secure co-processor when they are stored in memory. Another interesting application is to use memory encryption against remote code injection attack [6]. The technique presented in this paper, though discussed in the context of a secure processor with an integrated crypto-engine, can be extended to many scenarios of a security system design that uses memory encryption. The actual memory encryption can be performed either in software, co-processor, or in an integrated crypto-engine. However, since memory encryption is often a component of a security system, we do not elaborate on how to create a specific security system for a specific goal using memory encryption in this paper.

One crucial performance challenge of memory encryption is the decryption latency. Fast protection schemes based on counter mode encryption were introduced recently to meet this challenge as counter mode allows parallel execution of encrypted data fetching and decryption pad generation [20, 25]. But, a counter (also called *sequence number*) associated with each data block has to be cached inside the secure processor to exploit such parallelism enabled by counter mode. When the sequence number is missing in the on-chip cache, it needs to be brought back to the secure processor, then used to generate the corresponding encryption OTP using a block cipher. The whole process often takes hundreds of cycles to complete before the resultant OTP can be used for decryption data. Incorporating a large size sequence number cache inside the processor may help reduce the latency, however, by paying a substantial area overhead.

In this paper, we propose a novel technique called *OTP*

0-7695-2270-X/05/$20.00 (C) 2005 IEEE

prediction for addressing the long decryption latency issue. OTP prediction tackles the latency problem by using idle decryption engine cycles to predict sequence numbers and speculatively pre-compute their corresponding OTPs for missed memory blocks. The advantages of OTP prediction over sequence number caching are twofold. First, OTP prediction has far less demand on chip area than sequence number caching. Second, OTP prediction is far more effective in hiding memory fetch latency than sequence number caching. Performance analysis shows a significant improvement in both prediction rate and IPC using OTP prediction. We show that OTP prediction without any sequence number cache outperforms a large 512KB sequence number cache in many cases. In addition, we also propose and evaluate several prediction optimization techniques that can approach perfect prediction of sequence numbers. This means that using OTP prediction and given that the OTP generation latency is less than the memory latency, we can support memory protection without loss of performance. Note that unlike sequence number caching, all the performance improvement is achieved with little area overhead since OTP prediction is not based on caching, rather, it uses idle pipeline stages of decryption engine to speculate and pre-compute OTPs required for decrypting incoming memory blocks.

The major contributions of this work are:

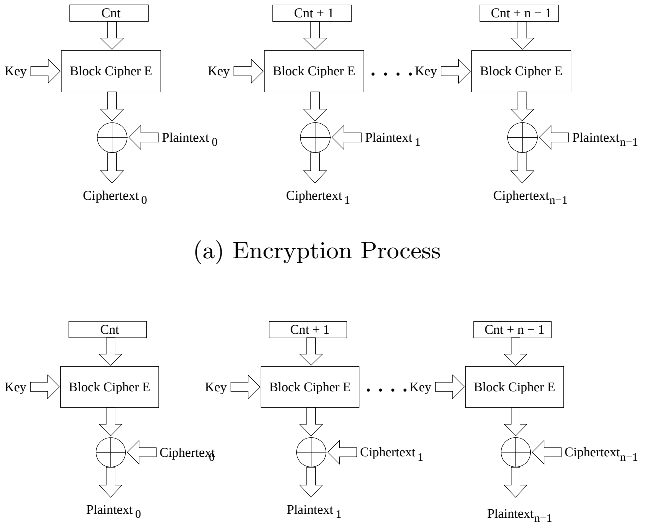
- We propose a novel OTP prediction and precomputation framework that successfully hides the latency overhead of fetching counter mode encrypted data from memory.
- We propose a unique adaptive OTP prediction mechanism that can predict both the static and infrequently updated data and the frequently updated dynamic data.
- We evaluate two OTP prediction optimization techniques: *Two-level OTP prediction* and *Context-based OTP prediction*. Both can improve the OTP misprediction rate substantially. Two-level prediction uses a range predictor that first predicts the relative range of a sequence number. Then the regular OTP prediction is carried out in that predicted range. Context-based prediction maintains a relative sequence number of the most recent memory access in a history register and uses it to generate predictions in addition to the regular OTP prediction.

The rest of the paper is organized as follows. Section 2 discusses the concept of counter mode in general and its application to secure processor design. Section 3 describes our OTP prediction and precomputation framework followed by security analysis in Section 4. We then evaluate and analyze the performance of our proposed techniques and optimizations from Section 5 to Section 8. Section 9 discusses prior work and finally, Section 10 concludes our work.

2. COUNTER MODE SECURITY

2.1 Counter mode encryption

Counter mode encryption is a common symmetric-key encryption scheme [7]. It uses a block cipher (e.g. AES [8]), a keyed invertible transform that can be applied to short fixed-length bit strings. To encrypt with the counter mode, one starts with a plaintext *P*, a counter *cnt*, a block cipher E, and a key. An encryption bitstream (OTP) of the form E(key, cnt) || E(key, cnt+1) || E(key, cnt+2) ...|| E(key, cnt+n-1) is generated as shown in Figure 1(a). This bitstream is XORed with the plaintext bit string P, producing the encrypted string ciphertext C. To decrypt, the receiver computes the same pad used by the sender based on the same counter and key, XORs the pad with C, then restores the plaintext P. P is padded, if necessary, to facilitate the OTP length.



(a) Encryption Process

(b) Decryption Process

Figure 1: Counter Mode Encryption

Counter mode is known to be secure against chosen-plaintext attacks, meaning the ciphertexts hide all partial information about the plaintexts, even if some a priori information about the plaintext is known. This has been formally proved in [3]. Security holds under the assumptions that the underlying block cipher is a pseudo-random function family (this is conjectured to be true for AES) and that a new unique counter value is used at every step. Thus a sequence number, a time stamp or a random number can be used as an initial counter. Note that the counter does not have to be encrypted. As most encryption modes, counter mode is malleable and thus is not secure against chosen-ciphertext attacks. For example, flipping one bit in a ciphertext results in the flipped bit in the plaintext. Also, counter mode does not provide authentication (integrity) of the data. For these reasons an additional measure such as message authentication code (MAC) should be used. If a secure MAC is applied to the counter mode ciphertext during the encryption and is verified during the decryption process, then the resulting scheme provides authentication and integrity, is non-malleable and is secure against chosen-ciphertext attacks. This is formally proved in [4]. As pointed out by [15], most other perceived disadvantages of counter mode are invalid, and are caused by lack of knowledge.

2.2 Counter mode security architecture

Memory encryption schemes based on counter mode were employed for its high efficiency by prior proposed security architectures [20, 25]. In these schemes, a counter[1] is associ-

[1] In many encryption mode descriptions the *counter* is some-

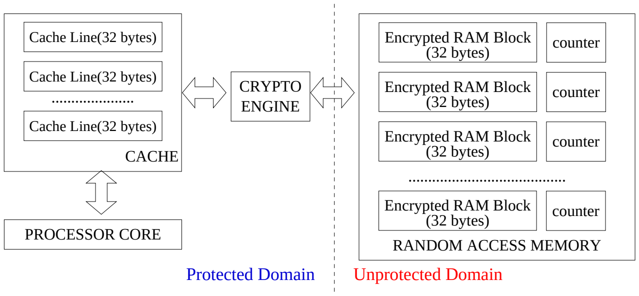


Figure 2: Concept of Counter Mode Security

ated with each cache line size memory block of the physical RAM as shown in Figure 2. Whenever a cache line is evicted from the secure processor, the corresponding counter is incremented and the result is forwarded to the crypto-engine to generate the OTP for the evicted cache line. Meanwhile, the corresponding counter value in the memory is updated. When an encrypted memory block is fetched, the corresponding counter must be fetched first from the memory to regenerate the OTP for decryption. As we mentioned earlier the counters do not need to be encrypted because the security strength of counter mode does not rely on their secrecy [3]. Unlike other regular block cipher based direct memory encryption schemes that serialize line fetching and decryption process, the potential advantage offered by counter mode is that it overlaps the OTP generation with fetching of encrypted program and data [20, 25]. However, to computer the OTP for a fetched cache line, the counter needs to be fetched from the memory first, eliminating the advantage of OTP pre-computation.

In [20, 25], a small sequence number (or counter) cache is used to cache counter values on-chip to exploit this advantage of counter mode architecture. In practice, however, these specialized caches do not hide decryption latency effectively because its hit rate does not grow steadily with its size. One possible explanation of this "plateau" effect of the sequence number cache is that the sequence number cache may contain (multiple) very large working sets . A small cache of several Kbytes may be enough to capture the first working set, but other working sets might be too large to be captured by a sequence number cache of tens or even hundreds of Kbytes. In other words, the area cost to improve the hit rate via simple caching can be prohibitively high.

We propose an alternative solution via prediction and precomputation to hide memory latency more effectively with minimal area overhead. The premise is that predictable counters are often used to generate the encryption OTP deterministically using some standard encryption function. Hence one can speculate the counter value and pre-compute the OTP before the counter is loaded from memory. It is important to point out that even though the counter value is deterministic and predictable, the cryptographic function used to generate the OTP is not. These functions involve a secret key only known to the processor itself. It is computationally infeasible for adversaries to predict the OTP even with a known counter value.

There are some assumptions behind a security processor in general for protecting software and data privacy. As mentioned earlier, memory encryption is the central component for building a variety of security systems, ranging from hardware-based tamper-resistant system, to systems countering remote or local software exploits on user data. Throughout this paper, we present our latency hiding techniques applied to memory encryption in the context of tamper resistant system supported with an integrated crypto-engine as shown in Figure 2. Nonetheless, the techniques can be generalized to other security systems based on memory encryption as well. We highlight the assumptions of our security processor model below based on prior security architecture frameworks [13, 19, 20].

- Like any other encryption mode, counter mode encryption itself does not provide integrity protection. Extra or additional measures such as Hash/MAC tree for integrity protection [21] must be used together with counter mode encryption.
- We assume a trusted OS kernel and a secure boot [1] that protect and properly manage security-related *process context* during context switch and interrupt handling [13, 21]. Proper management means maintaining privacy of information related to a protected process[2]. Methods for creating a trusted nucleus such as secure boot and secure kernel have been studied before [1].
- In a multiprogrammed environment, dynamic data of each process is protected with different cryptographic keys.

The above list of assumptions is by no means complete. But, it is important to note that many of these assumptions are design specific. For example, if the goal of a secure system is to use memory encryption to prevent remote code injection attacks [6] or countering malicious scan of memory space using illegitimate software, many of these assumptions are not needed.

3. OTP PREDICTION AND PRECOMPUTATION

In this section we explain the concept behind OTP prediction and pre-computation[3]. We first explain the technique using our proposed regular sequence number prediction algorithm and we point out a potential performance issue for the regular sequence number prediction. Next, we discuss a simple modification to the regular prediction scheme to redress the potential performance problem. It is also important to note that *OTP prediction* can be tied with any scheme based on stream cipher such as those discussed in [20, 25].

3.1 Regular OTP prediction

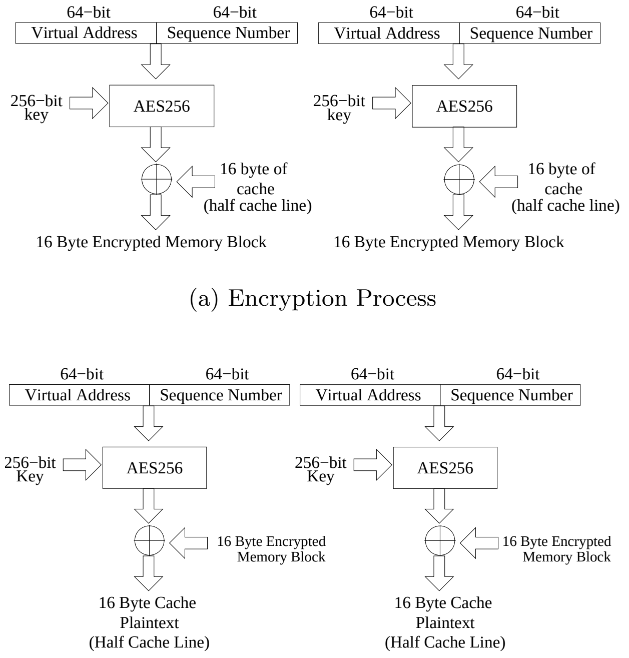
The concept of OTP prediction and pre-computation can be understood from the time-lines shown in Figure 4. We assume that the memory access latency and the encryption OTP generation latency are comparable[4]. We also assume

times referred to as *sequence number*, *nonce*, *initial vector*. We will use *sequence number* or *counter* interchangeably throughout this paper.

[2] Security related to context includes but not limited to register values, page table, hash/MAC tree node, dirty cache lines, and other per-process security related information, for instance root sequence numbers, context information used by OTP Prediction.

[3] Hereafter we use OTP prediction to represent OTP prediction and pre-computation.

[4] We later justify this assumption in Section 5.

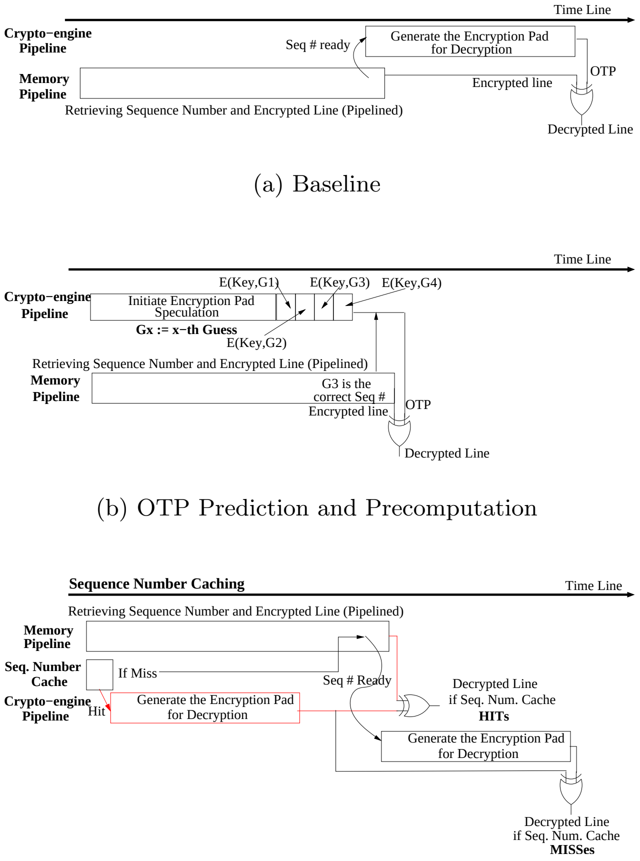

64−bit
64−bit
Virtual Address
Sequence Number
256−bit key
AES256
16 byte of cache (half cache line)
16 Byte Encrypted Memory Block
64−bit
64−bit
Virtual Address
Sequence Number
256−bit Key
AES256
16 Byte Encrypted Memory Block
16 Byte Cache Plaintext (Half Cache Line)

(a) Encryption Process

(b) Decryption Process

Figure 3: OTP Encryption Modes

a fully-pipelined encryption and decryption AES crypto-engine as illustrated in Figure 3. The input block of the AES is a concatenation of a 64-bit virtual address and a 64-bit sequence number. For 32-bit architecture, the virtual address is padded to 64-bit. The timeline of Figure 4(a) shows a scenario when fetching a cache line in a baseline security architecture without any support to accelerate the decryption process. It is obvious that the crypto-engine pipeline sits idle for the whole time between the time when the request is sent to the memory and the time when the sequence number is returned. Now suppose that in our architecture the sequence number of a given cache line is predictable and only depends on the page the cache line is associated with. We will justify our claim later. Figure 4(b) shows the timeline of our framework. A certain number of sequence number guesses, G1, G2, ..., Gx, can be tried and passed to the crypto-engine for pre-computing their corresponding OTPs while the actual memory request is being sent to memory. Since the AES cipher is pipelined, the predicted and precomputed OTPs will be available around the time the actual sequence number is obtained from memory. The precomputed OTPs are represented as E(Key,G1),E(Key,G2)... in the Figure 4(b). When the actual sequence number returns, we check it against all the guessed numbers. If one of them matches, in our case G3, we already have the encryption pad E(key,G3) for G3. We can now directly obtain the plaintext by XORing the pre-computed OTP with the encrypted cache line fetched. The rationale of our scheme is to utilize the idle time of the crypto-engine pipeline for pre-computing several OTPs and thus hide the memory access latency if one of the speculations succeeds. To demonstrate its difference from prior art, Figure 4(c) shows the sequence number caching technique.


Time Line
Crypto−engine Pipeline
Seq # ready
Generate the Encryption Pad for Decryption
Memory Pipeline
OTP
Encrypted line
Retrieving Sequence Number and Encrypted Line (Pipelined)
Decrypted Line
Time Line
E(Key,G1)
E(Key,G3)
E(Key,G4)
Crypto−engine Pipeline
Initiate Encryption Pad Speculation
Gx := x−th Guess
E(Key,G2)
Retrieving Sequence Number and Encrypted Line (Pipelined)
Memory Pipeline
G3 is the correct Seq # Encrypted line
OTP
Decrypted Line
Sequence Number Caching
Time Line
Retrieving Sequence Number and Encrypted Line (Pipelined)
Memory Pipeline
Seq. Number Cache
If Miss
Seq # Ready
Crypto−engine Pipeline
Hit
Generate the Encryption Pad for Decryption
Decrypted Line if Seq. Num. Cache HITs
Generate the Encryption Pad for Decryption
Decrypted Line if Seq. Num. Cache MISSes

(a) Baseline

(b) OTP Prediction and Precomputation

(c) Sequence Number Caching

Figure 4: Timeline Comparison of Different OTP Computation

A sequence number cache stores a finite set of sequence numbers for evicted lines. If a request hits the sequence number cache, the sequence number is obtained and the OTP generation can begin much before the cache line returns from memory. For a sequence number cache miss, the decryption process will be serialized similar to the baseline scenario. One issue of sequence number caching is the hit rate, which can be substantially reduced when the working set is large or in-between context switches. Another issue is the potential large hardware overhead dedicated to the sequence number storage for achieving decent hit rates. It should be kept in mind that the area overhead of our scheme is minimal compared to the caching scheme for we only need a small buffer to store the pre-computed OTPs. We will show in our results that the benefit we receive with our optimized design cannot be achieved even with a very large sequence number cache. Our scheme is also complementary to the sequence number cache. We can combine them in a design to gain benefits offered by both. Now we elaborate our prediction architecture in details.

Figure 5 shows the design of our OTP prediction and precomputation mechanism. A root OTP sequence number is assigned to each virtual memory page by a hardware ran-

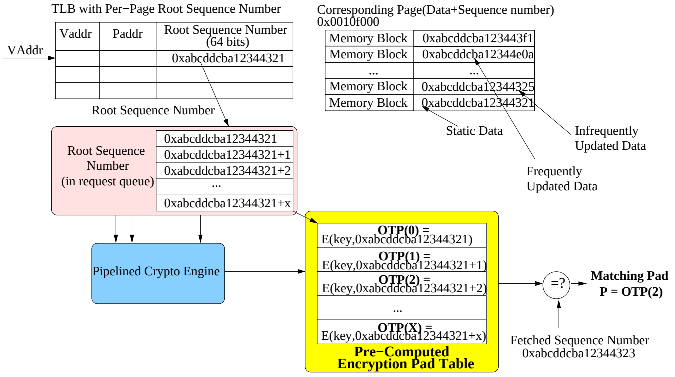

TLB with Per−Page Root Sequence Number
VAddr
Vaddr
Paddr
Root Sequence Number (64 bits)
0xabcddcba12344321
Root Sequence Number
Corresponding Page(Data+Sequence number)
0x0010f000
Memory Block
0xabcddcba123443f1
Memory Block
0xabcddcba12344e0a
...
Memory Block
0xabcddcba12344325
Memory Block
0xabcddcba12344321
Static Data
Infrequently Updated Data
Frequently Updated Data
Root Sequence Number (in request queue)
0xabcddcba12344321
0xabcddcba12344321+1
0xabcddcba12344321+2
...
0xabcddcba12344321+x
Pipelined Crypto Engine
OTP(0) = E(key,0xabcddcba12344321)
OTP(1) = E(key,0xabcddcba12344321+1)
OTP(2) = E(key,0xabcddcba12344321+2)
OTP(X) = E(key,0xabcddcba12344321+x)
Pre−Computed Encryption Pad Table
=?
Matching Pad P = OTP(2)
Fetched Sequence Number 0xabcddcba12344323

Figure 5: OTP Prediction

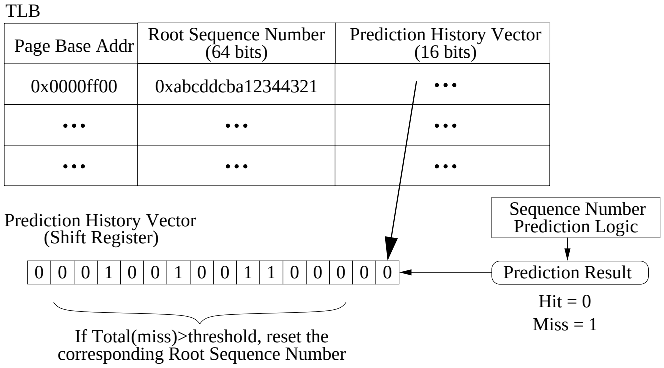

TLB
Page Base Addr
Root Sequence Number (64 bits)
Prediction History Vector (16 bits)
0x0000ff00
0xabcddcba12344321
...
Prediction History Vector (Shift Register)
0 0 0 1 0 0 1 0 0 1 1 0 0 0 0 0
Sequence Number Prediction Logic
Prediction Result
Hit = 0
Miss = 1
If Total(miss)>threshold, reset the corresponding Root Sequence Number

Figure 6: Prediction Tracking and Sequence Number Resetting

dom number generator each time the virtual page is mapped to a physical one. All the cache lines of the same page use the same root OTP sequence number for their initial values. Each TLB entry is tagged with the root sequence number of the corresponding page. Whenever a dirty line is evicted from the secure processor, the sequence number associated with the corresponding line is incremented. For each missing cache line, a request for the line itself and its associated sequence number is sent to memory. Simultaneously, the prediction logic takes the root OTP sequence number associated with the virtual page, and inserts a few sequence number guesses into the *request queue*. The pipelined crypto-engine takes each request and computes its corresponding encryption OTP. Upon the receipt of the correct sequence number from memory, the processor compares it with the set of sequence number predictions. If a match is found, then the corresponding pre-computed OTP is used to decrypt the fetched memory data. If no match can be found, the crypto-pipeline will take the newly received sequence number and computes its OTP, same as baseline. In summary, OTP prediction effectively hides the latency of encryption pad generation by speculatively pre-computes encryption pads using speculated sequence numbers.

Sequence number prediction is based on the observation, that during the whole lifetime a physical memory page is bound to a virtual memory page, many of its cache lines are only updated a very small number of times. Our profiling study of SPEC benchmarks indicates that many lines are rarely updated during the entire process lifetime, in other words, when a cache line missing the L2 cache, its sequence number is very likely to be within a small range of the first time initialized random sequence number associated with that memory page. Regular OTP prediction is designed to predict sequence numbers associated with static and infrequently updated data.

3.2 Adaptive OTP prediction for frequently updated data

There is a concern about the performance of OTP prediction over a large time window of execution. It is reasonable to suspect that prediction rate may drop as data are frequently updated. To address this issue, a dynamic prediction rate tracking and sequence number reset mechanism is proposed. The purpose of this mechanism is to identify those virtual pages with low prediction rate caused by frequent memory updates and reset its page root sequence number to a new random value so that high predictability can be maintained.

Prediction tracking is performed in hardware using a scheme as follows. There is a 16 bit prediction history vector (PHV) for each memory page. The PHV records hit or miss of the last 16 sequence number prediction on cache lines of the associated page. Every time a cache line is loaded from memory, the PHV of that page is updated by shifting the new prediction result into the vector. (1 for misprediction and 0 for hit). When the total number of mispredictions of the last 16 predictions is greater than a threshold, the root sequence number associated with that page will be reset to a new randomly generated number. After reset, blocks of the involved page will use this new number for OTP generation next time when it is evicted from the L2.

The adaptive predictor requires the ability to test whether a sequence number used by a cache line is counted based on the current root sequence number. Note that this function does not have to be 100% accurate because a wrong test result will only cause reset of the sequence number. A simple implementation is to use the distance between a sequence number and the current root sequence number as a criteria. To decide whether a sequence number started its count from the current root sequence number, its distance to the current root is calculated. If the distance is negative or too large, the sequence number is considered counting from an old root sequence number. If a mismatch is detected, the corresponding cache line will reset its sequence number to the current root sequence number.

4. SECURITY ANALYSIS

We discuss a few issues about OTP prediction security in this section.

- Security is guaranteed by the security analysis provided in [3] since we did not modify the counter mode encryption scheme itself, but rather showed how to decrease the long decryption process latency via architectural techniques.
- In OTP prediction, different memory blocks of the same page may use the same sequence number. This, however will not weaken security. When the OTP is generated, the address is used together with the sequence number and a prefix-padding. Since each memory block has a different address, the resultant OTP will be different for

different memory blocks. Knowing OTP of a particular memory block does not reveal any information about the OTPs of other blocks of the same page.

- The root sequence number is set and reset using a hardware random number generator. For the sequence number to wrap-around, it has to be incremented 2^{64} times. This equals hundreds of years under current processor clock speed.

5. SIMULATION METHODOLOGY AND IMPLEMENTATION

5.1 Simulation framework

Our simulation framework is based on SimpleScalar 3.0 running SPEC2000 INT and FP benchmark programs compiled with -O3 option. We implemented architecture support for *OTP Prediction* and root sequence number history over SimpleScalar's out-of-order Alpha simulator. We also integrated an accurate DRAM model [9] to improve the system memory modeling, in which bank conflicts, page miss, row miss are all modeled following the PC SDRAM specification. The architectural parameters used for performance evaluation are listed in Table 1. To model OTP prediction faithfully, we added memory profiling support to SimpleScalar that keeps track of memory transactions for evaluating OTP prediction, such as number of times a memory block is evicted from L2 cache, the sequence number assigned to each virtual page, and etc. Each benchmark is fast-forwarded at least 4 billion instructions and simulated in a representative place according to SimPoint [18] for 400M instructions in performance mode. During fast-forwarding, L1 cache, L2 cache, sequence number cache, sequence number prediction mechanism are simulated. The profiled memory status is also updated during fast-forwarding. To study the performance sensitivity of OTP prediction optimization, we also run each benchmark in a simplified mode that simulates the memory hierarchy and OTP prediction for 8 billion instructions. We used 16 bits for the prediction history window. By default, the sequence number of each virtual page is reset if the number of prediction misses over the last 16 is greater than or equal to 12. The prediction depth, that is the number of guesses generated for each missing sequence number, is set to 5. We also use a prediction swing of 3 for context-based prediction to be discussed in Section 7.4. Per-page root sequence numbers require small storage space. Given a 64-bit sequence number and 256 page entries cached, the total cost of storing root sequence numbers is about 2KB. Also, to simulate the effect of OS and system, dirty lines of caches are flushed every 25million cycles. Furthermore, we subset the SPEC simulations for those with high L2 misses. All the benchmarks are simulated under the security setting that data privacy must be protected.

5.2 Block cipher implementation

The AES cipher processes 128-bit data blocks by using key lengths of 128, 192 and 256 bits. It is based on a round function, which is iterated 10 times for a 128-bit length key, 12 times for a 192-bit key, and 14 times for a 256-bit key. Each round consists of four stages. The first stage is *subbytes* transformation. This is a non-linear byte substitution for each byte of the block. The second stage known as *shiftrows* transformation, cyclically shifts (permutes) the bytes within the block. The third stage involves *mixcolumns* transformation. It groups 4 bytes together forming 4-term polynomials and multiplies the polynomials with a fixed polynomial mod (x^4+1). The fourth stage is *addroundkey* transformation. It adds the round key with the block of data. For high throughput and high speed hardware implementation, AES is often unrolled and with each round pipelined into multiple pipeline stages (4-7) to achieve high decryption/encryption throughput [10, 16, 11]. The minimal total area of unrolled and pipelined Rijndael is about 100K - 150K gates to achieve 15-20Gbit/sec throughput [11]. Based on our real verilog implementation and synthesis results, each round of pipelined AES takes about 5nsec - 6nsec using 0.18 μ standard cell library. In this study, the default latency is 96ns for 256-bit AES.

Parameters	Values
Fetch/Decode width	8
Issue/Commit width	8
L1 I-Cache	DM, 8KB, 32B line
L1 D-Cache	DM, 8KB, 32B line
L2 Cache	4way, Unified, 32B line, Writeback, 256KB and 1MB
L1 Latency	1 cycle
L2 Latency	4 cycles (256KB), 8 cycles (1MB)
I-TLB	4-way, 256 entries
D-TLB	4-way, 256 entries
Memory Bus	200MHz, 8B wide
AES latency	14 rounds with an initial and a final round, 6 stages 1ns each 96ns
Sequence number cache	4KB, 128KB, 512KB (32B line)
Prediction History Vector	16 bit
PHV threshold	12
Prediction depth (used by all predictions)	5
Prediction swing (used by context-based only)	3

Table 1: Processor model parameters

6. EVALUATION OF ADAPTIVE OTP PREDICTION

In this section we summarize the prediction rates and IPC results of adaptive OTP prediction. The results are collected for two L2 cache sizes, 256KB and 1MB. Choosing a 256KB L2 cache is not only representative for many contemporary machines but also appropriate for our evaluation given the SPEC benchmark suite is known to have relative small working set.

6.1 OTP prediction rate over large execution time

One concern about OTP prediction is that its performance over execution time or its capability to predict dynamic data. To answer this question, we simulated performance of *OTP prediction* over a relatively large time window, 8 billion instructions. We also used two different sequence number cache sizes, 128KB, and 512KB, for our reference comparison. Figure 7 shows the sequence number hit rates for different sequence number cache sizes and our OTP prediction. As the results indicate, the hit rate of sequence number cache is relatively low even with a decent sized 128KB sequence number cache. The results also indicate that sequence number prediction can achieve good prediction rate over a long execution window. The average prediction rate

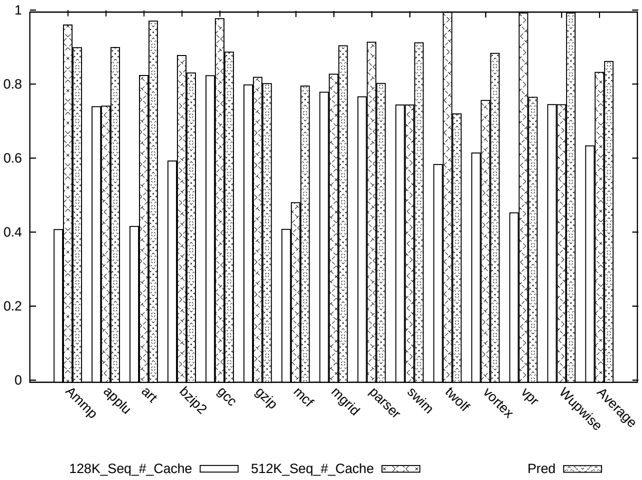


Figure 7: Sequence Number Hit Rates, 256KB L2, 8 billion instructions

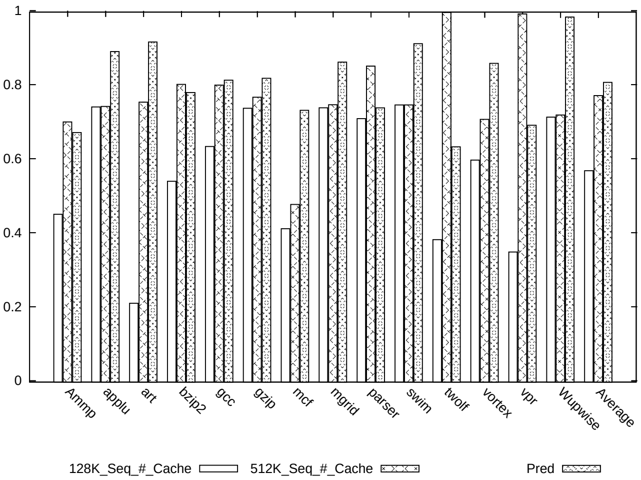


Figure 8: Sequence Number Hit Rates, 1MB L2, 8 billion instructions

is 82%, higher than that of a 128KB or a 512KB sequence number cache. Figure 8 shows the same comparison with a 1MB L2. Similarly, OTP prediction also achieves a better performance than sequence number caching for a fairly large L2. The average prediction rate is 80% compared to 57% for a 128KB sequence number cache.

The results also verify the initial assumption that sequence numbers tend to have large working set. One possible explanation is that for the sequence number cache to perform well, the processor has to miss on the same memory block many times within a short time window before the sequence number is evicted from the sequence number cache. Due to the temporal locality and memory working set, a processor rarely repeats missing on the same memory block many times in a short duration of time. This illustrates the limitation of using caching for improving performance since the area cost can be prohibitively high.

Figure 9 breaks the total number of hits into three categories, 1) hit both, a sequence number that is in the sequence number cache and can be predicted; 2) prediction only, a sequence number that is missing in the sequence number cache, but can be predicted; 3) sequence cache only, sequence number cannot be predicted but available in the cache. The results are collected from simulation using a 32KB sequence number cache plus prediction. As seen from the figure, OTP prediction can uncover more performance opportunities lost by the sequence number caching scheme.

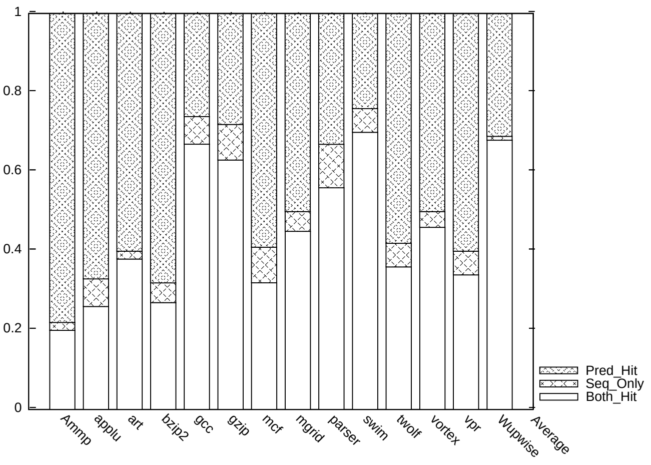


Figure 9: Breakdown of Contribution of Sequence Number Cache, and OTP Prediction

6.2 IPC improvement using OTP prediction

Increasing prediction rate has a great performance impact on memory-bound benchmarks. For example, without OTP prediction, the average IPC of the selected benchmarks only reaches 82% of IPC of an oracle scenario where every sequence number is cached. In particular, bzip2, mcf, mgrid, twolf, and vpr have their ratios in the range of 60% to 80%. If OTP prediction can achieve ideal 100% prediction rate, the potential performance improvement would be in the range of 20% to 40%.

Figure 10 and Figure 11 show normalized IPC performance of large sequence number caches vs. adaptive OTP prediction. The IPC is normalized to the oracle case. As shown, OTP prediction can effectively improve performance. On average, IPC is increased by 18% and 11% for a 256KB L2 and a 1MB L2, respectively. For ten of the fourteen SPEC2000 benchmarks, the improvement is in the range from 15% to 40%. Six benchmarks have their improvements over 20% and two over 30%. For every benchmark, OTP prediction outperforms a 128KB sequence number cache. For average IPC, *OTP prediction* even performs better than a very large 512KB sequence number cache. The results clearly show the advantage of OTP prediction over a pure sequence number caching.

Consistent with the prediction rate results, sequence number prediction is more effective than caching for overall performance. To achieve similar performance using caching, the required sequence number cache size needs to be unreasonably large, larger than a typical unified L2 cache.

7. OPTIMIZING OTP PREDICTION

Although adaptive OTP prediction can handle both infrequently and some frequently updated data, our study of prediction rate shows that there is still room for improvement. In this section, we propose and investigate some unique op-

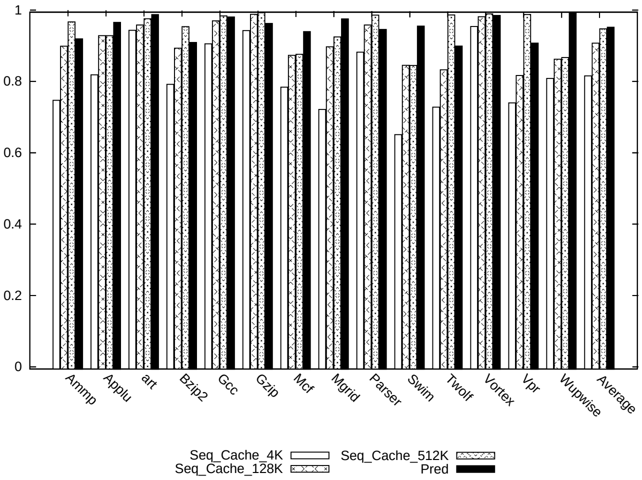


Figure 10: Normalized IPC Under Different Sequence Number Cache Sizes(4KB, 128KB and 512KB) vs OTP Prediction, 256KB L2

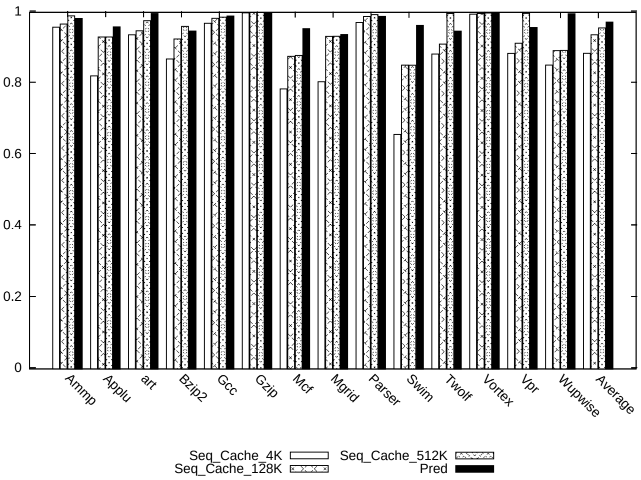


Figure 11: Normalized IPC Under Different Sequence Number Cache Sizes(4KB, 128KB and 512KB) vs OTP Prediction, 1MB L2

timizing techniques to increase the prediction performance for frequently updated data.

7.1 Profiling misprediction

First, we conduct profiling studies on OTP misprediction. We found there are two main contributors. One is *prediction depth* which is equivalent to the number of predictions can be made for each missing line. Profiling studies reveal that some lines are evicted far more often than the others and they are often outside the prediction range. Increasing the prediction depth, i.e. more predictions per line, does not solve the problem as too many predictions will overload the crypto-engine and could lead to negative impact on performance. The second contributor is due to the reset of the root sequence number. After the per-page root sequence number is reset, all the future predictions on cache lines of the same page will use the new root sequence number instead of the old ones that causes predictions of lines using old root sequence numbers to fail.

7.2 Two-level prediction

To reduce mispredictions caused by short prediction depth, we introduce a novel range prediction technique to be used in combination with the regular OTP prediction. Regular OTP prediction is good at predicting sequence numbers that are not updated frequently. For example, if the prediction depth is 8 and a cache line has been evicted 23 times. It is not possible for the regular OTP prediction to predict correctly. However, if we divide the distance between each sequence number to its root sequence number into multiple ranges, for instance, four ranges, [1, 8], [9, 16], [17, 24], [25, ∞] and have OTP predictions generated only under a particular range for each sequence number, it will greatly increase the hit rate of OTP prediction without adding pressure to the crypto-engine pipeline. We call this design, *Two-level OTP Prediction*, with the first level predicting the possible range of a sequence number and the second level using regular OTP prediction in that range. To implement a two-level OTP prediction, range information associated with each cache line needs to be encoded and stored. In fact, the cost of the first level prediction is small. For example, assume that there are four ranges, this information can be encoded with only 2 bits. Under a 4KB page and 32-byte lines, the cost to store range information for all the 128 lines of a page is only 256 bits.

When accessing a sequence number, the secure processor will look up the range prediction table, where each entry of the table stores the range information for all lines in a page. Then the retrieved range information is used by the regular OTP prediction where a number of predicted sequence numbers are inserted to the *prediction queue*. The starting predicted sequence number is *root sequence number + range lower bound*, and the last predicted sequence number is *root sequence number + range lower bound + prediction depth*. When a line is evicted from the processor, its associated entry in the range prediction table is updated.

The additional cost of range prediction is relatively small considering a 64 entry table costs only about 2KB with the benefit of quadruple the effective prediction depth.

7.3 Root sequence number history

Per-page root sequence number reset is important to maintain a satisfactory OTP prediction performance. However, resetting and discarding old root sequence numbers may cause predictions on sequence numbers based on discarded root numbers to fail. To prevent misprediction caused by resetting, we introduced a sequence number memoization technique that keeps a certain number of old root sequence numbers (usually very small, 1 or 2 at most) as history. When predicting a missing sequence number, predictions based on both the new root sequence number and the old sequence number(s) are generated. It is important to note that when a dirty cache line is being evicted, the writeback is always encrypted using a new OTP based on the current root sequence number.

7.4 Context Based Prediction

Context-based prediction is another OTP prediction optimization that can significantly improve the OTP prediction rate. The idea of context-based prediction is very simple. We use a register called *Latest Offset Register (LOR)* that records the offset sequence number of the most recent memory access (= sequence number - root sequence number).

For a new memory fetch, aside from the regular predictions based on the per-page root sequence number, the context-based OTP prediction mechanism also generates a few more predictions based on the LOR value. In context-based prediction, two sets of prediction are made. The first is our regular prediction using the prediction depth (*pred_depth*). Assume that root(addr) returns the root sequence number associated with fetched data, so the prediction of sequence numbers falls in the range of $[root(addr), root(addr)+pred\_dep$ in other words, $(pred\_depth + 1)$ predictions are generated. The second prediction set uses the LOR value with a *prediction swing (pred_swing)*. The prediction falls in the range of $[Max(root(addr)+LOR-pred\_swing, root(addr)), root(addr)$ $LOR + pred\_swing]$ with a maximum of $2 \cdot pred\_swing + 1$ predictions to be made. In our simulation, the *pred_depth* and *pred_swing* are set to 5 and 3 respectively.

Context-based prediction has far less overhead than Two-level prediction because it requires only one extra register rather than a range table. Comparing with the regular OTP prediction scheme, the additional cost is almost negligible. However, Context-based prediction does increase the workload on the OTP engine because it generates more predictions. As will be shown, context-based prediction in fact has the best prediction rate among all three prediction schemes.

8. PERFORMANCE EVALUATION

In this section, we present performance evaluation of the proposed optimizations for improving OTP prediction rate. We do not report the results for the root sequence number history method since it shows only marginal improvement over the regular OTP prediction scheme.

8.0.1 Two-level and Context based predictions

We use a 4-bit range predictor for each memory block and there are 64 entries in the range prediction table (about 4KB size). The prediction rates of two-level prediction are shown in Figure 12 and Figure 13. We can see a significant improvement over the regular OTP prediction. The average prediction rate of two-level prediction is almost 96% with a 256KB L2 and 95% with 1MB. Comparing with results in Figure 7 and Figure 8, two-level OTP prediction using only 4KB range prediction table actually outperforms both the 128KB and 512KB sequence number cache settings. However, the best prediction performance is achieved by context-based prediction. For nearly all benchmark programs, context-based prediction attains almost perfect prediction rate.

The prediction rate using a large L2 is often smaller than the prediction rate using smaller L2 size. Since a large L2 typically reduces memory traffic, the number of predictions made with a larger L2 is usually far less than that of a smaller one. In terms of absolute number of miss predictions, larger L2 has less number of overall misses than smaller L2. Figure 14 shows the absolute number of predictions.

Figure 15 and Figure 16 show normalized IPC results for two-level and context-based predictions. Both two-level and context-based predictions achieve better performance over their regular counterpart. Using 256KB L2, for some benchmarks such as ammp, bzip2, twolf, and vpr, the improvement is about 7%. With 1MB L2, the improvement is about 4% for a number of benchmarks including applu, bzip2, mgrid, swim, twolf and vpr. For most benchmarks, context-based prediction outperforms two-level prediction. Note that context-based prediction also generates more predictions than two-level prediction. Considering the extra range table cost of two-level prediction, context-based prediction is a better choice because on average, it delivers better performance with far less hardware requirement.

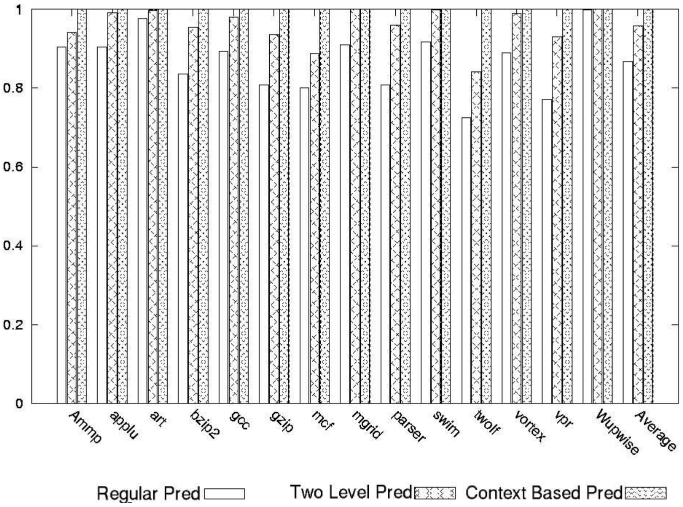


Figure 12: Hit Rate of Two-level Pred vs. Context-based Pred vs. Regular Pred, 256KB L2, 8 billion instructions

9. RELATED WORK

9.1 Sequence number caching

Caching sequence numbers or time stamps required for decryption was proposed in [20, 25]. In this paper, we have compared the difference between caching and our scheme. In contrast to caching, OTP prediction does not use any on-chip cache (either dedicated or shared) for storing sequence numbers of cache lines that might be used in the future. As we showed, OTP prediction has better performance with much less area overhead.

9.2 Memory Pre-decryption

Prefetch is a mature technique for latency hiding [5, 2, 22, 23]. When memory is encrypted, a prefetched cache line can be pre-decrypted as shown recently [17]. Prefetch and pre-decryption can cause cache pollution if the pre-decrypted data are stored in the regular caches. Furthermore, prefetch and pre-decryption can increase workload on the front side bus and memory controller if they become too aggressive. Different from pre-decryption, OTP prediction fetches only those lines absolutely required, thus no throttling on the bus. However, memory pre-decryption and OTP prediction are orthogonal techniques. A hybrid approach can be designed for further performance improvement.

9.3 Value prediction

Value prediction can be applied to tolerate memory latency [14, 24]. OTP prediction is different from value prediction because value prediction does not specifically address the issue of sequence number fetch on the critical path of memory decryption. Another major difference between

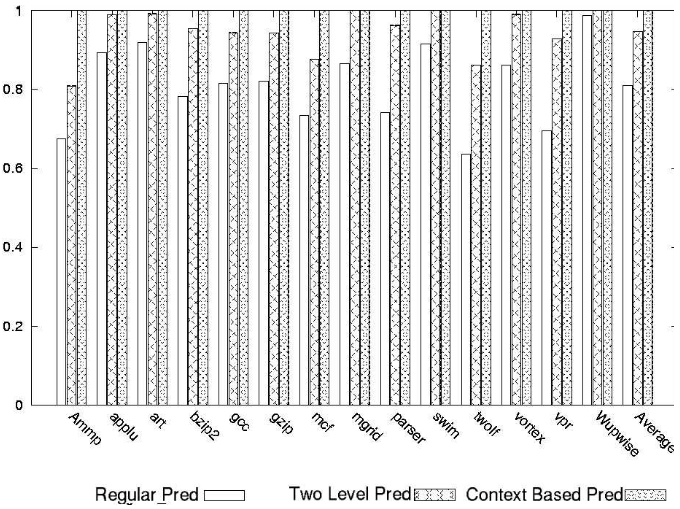


Figure 13: Hit Rate of Two-level Pred vs. Context-based Pred vs. Regular Pred, 1MB L2, 8 billion instructions

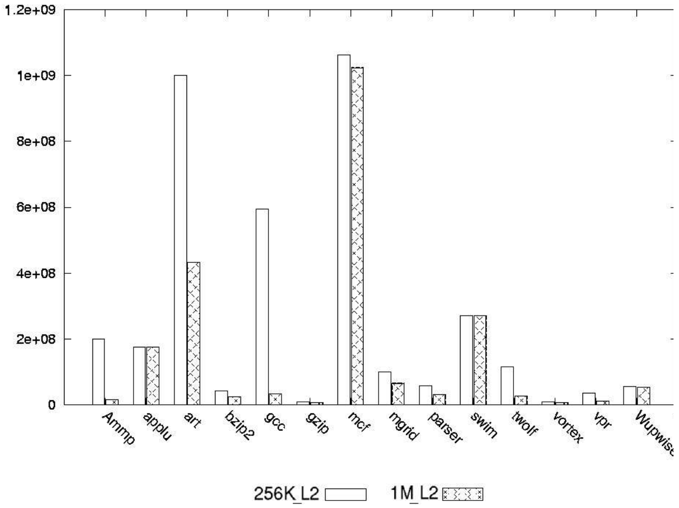


Figure 14: Number of Predictions under 256KB vs. 1MB L2

these two techniques is the source of predictability. *Sequence number predictability* mainly relies on the spatial and temporal coherence of the number of times cache lines are updated.

10. CONCLUSIONS

This paper introduces an OTP prediction and precomputation mechanism for hiding decryption latency in counter mode security architectures. In addition, we propose and evaluate several novel optimization techniques to further improve the performance of OTP prediction. The proposed adaptive OTP prediction improves performance of memory-bound applications significantly over prior sequence number caching method with a much smaller area overhead. Without any extra on-chip cache, our adaptive OTP prediction can achieve an average of 82% OTP prediction rate for both infrequently and frequently updated memory. For several memory-bound SPEC benchmark programs, the IPC improvement is in the range of 15% to 40%. In addition to the regular prediction technique, Two-level OTP prediction is proposed to further improve the prediction rate from 82% to 96%. Finally, we propose a context-based OTP prediction that performs even better than two-level prediction. It also outperforms the caching scheme with a very large 512KB sequence number cache. Two-level and context-based predictions can provide additional 7% IPC improvement on top of the regular OTP prediction. To summarize, with a minimal area overhead, our techniques succeed in achieving significant performance improvement for counter mode encrypted memory architectures. We envision that OTP prediction would help in making encrypted memory secure processors more practical in the future.

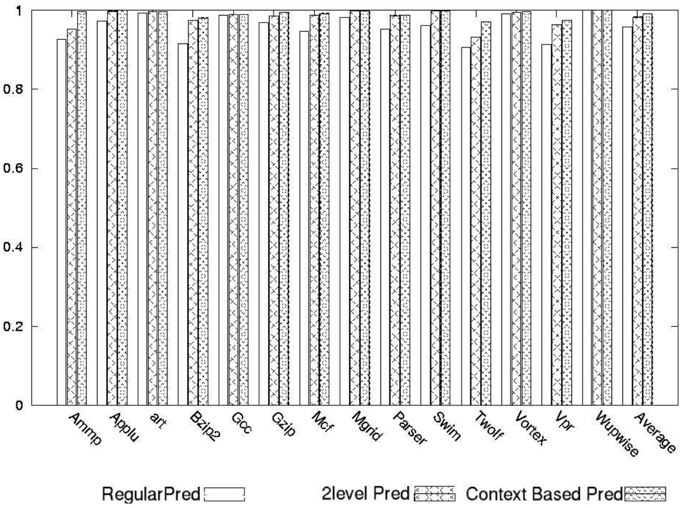


Figure 15: Normalized IPC of Two-level Pred vs. Context-based Pred vs. Regular Pred, 256KB L2

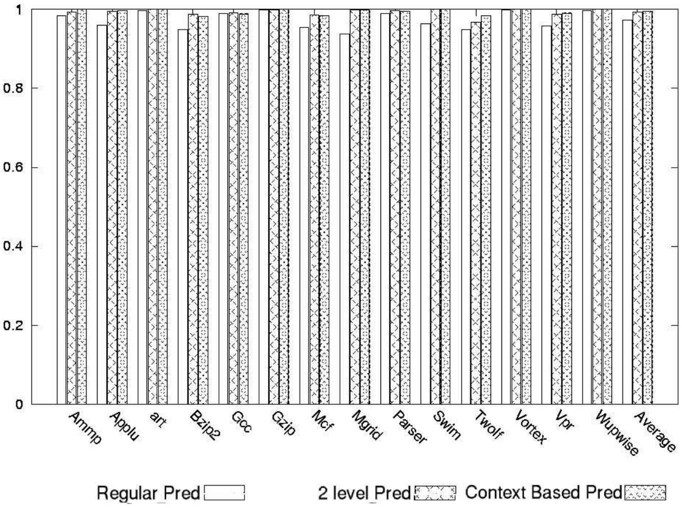


Figure 16: Normalized IPC of Two-level Pred vs. Context-based Pred vs. Regular Pred, 1MB L2

11. ACKNOWLEDGMENTS

This research was supported by NSF Grants CCF-0326396 and CNS-0325536.

12. REFERENCES

[1] W. A. Arbaugh, D. J. Farber, and J. M. Smith. A Secure and Reliable Bootstrap Architecture. In *Proceedings of the 1997 IEEE Symposium on Security and Privacy*, page 65. IEEE Computer Society, 1997.

[2] J.-L. Baer and T.-F. Chen. Effective Hardware-Based Data Prefetching for High-Performance Processors. *IEEE Transactions on Computers*, 44(5):609–623, 1995.

[3] M. Bellare, A. Desai, E. Jokipii, and P. Rogaway. A Concrete Security Treatment of Symmetric Encryption. In *Proceedings of the 38th Annual Symposium on Foundations of Computer Science*, page 394. IEEE Computer Society, 1997.

[4] M. Bellare and C. Namprempre. Authenticated Encryption: Relations among Notions and Analysis of the Generic Composition Paradigm. *In Advances in Cryptology — Asiacrypt 2000 Proceedings, Lecture Notes in Computer Science*, 1976, 2000.

[5] T.-F. Chen and J.-L. Baer. Reducing Memory Latency via Non-blocking and Prefetching Caches. In *Proceedings of the 5th International Conference on Architectural Support for Programming Languages and Operating System)*, volume 27, pages 51–61, New York, NY, 1992. ACM Press.

[6] C. Cowan, S. Beattie, J. Johansen, and P. Wagle. PointGuard$^{\mathrm{TM}}$: Protecting pointers from buffer overflow vulnerabilities. In *Proc. of the 12th USENIX Security Symposium*, Aug 2003.

[7] W. Diffie and M. Hellman. Privacy and Authentication: An Introduction to Cryptography. In *Proceedings of the IEEE, 67*, 1979.

[8] F. I. P. S. Draft. Advanced Encryption Standard (AES). National Institute of Standards and Technology, 2001.

[9] M. Gries and A. Romer. Performance Evaluation of Recent Dram Architectures for Embedded Systems. In *TIK Report Nr. 82, Computing Engineering and Networks Lab (TIK), Swiss Federal Institute of Technology (ETH) Zurich*, November 1999.

[10] A. Hodjat and I. Verbauwhede. Minimum Area Cost for a 30 to 70 Gbits/s AES Processor. In *IEEE Computer Society Annual Symposium on VLSI, pp. 498-502*.

[11] A. Hodjat and I. Verbauwhede. Speed-Area Trade-off for 10 to 100 Gbits/s. In *37th Asilomar Conference on Signals, Systems, and Computer, Nov. 2003*.

[12] D. Lie, C. Thekkath, M. Mitchell, P. Lincoln, D. B. J. Mitchell, and M. Horowitz. Architectual Support For Copy and Tamper Resistant Software . In *Proceedings of the 9th Symposium on Architectural Support for Programming Languages and Operating Systems*, 2000.

[13] D. Lie, C. A. Thekkath, and M. Horowitz. Implementing an Untrusted Operating System on Trusted Hardware. In *Proceedings of the 19th ACM Symposium on Operating Systems Principles*, pages 178–192, October, 2003.

[14] M. H. Lipasti, C. B. Wilkerson, and J. P. Shen. Value Locality and Load Value Prediction. In *Proceedings of the seventh international conference on Architectural support for programming languages and operating systems*, pages 138–147. ACM Press, 1996.

[15] H. Lipmaa, P. Rogaway, and D. Wagner. Comments to NIST Concerning AES-modes of Operations: CTR-mode Encryption. In *In Symmetric Key Block Cipher Modes of Operation Workshop, Baltimore, Maryland, US*, 2000.

[16] M. McLoone and J. V. McCanny. High Performance Single-Chip FPGA Rijndael Algorithm Implementations. In *Proceedings of the Third International Workshop on Cryptographic Hardware and Embedded Systems*, pages 65–76. Springer-Verlag, 2001.

[17] B. Rogers, Y. Solihin, and M. Prvulovic. Memory Predecryption: Hiding the Latency Overhead of Memory Encryption. *Workshop on Architectural Support for Security and Anti-Virus*, 2004.

[18] T. Sherwood, E. Perelman, G. Hamerly, and B. Calder. Automatically Characterizing Large Scale Program Behavior. In *Proceedings of the 10th Symposium on Architectural Support for Programming Languages and Operating Systems*, pages 45–57, Oct. 2002.

[19] W. Shi, H.-H. S. Lee, C. Lu, and M. Ghosh. Towards the Issues in Architectural Support for Protection of Software Execution. In *Workshop on Architectural support for Security and Anti-Virus*, pages 1–10, 2004.

[20] E. G. Suh, D. Clarke, B. Gassend, M. van Dijk, and S. Devadas. Efficient Memory Integrity Verification and Encryption for Secure Processors. In *Proceedings 0f the 36th Annual International Symposium on Microarchitecture*, December, 2003.

[21] E. G. Suh, D. Clarke, M. van Dijk, B. Gassend, and S.Devadas. AEGIS: Architecture for Tamper-Evident and Tamper-Resistant Processing . In *Proceedings of The Int'l Conference on Supercomputing*, 2003.

[22] S. P. Vanderwiel and D. J. Lilja. Data Prefetch Mechanisms. *ACM Computing Surveys*, 32(2):174–199, 2000.

[23] Z. Wang, D. Burger, K. S. McKinley, S. K. Reinhardt, and C. C. Weems. Guided Region Prefetching: A Cooperative Hardware/Software Approach. In *Proceedings of the 30th Annual International Symposium on Computer Architecture*, pages 388–398, June 2003.

[24] J. Yang and R. Gupta. Frequent Value Locality and its Applications. *ACM Transactions on Embedded Computing Systems*, 1(1):79–105, November 2002.

[25] J. Yang, Y. Zhang, and L. Gao. Fast Secure Processor for Inhibiting Software Piracty and Tampering. In *Proceedings of the 36th International Symposium on Microarchitecture*, December 2003.

[26] X. Zhang and R. Gupta. Hiding Program Slices for Software Security. In *Proceedings of the Internal Conference on Code Genration and Optimization*, pages 325–336, 2003.