Comparative Analysis of Various Cryptographic Algorithms

Er. Satish Kumar
Assistant Professor
Dept. of Information Technology
GIMET, Amritsar

Mr. Amit Puri
Assistant Professor
Dept. of Computer Application
GIMET, Amritsar

ABSTRACT

Does increased security provide comfort to fearful people? Or does security provide some very basic protections that we are inexperienced to believe that we don't need? During this time when the Internet provides essential communication between millions of people and is being increasingly used as a tool for commerce, trading, research & banking, security becomes a tremendously important issue to deal with. There are many aspects to security and many applications, ranging from secure commerce and payments to private communications and protecting passwords. This paper has two major purposes. The first is to define some of the terms and concepts behind basic cryptographic methods, and second is to offer a way to compare the myriad cryptographic algorithms in use today.

Keywords: Cryptography, Encryption, Decryption, Key-Setup, Throughput, Cores.

INTRODUCTION

Cryptography is the art/science of writing secret code. Some experts argue that cryptography appeared spontaneously sometime after writing was invented. In data and telecommunication systems, cryptography is necessary when communicating over any untrusted medium, which includes just about any network, particularly the Internet. Cryptography is the practice and study of hiding information. In modern times cryptography is considered as a branch of both mathematics and computer science and is affiliated closely with information theory, computer security and engineering. Cryptography is used in applications present in technologically advanced societies; examples include the security of ATM cards, computer passwords and electronic commerce which all depend on cryptography. Cryptography is the process of converting ordinary information (plaintext) into unintelligible gibberish (i.e., cipher text). The detailed operation of a cipher is controlled both by the algorithm and in each instance by a key. This is a secret parameter (ideally known only to the communicants) for a specific message exchange context. Keys are important, as ciphers without variable keys are trivially breakable and therefore less than useful for most purposes. Historically, ciphers were often used directly for encryption or decryption without additional procedures such as authentication or integrity checks. When communication takes place in between applications, then there are some specific security requirements, which include:

- **Authentication**: Authentication involves the process of proving one's identity. The primary forms of host-to-host authentication on the Internet today are name-based or address-based.

- **Privacy/confidentiality**: This ensures that no one can read the message except the receiver.

- **Integrity**: Integrity ensures the receiver that message has not been altered in any way from the original.

- **Non-repudiation**: This is a mechanism to prove that the sender really sent this message.

Cryptography not only protects data from theft or alteration but can also be used for user authentication. There are, in general, three types of cryptographic schemes typically used to accomplish these goals: secret key (or symmetric) cryptography, public-key (or asymmetric) cryptography, and hash functions. [1][2]

Symmetric-Key Algorithms

Symmetric-key algorithms are a class of algorithms for cryptography that use trivially related, often identical, cryptographic keys for both encryption & decryption. The encryption key is trivially related to the decryption key, in that they may be identical or there is a simple transform to go between the two keys. The keys, in practice, represent a shared secret between
two or more parties that can be used to maintain a private information link. Symmetric-key algorithms can be divided into stream ciphers and block ciphers. Stream ciphers encrypt the bits of the message one at a time, and block ciphers take a number of bits and encrypt them as a single unit. Blocks of 64 bits have been commonly used. The Advanced Encryption Standard algorithm approved by NIST in December 2001 uses 128-bit blocks. Some examples of popular and well-respected symmetric algorithms include Twofish, Serpent, AES (Rijndael), Blowfish.

Public Key Algorithms

Public-key cryptography is a cryptographic approach employed by many cryptographic algorithms and cryptosystems, whose distinguishing characteristic is the use of asymmetric key algorithms instead of or in addition to symmetric key algorithms. They do not require a secure initial exchange of one or more secret keys as is required when using in symmetric key algorithms. It can also be used to create digital signatures. The distinguishing technique used in public key - private key cryptography is use of asymmetric key algorithms because the key used to encrypt a message is not the same as the key used to decrypt it. Each user has a pair of cryptographic keys — a public key and a private key. The private key is kept secret, while the public key may be widely distributed. Messages are encrypted with the recipient's public key and can only be decrypted with the corresponding private key. The keys are related mathematically, but the private key cannot be feasibly (i.e., in actual or projected practice) derived from the public key. The Diffie-Hellman and RSA algorithms, in addition to being the first publicly known examples of high quality public-key algorithms, have been among the most widely used.

ANALYSIS

Candidate Algorithms for Analysis

In cryptography, the Advanced Encryption Standard (AES) is an encryption standard adopted by the U.S. government. The standard comprises three block ciphers, AES-128, AES-192 and AES-256. Each AES cipher has a 128-bit block size, with key sizes of 128, 192 and 256 bits, respectively. A new standard was needed primarily because DES (Data Encryption Standard) has a relatively small 56-bit key which was becoming vulnerable to brute force attacks. In addition the DES was designed primarily for hardware and is relatively slow when implemented in software. While Triple-DES avoids the problem of a small key size, it is very slow in software, is unsuitable for limited-resource platforms, and may be affected by potential security issues connected with the block size of 64 bits.

The AES cipher is specified as a number of repetitions of transformation rounds that convert the input plain-text into the final output of cipher-text. Each round consists of several processing steps, including one that depends on the encryption key. A set of reverse rounds are applied to transform cipher-text back into the original plain-text using the same encryption key.

The following AES algorithms are selected for the analysis:

1. MARS

MARS incorporates its "cryptographic core" into an innovative, heterogeneous overall structure. It also features a wide variety of operations, including the technique of rotating digits by a varying number of places that is determined by both the data and the secret key. Consequently, while MARS performs well in general, it performs particularly well on computer platforms that support its rotation and multiplication operations efficiently. National Institute of Standards and Technology (NIST) accepted a modification to MARS for Round 2 improves its ability and flexibility to function in some memory-constrained environments, such as low-end smart cards. MARS was submitted to the AES development effort by the International Business Machines Corporation.[3]

2. RC6

RC6 is an algorithm that is simple enough to memorize, and should be easy to implement compactly in both software and hardware. Its simplicity also should facilitate its further security analysis in Round 2, which is assisted by the analysis of its predecessor, RC5. RC6 does not use substitution tables; instead, the principal engine for its security is the technique of rotating digits by a varying number of places that is determined by the data. In general, RC6 is fast, and it is particularly fast on platforms that support its rotation and multiplication operations efficiently; its key setup is also fast. RC6 was submitted to the AES development effort by RSA Laboratories.[4][5]

3. Rijndael

Rijndael performs excellently across all considered platforms. Its key setup is fast, and its memory requirements are low, so it also should perform well in hardware and in memory-constrained environments. The straightforward design and the conservative choice of operations should facilitate its further analysis, and the operations should be relatively easy to defend against certain attacks on physical implementations. Even though parallel processing was not considered during the Round 1
selection process by the AES review team, Rijndael has the potential of benefiting from advances in computer processors that allow many instructions to be executed in parallel. Rijndael was submitted to the AES development effort by Joan Daemen and Vincent Rijmen.

4. Serpent

Serpent is ultra-conservative in its security margin: the designers chose to use twice as many iterations as they believed secure against currently known attacks. Consequently, Serpent’s performance is relatively slow compared to the other four finalists; however, in some settings this should be mitigated by the efficiency of optimized implementations using what the submitters call the “bitslice” mode, for which the algorithm was specially designed. Serpent should fit well in hardware (with potential tradeoffs of speed versus space) and in memory-constrained environments. The straightforward design and the conservative choice of operations should facilitate further analysis of this candidate, and the operations should be easy to defend against certain attacks on physical implementations. Serpent was submitted to the AES development effort by Ross Anderson, Eli Biham, and Lars Knudsen.

5. Twofish

Twofish exhibits fast and versatile performance across most platforms; it also should perform well both in hardware and in memory-constrained environments. It features variable substitution “tables” that depend on the secret key. The submitters believe that such tables generally offer greater security than tables with fixed values. The possibility of pre-computing these tables to varying degrees helps Twofish to offer a wide variety of performance tradeoffs depending on the setting. Twofish can be optimized for speed, key setup, memory, code size in software, or space in hardware. Twofish was submitted to the AES development effort by Bruce Schneier, John Kelsey, Doug Whiting, David Wagner, Chris Hall, and Niels Ferguson.

Technology Used for Analysis-FPGA

Processors and ASIC (Application Specific Integrated Circuits) are the cores of the two major computing paradigms of our days. Processors are general purpose and can virtually execute any operation. However, their performance is limited by the restricted interconnect, data path and instruction set provided by the architecture. Conversely, ASICs are application-specific and can achieve superior performance compared with processors. However, the functionality of an ASIC design is restricted by the designed parameters provided during fabrication. Any update to an ASIC-based platform incurs high cost. As a result, ASIC-based approaches lack flexibility.

FPGA (Field Programmable Gate Arrays) technology is a growing area of research that has the potential to provide the performance benefits of ASICs and the flexibility of processors. Application specific hardware circuits can be created on demand to meet the computing and interconnect requirements of an application. Moreover, these hardware circuits can be dynamically modified partially or completely in time and in space based on the requirements of the operations under execution. As a result, superior performance can be expected compared with the performance of the equivalent software implementation executed on a processor.

FPGAs were initially an offshoot of the quest for ASIC prototyping with lower design cycle time. The evolution of the configurable system technology led to the development of configurable devices and architectures with great computational power. As a result, new application domains become suitable for FPGAs beyond the initial applications of rapid prototyping and circuit emulation. FPGA-based solutions have shown significant speedups (compared with software and DSP based approaches) for several application domains such as signal & image processing, graph algorithms, genetic algorithms, and cryptography among others. The basic feature underlying FPGAs is the programmable logic element which is realized by either using anti-fuse technology or SRAM-controlled transistors. FPGAs have a matrix of logic cells overlaid with a network of wires. Both the computation performed by the cells and the connections between the wires can be configured. Current devices mainly use SRAM to control the configurations of the cells and the wires. Loading a stream of bits onto the SRAM on the device can modify its configuration. Furthermore, current FPGAs can be reconfigured very quickly, allowing their functionality to be altered at runtime according to the requirements of the computation.

FPGA-based Cryptography

FPGA devices are a highly promising alternative for implementing private-key cryptographic algorithms. Compared with software-based implementations, FPGA implementations can achieve superior performance. The fine-granularity of FPGAs matches extremely well the operations required by private-key cryptographic algorithms (e.g., bit-permutations, bit-substitutions, look-up table reads, Boolean functions). As a result, such operations can be executed more efficiently in FPGAs than in a general-purpose computer.

Furthermore, the inherent parallelism of the algorithms can be efficiently exploited in FPGAs as opposed to the serial
fashion of computing in an uni-processor environment. At the cryptographic-round level, multiple operations can be executed concurrently. On the other hand, at the block-cipher level, certain operation modes allow concurrent processing of multiple blocks of data.

For example, in the ECB mode of operation, multiple blocks of data can be processed concurrently since each data block is encrypted independently. Consequently, if \( p \) rounds are implemented, a throughput speed-up of \( O(p) \) can be achieved compared with a “single-round” based implementation (one round is implemented and is reused repeatedly). Moreover, by adopting deep-pipelined designs, the throughput can be increased proportionally with the clock speed. On the contrary, in feedback modes of operation (e.g., CBC, CFB), where the encryption results of each block are fed back into the encryption of the current block, encryption cannot be parallelized among consecutive blocks of data. As a result, the maximum throughput that can be achieved depends mainly on the encryption time required by a single cryptographic round and the efficiency of the implementation of the key-setup component of an algorithm.

Besides throughput, FPGA implementations can also achieve agile key-context switching. Key-context switching includes the generation of the required key dependent data for each cryptographic round (e.g., subkeys, key-dependent S-boxes). A cryptographic round can commence as soon as its key-dependent data is available. In software implementations, the cryptographic process cannot commence before the key-setup process for all the rounds is completed. As a result, excessive context switching is introduced making key-context switching inefficient. On the contrary, in FPGAs, each cryptographic round can commence as early as possible since the key-setup process can run concurrently with the cryptographic process. As a result, minimal latency can be achieved.

Security issues also make FPGA implementations more advantageous than software-based solutions. An encryption algorithm running on a generalized computer has no physical protection. Hardware cryptographic devices can be securely encapsulated to prevent any modification of the implemented algorithm. In general, hardware-based solutions are the embodiment of choice for military and serious commercial applications (e.g., NSA authorizes encryption only in hardware).

Finally, even if ASICs can achieve superior performance compared with FPGAs, their flexibility is restricted. Thus, the replacement of such application-specific chips becomes very costly while FPGA-based implementations can be adapted to new algorithms and standards.

**IMPLEMENTATION & DESIGN DECISIONS**

As a hardware target for the proposed implementations, we have chosen the Xilinx Virtex family of FPGAs. Virtex is a high-capacity, high-speed performance FPGA providing a superior system integration feature set. For mapping onto Virtex devices, we used the Foundation Series v2.1i software development tool. The synthesis and place-and-route parameters of the tool remained the same for all the implementations. All the results were based on placed-and-routed implementations that included both the key-setup component and the cryptographic core along with their control circuit.

Among the various time-space tradeoffs, our focus was primarily time performance. For each algorithm we have implemented the key-setup component, the control circuitry, and the encryption block cipher for 128-bit data blocks using 128-bit keys. A “single-round” based design was chosen for each implementation. Since one round was implemented, it was reused repeatedly. The key-setup component was processing data in parallel with the cryptographic core. While the cryptographic core was processing the data of the \( i \)th round, the key-setup component was calculating the key-dependent data for the \( (i + 1) \)th round. As a result, even if an algorithm does not support on-the-fly key generation in the software domain, the key setup can be executed on the \( y \) in FPGAs.

Our goal was to maximize throughput for each candidate algorithm. We have exploited the inherent parallelism of each cryptographic core and the low-level hardware features of FPGAs to enhance the performance. The performance metrics are throughput and key-setup latency. The throughput metric indicates the amount of data encrypted per time unit after the initialization of the algorithm. The key-setup latency denotes the minimum time required to commence encryption after providing the input key. While throughput indicates the bulk-encryption capability of the implementation, key-setup latency indicates the capability of agile key-context switching.

The key-setup latency issue was of primary interest, that is, the cryptographic core had to commence as early as possible. Based on the achieved throughput, we designed the key-setup component to sustain the processing rate of the cryptographic core and to achieve minimal latency. The key-setup latency metric is the key metric for applications where a small amount of data is processed per key and key-context switching occurs repeatedly. In software implementations, the cryptographic process cannot commence before the key-setup process for all the rounds is completed. As a result, the key-setup latency time equals the key-setup time.

To implement efficient key-setup circuits, we took advantage of the embedded memory modules (Block SelectRAM) of the
Virtex FPGAs [16]. The Virtex FPGA Series provides dedicated on-chip blocks of true dual-read/write port synchronous RAM, with 4096 memory cells each. Depending on the size of the device, 32-132 Kbits of data can be stored using the Block Select RAM memory modules. The key-setup circuit utilized these memory modules to pass its results to the cryptographic core. As a result, the cryptographic core could commence as soon as the key-dependent data for the first encryption round is available in the memory modules. Then, during each encryption round, the cryptographic core reads the corresponding data from the memory modules.

For each algorithm, we have also implemented the key-setup circuit and the cryptographic core separately. For all the implementations, the maximum clock speed of the key-setup circuit was higher than the maximum clock speed of the cryptographic core. Based on the results of these individual implementations, we also provide latency estimates for implementations that clock each circuit at its maximum speed.

Regarding the cryptographic cores, the majority of the required operations fit extremely well in Virtex FPGAs. The permutations and substitutions can be hard-wired while distributed memory can be used as look-up tables. In addition, boolean functions, data-dependent rotations, and addition can be mapped very efficiently onto Virtex FPGAs. Wherever a multiplication with a constant was required, constant coefficient multipliers were utilized to enhance the performance compared with “regular” multipliers. Regular multiplication is required only by the MARS and RC6 block ciphers. In both cases, two 32-bit numbers are multiplied and the lower 32-bit of the output are used in the encryption process. We tried the multiplier-macros provided for Virtex FPGAs but we found that they were a performance bottleneck. Besides the excessive latency that was introduced due to the numerous pipeline stages, excessive area was also required since the full multiplier was mapped onto the FPGA. As a result, the latency was reduced by more than 50% and the area requirements were also reduced significantly.

IMPLEMENTATION RESULTS

In the following, implementation results as well as relevant performance issues specific to each algorithm are provided. The key-setup latency results are represented both as absolute time and as the fraction of the corresponding encryption time over one 128-bit block of data. In addition, the throughput results are represented both as encryption rate and as encryption rate elaborated on area. Finally, area requirements results are provided for both the key-setup and the cryptographic core circuits.

MARS

The MARS block cipher is the IBM submission to AES. The time performance and area requirements results for our MARS implementation are shown in Table 1.

Key Setup

The MARS key expansion procedure expands the input 128-bit key into a 1280-bit key. First a linear-key expansion occurs following by stirring the key-words based on an S-box. Both processes involve simple operations performed repeatedly. However, the final stage of modifying the multiplication key-words involves string-matching operations that are relatively expensive functions. String-matching is an expensive operation compared with the rest of the operations required by MARS. A compact implementation of string-matching introduces high latency while a high-performance implementation increases the area requirements dramatically. In our implementation, the last stage of the key expansion process (i.e., string-matching) was not implemented. In spite of this, the introduced key-setup latency was still relatively high (the worst among all the implementations considered in this paper).

Cryptographic Core

The cryptographic core of MARS consists of a 16-round cryptographic layer wrapped with two layers of 8-round “forward" and “backward mixing". The achieved throughput depended mainly on the efficiency of the multiplier. In our implementation only one round of each layer was implemented that was used repeatedly. The encryption time for one block of data was 32 clock cycles. An interesting feature of our design is that by increasing the utilization factor of the processing stages (i.e., all the three processing stages execute in parallel), the average encryption time for one block of data can be reduced to 16 clock cycles for operation modes that allow concurrent processing of multiple blocks of data (e.g., non-feedback, interleaved).

<table>
<thead>
<tr>
<th>Key-Setup Latency</th>
<th>Throughput</th>
<th>Area Requirements</th>
</tr>
</thead>
<tbody>
<tr>
<td>ms</td>
<td>KSLT BET</td>
<td>Mbits/sec</td>
</tr>
</tbody>
</table>


The RC6 block cipher is the AES proposal of the RSA Laboratories and R. L. Rivest from the MIT Laboratory for Computer Science [12]. The implemented block cipher corresponds to \( w = 32 \)-bit round keys, \( r = 20 \) rounds, and \( b = 14 \)-byte input key. The time performance and area requirements results for our RC6 implementation are shown in Table 2.

**Key Setup**

The RC6 key setup expands the input 128-bit key into 42 round keys. The key for each round corresponds to a 32-bit word. The key scheduling is fairly simple. The round keys are initialized based on two constants. We have implemented the initialization procedure using a look-up table since it is the same for any input key. Then, the contents of the look-up table were used to generate the round keys with respect to the input key. As a result, remarkably low key-setup latency was achieved that was equal to the 15% of the time for encrypting a block of data.

**Cryptographic Core**

The cryptographic core of RC6 consists of 20 rounds. The symmetry and regularity found in the RC6 block cipher resulted in a compact implementation. The entire data block was processed at the same time by using two identical circuits. The achieved throughput depended mainly on the efficiency of the multiplier.

<table>
<thead>
<tr>
<th>Key-Setup Latency</th>
<th>Throughput</th>
<th>Area Requirements</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \mu s )</td>
<td>KSLT</td>
<td>BET</td>
</tr>
<tr>
<td>0.17</td>
<td>0.15</td>
<td>112.87</td>
</tr>
</tbody>
</table>

KSLT - Key Setup latency

BET - Block encryption time

**RIJNDAEL**

The Rijndael block cipher is the AES proposal of J. Daemen and V. Rijmen from the Katholieke University Leuven [7]. The implemented block cipher corresponds to \( N_b = 4, N_k = 4 \), and \( N_r =10 \) (i.e., 4x32-bit block data, 4x32-bit key, 10 rounds). The time performance and the area requirements results of our implementation are shown in Table 3.

**Key Setup**

The Rijndael key setup expands the input 128-bit key into a 1408-bit key. Simple operations are used that resulted in extremely low key-setup latency latency. ROM-based look-up tables were utilized to perform the SubByte transformation. The achieved latency was the lowest among all the implementations considered in this paper.

**Cryptographic Core**

The cryptographic core of Rijndael consists of 10 rounds. The cryptographic core is ideal for implementations on FPGAs. It combines fine-grain parallelism with look-up table operations. The round transformation can be represented as a look-up table.
resulting in extremely high speed. We have implemented a ROM-based fully-parallel version of the look-up table. By combining common references to the look-up table, we have achieved a 25% savings in ROM compared with the straightforward implementation suggested in the AES proposal. The simplicity of the operations and the inherent fine-grain parallelism resulted in the highest throughput among all the implementations. Furthermore, the Rijndael implementation had the highest area utilization factor (i.e., throughput per area unit).

<table>
<thead>
<tr>
<th>Key-Setup Latency</th>
<th>Throughput</th>
<th>Area Requirements</th>
</tr>
</thead>
<tbody>
<tr>
<td>µs</td>
<td>KSLT</td>
<td>Mbits/sec</td>
</tr>
<tr>
<td>BET</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0.07</td>
<td>0.20</td>
<td>353.00</td>
</tr>
</tbody>
</table>

KSLT - Key Setup latency time
BET - Block encryption time

Table 3: Implementation Details of Rijndael

SERPENT

The Serpent block cipher is the AES proposal of R. Anderson, E. Biham, and L. Knudsen from Technion, Cambridge University, and University of Bergen respectively. The time performance and area requirements results for our Serpent implementation are shown in Table 4.

Key Setup

The Serpent key setup expands the input 128-bit key into a 4224-bit key. First, the input key is padded to 256 bits and then it is expanded to an intermediate key by iterative mixing of the key data. Finally, by using look-up tables, the keys for all the rounds are calculated. The simplicity of the required operations resulted in extremely low key-setup latency (the second lowest among all the implementations considered in this paper).

Cryptographic Core

The cryptographic core of Serpent consists of 32 rounds. The round transformation is a linear transform consisting of rotations, shifts, and XOR operations. Neither multiplication nor addition is required. As a result, the lowest encryption time per round and the most compact implementation were achieved among all the implementations. Furthermore, the Serpent implementation had the second higher area utilization factor.

<table>
<thead>
<tr>
<th>Key-Setup Latency</th>
<th>Throughput</th>
<th>Area Requirements</th>
</tr>
</thead>
<tbody>
<tr>
<td>µs</td>
<td>KSLT</td>
<td>Mbits/sec</td>
</tr>
<tr>
<td>BET</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0.08</td>
<td>0.09</td>
<td>148.95</td>
</tr>
</tbody>
</table>

KSLT - Key Setup latency time
BET - Block encryption time

Table 4: Implementation Details of Serpent

Twofish

The Twofish block cipher is the AES proposal of the Counterpane Systems, Hi/ fn, Inc., and D. Wagner from the University of
California Berkeley. The time performance and area requirements results of our implementation are shown in Table 5.

**Key Setup**

The Twofish key setup expands the input 128-bit key into a 1280-bit key. Moreover, it generates the key-dependent S-boxes used in the cryptographic core. Four 128-bit S-boxes are generated. Since our goal was to minimize latency, we have implemented a parallel version of the key setup consisting of 24 q0/q1 permutation boxes and 2 MDS matrices. Moreover, the RS matrix was implemented for the S-box generation. The matrices are used for “constant matrix”-to-matrix multiplication over GF (2^8). The best known implementation of a constant coefficient multiplier in Virtex FPGAs is by using a look-up table. As a result, low latency was achieved but excessive area was required. The area requirements corresponded to the 70% of the total area. However, by implementing a more compact design (e.g., reusing processing elements), the key-setup latency would increase.

**Cryptographic Core**

The cryptographic core of Twofish consists of 16 rounds. The structure of the round transformation is similar to the structure of the key-expansion circuit. The only major difference is the S-boxes that the cryptographic core uses.

<table>
<thead>
<tr>
<th>Key-Setup Latency</th>
<th>Throughput</th>
<th>Area Requirements</th>
</tr>
</thead>
<tbody>
<tr>
<td>µs</td>
<td>KSLT</td>
<td>BET</td>
</tr>
<tr>
<td></td>
<td>Mbits/sec</td>
<td>Total</td>
</tr>
<tr>
<td></td>
<td># of slices</td>
<td>Key setup</td>
</tr>
<tr>
<td>0.18</td>
<td>0.25</td>
<td>173.06</td>
</tr>
<tr>
<td></td>
<td>6554</td>
<td>(70%)</td>
</tr>
</tbody>
</table>

KSLT - Key Setup latency time,
BET - Block encryption time

**Table 5: Implementation Details of Twofish**

**COMPARATIVE ANALYSIS OF FPGA IMPLEMENTATIONS**

In Figure 1 key-setup latency comparisons are made among our FPGA implementations. The comparisons are made in terms of absolute time and the ratio of the key-setup latency time to the time required to encrypt one block of data. The latter metric represents the capability of agile key-context switching with respect to the encryption rate.

![Fig. 1.1: µs](image1)

**Fig. 1.1: µs**
Fig. 1: Key-Setup Latency Comparison

Clearly, Rijndael and Serpent achieve the lowest key-setup latency times while the latency times for RC6 and Twofish are higher by a factor of 2.5. As we have mentioned in Section 4, the key-setup latency introduced by MARS is the highest. All the algorithms (except MARS) achieve key-setup latency time that is equal to the 7-25% of the time for encrypting one block of data.

Fig. 2: Throughput Comparison

In Figure 2, throughput comparisons are made among our FPGA implementations. The comparisons are made in terms of the encryption rate and the ratio of the encryption rate to the area requirements. The latter metric reveals the hardware utilization efficiency of each implementation. Rijndael achieves the highest encryption rate due to the ideal match of its algorithmic characteristics with the hardware characteristics of FPGAs. In addition, the encryption rate of Rijndael is higher than the ones achieved by the other algorithms by a factor of 1.7-3.12. Moreover, Rijndael also achieves the best hardware utilization. The latter metric combines, for each algorithm, the computational demands in terms of an FPGA implementation with the inherent parallelism of the cryptographic round.

Serpent achieves the second best hardware utilization while having the lowest encryption time per round. The latter suggests that, under the same area constraints, Serpent can achieve throughput equivalent to Rijndael for operation modes that allow concurrent processing of multiple blocks of data. Similar to Rijndael, the algorithmic characteristics of Serpent matches extremely well with the hardware characteristics of FPGAs.

Fig. 3: Area Requirement Comparison

Finally, in Figure 3, area comparisons are made among our FPGA implementations. The comparisons are made in terms of the total area as well as the area required by each of the key-setup and the cryptographic core circuits. Serpent and RC6 have the most compact implementations. Serpent also has the most compact cryptographic core circuit while RC6 has the most compact key-setup circuit. For the MARS block cipher, the result shown is based on an implementation that does not include the circuit for modifying the multiplication key-words.

CONCLUSION

In this paper, we discussed the provided time performance and area requirements results for the implementations of the five final AES candidates (MARS, RC6, Rijndael, Serpent, and Twofish) using FPGAs. In our implementations, the key-setup process can be performed in parallel with the encryption process regardless of the capability of the software implementation to support on-the-fly key setup. The most obvious conclusion that can be drawn from these performance comparison is that it is
very difficult to compare the various candidate algorithms for efficiency because efficiency means many different things in different context. Efficiency may be key-setup time in hardware or it may be bulk encryption speed in software. As a standard, AES must be efficient in all of these contexts, since it is used in wide variety of applications.

The implementations suggest that Rijndael and Serpent favor FPGA implementations the most due to the ideal match of their algorithmic characteristics with the characteristics of FPGAs. The Rijndael implementation achieves the lowest key-setup latency time, the highest throughput, and the highest hardware utilization. Comparing our results with software and ASIC implementations, we draw the conclusion that that Rijndael also achieves the best time performance across different platforms i.e., ASIC, FPGA, software.

REFERENCES