# LASE: Latency Aware Simple Encryption for Embedded Systems Security

*Kerry Courtright, Mohammad Iftekhar Husain and Ramalingam Sridhar,*

University at Buffalo, The State University of New York, Buffalo, NY, 14260

**Summary**
Security in the area of embedded systems has been drawing enormous attention lately. Although recent advances in hardware-based security models have shown promise for faster and more reliable security solutions, particular characteristics of embedded systems, such as computation and energy constraints, pose a unique challenge for the security researchers in this domain. Among hardware-based security models for traditional architectures, the Execute Only Memory (XOM) has used many different block ciphers to provide confidentiality as a part of the security module. However, implementations of many of these block ciphers in hardware do not meet the computation and energy constraints of embedded systems. In this paper, we propose to use a simple instruction encryption mechanism for embedded systems security based on the XOM model. As FPGAs provide reliable low-cost and high-performance implementations of cryptographic primitives, we implement our encryption algorithm on FPGA and compare with existing AES implementations for XOM-based models. The comparison has shown significant improvement on performance and power usage with a moderate level of security and hardware resources.

***Key words:***
*FPGA, Embedded Systems, Encryption, Security, AES, Execute Only Memory.*

## 1. Introduction

The pervasiveness of embedded systems has increased dramatically over the last decade. Embedded processors can be found in all kinds of applications, such as smart appliances, cell phones, GPS, navigation systems, cars, PDAs, personal media players, hand held games, and many more. The advancement of technology has allowed embedded systems to accomplish difficult and interesting tasks such as dealing with sensitive data and performing safety-critical tasks. These factors have heightened the need for increased security and dependability in the embedded systems domain.

Providing security for an embedded system is a difficult problem because of the usage environment, and the associated system constraints. The major factors in the design of embedded systems are cost, computational power, energy efficiency and real-time performance. The inclusion of extra security hardware increases a system's cost, as well as its power usage, and often directly affects the performance parameters of the system. Including security as a design requirement increases the difficulty of designing the system as a whole, due to an increase in competition for scarce resources. This difficulty is compounded by the fact that constrained environments do not enter into the design considerations of most security hardware and algorithms.

There has been a significant amount of research in the area of hardware-based security models for traditional computer architectures. The Execute Only Memory (XOM) model [1] and the AEGIS [2] architectures are at the forefront of such approaches and have been studied in depth. These secure architectures are designed to provide high levels of security, implementing such concepts as process separation, trusted computing, intellectual property protection, program and data integrity, and secure distributed computing. Implementations of these architectures often rely on symmetric encryption such as Advanced Encryption Standard (AES) for the encryption and integrity checking of memory operations.

However, existing encryption algorithms including AES are designed for traditional computer platforms, primarily focusing on providing the highest level of security for a given key size and encryption throughput. Issues relating to encryption latency, energy consumption, and real-time performance are treated secondary to these other security goals. In secure architectures, encryption often lies on the critical path, making encryption latency an important design consideration for maximizing computational and real-time performance. For embedded systems, energy consumption is also an important design consideration. These factors make many traditional encryption algorithms inappropriate for inclusion in secure architectures for the embedded systems domain.

In this paper we study the effect of encryption latency on performance for secure architectures that use memory encryption similar to the XOM model. We then propose a hardware-based simple encryption method that considers the constraints of embedded systems, for use with existing

Manuscript received October 5, 2009
Manuscript revised October 20, 2009

secure architectures such as XOM. Our encryption method is designed with FPGA implementation in mind, with its main operations performed by look up tables (LUTs) like those found in many FPGAs. We then compare our proposed encryption with implementations of the AES with respect to encryption latency, power usage, and implementation size. From this data we evaluate our algorithm as a possible replacement for AES in embedded secure architectures. This work will give embedded system designers insight into methods of better developing secure systems that fit in the constrained embedded systems domain.

In the next section we introduce current work in the area of secure architectures. Section 3 shows the effect of encryption latency on the overall performance of the architecture. It also shows the effect of cache performance compared to encryption latency. In Section 4, we introduce our encryption design and provide implementation details. Then in section 5, we compare the results of our encryption to other implementations of the AES algorithm. We then finish with our conclusions in Section 6.

## 2. Related Work

The XOM model is the most widely used model for developing secure architectures. The XOM model states that no program, not even the operating system, can be trusted for security [1]. When the operating system can't be trusted, the hardware must provide the trust for the system. In the XOM model the hardware is responsible for complete process separation. No process should be allowed to read or modify any of the private data or instructions of another process. The XOM model not only protects against other processes reading or modifying data, but also from hardware attacks to off-chip memory. XOM achieves this through encryption of the programs and data of each process. This creates a processor that greatly reduces the amount of information that is leaked about the programs and data that it processes.

AEGIS [2] is another secure architecture that follows along the same lines as XOM. AEGIS helped define the programming model of secure systems and explored the concept of how much trust can be put into software. XOM puts all the trust into the hardware, while AEGIS saves some hardware resources by having a small security kernel provide some of the security in software. AEGIS also exposed vulnerability in XOM's original memory integrity checking scheme [2]. Several other researchers [3, 4, 5] have worked on the improvement of the performance and quality of memory integrity checking for XOM type architectures. Both architectures provide methods to encrypt compiled code for a processor and to execute the encrypted code while protecting it from reverse engineering, tampering, and some additional attacks

The XOM and AEGIS models assume that the processor and all the signals and memory on it are secure from reading and modification. However, signals leaving the chip are vulnerable to attack. Because of this assumption, all the data and instructions that come onto or leave the chip boundary are encrypted. XOM originally used DES to encrypt memory transactions while AEGIS uses AES for memory encryption. Many XOM-type architectures now use AES or other well-known block ciphers. In both the XOM and AEGIS security model, data is encrypted when it leaves the physical boundary of the chip between the cache and the main memory. The data and instructions are decrypted when they are fetched from main memory and brought into the cache shows the general memory hierarchy for XOM and AEGIS systems. The addition of encryption to the memory hierarchy increases the latency of memory accesses and reduces the overall performance of the processor. AES encryption latency on the memory bus can cause an unacceptable performance reduction of 15%-20% or more for many programs [2].

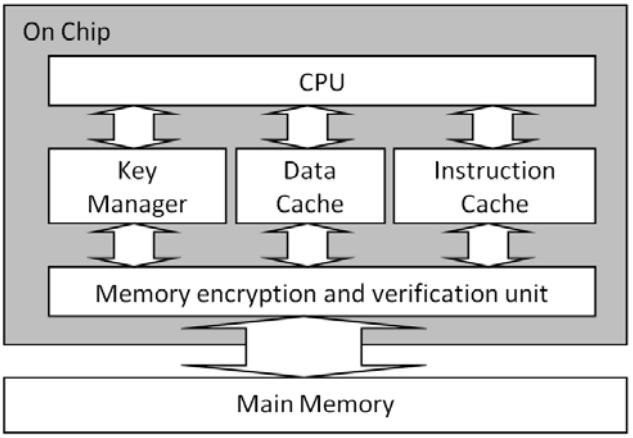


Figure 1: AEGIS & XOM Memory hierarchy

Zhuang *et al* [6] shows that information leaked on the address bus can be used to reverse engineer the running program. Zhuang *et al* [6] and Gao *et al* [7] address this problem through permuting the address bus, thus reducing the statistical correlation between memory accesses and the running program. Suh *et al* [5] and Yang *et al* [8] reduce the performance penalty imposed by long encryption times by taking the high latency of AES off of the critical path of the memory accesses. This is achieved by using a one-time pad encryption which is XOR-ed with the data. Lazy integrity checking was also used to reduce the performance degradation caused by their integrity checkers. Shi *et al* [9] shows that the onetime pad (OTP) encryption method

combined with lazy integrity checking can lead to the introduction of security vulnerabilities.

## 3. Latency and Performance

In [1] Lie *et al* shows the slowdown of a processor based on the XOM type architecture can be calculated with. This equation has shown to be accurate in calculating the slowdown of XOM type architectures when compared to experimental results. Using this formula, the effect of encryption latency can be shown for each of the separate parameters. .

$$\text{Slowdown} = 1 + \frac{S * \text{Encryption Latency}}{\text{Memory Latency}} \quad (1)$$

S = % of time stalled on instruction featchs

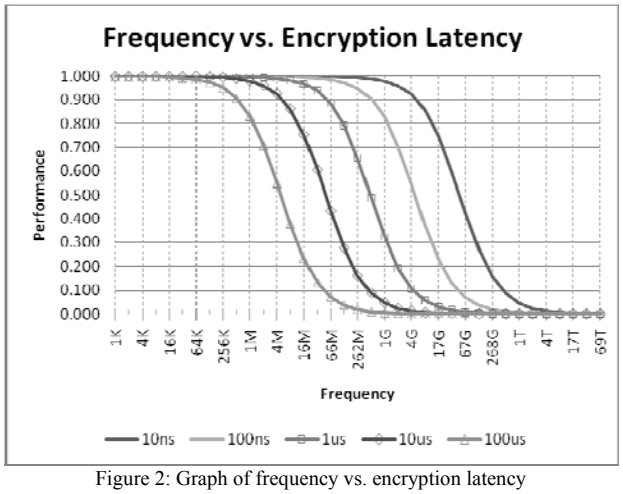


Figure 2: Graph of frequency vs. encryption latency

The Graph in shows how the latency of the encryption algorithm used affects the hardware performance over a range of clock frequencies. The graph assumes that the unmodified machine is stalled 20% of the time on off-chip memory accesses, and the memory latency is 100 clock cycles. This implies that the cache used stays the same and the memory speed is scaled with the processor speed. This graph can be used to help determine how much latency in the encryption algorithm can be tolerated while still achieving a given performance goal. At 1MHz the speedup for the selected encryption latencies of 10ns, 100ns, 1µs, 10µs, and 100µs are 99.80%, 99.80%, 99.60%, 97.85%, and 82.92% respectively. At this speed all encryption latencies of 10µs and below show excellent performance, with 100µs showing passable performance numbers. At 100MHz the speedup for the encryption latencies are 99.80%, 98.04%, 83.33%, 33.33%, and 4.76% respectively. The performance penalty for 100ns encryption latency or less is excellent, with 1µs encryption latency providing passable results. However, encryption latencies of 10µs and above are unfit for use at the given clock speed.

The graph in shows the effect of encryption latency on memory-bound processes. The performance degradation increases as the process becomes more memory-bound, causing memory-intensive programs to suffer. The percentage of time stalled for a memory fetch is determined by the overall on-chip cache miss rate, and the memory latency. The graph shows how the number of cache misses per instruction effect the performance of a 100MHz processor with a memory latency of 100 cycles. The miss rate of a cache is determined by the cache design and the memory access profile of the running program. The performance penalty increases dramatically with cache misses for encryption latencies that are inappropriate for a given clock frequency.

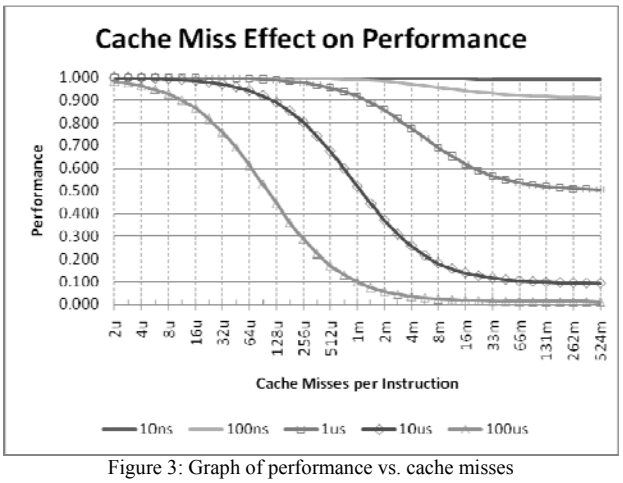


Figure 3: Graph of performance vs. cache misses

For a program exhibiting a 1% cache miss rate the performance for the selected encryption latencies of 10ns, 100ns, 1µs, 10µs, and 100µs are 99.50%, 95.24%, 66.67%, 16.67%, and 1.96% respectively. The performance values for a program with a .1% miss rate are 99.91%, 99.10%, 91.67%, 52.38%, and 9.91% respectively. For this example, encryption latencies in the range of 1µs to 100µs show a large performance difference depending on the cache miss rate. This is an undesirable effect, and encryption latencies should be chosen that reduce this effect.

This graph also shows that reducing the encryption latency in most usage scenarios has more effect than increasing cache performance. The 10ns encryption latency shows a worst-case performance around 99.00% for all programs. The 100ns encryption shows a worst-case performance of around 90%. To achieve the same performance as the 10ns encryption, the 100ns encryption cache design would have to have a miss rate per instruction

of less than .1% for all programs. This is not a feasible goal for many programs. Cache improvements often require drastic increases in both chip area and power usage, competing for already limited resources.

## 4. Latency Aware Simple Encryption (LASE)

The general encryption hardware that we have designed is based on look-up tables (LUT) similar to those found on a Field Programmable Gate Array (FPGA). shows the simple encryption hardware using a 128-bit message size. The first step is to split the message into 4 equal parts, in this case 32 bits. The 4 sub messages are then sent through a series of rounds consisting of a mixing stage and a rotate stage. The last round does not require a rotate stage. The mix stage performs a bitwise mix of the four inputs using a 4x4 LUT.

The Rotate stages rotate each of the 4 sub-messages by a different amount based on a constant multiple for the round. The multiple follows a pattern that is initially dependent on the round. In the initial rounds the multiple equals $4^{round-1}$. After $log_4(message\_bit\_length)-1$ rotations, each input has the possibility of affecting every output. If the sub-message bit length is an odd power of 2, the last rotation stage rotates by $(4^{round-1})/2$ bits. This is done to prevent two of the LUT outputs becoming inputs to the same LUT in the next round. The rotation pattern repeats if more rounds are desired.

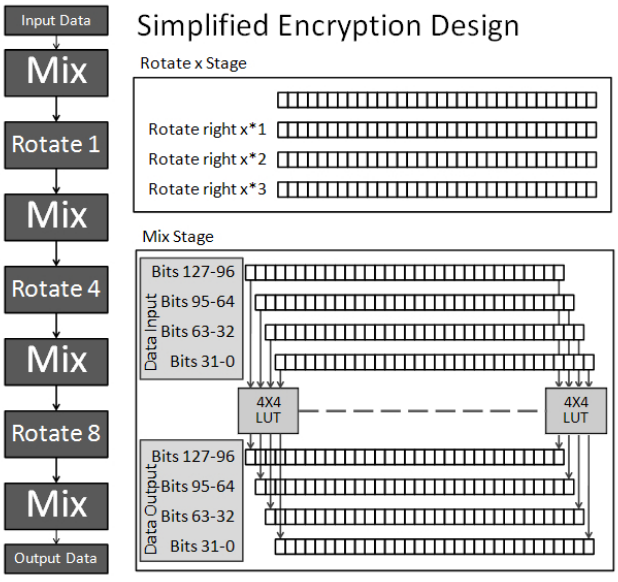


Figure 4: Example design of 128-bit LASE

### 4.1 Implementation of LASE

The encryption design was implemented and simulated using *Xilinx* design tools to determine its encryption latency, and power usage. A circuit was also designed to determine the number of gates it would require to implement the encryption algorithm on custom hardware. For the 128-bit block size, the encryption latency found when designing for a *Spartan 3e* FPGA using a 90nm process is less than 7ns. Varying the percentage of bit flips per cycle, the power was estimated to be between 50-150mW. The total area cost is 512 4-bit LUTs. The circuit that was also designed based on LUTs required 19,456 gates to implement. The encryption algorithm also requiring a total of 8192 bits of data to program the LUTs it uses.

A 32-bit block size was also simulated and found to have an encryption latency of approximately 5ns, power between 10-30mW, and a total of ninety six 4-bit LUTs. The 32-bit version consists of 3 mix rounds with four 8-bit sub-messages, and requires twenty four 4x4 LUTs. This requires 1536 bits of data to implement, and a hardware implementation of this 32-bit version would require 3,648 gates.

## 5. Comparison and Performance Evaluation

The major factors considered in designing embedded systems are cost, computational power, energy efficiency, and real time performance. The choice of encryption algorithm used by a XOM-type architecture directly affects all of these design constraints. The latency of the encryption directly affects the computational power and real time performance of the system. Its energy usage adds directly into the energy efficiency of the embedded system, and the encryption algorithms implementation size and complexity directly relates to its cost. In this section, we compare our encryption algorithm with published implementations of the AES algorithm, with respect to the silicon area used by the implementation, the latency, and the power usage.

Most AES implementations [10, 11, 12] focus on throughput and the implementation size of the hardware design over latency. Often, the latency can be calculated from the information given in the paper. To give an estimate of the power usage, we break the power into two categories: static power and dynamic power. The static power is estimated by the total number of gates used to implement the design. Given the same engineering process, the number of gates can be a good estimate of the relative static power usage between two designs. To estimate the total dynamic power per encryption operation, we calculate the total number of active gates per encryption. We

estimate that the number of active gates for each encryption cycle is 12.5% of the total gates. This was multiplied by the total number of cycles needed for each encryption. Table 1 summarizes the results.

The different implementations in the table use different feature sizes. The only parameter we compare that will change with scaling technology is that of latency. The comparison of our 90nm FPGA implementation should show comparable latency to that of a .11µm implementation, due to the reduced routing delays in a custom implementation over a FPGA. The other technologies in question will have lower latencies if scaled down to a .11µm process. However, the clock speed increase should not reduce the latency to a value close to our implementation.

Table 1: Comparison of AES and LASE

| clock cycles | Area (gates) | Max Freq. (MHz) | Throughput (Mbps) | latency (ns) | Estimated active gates /Encryption |
|---|---|---|---|---|---|
| 0.11µm [10] | | | | | |
| 54 | 5,398 | 131.24 | 311.09 | 412 | 36,437 |
| 54 | 10,338 | 222.22 | 526.74 | 243 | 69,782 |
| 44 | 6,292 | 137.55 | 400.15 | 320 | 34,606 |
| 44 | 10,990 | 219.30 | 637.96 | 201 | 60,445 |
| 32 | 7,998 | 137.17 | 548.68 | 234 | 31,992 |
| 32 | 14,777 | 218.82 | 875.28 | 187 | 59,108 |
| 22 | 8,836 | 137.17 | 798.08 | 161 | 24,299 |
| 22 | 17,016 | 217.86 | 1,267.55 | 101 | 46,794 |
| 11 | 12,454 | 145.35 | 1,691.35 | 76 | 17,124 |
| 11 | 21,337 | 224.22 | 2,609.11 | 50 | 29,338 |
| 0.35µm [11] | | | | | |
| 1 | 612,834 | 15.23 | 1,950.03 | 66 | 76,604 |
| 0.6µm [12] | | | | | |
| 92 | 8,541 | 50.00 | 70.00 | 1840 | 98,222 |
| 65 | 11,205 | 50.00 | 98.00 | 1300 | 91,041 |
| 35 | 15,850 | 50.00 | 183.00 | 700 | 69,344 |
| 90nm FPGA | | | | | |
| 1 | 19,456 | 142.86 | 18,285.71 | 7 | 2,432 |
| 1* | 38,912 | 142.86 | 18,285.71 | 7 | 4,864 |

When comparing LASE to the implementations of AES the biggest difference is in the latency and the active power requirement. The best implementation of AES has a latency of 7X more than LASE, and about 6X more active gates per encryption. With our encryption algorithm, both encryption and decryption can be done in parallel and still have less than 1/3 the number of active gates compared with any other implementation. The number of gates for our algorithm is large compared to the listed implementations of AES. This is not as big a problem because the design of the algorithm reduces the complexity of the circuit layout. The design also lends itself well to static-power-reducing hardware designs. In the worst case it is less than 8X the number of gates of any AES implementation, and compared to the fastest AES implementation it is less than 2X the number of gates.

The last thing that needs to be compared is security. AES has a long history of study, and as of yet for the 128-bit key size there have been no significant security breaks. This provides a lot of confidence in the AES algorithm. The encryption algorithm we describe has not gone through the rigorous scrutiny that the AES algorithm has endured. The 8192 bits of data used to program the LUTs can produce $O(16!^{128}) = O(2^{6144})$ unique permutations. The extremely large number of permutations our algorithm produces does have the advantage that even an attack that reduces the search space 16 orders of magnitude from its theoretical maximum still is more computationally intensive than breaking the AES algorithm.

## 6. LASE Design Discussion

The LASE design described in this paper give a general framework for the development of Low latency encryption hardware. LASE is a general substitution permutation network that is designed to be extensible and able to be tailored to the specific needs of the real-time embedded system. The block size and the number of rounds are not fixed for LASE. This allows system designers to tailor LASE to their specific requirements by exploiting trade-offs in power, size, latency, and security. Guidelines for increasing the effectiveness of LASE should be established (e.g. $number\_of\_rounds > log_4(message\_bit\_length)$).

The substitution permutation network used for LASE is incomplete without a key schedule for converting a secret key into the data needed to program the LUTs. Key schedules also can be developed to tailor the encryption to the specific needs of the system. Key schedules can be developed to address many specific issues including the following; tailor key length to security requirement, reduce the time of Key updates, reduce the difficulty and increase performance of software implementations, and reduce the possibility of weak keys.

The LASE hardware is designed to reveal the minimal amount of information about the key schedule used to program its LUTs as possible. Simple extensions to the general LASE hardware can be created that trades of some of LASEs general design to further customize the hardware to a given environment. Modification to the hardware to reduce the effective key size while maintaining as much of LASEs general nature and large permutation space is a work in progress. Extensions that allow encryption and decryption to be performed on the same hardware, as well as designs to reduce the number of LUTs used in designs with a large number of rounds are also being explored.

## 7. Conclusion

In this work we have shown the direct effect of encryption latency on performance for XOM-type architectures. We have also shown that increasing cache sizes to combat the added encryption latency is less practical a solution than reducing encryption latency. The encryption algorithm we have developed shows a very low latency compared to AES while occupying about the same silicon area as higher performing AES implementations. The lack of power usage data for low latency AES encryption algorithms has made it difficult to compare, so we estimate the power usage for both AES and our method. Our estimates, based on active gates per encryption, show our method has a large advantage in dynamic power usage. Detailed security analysis of our encryption design is still a work in progress. The development of key schedules for reducing weak keys and providing tighter bounds on security measures is also left for future research.

## References


[1] Lie, D., Thekkath, C., Mitchell, M., Lincoln, P., Boneh, D., Mitchell, J., and Horowitz, M. Architectural Support for Copy and Tamper Resistant Software. *SIGPLAN No.* 35, 11 (Nov. 2000), 168-177.

[2] Suh, G. E., Clarke, D., Gassend, B., van Dijk, M., and Devadas, S. 2003. AEGIS: Architecture for Tamper-evident and Tamper-resistant Processing. In *Proceedings of the 17th Annual international Conference on Supercomputing* (San Francisco, CA, USA, June 23 - 26, 2003). ICS '03. ACM, New York, NY, 160-171.

[3] Yan, C.; Englender; D., Prvulovic, M.; Rogers, B.; Solihin, Y. "Improving Cost, Performance, and Security of Memory Encryption and Authentication," *33rd International Symposium on Computer Architecture*, pp. 179-190, 2006

[4] Lu, C., Zhang, T., Shi, W., and Lee, H. S. 2006. M-TREE: A High Efficiency Security Architecture for Protecting Integrity and Privacy of Software. *Journal of Parallel and Distributed Computing* 66, vol 66, Issue 9 (Sep. 2006), 1116-1128.

[5] Suh, G. E., Clarke, D., Gassend, B., van Dijk, M, and Devadas, S. Efficient Memory Integrity Verification and Encryption for Secure Processors. In *Proceedings of the 36th Annual IEEE/ACM International Symposium on Microarchitecture* (December 03 - 05, 2003). IEEE Computer Society, Washington, DC, 339.

[6] Zhuang, X., Zhang, T., and Pande, S. HIDE: An Infrastructure for Efficiently Protecting Information Leakage on the Address Bus. *Proceedings of the 11th International Conference on Architectural Support for Programming Languages and Operating systems*, Boston, MA, 2004 pp. 72-84.

[7] Gao, L., Yang, J., Chrobak, M., Zhang, Y., Nguyen, S., and Lee, H. S. A Low-cost Memory Remapping Scheme for Address Bus Protection. In *Proceedings of the 15th International Conference on Parallel Architectures and Compilation Techniques* (Seattle, Washington, USA, September 16 - 20, 2006). PACT '06. ACM, New York, NY, 74-83.

[8] Yang, J., Zhang, Y., and Gao, L. Fast Secure Processor for Inhibiting Software Piracy and Tampering. In *Proceedings of the 36th Annual IEEE/ACM International Symposium on Microarchitecture* (December 03 - 05, 2003). IEEE Computer Society, Washington, DC, 351.

[9] Shi, W., Lee, H. S., Lu, C., and Zhang, T. Attacks and Risk Analysis for Hardware supported Software Copy Protection Systems. In *Proceedings of the 4th ACM Workshop on Digital Rights Management* (Washington DC, USA, October 25 - 25, 2004). J. Feigenbaum, T. Sander, and M. Yung, Eds. DRM '04. ACM, New York, NY, pp. 54-62.

[10] Satoh, A., Morioka, S., Takano, K. and Seiji Munetoh. A Compact Rijndael Hardware Architecture with S-Box Optimization. *In Proceedings of the 7th International Conference on the Theory and Application of Cryptology and Information Security: Advances in Cryptology*, pp. 239-254

[11] Ichikawa, T., Kasuya, T., and Matsui, M. Hardware Evaluation of the AES Finalists. In Proceedings *Third Advanced Encryption Standard Candidate Conference*, pp. 279–285. NIST, April 2000.

[12] Dobbertin, H., Rijmen, V., Sowa A. (Eds.) Efficient AES Implementations on ASICs and FPGAs. *AES 2004*, LNCS 3373, pp. 98–112, 2005. Springer-Verlag Berlin Heidelberg 2005


**Kerry Courtright** is a Ph.D. candidate in Computer Science and Engineering at the University at Buffalo. He graduated with a BS and MS in Computer Science and Engineering from the same University in 2002. His research interests are in Secure Architectures, Embedded Systems,High-Performance Architectures, Pervasive Computing, and Programming languages.

**Mohammad Iftekhar Husain** is currently in the University at Buffalo, The State University of New York Computer Science and Engineering doctoral program (Ph.D. expected 2011). He graduated with an MS in Computer Science and Engineering from the same school in 2008 and a BS in Computer Science from Yamagata University in Japan in 2006. His broad research interests are in the field of Wireless Network and Cloud Computing Security.

**Ramalingam Sridhar** received a B.E. (Honors) degree in Electrical and Electronics Engineering from Guindy Engineering College, University of Madras in 1980, MS and PhD in Electrical and Computer Engineering from Washington State University in 1983 and 1987 respectively. He is currently an Associate Professor in the Department of Computer Science and Engineering at University at Buffalo, The State University of New York. His research interests are in Wireless and sensor network security, pervasive and RFID systems, secure architectures, embedded technologies, deep submicron VLSI systems, Clocking and Synchronization, and memory circuits & architecture. He was an IEEE CAS Distinguished Lecturer. He has served as Program Chair and General Chair of ASIC/SoC Conference and has served in the editorial boards of many journals and technical committee of numerous conferences in wireless and embedded systems, security and VLSI.