Hindawi Publishing Corporation
Journal of Computer Systems, Networks, and Communications
Volume 2008, Article ID 923079, 6 pages
doi:10.1155/2008/923079

Research Article

Novel Hardware Implementation of the Cipher Message Authentication Code

H. E. Michail, G. Selimis, M. Galanis, D. Schinianakis, and C. E. Goutis

Department of Electrical and Computer Engineering, University of Patras, 26500 Patras, Greece

Correspondence should be addressed to H. E. Michail, michail@ece.upatras.gr

Received 3 April 2008; Revised 28 July 2008; Accepted 12 August 2008

Recommended by X. Du

A new algorithm for producing message authenticating codes (MACs) was recently proposed by NIST. The MAC protects both a message's integrity—by ensuring that a different MAC will be produced if the message has changed—as well as its authenticity because only someone who knows the secret key could have generated a valid MAC. The proposed security scheme incorporates an FIPS approved and secure block cipher algorithm and was standardized by NIST in May, 2005. In this paper is presented the first efficient hardware implementation of the CMAC standard.

1. INTRODUCTION

It was only on May 2005 when National Institute of Standards and Technology (NIST) decided to standardize a new algorithm for producing message authenticating codes (MACs) which are widely used in security schemes for transactions via public networks. This new method is the CMAC standard [1]. This algorithm was standardized recently, and thus no implementations have been presented concerning the CMAC standard. In this paper, CMAC's first implementation is presented and its throughput reaches a maximum of about 2 Gbps.

A message authentication code (MAC) is an authentication tag (also called a checksum) derived by applying an authentication scheme, together with a secret key, to a message. Unlike digital signatures, MACs are computed and verified with the same key, so that they can only be verified by the intended recipient.

MACs are generated with several different ways depending on the corresponding standard. One popular way of producing MACs is described in the HMAC standard that incorporates the usage of a secure hash function such as SHA-1 or SHA-256 and was proposed by NIST some years ago. However, in the case of applications based on block cipher algorithms, a hash function is used only for the production of MACs. Hence, it is essential to exploit an already existing cryptographic primitive IP in the system, such as common block ciphers and not a special purpose hash function. Thus, the corresponding security schemes are area efficient and they can be widely deployed in portable devices.

In 1985, a data authentication algorithm (DAA) [2] was proposed which could be used to detect unauthorized modifications, both intentional and accidental, to data. In DAA, a data authentication code (DAC) is produced that is similar to the currently used MACs. DAA is based on the algorithm specified in the data encryption standard (DES) [3], which is now considered as potentially insecure. Respectively, DAA is also insecure since its security level depends on the underlying block cipher algorithm.

DES has been phased out in favor of advanced encryption standard (AES) [4], and the next step was to standardize a secure mechanism that will provide data integrity and authenticity. Taking these facts into consideration, NIST decided to standardize a new process for generating MACs, using approved block cipher algorithms like AES or triple data encryption algorithm (TDEA) [3], and so forth. Moreover, in this standard, it is the first time that the notion of MAC is used in the way that is currently used to all widespread applications that call for security in antithesis to DAA.

MACs are used in public key digital signature techniques that provide data integrity and source authentication capabilities to enhance data trustworthiness in computer

networks. These techniques are widely used to a great number of applications, especially in the security layer of almost every communication protocol. One application that incorporates such techniques is the public key infrastructure (PKI) [5] that provides digital certificates to both clients and servers for secure transactions.

They are also used in the 802.16 standard for local and metropolitan area networks and in virtual private networks (VPNs) that companies are establishing in order to exploit the benefits of online collaboration. Moreover, MACs are widely used for digital signature algorithm (DSA) [6], which is used for producing digital signatures, and in secure electronic transactions (SETs) [7] which is a standard for secure electronic transactions via public networks that have been deployed by VISA, MasterCard, and many other leading companies providing financial services.

CMAC like any well-designed MAC algorithm provides stronger assurance of data integrity than a checksum or an error detecting code. The verification of a checksum or an error detecting code is designed to detect only accidental modifications of the data, while CMAC is designed to detect intentional perturbations, unauthorized modifications of the data, as well as accidental alterations.

The rest of this paper is organized as follows. In Section 2, issues about lack of related work and intended applications of CMAC are discussed, while in Section 3, general issues on CMAC are given. In Section 4, CMAC specifications are provided according to the standard that was proposed by NIST. The proposed implementation is presented in depth, providing details regarding the architecture and the logic in Section 5. In Section 6, experimental results of the implementations FPGA technology are discussed. Finally, the conclusion of our work is found in Section 7.

2. RELATED WORK. WHY CMAC?

It has to be noticed that, to the best of our knowledge, there are no other hardware implementations concerning CMAC (CMAC neural network has no relation to CMAC for producing MACs for integrity). This is partially due to the fact that this algorithm was recently proposed, and industry has not yet introduced to the market-related hardware products.

However, some software CMAC implementations do exist but obviously, these cannot be either commented or compared to our work which proposes a hardware implementation.

Another reason for the nondevelopment of hardware CMAC implementations is that the functionality of a CMAC can be incorporated in a security scheme with an HMAC which uses a hash function and this is the case in most of today's applications. Industry must be persuaded for the necessity of this algorithm if it is to start introducing CMAC hardware implementations, which are costly comparing to software implementations. Later, in this section, a specific case of this kind (e-voting) is commented so as to bring out this necessity.

This necessity mainly has to do with the numerous applications that do not call for high-throughput implementations (as long as the security part is concerned) but for small-sized or low-power implementations. In these cases, the usage of CMAC instead of HMAC is ideal since there is no need to incorporate both a hash function and a cipher block but only a block cipher algorithm like AES that can handle all aspects of security in the certain security scheme.

Such security applications are those used in e-voting, where there is a lot of time available for the transaction to be performed, and thus for the sake of small-sized and low-power implementations, CMAC seems to be a very good solution.

Moreover, it has to be noticed that this is a trend in our days considering many standards that are either under consideration or published by NIST in 2007 like XTS [8] and GCM [9], where security schemes tend to use a cipher block like AES in order to handle all security aspects in an application. It seems that there is a tendency to use only one basic cryptographic component, that is, AES and based on that to broaden offered security services over storage devices, wireless networks, and so on.

In order to make time-to-market shorter and reduce cost, the need of providing all kinds of security services with the usage of only one basic cryptographic component has emerged, as it is clearly seen especially during the last year with the hot debate about XTS [8] and GCM [9].

This highlights the importance of our work since in the same way, we propose a way to produce an MAC using AES cipher block instead of using a hash function like SHA-1 or SHA-256. CMACs like XTS and GCM focus on the service that intends to offer ensuring the security level and simplifying the whole implementation without using sophisticated algorithms for each case. This way, we manage to save a significant amount of integration area and power consumption, reduce the design and implementation complexity, and offer the chance to concentrate only on the optimization of the incorporated AES algorithm in order to improve the performance of the whole security scheme.

Thus, in applications where there are a few servers (for centralized data process) and a great number of distributed portable clients (for data collection), that need to be cheap, small, and have minor power consumption without great concern about throughput, like in e-voting, the adoption of CMAC for MAC generation seems to be ideal since it will result in reliable and efficient solutions.

3. GENERAL ISSUES ON CMAC

The CMAC algorithm incorporates a symmetric key block cipher. Block ciphers consist of two operations one for encryption and one for decryption. However, in the implementation of the CMAC algorithm, only one of these operations is needed, and usually the encryption process is selected. The cipher encryption process is a permutation on bit strings of a fixed length; the strings are called blocks and their size varies depending on the used block cipher. For the AES, it is 128 bits, whereas for TDEA is 64 bits.

For the implementation of the CMAC algorithm, a variety of block ciphers can be used. However, the block cipher will be secure. The CMAC algorithm implies the usage of a secret key which is the block cipher key.

The corresponding key will be protected for its secrecy and used exclusively for the CMAC mode of the chosen block cipher.

3.1. Input and output data

For a given block cipher and key, the input to the MAC generation function is a bit string called the message, denoted M. The bit length of M is denoted M_{len}. The value of M_{len} is not an essential input for the MAC generation algorithm if the implementation has some other means of identifying the last block in the partition of the message. Thus, in such a case, the computation of the MAC may begin "online" before the entire message is available. In principle, there is no restriction on the lengths of messages. In practice, however, the system in which CMAC is implemented may restrict the length of the input messages to the MAC generation function.

The output of the MAC generation function is a bit string called the MAC, denoted T. The length of T, denoted T_{len}, is a parameter that will be fixed for all invocations of CMAC with the given key.

3.2. Subkeys

The block cipher key is used to derive two additional secret values, called the subkeys, denoted $K1$ and $K2$. The length of each subkey is the block size. The subkeys are fixed for any invocation of CMAC with the given key. Consequently, the subkeys may be precomputed and stored with the key for repeated use; alternatively, the subkeys may be computed anew for each invocation.

Any intermediate value in the computation of the subkey, in particular, $\text{CIPH}_K(0^b)$ will also be secret. $\text{CIPH}_K(0^b)$ is the calculated ciphertext using the secret key K for the message that consists of b zeros. This requirement precludes the system in which CMAC is implemented from using this intermediate value publicly for some other purpose, for example, as an unpredictable value or as an integrity check value on the key. One of the elements of the subkey generation process is a bit string, denoted Rb that is completely determined by the number of bits in a block. In particular, for the two block sizes of the currently approved block ciphers, $R_{128} = 0^{120}10000111$ and $R_{64} = 0^{59}11011$.

In general, R_b is a representation of a certain irreducible binary polynomial of degree b, namely, the lexicographically first among all such polynomials with the minimum possible number of nonzero terms. If this polynomial is expressed as $u^b + c_{b-1}u^{b-1} + \cdots + c_2u^2 + c_1u + c_0$, where the coefficients $c_{b-1}, c_{b-2}, \ldots, c_2, c_1$ are either 0 or 1, then R_b is the bit string $c_{b-1}c_{b-2}\cdots c_2c_1c_0$.

3.3. MAC generation and verification

According to MAC algorithm, an authorized party applies the MAC generation process to the data to be authenticated to produce an MAC for the data. Subsequently, any authorized party can apply the verification process to the received data and the received MAC. Successful verification provides assurance of data authenticity and of integrity.

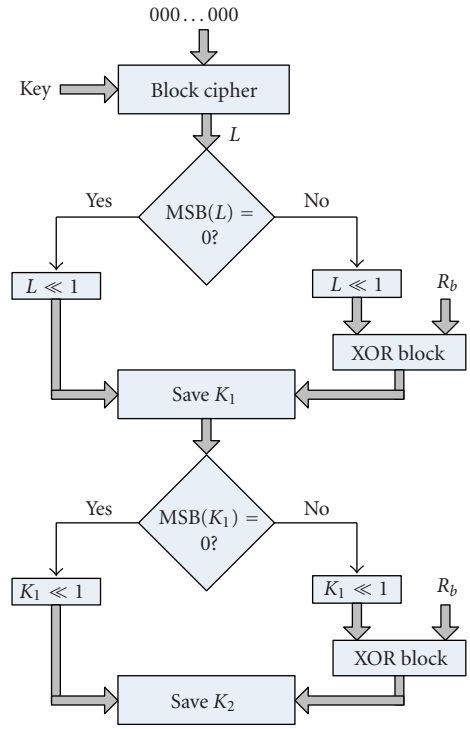

Figure 1: Subkeys generation process.

4. CMAC SPECIFICATION

Subkey generation, MAC generation, and MAC verification are specified in this section. The specifications include the inputs, the outputs, a suggested notation for the function, the steps, and a summary. The inputs that are typically fixed across many invocations of CMAC are called the prerequisites. The suggested notation does not include the block cipher.

4.1. Subkey generation

The subkeys generation process for the CMAC is described in Figure 1.

First of all the block cipher is applied to the block that consists entirely of "0" bits. In the next step, the first subkey is derived from the resulting string by a left shift of one bit, and, conditionally, by XORing a constant that depends on the block size. Next, the second subkey is derived in the same manner from the first subkey. Any intermediate value in the computation of the two subkeys, in particular, $\text{CIPH}_K(0^b)$, will be secret. For this reason, its computation should be done in the security scheme that will be implemented in hardware.

4.2. MAC generation and verification

In [1], the authentication procedure using a cipher block is described. There all the prerequisites are mentioned and thoroughly described. This process has some similarities with

the procedure that is followed in order to produce an MAC using a hash function which is the most common case.

The sender produces the MAC using the eight steps described in [1] and thus producing the CMAC, that is, the MAC with the usage of a cipher block like AES. It has to been noticed that the iterations of Step 5 may be executed "on the fly," that is, on each successive block of the message as soon as it is available for processing. Step 4 may be delayed until the final bit string in the partition is available; the appropriate case, and value of j, if necessary, can be determined from the length of the final bit string. In such an implementation, the determination in Step 2 of the total number of blocks in the formatted message may be omitted, assuming that the implementation has another way to identify the final string in the partition.

Similarly, the subkeys need not be computed anew for each invocation of CMAC with a given key; instead, they may be precomputed and stored along with the key as algorithm inputs.

For the verification process, the necessary procedure is fully specified in the standard [1] that has to be followed by the recipient of the message. This means producing an MAC using the same cipher block with the sender of the message. Finally, the two MACs are compared and in case they match, the transaction is considered as valid.

The CMAC standard implies the usage of a Federal Information Processing Standards (FIPSs) approved cipher block and the most widely currently used block cipher algorithm is the advanced encryption standard (AES) which is considered as absolutely secure. For implementation's sake, this block cipher was designed in order to be used in the whole CMAC structure as it will be presented in Section 5.

It should be stressed that only the encryption process was developed since the CMAC uses either the encryption or the decryption process of the selected block cipher algorithm. However, in a real world security scheme, both processes exist for encrypting and decrypting the exchanged data between the communicating parts and at the computation of a CMAC value simply only the one process is used. Other block cipher algorithms that can be used in the CMAC mechanism are all FIPS approved block ciphers such as TDEA. However, we decided to use the encryption process of the AES block cipher because it is the most widespread worldwide.

Many AES implementations have been proposed and for the scope of the paper one conventional implementation of AES has been developed. As it will be referred, the critical path of the whole CMAC mechanism is located at the AES core and so at a later time effort can be paid in optimizing the used cipher block (i.e., AES) and thus optimize the CMAC mechanism.

5. PROPOSED CMAC IMPLEMENTATION

The CMAC algorithm incorporates the usage of a symmetric key block cipher like AES or TDEA. In the implementation scheme, in this paper, existing implementations of these two block cipher algorithms were used as well as similar implementations that were developed by the authors. In

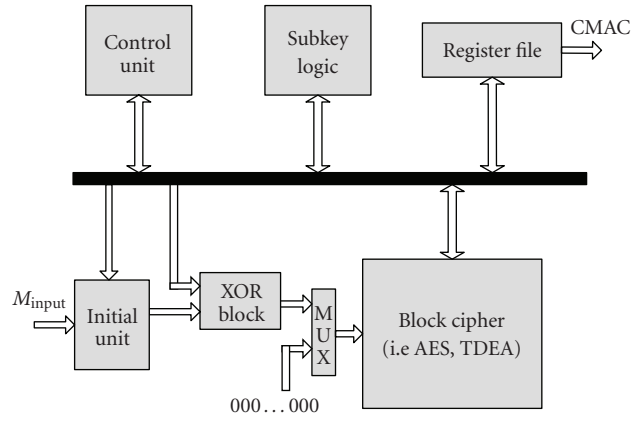

Figure 2: CMAC implementation.

Figure 2, the CMAC architecture is illustrated where one of these implementations regarding the block cipher algorithm has been used.

Any FIPS approved cipher block algorithm can be used instead of AES and TDEA in the same way as it is shown in Figure 1. The interface for the communication of the different components will remain the same. The CMAC implementation consists of several parts.

The overlying software-based system feeds the CMAC scheme (implemented in hardware) with the properly padded (when needed) input blocks M_{input} which are parts of the whole message that is intended to be protected for authentication. The padding process is a quite simple process that can be carried out by the software. This does not imply security problems since there is no secrecy or keys information that should be protected at this point.

The initial unit part is the hardware component responsible for the conditional bitwise operation xor between the M_{input} and one of the subkeys K_1 or K_2.

This part must be implemented in hardware although it could easily be carried out in software and is illustrated in Figure 3. The necessity for hardware implementation of the specific parts arises from the fact that in this unit certain operations concerning the two subkeys exist. If this process is carried out in software then the whole security system is vulnerable since the two subkeys can easily be retrieved. Hardware implementation increases dramatically security, since the K_1 and K_2 values are stored in registers. Furthermore, in order to ensure an even more higher level of security in our implementation, the key K and the two subkeys K_1 and K_2 are not stored in an RAM memory but in shadow registers. This way, we avoid the danger someone who manages to access and read the security system's memory (RAM) to reveal our keys and subkeys that are stored there.

At the initialization phase, the two subkeys K_1 and K_2 must be computed from the main key K. This procedure is carried out by the hardware component "subkey logic" that is illustrated in Figure 4. This component loads the key K from the register file and an already available output of the used block cipher algorithm and in two clock cycles it computes

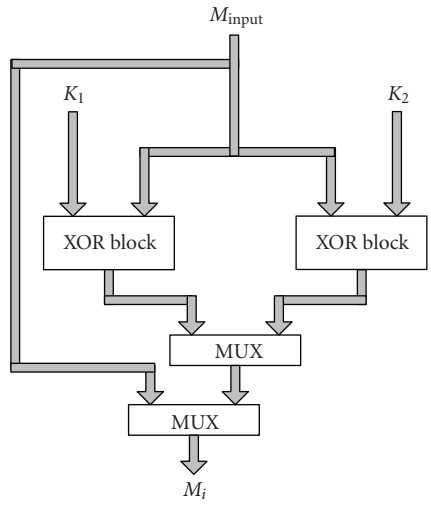

Figure 3: Initial unit.

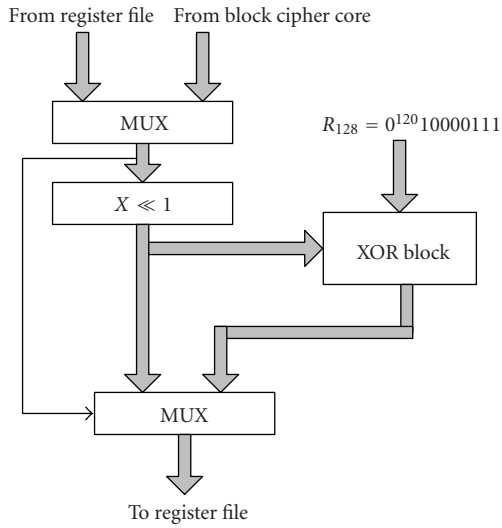

Figure 4: Subkeys logic.

the subkeys K_1 and K_2 that are then saved in the register file. The selected output from the cipher block is the $\text{CIPH}_K(0^b)$, so before K_1 and K_2 computations take place the $\text{CIPH}_K(0^b)$ must be first computed. Thus, this component is enabled only after $\text{CIPH}_K(0^b)$ has been computed and is disabled after two clock cycles resulting to a low power operation. This component is enabled again only if the used main key K has changed for some reason.

The "XOR BLOCK" component is a simple component performing the bitwise xor operation. The "register file" consists of some registers that are used for storing keys, subkeys, the final CMAC value as well as some useful intermediate values.

Finally, the control unit consists of some small counters and is responsible for the correct synchronization of all the components that exist in the CMAC scheme. It is also responsible for the enabling of the several components just when they are needed and disabling them just after that.

6. EXPERIMENTAL RESULTS

The presented CMAC implementation was captured in VHDL and was simulated and verified using the Model Technology's ModelSim Simulator. The designs were fully verified using a large set of test vectors, apart from the test example proposed by the standards.

The synthesis tool used to port VHDL to the targeted technologies was Synplicity's Synplify Pro Synthesis Tool. Simulation of the designs was also performed after synthesis, exploiting the back annotated information that was extracted from the synthesis tool. Xilinx Virtex-II XC2V1000bg575 has been chosen as the target device. This device has 5120 CLB Slices and it also provides fast on-chip memories, Block RAMs, which are ideal for the kind of design used in the CMAC implementation. Detailed analysis of the Virtex–II structure can be found in [10].

Probing of the FPGA's pins was done using a logic analyzer. No scaling frequency technique was followed, selecting one master clock for the system, which was driven in the FPGA from an onboard oscillator. The behavior of the implementation was verified exploiting the large capacity of the FPGA device. Thus, a BIST unit was developed which was responsible for providing predefined inputs to the CMAC and monitoring of the output response. This allowed additionally to cope with high output throughput.

As it was expected the critical path is located at the block cipher core which determines the operating frequency of the whole CMAC implementation. The corresponding throughput can be calculated using the formula

$$\text{Throughput} = \frac{256 * \text{frequency}}{21 * n}. \tag{1}$$

Since in the AES implementation that has been used in the CMAC core the system produces 256 bits of ciphertext data in 21 clock cycles and n is the number of iterations that are required in order to obtain the CMAC value. As we have already said n is specified by the number of blocks that exist in the whole message that we are supplying in the CMAC system.

In our implementation, only the encryption process of the AES core has been used. The operating frequency of the block cipher is about 160 MHz (pipelined stages) and the achieved throughput is over 1.9 Gbps.

Table 1 presents detailed implementation results for the CMAC implementation, where it is observed that the clock frequency is determined by the incorporated block cipher core. The referred value for throughput is the ultimate one assuming that the supplied message consists of only one block. This is a quite usual case. In any other case, the corresponding throughput is calculated by dividing the maximum throughput with the number of blocks that exist in the message that is supplied in the CMAC implementation.

It has to be mentioned that this work is the first presented (at the best of the authors knowledge) implementation regarding the CMAC standard that has been recently adopted by NIST. Since no other implementations exist, no comparisons can be made regarding our implementation. However,

Table 1: Implementation results for CMAC core.

Device utilization (XC2V1000bg575-5)			
Resources	Used	Available	Coverage(%)
CLB Slices	1822	5120	36
BlockRams	10	40	25
Timing report (XC2V1000bg575-5)			
Clock frequency (MHz)		159.21	
Clock cycle (ns)		6.28	
Throughput (Mbps)		1940.90	

in the near future, other implementations are expected to be presented both by academia and commercial companies.

7. CONCLUSION

In this paper, the implementation of a new standard is presented. This standard is related to a new method for producing message authenticating codes (MACs) other than the Hash-based MACs (HMACs). The new MAC process, standardized by NIST in May 2005 and is called CMAC, incorporates the usage of a cipher block algorithm instead of a hash function. The architecture of the CMAC implementation was presented thoroughly. The proposed CMAC system was captured in VHDL and was fully simulated and verified using the Model Technology's ModelSim Simulator. The design was fully verified using a large set of test vectors, apart from the test example proposed by the standards. This is the first implementation of CMAC in literature. The achieved throughput is 1,9 Gbps for a common FPGA family, Xilinx Virtex–II.

Going the presented work a step further, we will try to incorporate an improved version of the AES implementation with higher throughput so as to speed up the whole CMAC construction. This means that effort will be focused on optimization of the internal AES operations which have to do with substitution of memory modules with combinational logic (where possible) and application of pipeline stages. Our research team is already working toward this direction.

Other possible expansion of this work is to investigate applications, where the adoption of the CMAC standard would result in better and more efficient implementations. Toward this direction, we are examining ways to combine the CMAC standard with others like XTS [8] and GCM [9] due to the obvious tend to use AES as a cryptoprocess for many applications. This way, we will manage to propose a complete security scheme with advanced services requiring limited area for integration.

ACKNOWLEDGMENT

This work was supported by the Project PENED 2003 no. 03ED507, which is funded in 75% by the European Union-European Social fund and in 25% by the Greek state-Greek Secretariat for Research and Technology.

REFERENCES

[1] "Recommendation for Block Cipher Modes of Operation: The CMAC Mode for Authentication," 2005, http://csrc.nist.gov/publications/nistpubs/index.html#sp800-38B1998.

[2] "Data Authentication Algorithm (DAA)," FIPS PUB 113, http://www.itl.nist.gov/fipspubs/fip113.htm.

[3] "Data Encryption Standard (DES)," FIPS PUB 46-2, http://csrc.nist.gov/publications/fips/fips46-3/fips46-3.pdf.

[4] "Advanced Encryption Standard (AES)," 2001, http://csrc.nist.gov/encryption/aes.

[5] "The Public Key Infrastructure page," http://www.pki-page.org.

[6] "Digital Algorithm Standard (DSA)," FIPS PUB 186, http://csrc.nist.gov/publications/fips/fips1861.pdf.

[7] "Verifying Electronic Commerce Protocols," http://www.cl.cam.ac.uk/users/lcp/Grants/SET.html.

[8] "The XTS-AES Tweakable Block Cipher," IEEE 1609 approved, under consideration from NIST for FIPS 140-2, http://grouper.ieee.org/groups/1619tmp/1619-2007-NIST-Submission.pdf.

[9] "Recommendation for Block Cipher Modes of Operation: Galois/Counter Mode (GCM) and GMAC," FIPS PUB 800-38D, http://csrc.nist.gov/publications/nistpubs/800-38D/SP-800-38D.pdf.

[10] Xilinx Inc., "Virtex-4 Multi-Platform FPGA Datasheets," San Jose, Calif, USA.