Accelerating Memory Decryption and Authentication with Frequent Value Prediction

Weidong Shi
Motorola Application Research Lab
Motorola, Inc.
Schaumburg, IL 60196
larry.shi@motorola.com

Hsien-Hsin S. Lee
School of Electrical and Computer Engineering
Georgia Institute of Technology
Atlanta, GA 30332
leehs@gatech.edu

ABSTRACT

This paper presents a novel architectural technique to hide fetch latency overhead of hardware encrypted and authenticated memory. A number of recent secure processor designs have used memory block encryption and authentication to protect un-trusted external memory. However, the latency overhead of certain encryption modes or authentication schemes can be intolerably high. This paper proposes novel techniques called frequent value ciphertext speculation and frequent value MAC speculation that synergistically combine value prediction and the inherently pipelined cryptography hardware to address the issue of latency for memory decryption and authentication. Without sacrificing security, frequent value ciphertext speculation can speed up memory decryption or MAC integrity verification by speculatively encrypting predictable memory values and comparing the result ciphertext with the fetched ciphertext. In MAC speculation, a secure processor pre-computes MAC for speculated frequent values and compares the MAC result with the fetched MAC from memory. Using SPEC2000 benchmark suite and detailed architecture simulator, our results show that ciphertext speculation and MAC speculation can significantly improve performance for direct memory encryption modes based on only 8 most frequent 64-bit values. For eight benchmark programs, the speedup is over 10% and some benchmark programs achieve more than 20% speedup. For counter mode encrypted memory, MAC speculation can also significantly reduce the authentication overhead.

Categories and Subject Descriptors

C.4 [**Computer Systems Organization**]: Performance of Systems—*Reliability, Availability, and Serviceability*

General Terms

Design, Experimentation, Performance.

Keywords

Secure processors, message authentication, value prediction.

1. INTRODUCTION

There is a growing interest in creating a secure and tamper resistant software execution environment that combines the strength of hardware cryptography and secure systems to battle against attacks [13, 30, 29, 35, 36, 14]. Such tamper resistant systems [27, 28, 25] have a great future for solving various problems in the security domain such as tamper-proof digital rights protection, tamper-proof digital privacy, secure distributed computing, secure mobile agents, anti-reverse engineering, tamper-proof sensor devices, etc. One critical security measure provided by these systems is hardware-assisted cryptography that protects confidentiality and integrity of programs and data from both software and physical attacks. The strong tamper-proof protection provided by such secure processors appeals greatly to many applications such as military embedded systems, tamper-proof sensor devices, specialized computing platforms, to name a few. The research has inspired recent multiple commercial thrusts to design and build such tamper-proof processors.

An essential component shared by all the hardware-based tamper resistant systems is hardware protection of software secrecy and integrity. When a memory block containing data or instruction (e.g. a cache line) is brought into the secure processor, it is decrypted and verified. When a cache line is evicted from a secure processor, it is encrypted prior to being stored to any external memory. As addressed in [27], hardware cryptography is not only necessary for making systems difficult to reverse engineer but is also a critical component for content-based digital rights protection.

For protecting randomly accessed memory, several standard encryption modes are viable and some of them have been studied for secure processor design. Throughout this paper, any encryption mode that directly feeds plaintext memory blocks or derived bit string from plaintext to a block cipher in order to create encrypted memory blocks, are called *direct memory block encryption modes*. Well known encryption modes such as *Electronic Code Block (ECB mode)*, *Cipher Block Chaining (CBC mode)*, and *Offset Code Block (OCB)* belong to this category. CBC mode based secure processor design can be found in systems such as AEGIS [30]. In contrast with direct memory encryption, another well known encryption mode that uses a transaction number for encrypting memory blocks is *counter mode*. Counter mode based systems have been discussed in recent publications [35, 29, 26, 33]. Applying these modes involves splitting memory space into equally-sized blocks (often the same size as one L2 cache line) and encrypting each memory block separately to support random access of encrypted memory space.

One major overhead of hardware protection on software confidentiality and integrity is increased memory latency since memory blocks, once fetched from memory, need to be decrypted and authenticated before they can be used [1]. A number of techniques have been proposed such as employing a counter cache [35, 29], using prediction and pre-computation [26], or combined encryption/authentication scheme [33] to hide latency of *counter mode* encrypted memory. In this paper, we propose a novel *ciphertext* and *message authentication code* (MAC) speculation technique based on frequent value prediction to effectively hide the decryption and message authentication latency of encrypted static and dynamic data in memory. The technique is a synergistic approach combining *value prediction* and *speculative encryption* of predicted data values and speculative MAC computation to hide the decryption and authentication latency. The major contributions of this work are:

- Proposed an architectural framework to enable ciphertext speculation, a latency-hiding mechanism that combines value prediction and hardware cryptography for directly encrypted memory.
- Presented a scheme to speed up memory integrity verification by pre-computing of memory block's MAC value based on frequent value prediction.
- Provided further architecture enhancements to show how the proposed techniques can be efficiently applied to some popular encryption modes

The rest of the paper is organized as follows. Section 2 briefly discusses different modes of memory encryption and their applications in memory protection. Section 3 present *ciphertext speculation*. Performance evaluation and results are analyzed in Section 5 and Section 6, followed by Section 7 of related work. Finally, Section 8 concludes the paper.

2. BACKGROUND OF HARDWARE BASED MEMORY ENCRYPTION

This section discusses the role of memory encryption and integrity verification for secure processor design. It enumerates the desirable security features of memory encryption and authentication. It is important to point out that memory encryption is only one component of a secure processor. A hardware security model often requires other security architecture features to qualify as a tamper-resistant security system [13, 30, 29, 35, 36, 14]. There are also other assumptions on designing a secure system based on a secure processor and conditions on how to use a tamper-resistant system to protect static and dynamic information of a piece of software. The objective of this paper is not to propose another new security model, rather, we focus on architecture optimization and latency reduction. Our techniques are general and can be incorporated into any secure processor design that uses hardware crypto engine for encrypting and authenticating memory.

[1]Throughout this paper, we consider decryption and authentication latency part of the memory fetch latency.

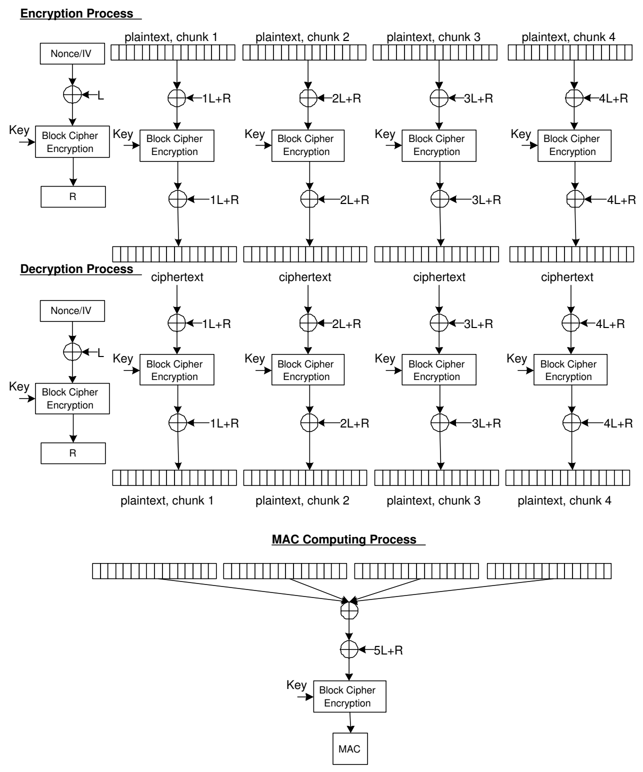


Figure 1: OCB Based Protection

2.1 Encryption Modes

Encryption modes define the mechanism to encrypt information using a particular encryption algorithm (or cipher). A cipher typically encrypts 64-bit or 128-bit data. Encryption modes specify how to encrypt much longer information such as a cache line (e.g., 32 bytes or 64 bytes) using a standard cipher. Electronic code block (ECB), Cipher block chaining (CBC), Offset code block (OCB), and counter mode are all encryption modes. Different encryption modes not only have different security characteristics but also demonstrate dramatic differences in performance under different hardware implementations. Table 1 compares several encryption modes from the perspective of performance. A parallelizable mode supports decryption of all words of a memory block in parallel. In addition to parallelizability, some modes such as counter mode also allows latency to be hidden by overlapping the memory decryption process with its own fetch.

In this paper, we mainly focus on the OCB mode although our technique is also easily applicable to other direct memory block encryption modes. The OCB mode was introduced in [22, 21]. Its strong achieved security level is proved in [23]. From security perspective, OCB provides higher protection than other standard modes such as ECB and CBC. Unlike CBC, OCB is non-malleable under *chosen-ciphertext attack*. OCB also belongs to the set of *authenticated encryption mode* that supports computing of a MAC with proven security. Figure 1 illustrates how OCB encrypts and decrypts a cache line with 4 data chunks and how the MAC is computed. For memory block encryption, we use a nonce consisting of the cache line virtual address concatenated with a 64-bit random initial value, and a 32-bit padding. As

Table 1: Comparison of Encryption Modes For Memory Encryption

Mode	ECB/OCB Mode	CBC Mode	Counter Mode
Parallelizability (decrypt memory block in parallel)	fully parallelizable	sequential decryption of memory blocks	fully parallelizable
Critical word first	yes	no	yes
Decryption latency hiding (overlap memory decryption with memory fetch)	no	no	yes

such, the weakness of having detectable ciphertext patterns can be avoided. This prevents an adversary from guessing encrypted data value based on encrypted values, in other words, frequently used values will not be encrypted into the same ciphertext. Similar to [30], the nonce associated with dynamic data blocks is incremented each time the corresponding dirty data block is evicted from the on-chip cache.[2] Also note that the L in Figure 1 is a secret pseudo-random bit string computed by encrypting a constant while R is a secret computed by encrypting the nonce [21, 22]. The aL+R inputs for different chunks are then computed using L and R to satisfy certain properties in number theory.

From the performance side, OCB achieves high level parallelism for decrypting/encrypting data chunks. Because the decryption/encryption of each data chunk is fully independent of other data chunks, OCB is fully parallelizable and suitable for high performance hardware implementation. In the rest of the paper, our direct memory block encryption mode is based on OCB. The block ciphers used in the OCB will be described in Section 3.2.1 and Section 3.2.2.

2.2 Integrity Verification

Integrity verification is a critical component of secure processor design. Integrity verification, achieved by employing message authentication codes (MAC) [17], guarantees detection of any unauthorized data modification. The MACs are stored along with each encrypted memory block such as a cache line. Similar to the case of encryption, there are many approaches and standards for generating a MAC, for example, HMAC [12], CBC-MAC [2], etc. For each dirty write-back, plaintext of the dirty cache line must be re-encrypted and stored with its updated MAC value. It is possible to conduct decryption and integrity verification concurrently. Some encryption mode called *authenticated encryption* went even further to combine decryption and integrity verification into one pass process. Examples include LAPM [11] and XCBC [7]. Authenticated decryption has less implementation and design complexity but makes decryption latency roughly the same as authentication latency and vise versa. In general, in secure processors, authentication takes a longer time than the decryption because in theory authentication can only be initiated after data is fetched from the memory. Due to the concerns of side-channel exploits on software confidentiality, close-coupling between integrity verification and memory decryption is required. In other words, to issue decrypted instructions or data to a processor pipeline without integrity verification may put code and data confidentiality at risk.

One misconception about integrity verification is that some designers consider it is secure by decoupling decryption and authentication so long as the secure processor does not retire the unauthenticated instructions. Such design prevents processor and memory states from being modified by unauthenticated instructions. It appears to be secure but in fact it does not prevent side-channel exploits of disclosing sensitive information through memory fetches because such fetches will be speculatively issued by a processor regardless of their authenticity. If program and data integrity is not verified promptly before those fetches are issued, an adversary may alter the program or data in such a way that sensitive data might be disclosed as fetch address [25, 28]. A secure processor can prevent such exploit by only issuing authenticated instructions and data.

[2]Note that hundreds of years are needed for 64-bit RV to wrap around with a 1GHz clock rate.

3. CIPHERTEXT AND MAC SPECULATION

In this section we explain how to apply value prediction to optimize the design of a secure processor.

3.1 Value Prediction

Previous studies have shown the predictability of (frequent) values among data fetched from external memory [15, 34]. For SPEC2000 benchmark suite, up to 80-90% fetched dynamic data are from a small set of application dependent data values (8 to 64 most frequent values). Memory profiling results in [34] also show that on average, about 40% of the data stored in the entire memory space of an application are frequent values. The high predictability of frequent data enables value prediction mechanisms for reducing latency overhead associated with direct encrypted memory. Here we propose *ciphertext speculation* and *MAC speculation* that effectively hide decryption and integrity verification latency by *speculatively encrypting* predictable values or pre-computing MACs for predictable values and match the results against fetched encrypted data block or MACs. In this study, we used a similar dynamic frequent value tracking mechanism proposed in [34]. In which, the most frequent values are dynamically determined. It can capture most of the frequently encountered values in-between working set changes or software context switches.

3.2 Ciphertext Speculation

The rationale of *ciphertext speculation* includes the following.

- The decryption latency of block cipher can be significant. It ranges from 50ns to a few hundreds of nsec depending on the cipher, design method, area constraint, and fabrication process [16, 9, 6, 5].
- The decryption of a direct encrypted memory block contains serialized operations composed of two parts: the *demand fetch* from memory and the *decryption* of encrypted memory block.
- A large portion of data stored in memory and fetched from memory are predictable [34].
- It is easier to increase and maximize the block cipher throughput than to reduce its latency [9].

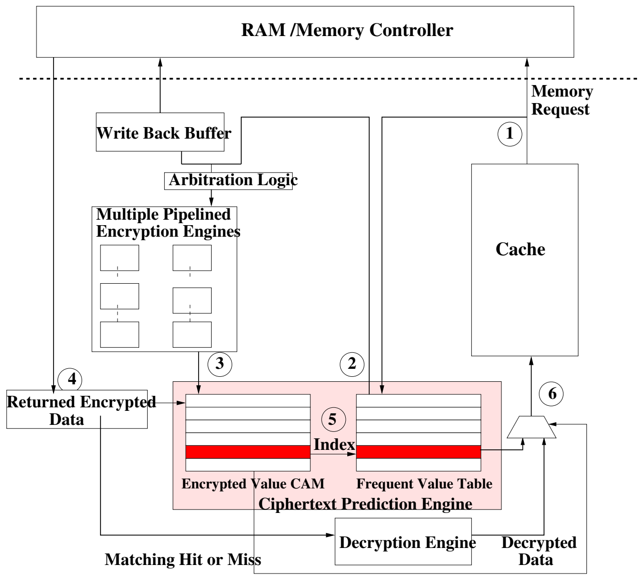


Figure 2: Ciphertext Speculation Mechanism

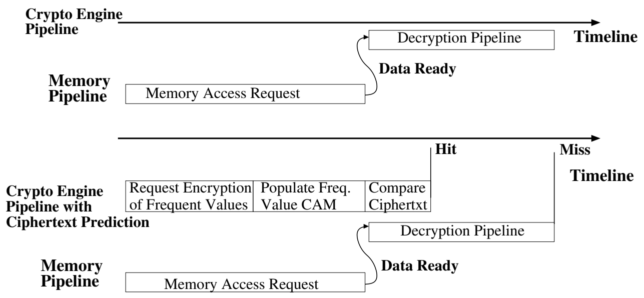


Figure 3: Timeline of Ciphertext Speculation

- For some block cipher such as AES, the encryption process is faster than decryption process.

Figure 2 illustrates the principle of *ciphertext speculation*. To enable prediction, a few new microarchitectural blocks are introduced — they are a Frequent Value Table (FVT), Pipelined Encryption Engines, and an Encrypted Value Content-Addressable Memory (CAM). The FVT keeps the top N most frequently used data values managed with an LRU policy. In our experiments, N is 8, 16 or 32. The ciphertext speculation mechanism is illustrated in the timeline given in Figure 3. When a cache line misses the on-die L2 cache and needs to be brought in from the memory, a fetch request is issued to the memory controller. At the same time, a request is also posted to the ciphertext speculation engine (① in Figure 2). The engine will read all the frequent values from the FVT, and send the data together with the fetch address to the encryption engine (②), which will encrypt them and store the resulting ciphertexts in the CAM (③). Note that the encryption engine is pipelined and replicated, thus multiple encryptions can be performed concurrently to accelerate the value speculation. When the missed *encrypted* cache line arrives, it is compared against the ciphertext waiting in the CAM (④). If a match is found, the original frequent value that corresponds to the matched ciphertext is returned and no decryption is needed (⑤ and ⑥). However, if there is no match (i.e., a misprediction), the secure processor will fill the missed cache line with its decrypted value from the fetched ciphertext. A secure processor can perform the ordinary decryption process and ciphertext matching concurrently. If there is a speculation hit, data value from FVT will take precedence over the decryption process. Note that one of the reasons to re-encrypt the frequent values for each missed cache line instead of using a simple *encrypted frequent value table* is that the ciphertext for each cache line size memory block is unique. Our encrypted ciphertext takes its memory address and a nonce into account, rather than simply encrypts the frequent values themselves.

Observing the timeline we should note that increasing the size of the FVT does not necessarily lead to a better performance. Even though more hits in the CAM will occur when more frequent values are kept inside the FVT, however, it needs more time to encrypt these values, thus delaying the value speculation of subsequent misses when there is a burst of back-to-back L2 misses. Ciphertext prediction applies to memory encrypted using parallelizable encryption modes such as the one proposed by XOM [13], in which the decryption of any word in a memory block is independent of the decryption of its neighbor words.

In the following sections, we consider two popular block cipher algorithms and discuss how to apply our *ciphertext speculation* to them.

3.2.1 Triple-DES/DES

The Data Encryption Standard (DES) is a symmetric encryption standard [20]. It encrypts a block of 64-bit message using a 56-bit secret key. In practice, a multiple encryption scheme based on DES, called Triple-DES is widely used [19] for its improved security. Triple-DES applies DES three times to achieve a better effective key length. The security of multiple encryptions is proved in [18] and Triple-DES is the NIST approved replacement for DES in 1999. Applying *ciphertext speculation* to Triple-DES/DES is straightforward because the prediction can be made on each 64-bit data block, the same size of Triple-DES/DES' blocks. For each 64-byte cache line, the line is divided into 8 64-bit chunks and each chunk is encrypted using parallelizable encryption modes with Triple-DES as the block cipher. The granularity of value prediction is made for every fetched 64-bit data. So, the size of each predicted ciphertext is also 64-bit. With a pipelined design, all the 8 chunks of the same cache line can be encrypted or decrypted simultaneously.

3.2.2 AES

Advanced Encryption Standard (AES) encrypts a 128-bit block with variable key lengths of 128-bit, 196-bit, or 256-bit. Since value prediction is performed for every 64-bit data[3], to predict each 128-bit ciphertext, combination of frequent values has to be used. For example, if the number of 64-bit frequent/predictable values used is 8, then there are 64 possible combinations of the eight frequent values, which gives 64 128-bit ciphertext speculations. Although such a prediction scheme may work, the prediction rate will drop

[3]Frequent value can also be dynamically tracked for 16 or 32-bit or even 128-bit values. But 64-bit is close to the block size of AES encryption (128-bit). Profiling results show that using 128-bit as units of frequent values generate poor frequent value profile.

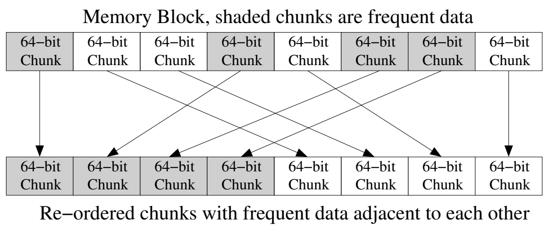

Memory Block, shaded chunks are frequent data
64−bit Chunk
Re−ordered chunks with frequent data adjacent to each other

Figure 4: Data Chunk Re-ordering

because it requires both 64-bit values of each 128-bit chunk to be predictable frequent values. To solve this problem, we propose a data chunk re-ordering mechanism that re-orders the eight data chunks so that chunks of predictable values are grouped and encrypted together. Figure 4 illustrates how the data chunk re-ordering is performed for each cache line. In this example, there are eight 64-bit data chunks and four of them are frequent values (shaded boxes). If a prediction is made for every two chunks, none of the four 128-bit ciphertext can be predicted correctly. Nevertheless, if the chunks are re-ordered as shown below the original data block, two of the four 128-bit ciphertext will be predictable.

To restore decrypted chunks back to the original order, a secure processor has to maintain additional information. One choice is to store a bitmap of frequent values. For a 64-byte cache line, the bitmap requires 8 bits to encode whether any of the eight 64-bit chunks is a frequent value.

The bitmap associated with each memory block is encrypted also when stored in the external memory. Since the bitmap contains only 8 bits per cache line, so bitmaps of several memory blocks can be encrypted and stored together. Decrypted data chunks or correctly predicted data chunks cannot be used if the secure processor cannot determine their original order before the corresponding bitmap is decrypted. This performance problem can be tackled in two ways. First, a small cache can be applied to cache bitmaps in the secure processor. The overhead to cache bitmaps is very small. Considering a data TLB of 128 entries and 4KB page size, to cache the frequent value bitmaps for all the 128 pages requires an 8KB cache. In fact, instead of requiring more power or space, such frequent value bitmap cache may save power or space. Since frequent values have less entropy, a significant portion of a cache line may be turned off or put into a lower power drowsy state knowing that many of the chunks are frequent values. For example, given eight frequent values, if the frequent value bitmap indicates that all the chunks of a cache line are frequent values, it requires only 24 bits (3 bits x 8) instead of 512 bits (64 bits x 8) for storing the cache line. Second, bitmaps can be prefetched using simple hardware prefetch technique.

In summary, *ciphertext speculation* is practical under both AES based and Triple-DES based direct memory block encryption. For the AES based design, frequent value bitmap cache is more desirable for its better performance. Considering that frequent value bitmap is also a compression technique, introducing such bitmap cache may actually make the secure processor more power efficient.

3.3 MAC Speculation

Similar to ciphertext speculation, a secure processor can also speculatively compute MACs for frequent value chunks. This requires each cipher size chunk to have its own MAC. One performance advantage of assigning a MAC to each chunk is that it supports parallel authentication of each individual memory chunk. When combined with parallelizable encryption mode, a secure processor can decrypt and authenticate critical fetched chunk of a cache line first before its trailing chunks. As aforementioned, a simple and secure design is to have a secure processor to issue instructions and data only after they are authenticated. Such design will not have the risk of disclosing sensitive information due to maliciously altered instructions or data. To implement MAC speculation, a secure processor will along with each speculated ciphertext speculate its corresponding MAC value. The speculated MAC will be matched against the MAC fetched from the memory. If both the speculated ciphertext and the speculated MAC match with the fetched ciphertext and MAC, the secure processor knows that the fetched value is a frequent value and there is no additional decryption and MAC verification required. Given that a large percentage of fetched memory blocks contain frequent values, such technique can significantly reduce the decryption and authentication overhead.

3.4 Security Analysis

It is important to point out that the proposed prediction technique is completely an architectural optimization. It does not change the security strength of the underlying encryption modes and the MAC integrity check schemes, nor does it modify the original encryption modes and MAC schemes.

Some audience may have concerns that since there are so many frequent values in the applications, memory encryption may not be able to provide sufficient protection against statistical analysis of ciphertext. Proper application of encryption modes will even out the frequency distribution. In theory, once encrypted, the encrypted data will be evenly distributed, in other words, without any frequent value bias [4]. Thus frequent values in plaintext will not manifest itself in ciphertext. To further clarify the confusion, even though our technique is named *ciphertext speculation*, by no means it implies that the ciphertext or the plaintext is predictable by adversaries. Predictions are made only by a secure processor within the secure boundary because the secure processor has the complete knowledge of necessary information such as a secret key used for encryption to create ciphertext. Also note that we are not proposing any new security model or new secure processor design paradigm but simply provide architectural enhancement on top of the existing secure architectures to address the latency issues associated with memory decryption and authentication.

4. IMPLEMENTATION AND OVERHEAD

Implementation of crypto engines is relatively straightforward. Most commercial crypto engines support standard ciphers such as Triple-DES (TDES), AES and standard MAC schemes. The trade-offs between performance, area, and energy consumption have been studied. In general, when

[4]Memory address or offset unique to each cache line is used as part of the nonce when applying OCB.

comparing against a fully fledged processor implementation, the overhead of a crypto engine in terms of area and power consumption has been shown to be rather insignificant.

A pipelined implementation of TDES can achieve about 20Gbits/sec throughput using roughly 55K gates. Considering that the critical path of a single stage of TDES implemented as a customized design using dynamic logic [6], and a 0.18 μ process, is about 2nsec, the total latency would be about 96nsec. The throughput can be further increased significantly when multiple pipelines are deployed to exploit the concurrency.

The AES cipher can process data blocks of 128 bits by using key lengths of 128, 192 or 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. For high throughput and high speed hardware implementation, Rijndael is often unrolled and with each round pipelined into multiple pipeline stages (4-7) to achieve high decryption/encryption throughput [16, 10, 9]. As shown in [10], the total area of unrolled and pipelined AES is about 100K - 150K gates to achieve a throughout of 15-20Gbit/sec. Based on our own synthesized Verilog implementation, each decryption round of pipelined AES-Rijndael takes less than 5nsec using 0.18μ standard cell library and each encryption round takes even less time. The area overhead of encryption process is also small as it requires less number of gates. In this study, unless specified, the default latency is 65ns for the 256-bit AES.

Since implementation of a crypto engine occupies a very small area, the extra power consumption is also very low. According to both industry standard and our implementation, it is estimated that a crypto engine consumes about tens of mW when active.

5. SIMULATION FRAMEWORK

Our simulation framework is based on SimpleScalar [3] running SPEC2000 INT and FP Alpha binaries compiled with -O3 option. We implemented architectural support for dynamic frequent value tracking [34] and our *ciphertext speculation* scheme. We also integrated a more accurate DRAM model [8] to improve the system memory modeling, in which bank conflicts, page miss, row miss are modeled based on the PC SDRAM specification. The architectural parameters used for performance evaluation are listed in Table 2. Each benchmark program is fast-forwarded and simulated at representative places according to SimPoint [24] for 400M instructions in performance mode. During fast-forwarding, L1 cache, L2 cache, and frequent value tracking are simulated. We subset the simulations for those with high L2 misses and memory throughput requirements.

6. PERFORMANCE ANALYSIS

In this section we summarize and analyze performance results of our *ciphertext speculation* technique. The results are collected for two L2 cache settings, 256KB and 1MB. The choice of a smaller L2 cache is critical for our analysis, because protection on data confidentiality is not only deemed for high-end systems but commodity platforms as well. Note that a very large L2 size for SPEC2000 may be inappropriate in evaluating our technique since the entire working set of most SPEC2000 benchmarks can easily fit into a very large L2.

Parameters	Values
Frequency	1GHz
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, 64B line, WB cache, 256KB and 1MB
L1 Latency	1 cycle
L2 Latencies	4 cycles (256KB), 8 cycles (1MB)
I-TLB/D-TLB	both 4-way, 256 entries
Memory Bus	200MHz, 8B wide
Memory Latency	X-5-5-5 (core clocks) X depends on page status
CAS latency	20 mem bus clocks
Precharge latency (RP)	7 mem bus clocks
RAS-to-CAS (RCD) latency	7 mem bus clocks
Triple-DES latency	96ns, 48 stages
AES latency	65ns

Table 2: Processor model parameters

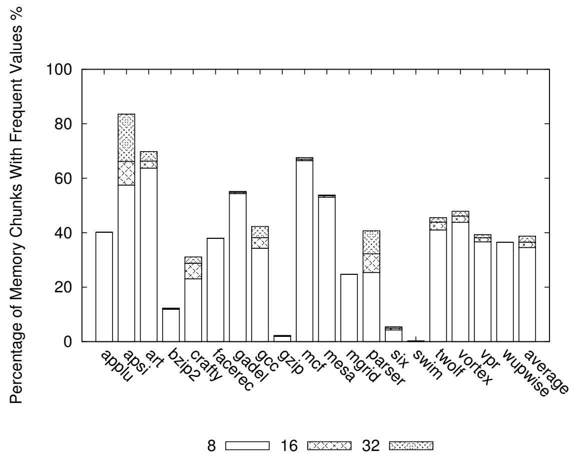
6.1 Frequent Values

Figure 5 shows the value predictability of each 64-bit memory chunks fetched due to L2 misses using 8, 16, and 32 frequent values. For several benchmark programs, the prediction rates are over 40% or higher. Some benchmark programs such as 164.gzip show poor prediction rates. Note that the relationship between ciphertext predictability and IPC is rather complicated and by no means proportional to ciphertext predictability. It is not guaranteed that every 64-bit data chunk fetched to L2 will be accessed because memory fetch is based on cache line size. For example, a piece of predictable data might be fetched into the processor because data adjacent to it mapped to the same cache line is accessed by the processor. Despite being predictable, the predictability of this un-used data may contribute little to the performance.

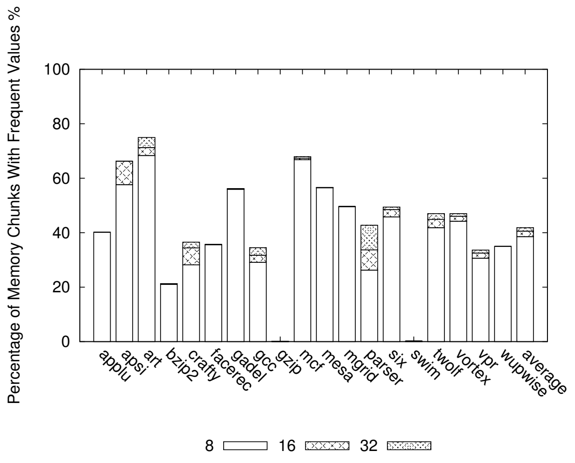
Figure 6 shows the advantages of using data chunk re-ordering for frequent value prediction. When a hardware uses a block cipher whose encryption size is 128-bit and one 64-bit frequent value is encrypted together with a 64-bit non-frequent value, the predictability is lost. Since a memory line often contains several frequent value data chunks, the hardware can re-arrange the data chunks so that frequent value data chunks are adjacent to each other. If a block cipher encrypts two 64-bit frequent value data chunks, both of them can be predicted. Figure 6 compares the ratios of predictable data chunks out of the total number of frequent value data chunks with and without the data chunk re-arrangement under 128-bit encryption unit size. According to the figures, without any re-arrangement, few 64-bit frequent value data chunks can be predicted under 128-bit block cipher. But when the re-arrangement is applied, over 95% 64-bit frequent value data chunks are predictable.

6.2 Ciphertext and MAC Speculation

Under parallelizable encryption modes, *ciphertext and MAC speculation* can be performed independently for each chunk of a cache line. In this section, we investigate the performance impact of *ciphertext and MAC speculation.* For TDES, a prediction is made on per 64-bit memory chunk of a cache line; for AES, each prediction is made on per 128-bit memory chunk using memory chunk re-ordering. We use a set of 8 64-bit frequent memory chunks dynamically tracked using an approach similar to that in [34].

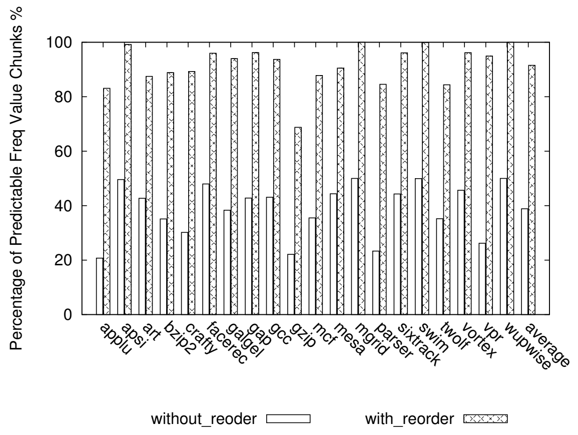


(a) 256KB L2

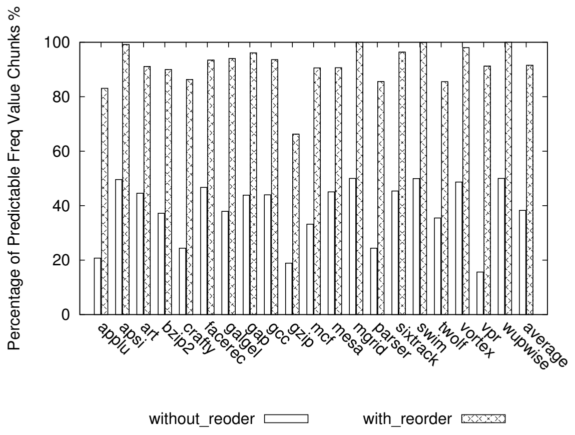


(b) 1MB L2

Figure 5: Percentage of frequent value memory chunks by keeping top 8, 16, 32 64-bit frequent values



(a) 256KB L2

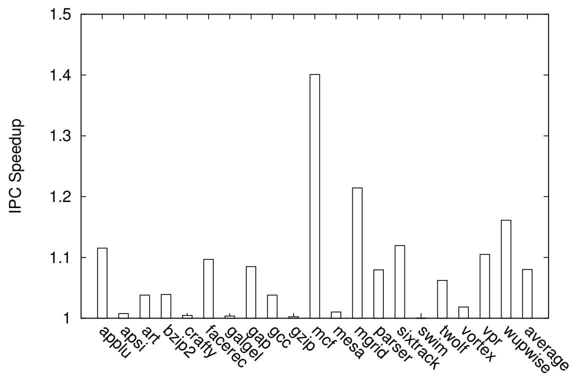


(b) 1MB L2

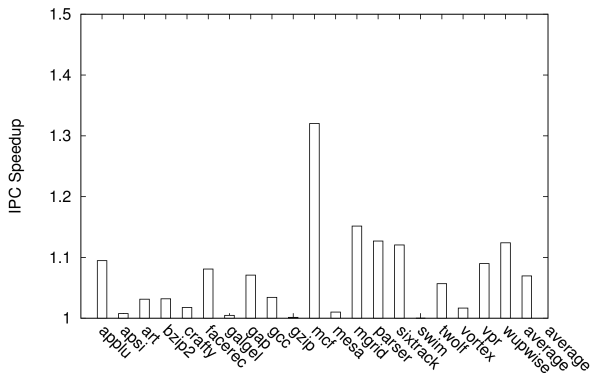
Figure 6: Percentage of Predictable Data Chunks Over All Frequent Value Data Chunks Under 128-bit Block Cipher

The performance improvement of *ciphertext and MAC speculation* is shown in Figure 7 for TDES and AES, both directly encrypted memory. Apparently, *frequent value based speculation* improves performance for most benchmarks for both ciphers. For some memory-bound applications such as 181.mcf, 175.vpr, 172.mgrid, the improvement is more significant. For eight benchmark programs, the IPC speedup's are over 10% for both TDES and AES. In general, using 64-bit speculation chunks achieves more performance improvement than the 128-bit ones. This is because 128-bit based speculation generates more speculations and is more likely to saturate the crypto engine with speculative encryption requests. When the L2 is increased to 1MB, the IPC improvement, as expected, decreases. But still for six benchmark programs, the speedup's are at least 10% and some of them attain almost 30%. Note that when the cache size is increased to 1MB, most of the memory bound SPEC2000 benchmark programs such as 181.mcf are less memory bandwidth limited, reducing memory traffic between the secure and insecure boundaries substantially

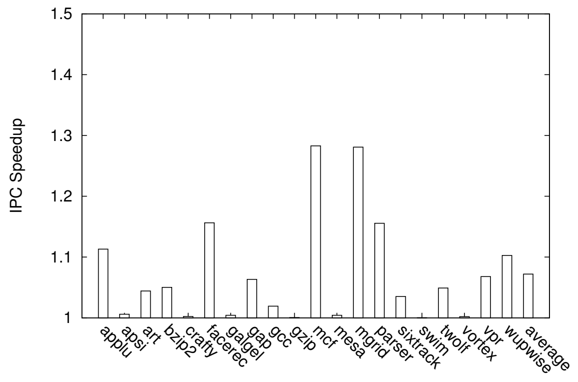
Figure 8 studies the performance impact of *MAC speculation* on memory encryption using counter mode encryption. Unlike direct encryption modes, counter mode receives less benefit from *ciphertext speculation*. That is because in counter mode, pre-computation of decryption pads is the dominating factor of decryption performance. But counter mode can still take advantage of frequent value based *MAC speculation* by significantly reducing authentication latency. The results indicate that many benchmark programs achieve performance increase of more than 5% and several of them have IPC speedup over 10%. With a 1MB L2, some benchmark programs such as 181.mcf, 172.mgrid, 197.parser also attain significant IPC speedup from 15% to 20%.



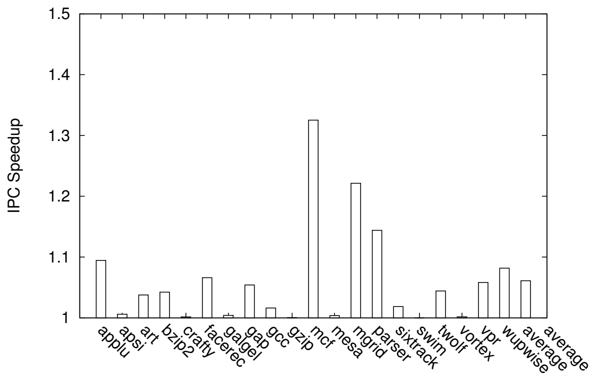
(a) TDES, 256KB L2



(b) AES, 256KB L2



(c) TDES, 1MB L2



(d) AES, 1MB L2

Figure 7: IPC Speedup Using Ciphertext and MAC Speculation For Direct Memory Encryption

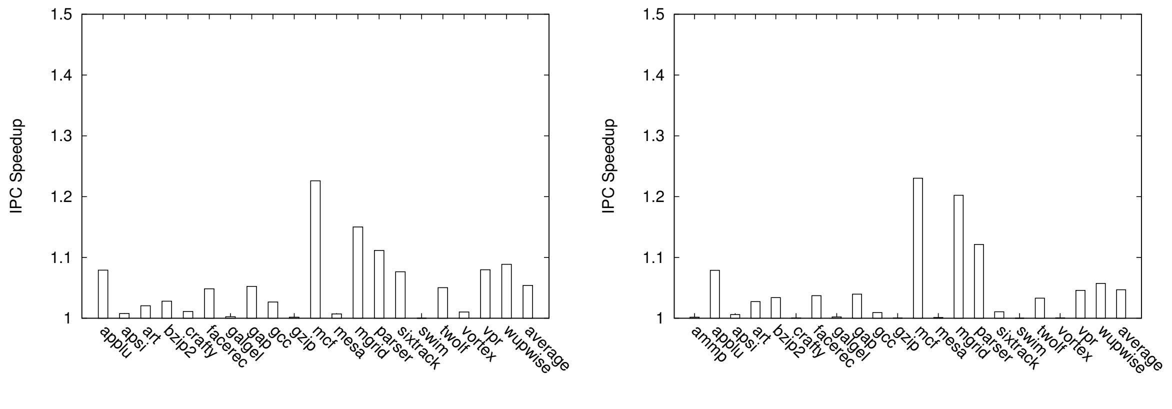
6.3 Sensitivity of Memory Latency

Essentially, frequent value based ciphertext and MAC speculation is a latency hiding technique. We now evaluate its effectiveness under different settings of memory speed. Since our simulation is based on a detailed SDRAM model, there is no single fixed number for memory fetch latency. The relative speed between CPU and memory is captured by the CPU-to-memory clock ratio. We experimented three different CPU memory clock ratio settings. They are 1:4, 1:5, and 1:6. Higher ratios infer longer memory fetch latencies. There are many factors that need to be considered when interpreting results obtained under different CPU memory clock ratios. Sometimes, it may not be meaningful to directly compare the speedup data under different CPU-memory ratios because they are computed using different baselines. The baseline IPC under 1:4 ratio is different from that under 1:6 ratio.

Figure 7 shows the result under 1:5 ratio. Figure 9 shows the impact of memory latency on the effectiveness of ciphertext and MAC speculation under 1:4 and 1:6 CPU-to-memory clock ratios using AES. First, the results indicate that ciphertext and MAC speculation is quite effective under all the three settings. For each benchmark, the differences of IPC improvement rates under the three settings are relatively insignificant. This shows robustness of the proposed frequent value based prediction technique. Second, the results show that on average ciphertext speculation improves IPC slightly more under larger CPU memory clock ratios. The average IPC speedup of all the benchmarks under 1:4 ratio is about 7.5% and the average IPC speedup of all the benchmarks under 1:6 ratio is close to 9%. In general, larger CPU-memory ratio leaves relative more time space for prediction and speculation. This may contribute to the slight improvement of speed-up under 1:6 ratio.

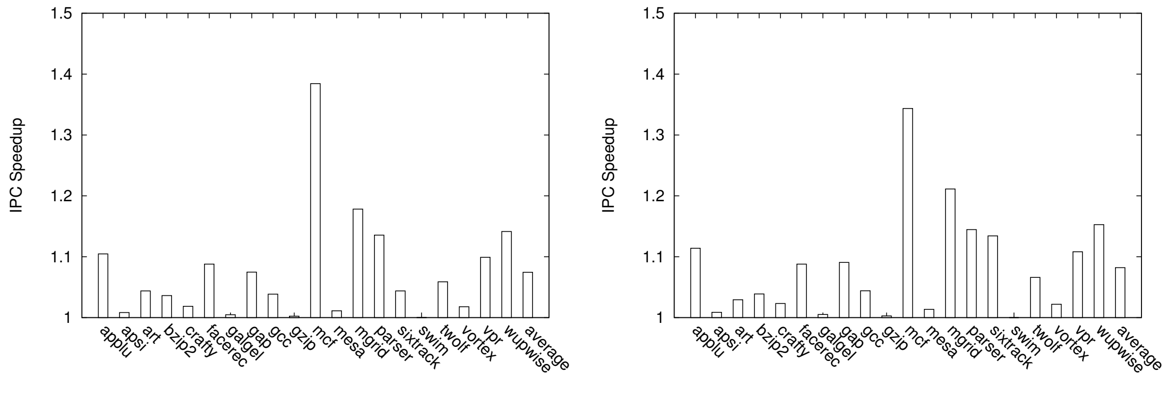
6.4 FVT Size

To evaluate performance sensitivity with respect to the size of the Frequent Value Table (FVT), we increase the number of frequent values kept, i.e., the number of entries inside the FVT, from 8 to 16 and 32. Ideally, more entries should produce more predictions of the missed ciphertext. As explained earlier, however, they can also throttle the performance due to much more encryption work that needs to be done for each L2 miss when a series of bursty L2 misses occurs. Figure 10 shows the IPC results for TDES encrypted



(a) 256KB L2　　(b) 1MB L2

Figure 8: IPC Speedup Using Ciphertext and MAC Speculation For Counter Mode



(a) CPU-to-Memory Ratio 1:4　　(b) CPU-to-Memory Ratio 1:6

Figure 9: Effect of Memory Speed Relative to CPU Clock Speed, 256KB L2 (AES)

memory. As indicated, there is little improvement with more entries in the FVT. In some cases, the performance was even reduced a little bit because the number of predictions is too large. The reason is that even though adding more predictions improves prediction rate, but at the cost of increasing workload on the speculative encryption engine, which potentially increases the latency of generating encrypted frequent values for the succeeding misses. For the AES-based scheme, the cost and workload to use more frequent values is higher than the TDES-based scheme because it uses combination of frequent values. Since increasing the number of frequent values and predictions does not have a significant performance advantage, we did not evaluate its effect on the AES-based scheme.

7. RELATED WORK

In this section, we address the difference of *ciphertext speculation* with other approaches on reducing fetch latency of encrypted memory.

Decryption pad prediction and precomputation was first proposed in [26]. It is a prediction technique in contrast with sequence number caching [29, 35] to facilitate pre-computing of decryption pads allowed by *counter mode* decryption. One advantage of prediction over sequence number caching is its efficiency in area. To address the latency issues for different security architecture, ciphertext speculation is designed when *directly encrypted memory* is employed. It predicts encrypted memory block (ciphertext) by virtue of frequent values. Unlike *counter mode*, modes of direct memory block encryption themselves do not support any pre-computation. One commonality of both techniques is that, both of them use idle cycles of pipelined crypto engines to speculatively compute either decryption pads or ciphertexts in order to hide decryption latency of encrypted memory. Table 3 summarizes their differences.

Prefetch was intensively studied for hiding memory latency [1, 4, 31, 32]. When memory is encrypted, a prefetched memory block can be pre-decrypted. There are some unique

Table 3: Compare counter prediction with ciphertext speculation

Technique	Supported Mode	What is predicted	Source of predictability
Counter prediction & Decryption pad precomputing	counter-mode	sequence numbers/pads	predictability of memory update frequency
Ciphertext prediction	direct memory block encryption	ciphtertext itself	frequent data chunk

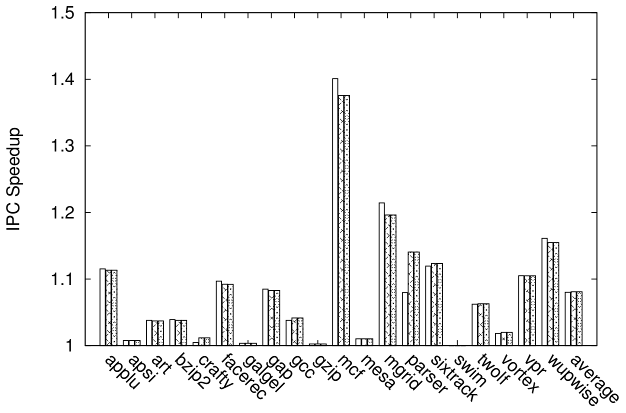


Figure 10: Effect of number of guesses/per chunk on performance with TDES encrypted memory, 256K L2

properties of *ciphertext speculation* when compared with pre-decryption. First, *ciphertext speculation* does not have the concern of potential cache pollution. Second, *ciphertext speculation* does not require significant additional bus throughput. Predictions are made inside the secure processor for memory blocks that must be fetched. A third difference is that *ciphtertext speculation* uses encryption process for prediction and prefetch/pre-decryption uses decryption process. *Ciphertext prediction* can take benefit of some encryption standard such as AES where encryption takes less time than decryption. Despite of the differences, pre-decryption and *ciphertext speculation* can be complementary techniques for latency hiding. It is possible to design a combined scheme that benefits from both approaches.

More recently, Yan et al. [33] described an enhanced counter mode based approach that provides smaller counter storage overhead using split counters and faster authentication operation using Galois/Counter Mode authentication (GCM). Different from their work, our frequent value prediction technique is mainly aimed at enhancing performance for direct memory encryption modes that by themselves cannot overlap the process of decryption key generation and authentication with memory fetch operation.

8. CONCLUSION

Minimizing the latency overhead of memory decryption and authentication is a crucial issue for designing a high performance secure processor. This paper proposes novel latency-hiding techniques — *frequent value ciphertext speculation* and *frequent value MAC speculation* to hide decryption and MAC authentication latency. *Ciphertext speculation* reduces or eliminates the decryption latency by speculatively encrypting frequent values and matching their ciphertext results with the fetched one. Our simulation profile indicates that on average over 40% fetched data values are frequent values. By exploiting these properties, the decryption latency for secure processors using direct memory encryption modes can be substantially reduced. We also propose *MAC speculation* which pre-computes MAC for frequent values and can accelerate authentication process by comparing the speculated MAC with the corresponding MAC fetched from memory. *MAC speculation* improves performance for all memory encryption schemes including counter mode security architecture that supports parallel MAC verification. As shown in our experiments, memory bound benchmark programs show significant IPC improvements using *ciphertext speculation* and *MAC speculation* with speedup ranging from 10% to 30%.

9. ACKNOWLEDGMENT

This research was supported by NSF Grant CNS-0325536, an NSF CAREER Award (CNS-644096), and a DOE Early CAREER PI Award.

10. REFERENCES

[1] J.-L. Baer and T.-F. Chen. Effective hardware-based data prefetching for high-performance processors. *IEEE Trans. Comput.*, 44(5):609–623, 1995.
[2] J. Black and P. Rogaway. Cbc macs for arbitrary-length messages: The three-key constructions. *J. Cryptol.*, 18(2):111–131, 2005.
[3] D. Burger and T. Austin. The simplescalar toolset, version 2.0. Technical Report 1342, University of Wisconsin, June 1997.
[4] 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 (ASPLOS)*, volume 27, pages 51–61, New York, NY, 1992. ACM Press.
[5] H. Eberle. A high-speed DES implementation for network applications. *Lecture Notes in Computer Science*, 740:521–539, 1993.
[6] S. Flgel, F. Grassert, M. Grothmann, M. Haase, P. Nimsch, H. Ploog, D. Timmermann, and A. Wassatsch. A design flow for 12.8 gbit/s triple des using dynamic logic and standard synthesis tools. *Synopsys User Group (SNUG) Europe, S. E3.2.*, pages 1–8, Mnchen, Mrz 2001.
[7] V. D. Gligor and P. Donescu. Fast Encryption and Authentication: XCBC Encryption and XECB Authentication Modes. In *FSE '01: Revised Papers from the 8th International Workshop on Fast Software Encryption*, pages 92–108, London, UK, 2002. Springer-Verlag.
[8] 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.

[9] 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 (ISVLSI '04), pp. 498-502.*

[10] 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.*

[11] C. S. Jutla. Encryption modes with almost free message integrity. In *EUROCRYPT '01: Proceedings of the International Conference on the Theory and Application of Cryptographic Techniques*, pages 529–544, London, UK, 2001. Springer-Verlag.

[12] H. Krawczyk, M. Bellare, and R. Canetti. HMAC: Keyed-hashing for message authentication. *Internet RFC 2104*, February, 1997.

[13] 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.

[14] 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. ACM Press, October, 2003.

[15] 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.

[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] A. J. Menezes, S. A. Vanstone, and P. C. V. Oorschot. *Handbook of Applied Cryptography.* CRC Press, Inc., Boca Raton, FL, USA, 1996.

[18] R. C. Merkle and M. E. Hellman. On the security of multiple encryption. *Commun. ACM*, 24(7):465–467, 1981.

[19] NIST. Recommendation for the Triple Data Encryption Algorithm (TDEA) Block Cipher. *SP-800-67, NIST, Mar.*, 2004.

[20] N. B. of Standards. Data encryption standard, fips-pub.46. *National Bureau of Standards, U.S. Department of Commerce, Washington D.C.*, January 1977.

[21] P. Rogaway. Proposal to nist for a block-cipher mode of operation which simultaneously provides privacy and authenticity.

[22] P. Rogaway. Ocb mode: Parallelizable authenticated encryption. In *P. Rogaway. OCB mode: Parallelizable authenticated encryption. presented in NIST's workshop on modes of operations, in October, 2000.*, 2000.

[23] P. Rogaway, M. Bellare, and J. Black. Ocb: A block-cipher mode of operation for efficient authenticated encryption. *ACM Transactions on Information System Security*, 6(3):365–403, 2003.

[24] 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.

[25] W. Shi and H.-H. S. Lee. Authentication Control Point and its Implications for Secure Processor Design. In *Proceedings of IEEE/ACM 39th International Symposium on Microarchitecture*, pages 103–112, 2006.

[26] W. Shi, H.-H. S. Lee, M. Ghosh, C. Lu, and A. Boldyreva. High Efficiency Counter Mode Security Architecture via Prediction and Precomputation. In *Proceedings of the 32nd Annual International Symposium on Computer Architecture*, June 2005.

[27] 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 in conjunction with the 11th International Conference on Architectural Support for Programming Languages and Operating Systems*, Boston, MA, 2004.

[28] W. Shi, H.-H. S. Lee, C. Lu, and M. Ghosh. Towards the issues in architectural support for protection of software execution. *SIGARCH Computer Architecture News*, 33(1):6–15, 2005.

[29] 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.

[30] 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.

[31] S. P. Vanderwiel and D. J. Lilja. Data prefetch mechanisms. *ACM Comput. Surv.*, 32(2):174–199, 2000.

[32] 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.

[33] C. Yan, B. Rogers, D. Englender, Y. Solihin, and M. Prvulovic. Improving Cost, Performance, and Security of Memory Encryption and Authentication. In *Proc. of the International Symposium on Computer Architecture*, 2006.

[34] J. Yang and R. Gupta. Frequent value locality and its applications. *Trans. on Embedded Computing Sys.*, 1(1):79–105, 2002.

[35] J. Yang, Y. Zhang, and L. Gao. Fast Secure Processor for Inhibiting Software Piracty and Tampering. In *36th Annual IEEE/ACM International Symposium on Microarchitecture*, December, 2003.

[36] X. Zhang and R. Gupta. Hiding program slices for software security. In *Proceedings of the 2003 Internal Conference on Code Genration and Optimization*, pages 325–336, 2003.