Optimizations in the Design of Cryptographic Hardware

M. H. Sinky

Abstract—**The architectures that are used in the increasingly popular area of cryptography are discussed. In the design of cryptographic systems optimization in speed and area are important for surpassing acceptable standards. Many methods have been explored in the literature with regard to optimizations in speed. Although area is not as critical as the aforementioned constraint, it is closely related to the speed of the hardware. Area becomes increasingly important when implementing embedded systems. We have proposed a design that takes into account both area and speed resulting in a system that achieves results surpassing today's standards and may set the standard for years to come. The CryptoKnight system has considerably outperformed conventional cryptographic systems presented in various literatures.**

I. Introduction

As cryptography becomes more significant in contemporary systems and more wide-spread among users in the general public, there arises a need for performance improvements in the hardware dedicated to cryptographic systems. Faster and smaller are two desirable characteristics in today's technology. Although speed has reached a more than acceptable level within some systems such as the SNL DES ASIC [1] which has the capability of achieving throughputs of up to 6.7 Billion bits per second, we will need to continue finding ways to improve this as security is taken to higher levels (increasing of bit strength) in order to provide stronger protection.

A. Design

Much work has been done in the area of cryptographic hardware and numerous design approaches have taken place. W. P. Choi and L. M. Cheng [2] model a cryptoprocessor implementing the DES algorithm in 3 forms in a Xilinx FPGA chip. Their results have demonstrated that modular design leads to better optimizations in the space and speed requirements. In a comparison between a partitioned model and an un-partitioned model, they show that the partitioned model is better by 60% in area and 80% faster in terms of speed.

The approach taken by the authors was a step-by-step implementation of the Rapid Prototyping of Application Specific Signal Processors (RASSP). This is a design model that exploits top-down design, re-use and model-year design. Along with the advantages mentioned earlier, this concept of design leads to "shorter time-to-market and first-time silicon success fabrication" [2], both important industrial goals.

RASSP consists of 3 main phases: (1) Design Specification, (2) Executable Specification, and (3) Detailed Design. For more information about this design process the reader may consult [2].

One of the benefits stemming from the re-use concept of the RASSP design methodology is that it greatly simplifies the modification process applied to cryptographic algorithms. The authors use as an example the DES algorithm modified to RDES and EDES. RDES is known as Randomized-DES where the swapping of the left and right halves of the output at each round is done randomly. Either the two halves are swapped or not depending on a randomly generated bit S_n:

$$(R_{n+1}) = \begin{cases} (R_{nL}, R_{nR}) & \text{if } S_n = 0 \\ (R_{nR}, R_{nL}) & \text{if } S_n = 1 \end{cases}$$

$$\text{where } S_n = G_0(R_n)$$

EDES or Extended-DES is the same DES algorithm extended to a 96-bit plaintext and a 128-bit key size. In this algorithm the order of the S-boxes are randomly arranged and the key scheduling is operated on two keys resulting from the division of the 128-bit key.

As far as RDES is concerned, all that needs to be done is to introduce a "swap" module that performs the actions described above. The rest of the system remains unchanged. For EDES the modification that needs to be performed is to increase the data block and key lengths appropriately, and make the necessary arrangements for the order of the S-boxes. This requires redesign of the top-level structure. Changes in functionality need not be performed as it is only an extension of the actual DES algorithm. It is clear that the modular and re-use design principles play a big role in the modification processes.

B. Custom Architectures

Implementation of cryptographic hardware can be customized to execute a specific algorithm. This generally offers an architecture capable of high-speed execution when optimized. A survey put forth by Sandia National Labs (SNL) showed that previous implementations of DES were unable to cross the 0.5 Gbps threshold [1]. Current high-speed networks are in the range of 1-10 Gbps, therefore previous implementations were not suitable for network packet encryption. SNL researchers designed an ASIC (Application Specific Integrated Circuit) tailored to network security, using the DES architecture.

The SNL DES ASIC, which was over 10 times faster than other available DES chips at the time, is a high-speed,

Author is with the Department of Electrical & Computer Engineering, Oregon State University, Corvallis, Oregon 97331. E-mail: sinky@ece.orst.edu

This work is supported by ECE 679.

fully pipelined implementation providing encryption, decryption, unique key input, or algorithm bypassing on each clock cycle. This allows the options of encrypting, decrypting or allowing the data to pass through the ASIC with no modification at each clock cycle.

One strong feature of the SNL DES ASIC is that it is capable of encrypting data with one key on one clock cycle, and encrypting new data with a different key on the next clock cycle. The design can also pass user-defined control bits in syncrhonism with the data being encrypted, decrypted, or bypassed which is critical when dealing with systems that need to flag data as "valid" or "not valid" in the encryption/decryption pipeline.

Performance of the SNL DES ASIC revealed that the chip was able to reach 6.7 Gbps operating on 64-bit words in ECB (Electronic Codebook) mode. Increased performance can be achieved, as the authors pointed out, by use of improved design tools, increasing pipeline stages, higher performance IO buffers, advancements in Gallium Arsenide (GaAs) technology, and use of multiple SNL DES ASICs.

The authors suggest that use of several ASICs in parallel would be very substantial in performance gains. Two DES ASICs operated using opposite clock phases could double the data throughput to greater than 13 Gbps. For use with Asynchronous Transfer Mode (ATM), 6 SNL DES ASICs could operate in parallel on 64-bit blocks (384-bit payload for ATM) obtaining 40 Gbps rates.

C. Flexible Architectures

As the design process is critical to the overall performance of crypto-systems and ASICs are well suited for implementation of high-speed devices, there are other factors involved that need to be considered. In the case of cryptographic hardware, architectures should be able to adapt to changing or flawed algorithms. Cryptographic methods are constantly under attack. As the cryptanalyst tries to exploit flaws, the cryptographer attempts to improve algorithms. Therefore, it is not unusual for weaknesses to arise within cryptographic algorithms creating the need to change or add to the implementation.

Taylor and Goldstein point out that custom hardware, although better performance-wise than software in running cryptographic algorithms, is not able to respond to such changes occurring in cryptography [3]. They emphasize the benefits of reconfigurable architectures in conjunction with cryptographic hardware implementations. Among these benefits are the ability to configure hardware to implement any algorithm as well as have the implemented algorithm highly customized. Another incentive for reconfigurable architectures is that they also present high performance.

The authors explain that the types of functions that work well with a reconfigurable fabric and provide significant benefits over customized hardware implementations contain specific characteristics. Among these, are that the function operates on bit-widths that are different from a general purpose processor's basic word size. Also, functions containing data dependencies that allow multiple function units to operate in parallel is another trait that leads to benefits when implemented in reconfigurable architectures. A third trait that lends itself to flexible hardwares is a function that can be pipelined. Several other properties are mentioned by Taylor and Goldstein [3] that make implementation of the desired function feasible with reconfigurable fabrics.

There are generally four possible ways in which fabrics are integrated into a system [3]. These are:

1. As an attached processor on the I/O or memory bus.
2. As a coprocessor.
3. As a functional unit on the main CPU.
4. As an on-chip-system (used for embedded computing systems).

The system architecture employed was that of a reconfigurable function unit (RFU) integrated into the processor. The authors used the PipeRench reconfigurable fabric developed at CMU to conduct their experiments and tests. This reconfigurable architecture has several important characteristics. The three most important characteristics of PipeRench in terms of cryptographic algorithms according to [3] are: it supports hardware virtualization, it is optimized to create pipelined datapaths for word-based computations, and it has zero apparent configuration time.

Hardware virtualization is similar to the concept of virtual memory in memory hierarchies. It allows configurations larger than the size of the physical fabric to be executed. The way this is done in PipeRench is the fabric and configurations are structured into pipeline stages, referred to as *stripes* by the authors. The stripes of a particular application are time multiplexed onto the physical stripes of the fabric. This provides the illusion of unlimited hardware resources.

There are N processing elements (PEs) in each stripe in PipeRench and each PE is composed of B identically configured 3 input look-up-tables (3-LUTs), P B-bit pass registers and some control logic.

One of the drawbacks of the version of PipeRench used by the authors is that it cannot perform large table lookups. Table lookups are generally used in place of S-boxes when implementing cryptographic algorithms.

The authors implemented 4 cryptographic algorithms using the PipeRench reconfigurable fabric in a 0.25 micron process, using 16 8-bit PEs, and 8 pass registers. The final chip uses 100mm^2 for 28 stripes and an on-chip cache with a capacity of 512 virtual stripes. The algorithms implemented were IDEA, Crypton, RC6, and Twofish.

IDEA

The IDEA algorithm makes use of 3 main cipher components: addition modulo 2^{16}, 16-bit XOR and 16x16 multiplication modulo $2^{16}+1$. PipeRench is able to accommodate the 64-bit input and output of IDEA. When implementing this algorithm in PipeRench the greatest point of optimization and improvement lies in the multiplication. Two ways of optimizing the multiplication operation in IDEA are by allowing the subkeys to be constants and by

Processor	Clock Speed	Clocks per Block	Throughput (MBytes/sec)
PipeRench (template)	100 MHz	6.3	126.6
PipeRench (compiler)	100 MHz	12	66.3
Pentium-II using MMX [21]	450 MHz	358	10.0
Pentium [23]	(scaled) 450 MHz	590	6.1
IDEACrypt Kernel [22]	100 MHz	3	90.0

TABLE I
COMPARISON OF IDEA IMPLEMENTATIONS.

using a redundant coding scheme. Taylor and Goldstein try both of these methods. When implementing the subkeys as constants, the compiler performs constant propagation reducing the number of partial products from 16 (general-purpose two-operand shift-and-add multipliers) to an average of 8 per multiplier. The redundant coding scheme implemented by transforming the shift-and-add multiplier into a constant canonical signed digit (CSD) multiplier yields further optimization. Table I is taken from [3] which compares template- and compiler-generated IDEA to optimized software implementations running on sate-of-the art processors at the time, and to custom VLSI designs. The citations within the table are the ones used by Taylor and Goldstein and can be located by checking [3].

Crypton

The Crypton cipher posed some problems when attempting to map to PipeRench. Although most components of the algorithm were easily transferrable, the non-linear S-box substitution was not easy to implement. In creating the S-boxes, they are either implemented as logic or as look-up tables. Both methods waste resources and occupy stripes in PipeRench. One round of Crypton uses 16 S-boxes where each S-box uses around 9 stripes. This amounts to nearly 150 stripes for a single round, resulting in a total of 1800 virtual stripes needed for the 12-round cipher. Size limitations on the storage and space for virtual stripes can place a heavy strain on PipeRench. The authors suggested that the application of Crypton can be re-pipelined to allow for re-use of the S-box in each round on all four 32-bit words cutting the virtual pipeline by factor of four. The resulting implementation still contains a large number of stripes but gives 24.8 MByte/sec of throughput compared with 18.46 MByte/sec on a 450 MHz Pentium Pro coded with in-line assembly.

Greater improvements are made on Crypton when placed on PipeRench+16 (original PipeRench with 1K bytes of extra memory and capability of 16 simultaneous reads) allowing the application to shrink, taking up a total of 288 stripes, and run faster at 87 MByte/sec.

RC6

The RC6 algorithm is easily implemented as a stream cipher. The speedup over a 200 MHz Pentium Pro however is not spectacular at 4.7x. The variable rotates in the algorithm require a big portion of hardware in order to perform the rotates in parallel. The multiplication in RC6 is does not contain any constants and therefore reduces the chances of optimization. Optimization that can be done comes from the fact that the result is only 32-bits, which allows the size of the multiplier to be cut in half.

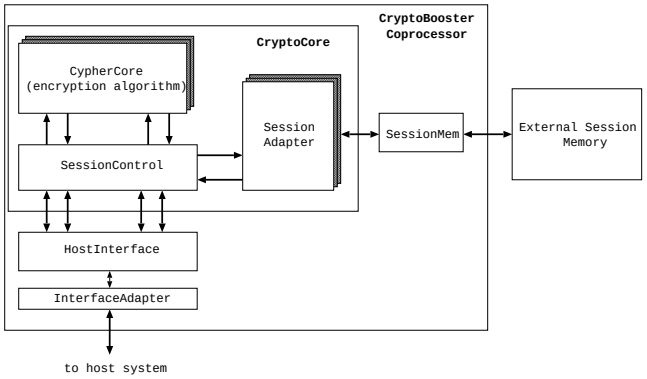

Fig. 1. Block diagram of CryptoBooster architecture.

Twofish

As in the case of Crypton, Twofish is difficult to implement due to its use of S-boxes. Parts of the algorithm can be mapped. The four table lookups of Twofish occupy much space but when using PipeRench to compute S-boxes the time for key setup is actually reduced. The authors used PipRench+4 and PipRench+16 (4 and 16 simultaneous reads, respectively) to implement Twofish. Each round of Twofish makes use of rotating, XORing, addition, and requires 16 loads. These are not a problem when realizing in PipeRench+. Results revealed that the hand-coded version of the algorithm in PipeRench+16 performed best at 164.7 MBytes/sec.

D. More on Flexible Architectures

As mentioned earlier, one of the methods of integrating cryptographic algorithms into systems is by use of a coprocessor. This system configuration was applied by [4] in their CryptoBooster system. The CryptoBooster is a modular and reconfigurable cryptographic coprocessor that takes full advantage of current high-performance reconfigurable circuits and their partial reconfigurability. The CryptoBooster is interfaced with a host system (typically a PC) and a session memory. The session memory is responsible for storing session information where a session is characterized by a set of parameters describing the cryptographic packets, the algorithm used, the key(s), the initial vector(s) for block chaining and other pertinent information.

The internal structure of the CryptoBooster is highly modular. It consists of the InterfaceAdapter module which takes care of the physical link to the host system, the HostInterface module which is the application layer, or software interface to the host and the SessionMem module which allows interfacing for different types and configurations of memory. Figure 1 illustrates the CryptoBooster architecture.

As can be seen from Figure 1 the CryptoCore module is divided into 3 parts [4]:

- *CypherCore*: encryption algorithm,
- *SessionAdapter*: session parameter management,
- *SessionControl*: central controller for session management.

The modularized architecture provides the opportunity for partial reconfiguration of the coprocessor. It is only the CypherCore and SessionAdapter modules that need to be modified when downloading a new design or algorithm onto the FPGA. This feature of the CryptoBooster cuts down on the interruption of service time.

Mosanya *et. al* [4] explain how the IDEA block encryption algorithm works and discuss its implementation in several different hardware scenarios ranging from the custom VLSI implementation VINCI and Ascom's ASIC to some FPGA designs. Common between all designs is the fact that the algorithm is pipelined.

In the CryptoBooster, implementation of IDEA contains two main properties: pipeline scalability and an associated block chaining module. The encryption algorithm is located on the CypherCore module and labelled IDEACore [4].

The solution introduced by Mosanya *et al* [4] involves a highly scalable IDEA pipeline which allows for selection of the length of the pipeline to be chosen at compile time. The default length of the pipeline follows the VINCI datapath which is 7 pipeline stages long for one round of IDEA. The flexibility of the datapath allows the total pipeline length to be cut down to 4, 2, and 1 round. In the case of one round for example, since IDEA is 8 rounds, the data must be passed through the regular round 8 times before going through the output round. For the normal configuration of 8 rounds, data passes once through all 8 rounds and then to the output round.

The block-chaining aspect of IDEACore was a module that implemented the commonly used block-chaining algorithms (EBC, CBC, CFB and OFB). This adds another piece of flexibility to applying IDEA and would also be useful for other algorithms that use such block-chaining techniques.

Performance results of the IDEACore CypherCore reveal a peak performance of 200 Mbits/s for a 1-round pipeline which fits easily into state-of-the-art FPGAs. A full-length pipeline of 8-rounds has a throughput estimated at more than 1500 Mbits/s, however requires a large amount of area. Factors that affect the overall performance are session initialization, key calculation, and the type of block-chaining mode used. Another important factor that limits performance is the multiplication modulo $(2^{16}+1)$ which is crucial because of the area such an operation takes up and the combinatorial delay introduced. The authors, although they admit it is not the best way, used bit-parallel multipliers. This will be a point of optimization for their coming designs.

E. Public-Key optimizations in Cryptographic Hardware

Modern information technology such as e-commerce, private networks, digital signatures, and secure internet makes use of public-key cryptography (PKC) to provide security. As the world moves more and more towards electronic means in all aspects of society, PKC becomes very important. The brand of algorithms that implement PKC rely heavily on long integer multiplication. As computing power improves however, such algorithms become easier to break in terms of brute-force means. PKC (and any well-designed encryption cipher for that matter) is capable of avoiding such attacks by increasing the length of the modulus. This, however creates problems in that by increasing the modulus length, the encryption/decryption time of the algorithm follows the same trend. Modern security demands require that the modulus have a length of at least 1024 bits [5].

Großschädl [5] analyzes the optimizations that can be made in RSA's modular arithmetic and presents design considerations of the RSA_γ crypto chip.

The RSA_γ crypto chip is specifically designed for high-speed RSA encryption/decryption. The chip consists of an interface and control unit, multiplier core and I/O register. These constitute the main components of the system. The performance of the chip mainly relies on the efficiency of the modular arithmetic, therefore it is the multiplier core that is discussed further in more detail.

As [5] explains, two main algorithms are preferred for hardware implementation of modular multiplication: the Montgomery algorithm and the Barrett modular reduction method. RSA_γ uses an optimized version of Barrett's modular reduction method named *FastMM algorithm*. The way the reduction works is that it replaces the integer division (when computing a modulus) with a multiplication due to the fact that division is very costly.

$$Z \bmod N = \underbrace{\left\lfloor \frac{Z}{N} \right\rfloor}_{replaced} N = Z - qN \text{ with } q = \left\lfloor \frac{Z}{N} \right\rfloor \text{ [5]}$$

The replacement of q is $\tilde{q}$:

$$\tilde{q} = \left\lfloor \frac{\left\lfloor \frac{Z}{2^{n-1}} \right\rfloor \left\lfloor \frac{2^{2n}}{N} \right\rfloor}{2^{n+1}} \right\rfloor \text{ [5]}$$

As Großschädl points out the divisions by 2^{n-1} and 2^{n+1} are done by simply truncating the least significant $n-1$ or $n+1$ bits of the operands whereas the expression $\left\lfloor \frac{2^{2n}}{N} \right\rfloor$ only depends on mod N, which would remain constant as long as the modulus does not change. The modification performed to produce the *FastMM algorithm* was to have the truncations be applied at multiples of the word-size w of the multiplier hardware (usually 16 or 32 bits):

$$\tilde{q} = \left\lfloor \frac{\left\lfloor \frac{Z}{2^{n+w}} \right\rfloor \left\lfloor \frac{2^{2n-w}}{N} \right\rfloor}{2^{n+2w}} \right\rfloor \text{ [5]} \qquad (1)$$

This was to allow for more natural operations given the word size. Following the above equations, when modular reduction is performed, the result may not be fully reduced.

For this reason equation (1) is combined with three multiplications to implement the modular multiplication according to the formulas explained in [5] to obtain the final result although there may be some error introduced due to an inaccurate approximation of $\frac{2^{2n+w}}{N}$.

Großschädl further discusses the physical architecture of the multiplier used in implementing the *FastMM algorithm*. The RSA$_\gamma$ prototype is optimized for $n = 1024$, therefore the multiplier core has the dimensions $(n + 2w) * w = 1056 * 16$ bits. One of the more important optimizations implemented in the hardware is the use of Modified Booth Recoding which reduces the number of partial products required in computation in half.

Performance results of the RSA$_\gamma$ crypto chip with a modulus length of $n = 1024$ and a word-size $w = 16$ give a decryption rate of 560 kbit/s where the multiplier core is clocked at 200 MHz (227 clock cycles for $n = 1024$). Use of the Chinese Remainder Theorem speeds up the decryption to 2 Mbits/s. The area of the multiplier core occupies 70 mm^2 and contains around 10^6 transistors. Overall, performance results show the efficiency of the hardware algorithms implemented and the multiplier architecture.

II. The CryptoKnight System

The design of our system encapsulates what has been discussed in the previous section and extracts the advantages proposed in the preceding designs. The CryptoKnight system resides on an ASIC fabricated by the semiconductor group at Oregon State University. The die is 2×2 mm making it more than ideal for embedded systems. The chip applies the Rijndael algorithm using a key length of 128 bits (requiring 149 trillion years to break given a machine capable of cracking DES in one second). We chose Rijndael because it has just recently been selected as the standard of choice and will remain secure for at least the next 20 years given there is no breakthrough on cracking the algorithm.

A. The Algorithm

We start off with a brief overview of the Rijndael algorithm. Rijndael consists of 10 rounds where each round involves the implementation of specific steps known as *layers*. The four basic layers are:

1. The ByteSub (BS) Transformation.
2. The ShiftRow (SR) Transformation.
3. The MixColumn (MC) Transformation.
4. The AddRoundKey (ARK) Transformation.

Rijndael operates on the input bits by grouping them into 16 bytes and arranging those bytes into a 4×4 array. The BS transformation basically substitutes each byte in a matrix by another byte using an S-box for this process. The SR transformation shifts the rows of a matrix to the left (cyclically) by offsets of 0, 1, 2, and 3. The MC transformation takes a matrix and multiplies each column of the matrix by a fixed ploynomial within the same field and reduces by modulo $x^4 + 1$. This polynomial is

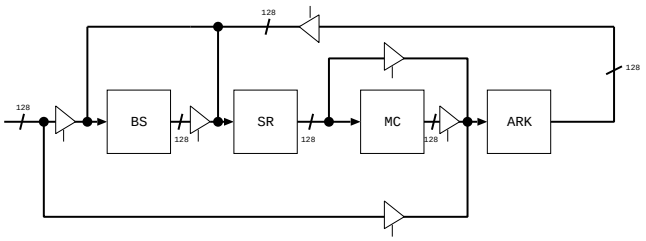

Fig. 2. Basic Modules within the CryptoKnight Chip.

$a(x) = \{03\}x^3 + \{01\}x^2 + \{01\}x + \{02\}$. Finally, the ARK transformation takes the round key (derived from the original key) and XORs it with the output of the MC step.

Key scheduling in Rijndael involves expanding the original 128 bit key by 40 more columns which are generated recursively based on the four columns of the key. These 40 extra columns are used to construct the keys for the last ten rounds of the algorithm. The key scheduling involves XORing, shifting, and computation of a round constant ($r(i) = 00000010^{(i-4)/4}$, for the ith round).

Rijndael Encryption makes use of the layers described above by using ARK for the zeroth round then BS, SR, MC, and ARK through rounds 1-9 (using the round key for each corresponding round). The final round goes through BS, SR, and ARK using the 10th round key.

Further details about Rijndael which is now the Advanced Encryption Standard can be found in [6].

B. Chip Design

The ASIC design implemented was constructed using one module per layer. The modules BS, SR, and MC each have one input, OP (aside from the data), that signifies either encryption or decryption. A value of 1 for OP makes those modules function in their normal manner, a value of 0 signifies decryption setting the modules BS, SR, and MC to invert their operation. The datapath as can be seen in Figure 2 is 128 bits wide. A counter array is used to determine the current round for an encryption or decryption operation and the system operates using a 4 stage pipeline. Data blocks which are matrices in the case of the Rijndael algorithm are tagged as up to 4 blocks can be in the pipeline at once. Comparators check to see which phase the encryption or decryption process is in (zeroth round, rounds 1-9, or last round).

Although the CryptoKnight system resides on an ASIC, it does exhibit some flexibility with regard to Rijndael. The number of rounds to execute can be defined by the user. The default is 10 rounds, and the maximum number of rounds is 32 defined by the input to the counter.

The algorithm is easily transferable to an FPGA allowing for certain flexibilities with regard to the implementation. This can accommodate for maintenance or changes to the AES algorithm that may occur in the future.

Operating on a 1 GHz clock the CryptoKnight system is capable of achieving throughputs of up to 200 Gbps. By operating the chips in parallel, this can be doubled or tripled depending on how many are used. Table II illustrates that

# of chips	Throughput
1	200 Gbps
2	400 Gbps
3	600 Gbps
4	800 Gbps
.	.
.	.
∞	∞ Gbps

TABLE II
POSSIBLE THROUGHPUTS OF OPERATING CRYPTOKNIGHT IN PARALLEL.

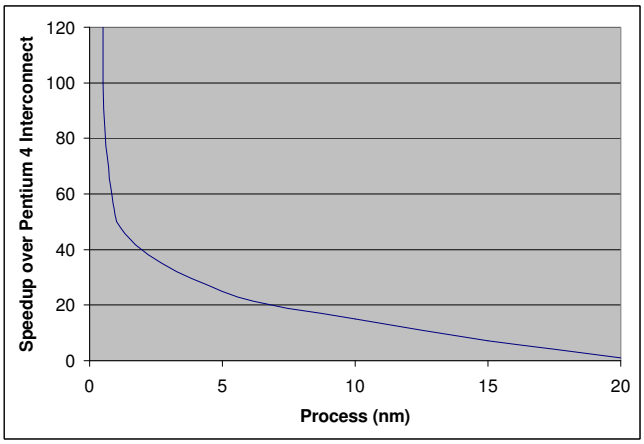

Fig. 3. The speedup of CryptoKnight interconnects compared to Pentium 4.

by using an unlimited number of chips, an infinite amount of data would be processed in a second.

C. Optimizations within CryptoKnight

The reason the chip size is quite small in comparison to other on-chip crypto-systems has much to do with the advances made by the Oregon State University semiconductor group. They have shrunk processes down to .5 nanotechnology which is extremely advanced. OSU now leads the world in this area of expertise. These advances have allowed the KryptoNight ASIC to total 4mm^2 in dimension which surpasses the area of any stand-alone cryptographic chip.

Further optimizations that allow for a smaller design are that the S-boxes are provided via a wireless method, i.e. they do not reside on-chip. Since such parts of cryptographic algorithms are known to the public this does not compromise the security of AES in any way. A separate ROM holds the S-box values and communicates remotely with the chip.

The modular design of Rijndael used in the CryptoKnight system reduces the total area that would be needed. As in the case of the FPGA Design of DES in [2] this proved to be a useful way of cutting down area requirements and simplifying the overall system. If changes occur in AES in the near future, only the corresponding module needs to be addressed.

The contributions to speed within CryptoKnight are closely related to the technology used as they allow for extremely high-speed interconnects for use in the integrated circuit. In comparison to Pentium 4 interconnects, the CryptoKnight is 100 times faster as is shown in Figure 3. This allows for extremely fast execution within each individual module. Also, The MixColumn transformation involves arithmetic computations modulo $x^4 + 1$. The GF Optimization Method (GOM) is used to speed up the process allowing such polynomial arithmetic operations to be calculated in less than a picosecond.

CryptoKnight also has very desirable power consumption running at 1/2 milliwatt. This is remarkable given the clock is running at 1GHz. Power management techniques are implemented through the use of the new ice-blocks located within the die. These are placed at the corners of the chip and keep the balance level such that the chip does not exceed maximum power capabilities. This allows the chip to operate for an indefinite amount of time making it well-suited for embedded systems.

III. CONCLUSION

As we move more towards mobile technologies and wireless systems, cryptography increases in importance. Effective cryptographic systems are required to allow for such advances in technology. Since the embedded system is increasing in popularity, researchers have focused towards improvements in the speed and area of cryptographic systems. Such improvements will need to be made to be able to fully incorporate cryptography with mobile and embedded systems. Our proposed system, CyptoKnight has proven to be an elite candidate for such future needs.

Further research and testing is currently being conducted in designing CryptoKnight2K5, the next in the line of CryptoKnight processors. We are attempting to incorporate quantum computing to further increase speed and reduce area such that the chip will not even be physically seen. Current obstacles are the unpredictability of quantum states, and 4th dimension interference with data integrity.

REFERENCES

[1] D. C. Wilcox, L. G. Pierson, P. J. Robertson, E. L. Witzke, and K. Gass, "A DES ASIC suitable for network encryption at 10 gps and beyond," in *Cryptographic Hardware and Embedded Systems - CHES 1999*, Ç. K. Koç and C. Paar, Eds. 1999, Lecture Notes in Computer Science No. 1717, pp. 37–48, Springer, Berlin, Germany.

[2] W. P. Choi and L. M. Cheng, "Modeling the crypto-processor from design to synthesis," in *Cryptographic Hardware and Embedded Systems - CHES 1999*, Ç. K. Koç and C. Paar, Eds. 1999, Lecture Notes in Computer Science No. 1717, pp. 25–36, Springer, Berlin, Germany.

[3] R. R. Taylor and S. C. Goldstein, "A high-performance flexible architecture for cryptography," in *Cryptographic Hardware and Embedded Systems - CHES 1999*, Ç. K. Koç and C. Paar, Eds. 1999, Lecture Notes in Computer Science No. 1717, pp. 231–245, Springer, Berlin, Germany.

[4] E. Mosanya, C. Teuscher, H. F. Restrepo, P. Galley, and E. Sanchez, "CryptoBooster: A reconfigurable and modular cryptograpic coprocessor," in *Cryptographic Hardware and Embedded Systems - CHES 1999*, Ç. K. Koç and C. Paar, Eds. 1999, Lec-

ture Notes in Computer Science No. 1717, pp. 246–256, Springer, Berlin, Germany.

[5] J. Großschädl, "High-speed RSA hardware based on barrett's modular reduction method," in *Cryptographic Hardware and Embedded Systems - CHES 2000*, Ç. K. Koç and C. Paar, Eds. 2000, Lecture Notes in Computer Science No. 1965, pp. 192–204, Springer, Berlin, Germany.

[6] National Institute of Standards and Technology (NIST). Advanced Encryption Standard (AES)., "Federal information processing standards (FIPS) publication 197," November 2001, Available for download at http://csrc.nist.gov/encryption.