

Cryptography Engine Design for IEEE 1609.2 WAVE Secure Vehicle Communication using FPGA

Chanbok Jeong

School of Electrical Engineering
Graduate school of UNIST

2014

Cryptography Engine Design for IEEE 1609.2 WAVE Secure Vehicle Communication using FPGA

A thesis

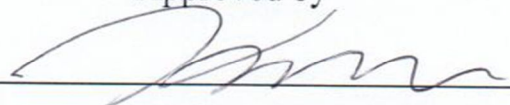
Submitted to the Graduate School of UNIST

in partial fulfillment of the
requirements for the degree of
Master of Science

Chanbok Jeong

12. 15. 2014

Approved by



Major Advisor

Youngmin Kim

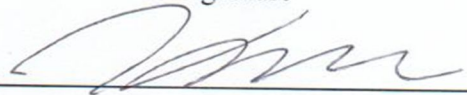
Cryptography Engine Design for IEEE 1609.2 WAVE Secure Vehicle Communication using FPGA

Chanbok Jeong

This certifies that the thesis of Chanbok Jeong is approved.

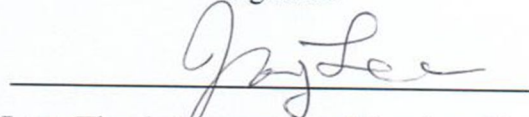
12. 15. 2014

Signature



Thesis Supervisor: Youngmin Kim

Signature



Jongeun Lee: Thesis Committee Member #1

Signature



Seokhyeong Kang: Thesis Committee Member #2

Abstract

In this paper, we implement the IEEE 1609.2 secure vehicle communication (VC) standard using FPGA by fast and efficient ways. Nowadays, smart vehicle get nearer to our everyday life. Therefore, design of safety smart vehicle is critical issue in this field. For this reason, secure VC is must implemented into the smart vehicle to support safety service. However, secure process in VC has significant overhead to communication between objectives. Because of this overhead, if circumjacent vehicles are increased, communication overhead of VC is exponentially increased along the number of adjacent vehicles.

To remove this kind of overhead, we design fast and efficient IEEE 1609.2 cryptography engine using FPGA. This engine consists of AES-CCM encryption, SHA-256 hash function, Hash_DRBG random bit generator, and ECDSA digital signature algorithm and each algorithm is analyzed carefully and optimized with specific technics.

For the AES-CCM, we optimized AES encryption engine. First, we use 32-bit S-box structure to remove 8-bit operation of AES. Second, we employ the key save register file architecture to reduce frequently key expansion operation when input of key value is always same for AES encryption engine. Third, to protect external attacks, we use internal register files to save processed data. Finally, we design parallel architecture for both CBC-MAC and counter in AES-CCM algorithm.

SHA-256 hash function is frequently used in ECDSA algorithm that is significant reason of optimization. So, we use parallel architecture for the preprocessing block and the hash computation block. And, we design latest schedule block to reduce usage of register and combinational logics.

In ECDSA, Hash-DRBG is used to generate key value and signature for vehicle message. To make Hash-DRBG, we use our SHA-256 design much fast generation of random value.

ECDSA is most critical and complex module in our cryptography engine. For this module, we use affine representation of elliptic curve in ECDSA. So, we can replace the prime arithmetic operation by right shift operation and bit operation. And, we implement scalar multiplier to optimize arithmetic operation of ECDSA. This kind of replacement is hardware kindly, so we can reduce complexity of ECDSA hardware design.

To implement all of algorithm in IEEE 1609.2 standard, we use Xilinx Virtex-5 FPGA chip with ISE 14.6 synthesis tool and Verilog-HDL.

Contents

I.	Introduction	1
II.	Related Works	2
III.	IEEE 1609.2 WAVE Protocol Security Service	5
	3.1 AES-CCM	6
	3.1.1 AES	7
	3.1.2 CBC-MAC	8
	3.1.3 Counter	8
	3.2 SHA-256	9
	3.3 Hash-DRBG	9
	3.4 ECDSA	11
IV.	Hardware Implementation of IEEE 1609.2 WAVE Protocol Security Service	15
	4.1 AES-CCM	15
	4.1.1 AES	15
	4.1.2 AES-CCM	17
	4.2 SHA-256	20
	4.3 Hash-DRBG	21
	4.4 ECDSA	26
V.	Experiment	28
	5.1 AES-CCM	28

Result

5.2 SHA-256	
.....	30
5.3 Hash-DRBG	31
VI. Conclusion	33

List of Figures

Figure 2 - 1. Environment and result of [5]

Figure 2 - 2. Overhead of ECDSA using the NIST elliptic curves in IEEE 1609.2 [8]

Figure 3 - 1. Figure 3 - 1. WAVE Communication Stack with OSI 7-Layer [4]

Figure 3 - 2. Entire algorithms in the IEEE 1609.2

Figure 3 - 3. The AES-CCM structure

Figure 3 - 4. Round structure of AES (128 bits key size) [13]

Figure 3 - 5. Standard structure of hash-DRBG [18]

Figure 3 - 6. Key generation of ECDSA

Figure 3 - 7. Signature generation of ECDSA

Figure 3 - 8. Signature verification of ECDSA

Figure 4 - 1. Proposed architecture of key expansion process

Figure 4 - 2. 8 bits data path architecture of AES encryption engine

Figure 4 - 3. The AES-CCM structure

Figure 4 - 4. Round structure of AES (128 bits key size) [13]

Figure 4 - 5. Standard structure of hash-DRBG [18]

Figure 4 - 6. Key generation of ECDSA

Figure 4 - 7. Signature generation of ECDSA

Figure 4 - 8. Signature verification of ECDSA

Figure 4 - 1. Proposed architecture of Hash_DRBG

Figure 4 - 2. Proposed architecture of Hash_df

Figure 4 - 3. Proposed architecture of Hashgen

Figure 4 - 4. Finite state machine of instantiation

Figure 4 - 5. Finite state machine of reseed

Figure 4 - 6. Finite state machine of generate

Figure 4 - 7. Proposed architecture of binary_multiplication

Figure 4 - 8. Proposed architecture of parallel_adder

Figure 5 - 1. RTL synthesis result of AES-CCM

Figure 5 - 2. Functional simulation result of AES-CCM

Figure 5 - 3. Result of functional simulation of SHA-256

Figure 5 - 4. RTL view of the SHA-256

Figure 5 - 5. RTL view of proposed hash-DRBG

Figure 5 - 6. Timing simulation result of hash-DRBG

List of Tables

Table 3 - 1. Key and round of the AES [13]

Table 3 - 2. Secure hash algorithm properties [16]

Table 3 - 3. Definitions for hash-DRBG mechanisms [18]

Table 3 - 4. List of parameters for curve P-224 [20]

Table 3 - 5. List of parameters for curve P-256 [20]

Table 5 - 1. FPGA implementation result and comparison with previous works of AES-CCM

Table 5 - 2. Usage of register files

Table 5 - 3. FPGA implementation result of SHA-256 and comparison with previous works

Table 5 - 4. Synthesis result of proposed hash-DRBG architecture

Nomenclature

AES-CCM	Advanced Encryption Standard-CBC-MAC
AES-CCMP	Advanced Encryption Standard-Counter with CBC-MAC Protocol
ALU	Arithmetic Logic Unit
C	Constant Value
CBC-MAC	Cipher Block Chaining Message Authentication Code
Ch	Chance
DES	Data Encryption Standard
ECC	Elliptic Curve Cryptosystem
ECDSA	Elliptic Curve Digital Signature Algorithm
ECIES	Elliptic Curve Integrated Encryption Scheme
FPGA	Field Programmable Gate Array
GF	Galois Field
Hash-DRBG	Hash-Deterministic Random Number Generator
HMAC	Hash-based Message Authentication Code
ITS	Intelligent Traffic Systems
IVHS	Intelligent Vehicle Highway Systems
LSB	Least Significant Bit
MAC	Message Authentication Code
Maj	Majority
MD	Minimum Delay
MIC	Message Integrity Code
MixColumn	Mix Column

NITSA	National Intelligent Transportation Systems Architecture
S-Box	Substitution Box
SHA	Secure Hash Function
ShiftRow	Shift Row
SubByte	Substitution Bytes
TTA	Telecommunications Technology Association
V	Seed Value
V2X	Vehicle to X
VC	Vehicle Communication
WAVE	Wireless Access in Vehicular Environments

Chapter I

Introduction

Nowadays, many countries enacted or were enacting new traffic safety criteria toward development a new car. Customer and market also demand smart vehicle for driving efficiency, prevent environmental pollution, safety, and convenience. To satisfy these requirements, vehicle communication (VC) is necessary technology for development the smart vehicle or the brand new cars.

To support VC, academy and enterprise push ahead with project and standardization work. For instance, SEVECOM, simTD, working group 5 security of ETSI, and eSafety security working group of eSafety forum in EU. And, in US IEEE 1609 Wireless Access in Vehicular Environments (WAVE) is standardized in order to support VC. It is also standardized in KOREA by the Telecommunications Technology Association (TTA) [1, 2].

But, VC has critical security issue because VC is concerned in safety of vehicle and human in vehicle or other traffics. Therefore, to provide VC service, we must resolve security issue. Actually, abovementioned VC standards or project define security mechanisms to solve this issues. Typical standards is the IEEE 1609.2 WAVE security service. In this standard has 5 security methods these are AES-CCM, SHA, DRBG, ECDSA, and ECIES. But, these security methods yield critical overhead in VC process. In order to propagate VC rapidly, we must reduce security overhead in VC.

In this paper, we propose the crypto engine to reduce security overhead in IEEE 1609.2 WAVE security methods using FPGA. The crypto engine has 4 security methods these are AES-CCM, SHA, DRBG, and ECDSA. To optimize AES-CCM, we use 32 bits S-Box and 32 bits data path to reduce required clock cycles in system, using key-box and saving expanded key to remove iterated key expansion step, internal registers are used to protect the processed data from external attacks, and parallel architecture to improve the throughput of AES-CCM. For the SHA algorithm, we employee LMS to reduce usage of register, embed Xilinx IP adder to improve clock cycles, cutting the critical path of algorithm using registers, and implement preprocessing unlike other papers. For the DRBG, we use our own SHA algorithm to generate the random number and internal finite state machine to improve efficiency of system control. In ECDSA algorithm, that use SHA, DRBG algorithms to make digital signature. So, we use already implemented IPs these are implemented by us. And, special mathematical routines are used to generate and process the elliptic curves for digital signature. That routines is much fitted for hardware implementation [3].

Chapter II

Related Works

The authors of [4] trace the history of VC in the United States and introduce WAVE protocol. In 1991, Intelligent Vehicle Highway Systems (IVHS) are created by the United States of Congress. The goals of IVHS are increased safety, improved congestion, decreased pollution, and saved fuels in traffic infrastructure. This plan was served by the National Intelligent Transportation Systems Architecture (NITSA) after 13 years and it was become to mater plan of the Intelligent Traffic Systems (ITS) in the US. In 2004, an IEEE task group was developing an amendment to the 802.11 standard to include vehicular environments as 802.11p. Another IEEE team, working group 1609, was developing IEEE 1609 standard set that was consisted IEEE 1609.1, IEEE 1609.2, IEEE 1609.3 and IEEE 1609.4. Finally, IEEE 1609.2 and IEEE 802.11p are combined by WAVE for the wireless accessing in vehicular environment. These 5 components in WAVE are described in this paper, very briefly. Especially, security service as IEEE 1609.2 is described with some cryptography algorithms.

As mentioned above chapter 1, VC has critical security issue for the safety of vehicle environments. This issue is analyzed by the authors of [5]. In this article show possibility of cars could be next victim of cyber-attacks. Because, vehicle already contain a huge amount of electronics controlled by software code and hardware. The authors are performed experiment to analyze security of a modern automobile. In the result, they can be hacked to even road test. To hack in to the car, they use OBD-2 connector of experimented-on car, diagnostic connector, and AVR-CAN module to connect CAN protocol in vehicle. And they can be killing engine and malfunction to break system at speeds of up to 40 MPH. Figure 2-1 shows the experiment environment and result of that. This result was published in the Financial Times at March 22th, 2013 [6].

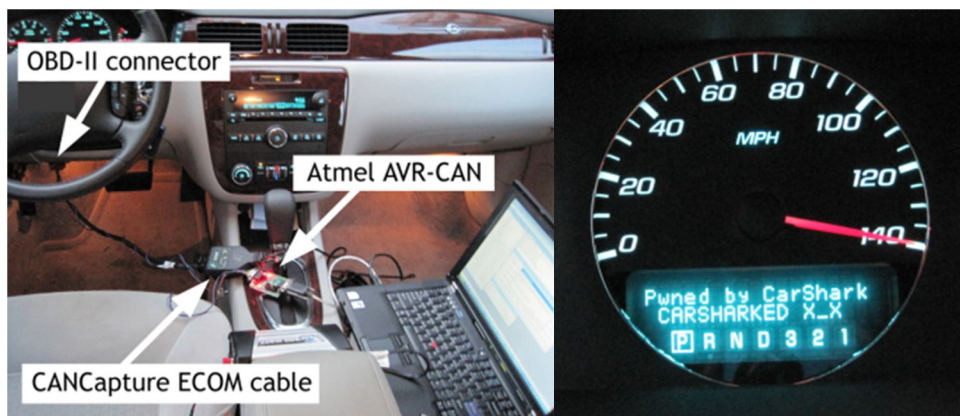


Figure 2 - 1. Environment and result of [5]

According to the authors of [7, 8], the data throughput and minimum delay (MD) limits of 802.11p VC protocol are analyzed. The MD of VC for 27 Mbps data rate of 1000 Bytes payload data is 565.5 μ s. The authors of [8], they claim that ECDSA algorithm in IEEE 1609.2 standard yield overhead in VANET. Because, ECDSA has require a lot of arithmetic operation to generate digital signature and verification of signature. Figure 2-2 shows that overhead of ECDSA rise breaking distance when emergency breaking situation in highway. To remove overhead of ECDSA, they use Montgomery multiplication that can be used for multiplication, inversion, modular operations of key generation, signature generation, and verification.

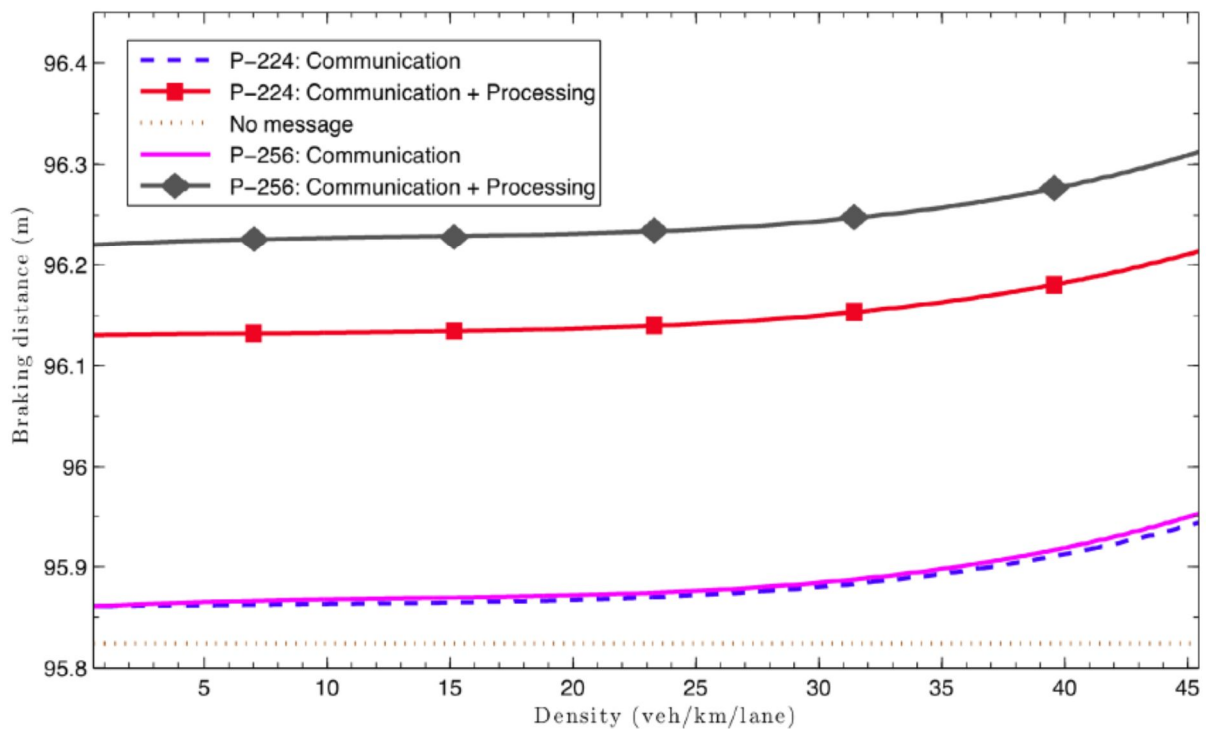


Figure 2 - 2. Overhead of ECDSA using the NIST elliptic curves in IEEE 1609.2 [8]

In [2], the author analyzes about IEEE 1609 and explains security objectives of C2X and IEEE 1609.2. Interestingly, the author analyzes implementation methods of security algorithms in IEEE 1609.2 with pure software, standard smart cards, and FPGAs or ASICs. As a result, FPGA implementation is a much feasible solution. But, in this implementation, they didn't use NIST prime EC curves in IEEE 1609.2.

In [9], they designed AES-CCMP (Advanced Encryption Standard-Counter with CBC-MAC Protocol) FPGA (Field Programmable Gate Array) hardware for IEEE 802.11i-2004, they are

carefully analysis AES-CCM architectures to exploit parallelization of some processes and the design of highly specialized processing modules. They aim to design a fast simple iterative AES-CCMP hardware architecture with low hardware requirements. In this work, Virtex-4 FPGA implementation has 12.259 Efficiency with 149.00 MHz that consist of 1921 FPGA slices and 20 BRAM. Also, this paper compare to software implementation and hardware implementation results. Consequently, software implementation has high frequency and throughput but it has low efficiency than hardware implementation.

In [10], authors have implemented the hash function in SHA-256 module for HMAC (Hash-based Message Authentication Code). To improve the performance, they use four pipeline stages for hash computation of four different input messages, and a carry save adder.

In [11], a compact FPGA processor for the SHA-256 algorithm is implemented without preprocessing unit. To optimize the SHA-256 hash function, they have proposed several techniques, such as minimization of the critical path, reducing of the memory access by using data reuse, and a specific 4-input arithmetic logic unit (ALU).

The authors of [12], they implement elliptic curve cryptosystem (ECC) with optimized arithmetic operation logic. Scalar multiplication operation is the most time consuming operation in ECC. But, this logic is used frequently to generate signature and verification. Authors implement multiplier and inverter co-processor on F_2^m with binary shifter. To implement co-processor, authors use a Xilinx Virtex-2 XC2V1000-4FG456 FPGA chip with VHDL. As a result, eight percent slices in FPGA is used to calculate multiplication with 166 MHz. For the inverter, thirty-five percent slices in FPGA is used to calculate inverter with 115 MHz.

Chapter III

IEEE 1609.2 WAVE Protocol Security Service

An IEEE 1609.2 is part of an IEEE 1609 WAVE Protocol (WAVE) to provide security services for application and management messages. As mentioned above, the WAVE is defined for safety, comfort and efficiency in vehicle. For these reasons, the WAVE has four sub-standards and also an IEEE 802.11p physical layer to support Vehicle to X (V2X) communication. An IEEE P1609.1, the first sub-standard, defines an application. An IEEE P1609.3 and An IEEE P1609.4 define network service and multi-channel operation, respectively [1, 4]. Figure 3-1 shows the WAVE communication stack. In this figure, we can show the relationship of each IEEE 1609 sub-layer.

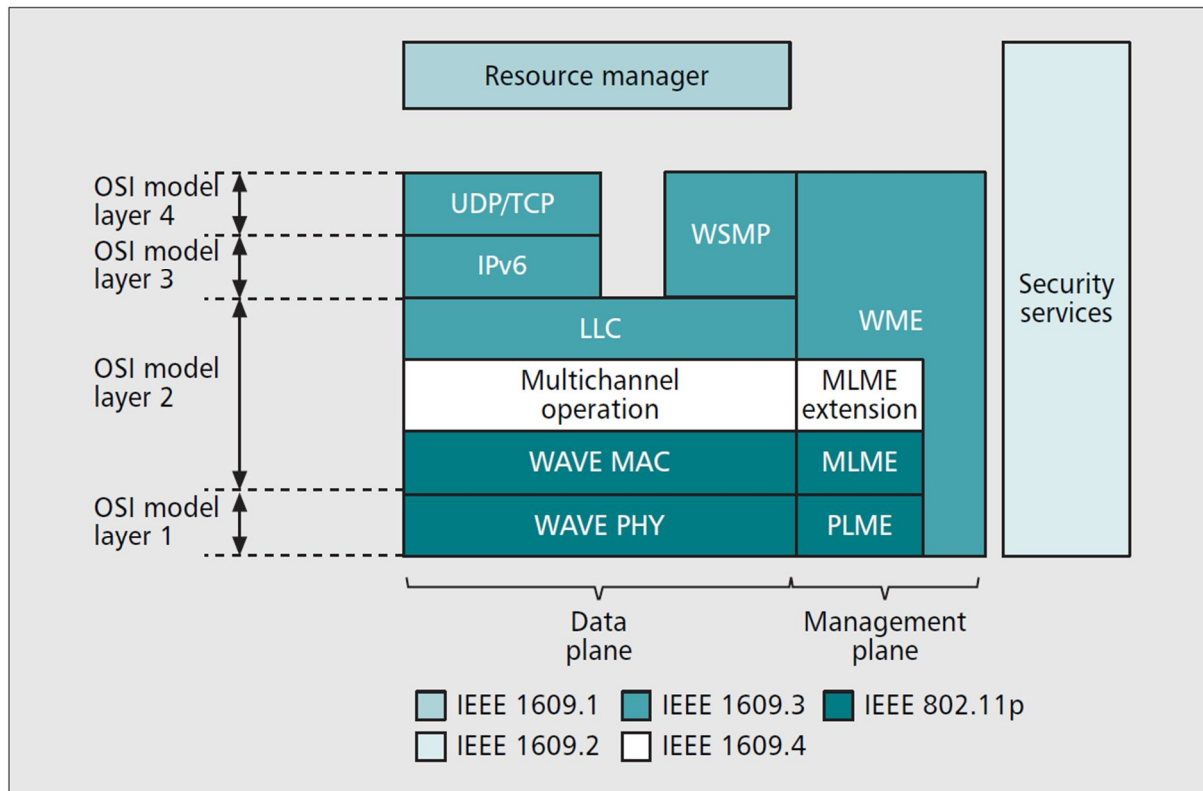


Figure 3 - 1. WAVE Communication Stack with OSI 7-Layer [4]

The IEEE 1609.2 is consisted of 5 algorithms these are Advanced Encryption Standard – CBC-MAC (AES-CCM), Secure Hash Function (SHA), Hash-Deterministic Random Number Generator (Hash-DRBG), Elliptic Curve Digital Signature Algorithm (ECDSA) and Elliptic Curve Integrated Encryption Scheme (ECIES). Connection of each algorithm in the IEEE 1609.2 is represented in figure 3-2. In this chapter, we explain each secure algorithm in the IEEE 1609.2, exclude the ECIES.

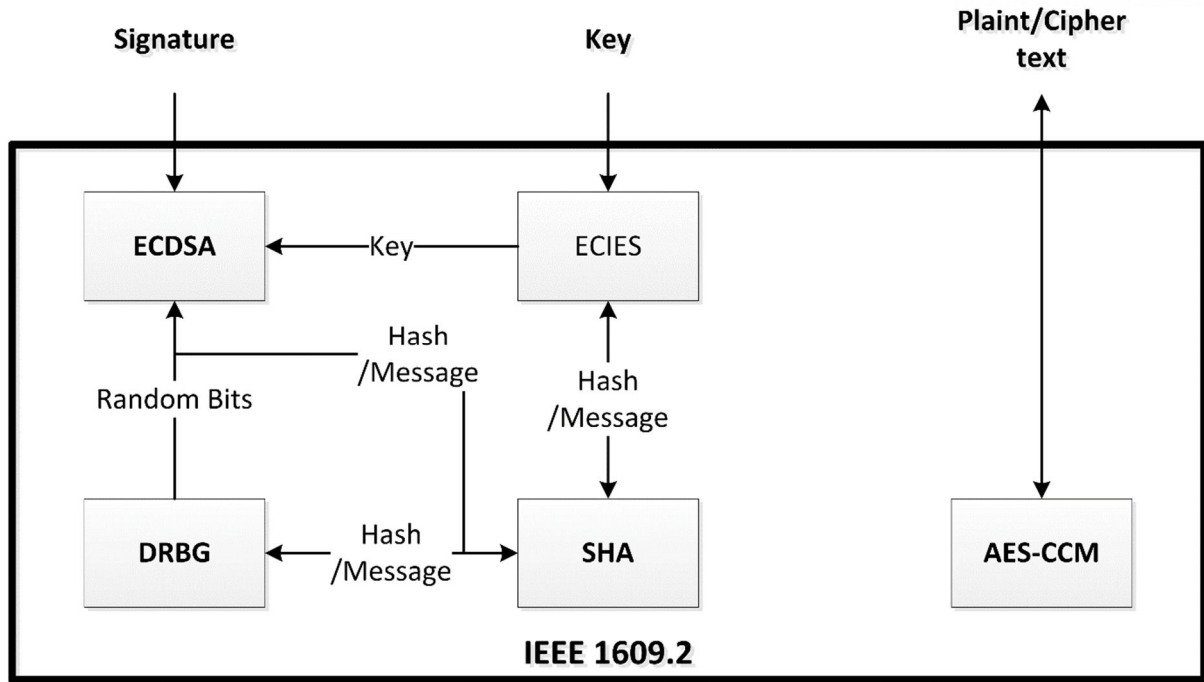


Figure 3 - 2. Entire algorithms in the IEEE 1609.2

3.1 AES-CCM

The AES-CCM is a unique symmetric key block cipher algorithm in the IEEE 1609.2 to encrypt and decrypt data also defined by a NIST SP 800-38C. It is also used for other wireless communication protocol like an IEEE 802.11 to encrypt the data. The AES-CCM is consisted of an AES-CBC-MAC and a Counter using an AES algorithm as shown in figure 3-3 [1, 9].

Figure 3 - 3. The AES-CCM structure

The input of the AES cipher (Plaintext) is formatted to 128-bit formatted block by the formatting function. This block is processed by a Cipher Block Chaining Message Authentication Code (CBC-

MAC) and a Counter to encrypt the Plaintext. Both results are XOR-ed to make a Message Integrity Code (MIC) data for authentication of data. And, the ciphertext is generated by the Counter. The CBC-MAC and the Counter are processed using the AES algorithm for each step.

3.1.1 AES

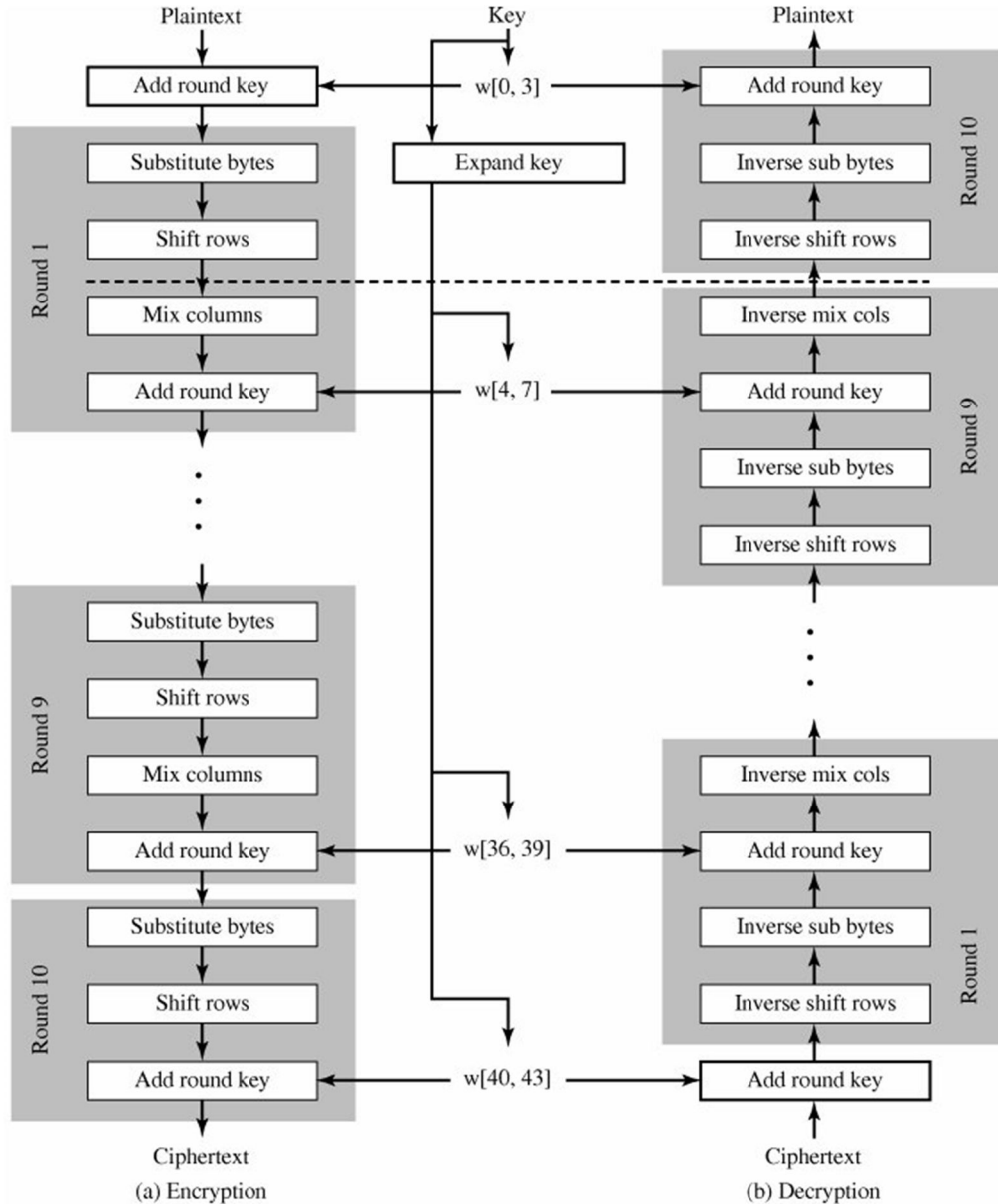


Figure 3 - 4. Round structure of AES (128 bits key size) [13]

The AES is encryption standard based on symmetric key block cipher. The AES is defined as FIPS-197 by NIST to substitute a Data Encryption Standard (DES) encryption algorithm. Because, the DES has been reported to have performance limitation and security weakness issues in academic research.

To develop the AES, NIST require the following standard criteria: minimum system resource usage, open source algorithm, ability for hardware and software implementation, robustness to any security attack, low complexity for encryption calculation, and implementation in any system environment.

The AES has three variable lengths of 128, 192, and 256 bits for an encryption key and input block size is 128-bit. The encryption key is expanded by a key expansion process that expand the key obey the length of key as shown table 3-1. The expanded keys are used to encryption the plaintext at each round [13, 14].

Table 3 - 6. Key and round of the AES [13]

Input key size	4 words / 16 bytes	6 words / 24 bytes	8 words / 32 bytes
Expanded key size	44 words / 176 bytes	52 words / 208 bytes	60 words / 240 bytes
Round	10	12	14

To encrypt the plaintext, the AES has round structure equal to figure 3-4. First step of round is substitution bytes (SubByte) step that substitute a byte-to-byte of plaintext or previous encrypted data using a substitution box (S-Box). The shift row (ShiftRow) step shifts the substitution step's data. Input of ShiftRow is shifted compliant with index of row in a processing block. The mix column (MixColumn) step permutes bit data in column of the processing block using Galois Field ($GF(2^8)$). Final step is add round key that operates bit-wise XOR for the mixed column results with expanded key value. These step are same during round 1 and 9 except first and final round.

3.1.2 CBC-MAC

The CBC-MAC step in AES-CCM generates message authentication code (MAC) using chaining block cipher method with the AES encryption. To make chaining block, the CBC-MAC is iterated encryption using AES and XOR operation with previous chaining result. At the end, a chaining block is became MAC that result of AES-CCM [15].

3.1.3 Counter

Ciphertext of AES-CCM is generated in counter step. In this step, counter blocks are generated for each plaintext block. The counter blocks are consisted with nonce data from an input of AES-CCM and simple counter value. The first block in ciphertext is XOR-ed with MAC data from the CBC-MAC. This value is used to checksum of the ciphertext. So, length of ciphertext is same to plaintext without MAC length [15].

3.2 SHA-256

SHA-256 is kind of the one-way hash function that is published as FIPS 180-3 by the NIST and included in SHA-2 family. Through table 3-2, we can show the properties of SHA-256. SHA-256 can has up to 2^{64} bits input data. This input data is processed by preprocessing and hash computation. Preprocessing stage is padding a message, parsing the padded message into $N \times 512$ -bit blocks and setting initialization values. When preprocessing stage is completed, hash computation stage generates a message schedule blocks using the padded message and a series of hash value using the message schedule blocks. Consequentially, the output of SHA-256 is generated as the message digest. The SHA-256 has 256 bits message digest. The message digest always has unique value depending on the input message [16][17].

Table 3 - 7. Secure hash algorithm properties [16]

Algorithm	Message Size (bits)	Block Size (bits)	Word Size (bits)	Message Digest Size (bits)
SHA-1	$< 2^{64}$	512	32	160
SHA-224	$< 2^{64}$	512	32	224
SHA-256	$< 2^{64}$	512	32	256
SHA-384	$< 2^{128}$	1024	64	384
SHA-512	$< 2^{128}$	1024	64	512

In SHA-256, all operation based on 32-bit unit as a word. Also, all arithmetic operations are based on 2^{32} . The SHA-256 has six logical functions, these are Ch, Maj, $\Sigma_0^{256}(x)$, $\Sigma_1^{256}(x)$, $\sigma_0^{256}(x)$ and $\sigma_1^{256}(x)$, eight working variables, a, b, c, d, e, f, g, and h and two temporary variables, T1 and T2. The logical functions, working variables and temporary variables are used to process hash computation stage [16][17].

3.3 Hash-DRBG

The hash-DRBG is one of the DRBG mechanisms based on hash functions. As shown in figure 3-5 the hash-DRBG has many inputs to generate the random bits these are an entropy input, a nonce, a personalization string, an additional string, a seed length (seedlen), and a security strength. But, the personalization string, and the additional string are optional input to generate the random bits. A reseed count in figure 3-5 isn't input of the hash-DRBG that is initialized at initialization step and

reseed step.

The hash-DRBG has 3 steps to make random bits. These are an instantiate, a generate algorithm, and a reseed. The inputs of instantiate algorithm are used to perform initialization for values of hash-DRBG to generate the random number. These input are concatenated and entered to a hash_df that sub algorithm to create instantiation values. The initialization values of in this algorithm are a reseed counter, seed value (V), and constant value (C). V and C are generated by the hash_df and reseed counter is just assigned value '1'.

The generate algorithm generate the random bit using V, C, the additional input and the seedlen. As mentioned above, V and C is generated by instantiation algorithm or regenerated values from algorithm itself. In this algorithm use SHA-256 and hashgen algorithm, sub algorithm of DRBG, are used to create random bit (returned_bit). In this algorithm, for the robustness of the security strength of the algorithm, V and C are regenerated during the generate algorithm and these value are feed to algorithm itself to generate the next random bits, V, and C.

Similarly to instantiate algorithm, the reseed algorithm generate the V and C for generate the random bits. Methodology also like the instantiate algorithm. However, the reseed algorithm is called when reseed counter is reached to the number of requests between reseeds in table 3-3 and reseed counter is initialized to 1 because the V and C doesn't have enough security strength for robustness. For the reseed algorithm, additional input can be entered to reseed algorithm [18].

Table 3 - 8. Definitions for hash-DRBG mechanisms [18]

	SHA-1	SHA-224 and SHA- 512/224	SHA-256 and SHA- 512/256	SHA-384	SHA-512
Output block length	160	224	256	384	512
Maximum entropy input length	$\leq 2^{35}$ bits				
Seed length for Hash-DRBG	440	440	440	888	888
Maximum personalization string length	$\leq 2^{35}$ bits				
Maximum additional input length	$\leq 2^{35}$ bits				
Number of requests between reseeds	$\leq 2^{48}$				

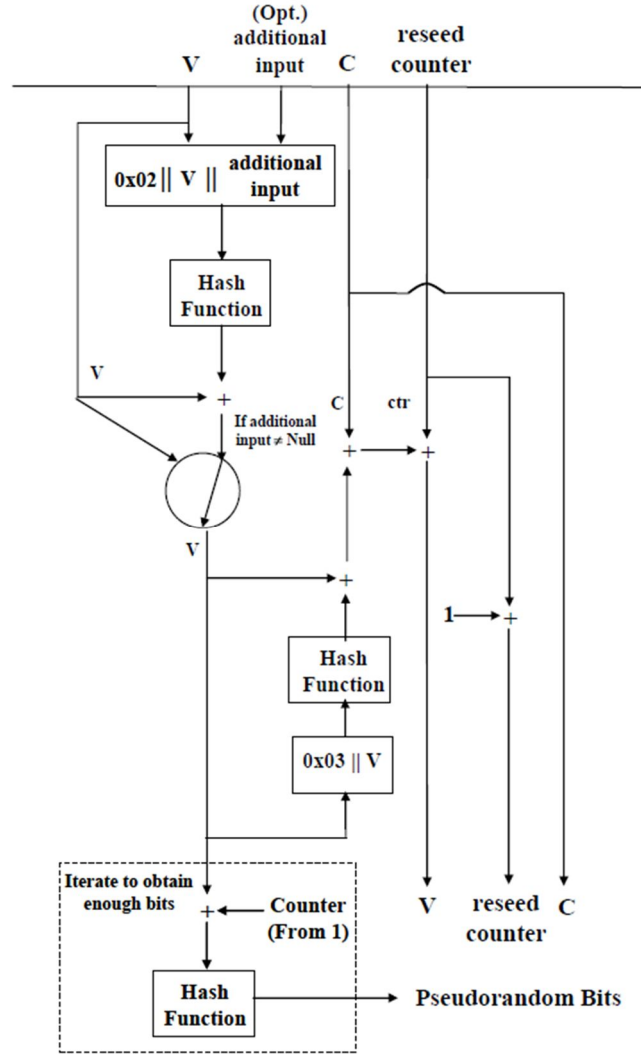


Figure 3 - 5. Standard structure of hash-DRBG [17]

3.4 ECDSA

ECDSA is typical digital signature generation algorithm in WAVE. In ECDSA algorithm, all values are defined over points on elliptic curves. If some attacker try to break the ECDSA algorithm, he/she solve the discrete logarithm problems similarly to DSA. The elliptic curve is defined in Equation 3.1. In this equation has parameters a and b these are constants [19].

$$y^2 = x^3 + ax + b \quad (\text{Eq. 1})$$

For the ECDSA in WAVE protocol, it use special recommended NIST elliptic curves, NIST P-224 and NIST P-256. These curves defined over prime fields. Table 3-4 and table 3-5 shows the parameter of each NIST prime curve, respectively [20].

Table 3 - 9. List of parameters for curve P-224 [20]

Parameter	Value
p	26959946667150639794667015087019630673557916260026308143510066298881
n	26959946667150639794667015087019625940457807714424391721682722368061
SEED	0x bd713447_99d5c7fc_dc45b59f_a3b9ab8f_6a948bc5
a	-3
c	0x5b056c7e_11dd68f4_0469ee7f_3c7a7d74_f7d12111_6506d031_218291fb
b	0xb4050a85_0c04b3ab_f5413256_5044b0b7_d7bfd8ba_270b3943_2355ffb4
G_x	0xb70e0cbd_6bb4bf7f_321390b9_4a03c1d3_56c21122_343280d6_115c1d21
G_y	0xbd376388_b5f723fb_4c22dfe6_cd4375a0_5a074764_44d58199_85007e34

Table 3 - 10. List of parameters for curve P-256 [20]

Parameter	Value
p	115792089210356248762697446949407573530086143415290314195533631308867097853951
n	115792089210356248762697446949407573529996955224135760342422259061068512044369
SEED	0xc49d3608_86e70493_6a6678e1_139d26b7_819f7e90
a	-3
c	0x7efba166_2985be94_03cb055c_75d4f7e0_ce8d84a9_c5114abc_af317768_0104fa0d
b	0x5ac635d8_aa3a93e7_b3ebbd55_769886bc_651d06b0_cc53b0f6_3bce3c3e_27d2604b
G_x	0x6b17d1f2_e12c4247_f8bce6e5_63a440f2_77037d81_2deb33a0_f4a13945_d898c296
G_y	0x4fe342e2_fe1a7f9b_8ee7eb4a_7c0f9e16_2bce3357_6b315ece_cbb64068_37bf51f5

ECDSA has three process these are key generation, signature generation, and verification of signature of WAVE message. Each process use scalar addition, scalar multiplication, scalar inversion, point addition, and point multiplication.

Figure 3-6 shows the key generation process of ECDSA. In this process, private key (d) is generated by Hash-DRBG. Public key is generated by multiplication d and point G(x, y).

Figure 3-7 is process of the signature generation for the message. Signature is consisted with r and s. To generate r, a random value k is generated by the Hash-DRBG and performed point multiplication with G. At this time, r is reduced by modular p and q operations. Another signature pair s is generated using r and k. In s, hashed WAVE message is contained with k, private value d, and r. s is also reduced by modular q.

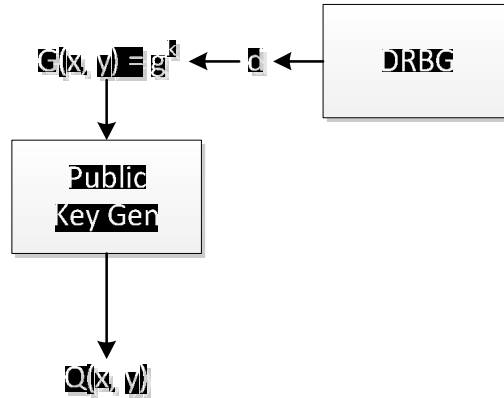


Figure 3 - 6. Key generation of ECDSA

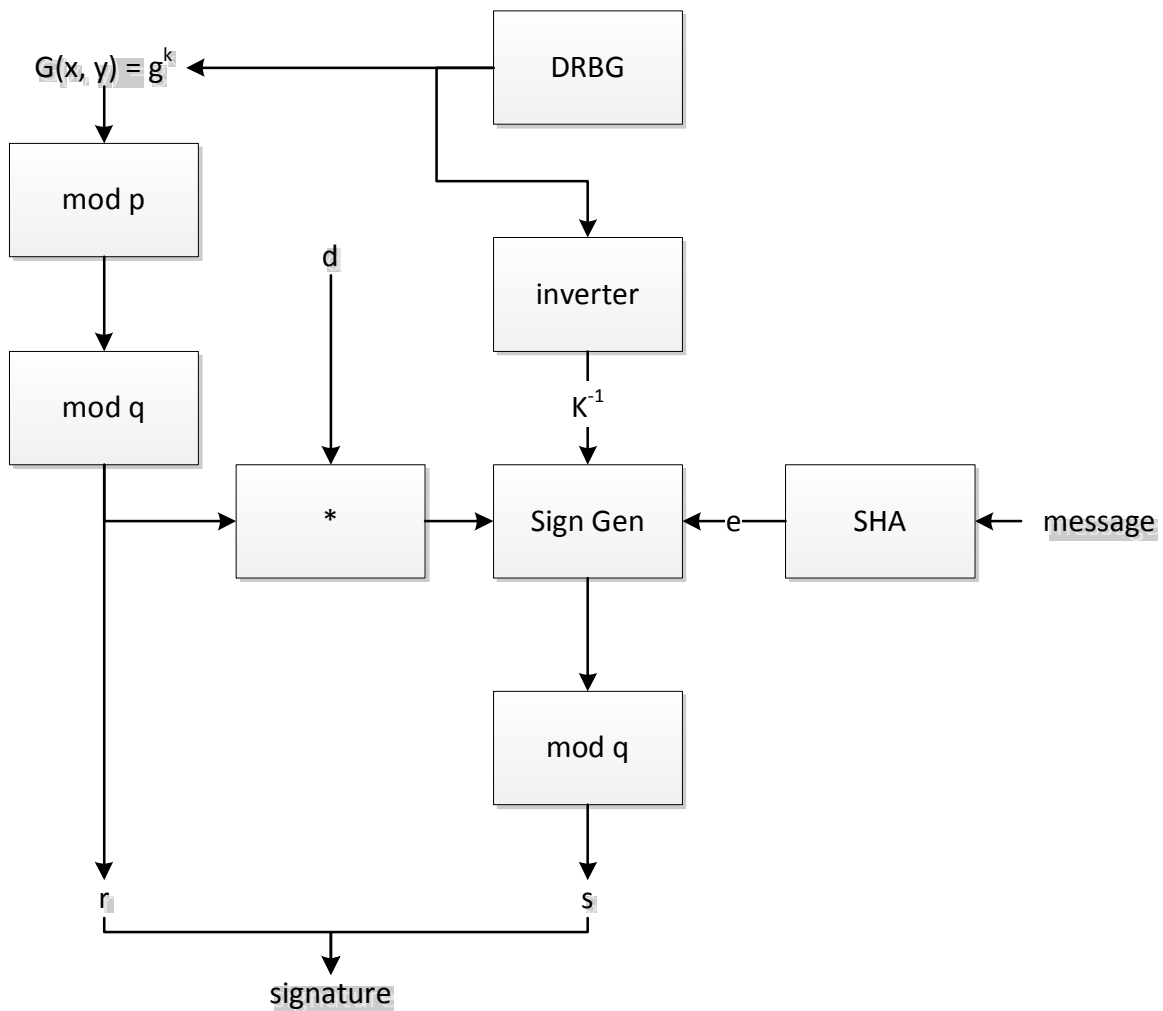


Figure 3 - 7. Signature generation of ECDSA

In figure 3-9, we can show the signature verification process of ECDSA. To verification signature, signature is hashed using SHA, as e , and separated to s and r by integer checker. s is became u_1 using

modular p inversion and scalar multiplication with e and reduced by modular q . r is also became u_2 using scalar multiplication with w and modular q . v_1 and v_2 are performed point multiplication with G and Q , respectively, and point addition each other with modular reduction p and q . It is became v that is compared with r . If r and v is equal than signature is valid and message of the signature is accepted to system. Else, signature is invalid and message is thrown out [20].

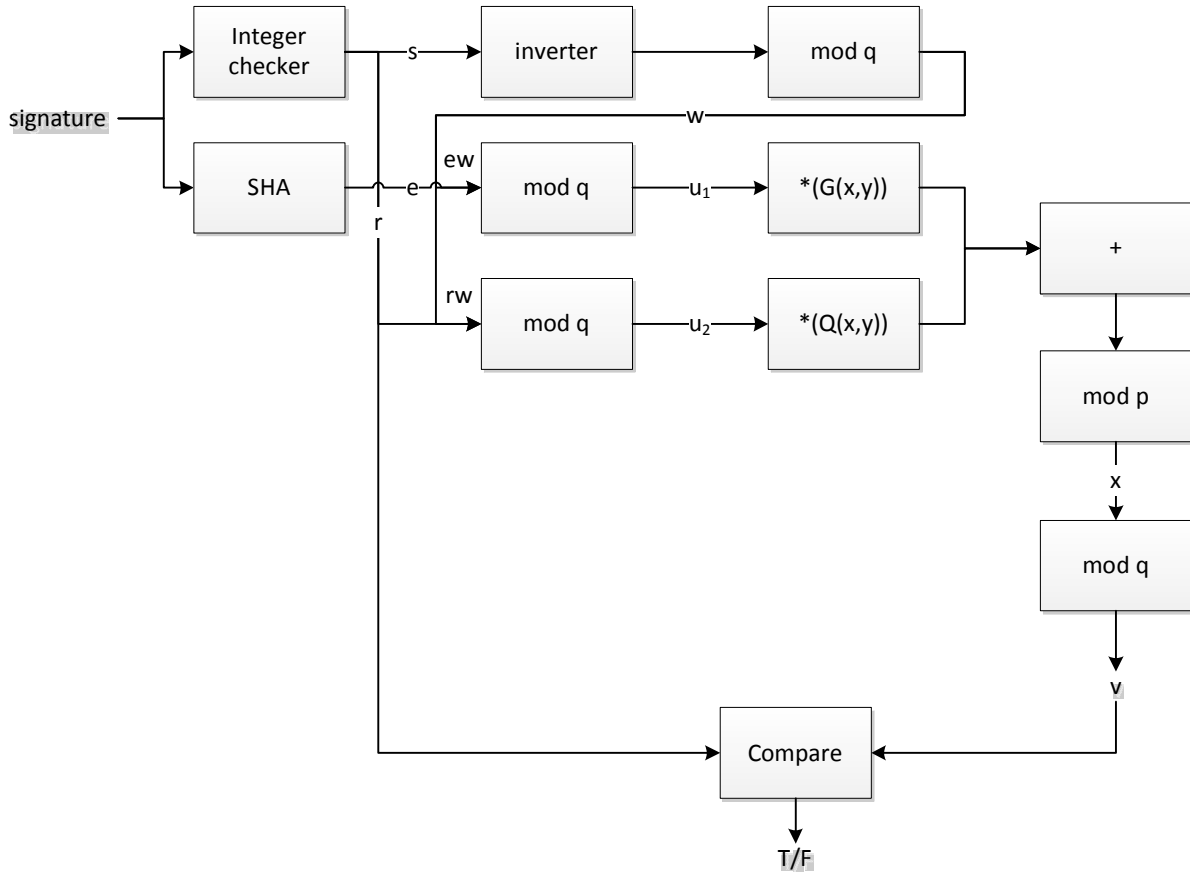


Figure 3 - 8. Signature verification of ECDSA

Chapter IV

Hardware Implementation of IEEE 1609.2 WAVE Protocol Security Service

In this chapter, we explain how to implement our crypto engine for the security algorithms in IEEE 1609.2 WAVE. To design cryptography standard algorithm, we need to change that is the software kindly described cryptography algorithm in the standard document to hardware structure for the RTL coding. There has many issues to overcome to describe the cryptography algorithm. First, because of it has sequentially described algorithms, to change the parallel structure for the faster circuit design. Second, many for, while, and if statements are used to describe cryptography algorithm. To implement these statements to RTL, many big comparators are used to control routines of the number of iteration or to finish the if statements. Third, reuse variable to calculate variable itself. Fourth, too long arithmetic combination logic path are used to generate cryptography result. Fifth, too wider and many operand in arithmetic operation. It is yield huge critical paths for the arithmetic operation. To overcome implementation issues, we use parallel architecture, analysis data dependency between variables, using bit operations to make simple arithmetic logics, register insertion to long combination logic to cut the critical path, and employment Xilinx adder/substracter IP core to improve arithmetic combination logic [21].

4.1 AES-CCM

4.1.1 AES

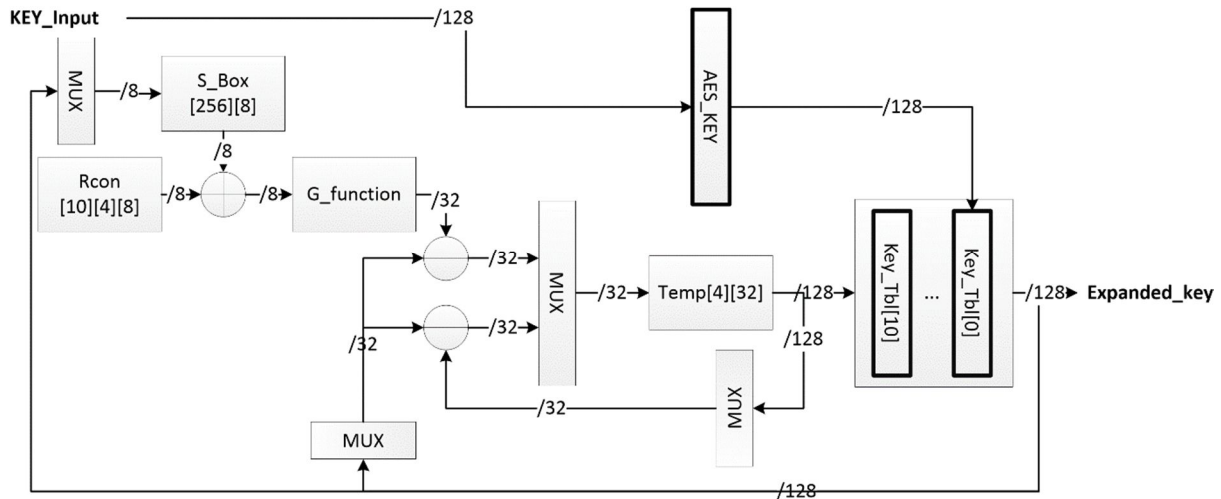


Figure 4 - 9. Proposed architecture of key expansion process

To make best performance of AES-CCM, we analyze and implement AES algorithm very carefully because of AES is frequently used in AES-CCM. AES has key expansion and AES engine module to generate encryption result. Key expansion module is performed expansion operation of input key to expanded key. Expanded keys are used to process of each round in AES. In this paper, 128 key length of AES engine is implemented.

In the figure 4-1, there has 8 bits operation data paths and these lead to delay in key expansion path. However, because of key expansion is executed only one time during same input key, we didn't optimize the data path. But, it is required key saving operation. So, we employee 128 bits x 11 key table (Key_Tbl) to save expanded key values. The substitution box (S_Box) is same module in proposed AES engine. The input and output length of key expansion is same 128 bits.

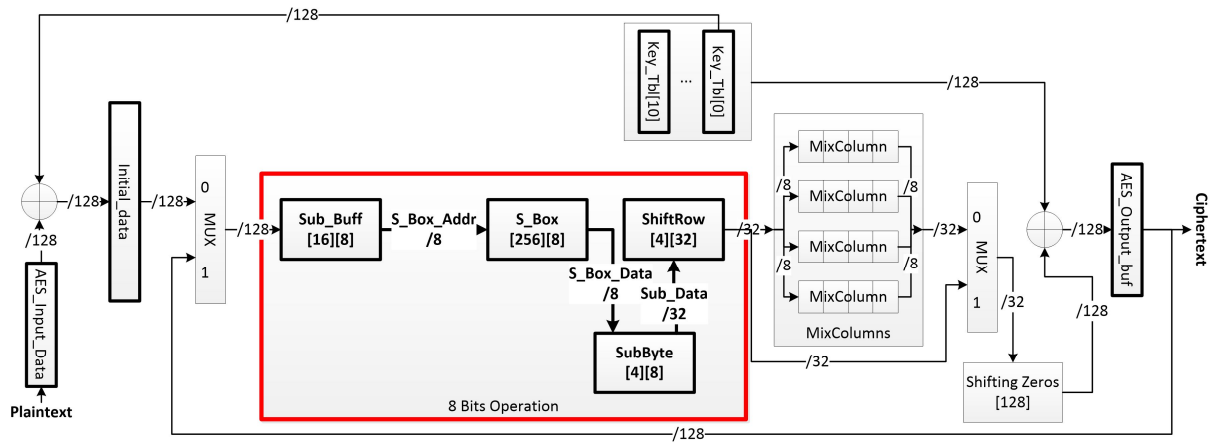


Figure 4 - 10. 8 bits data path architecture of AES encryption engine

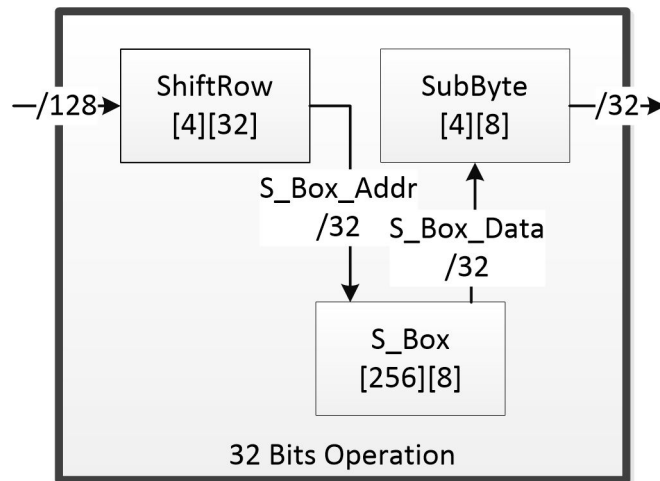


Figure 4 - 11. Proposed 32 bits data path architecture alter the 8 bits data path

Figure 4-2 shows the AES encryption engine structure with 8 bits data path. Input length of the AES encryption engine is 128 bits. Output ciphertext length is also 128 bits. In first round, add round key operation is performed to generate initial data. After that, second round through tenth round is performed to generate encryption result, sequentially. In these rounds, perform substitution byte, shift row, mix column, add round key, sequentially. But, in figure 4-3 that is proposed architecture, shift row (ShiftRow) is performed before the substitution byte (SubByte). Because, likes sequence of figure 4-2, changing row data (ShiftRow) to column data (mix column) is required additional buffer to transform. But, likes sequence of figure 4-3, changing row data to byte data (SubByte) and byte data to column data can be transform without the data buffer. Because, byte data operation is performed likes buffer. And, we consider that 32 bits data substitution using substitution box (S_Box) and 32 bits data is considered order of mix column with already row-wise shifted bus interface using wire index syntax in Verilog-HDL. By the proposed 32 bits operation, we can reduce 4 clock cycles in each round and remove substitution buffer in 9 bits operation data path.

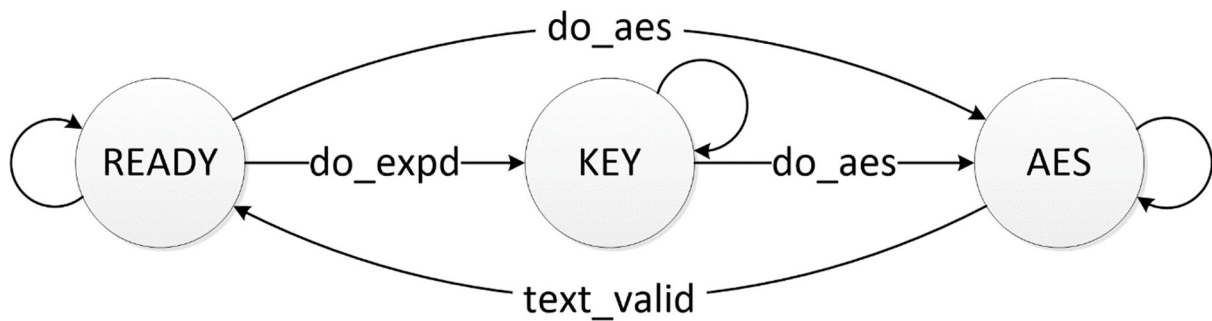


Figure 4 - 12. Finite state machine of AES encryption engine

Figure 4-4 shows the finite state machine (FSM) of AES module. To control key expansion and AES encryption engine, control signal is generated in FSM. Because, key expansion and AES encryption engine have different round counters and timing of beginning process (do_exped, do_aes) and final process (text_valid).

4.1.2 AES-CCM

In the figure 4-5, we can show the proposed architecture of AES-CCM. It has parallel structure for counter and CBC-MAC process. AES-CCM has nonce, plaintext, and key input signal and ciphertext and MAC data are output signal. Formatted block process in figure 4-5 generate formatted block from plaintext data into the 128 bits data blocks. But, the first block is consisted with the nonce, the length of payload and the length of MAC. For the formatted block process, we use case statement with modular to length of plaintext with 128. As shown in figure 4-5. (a), counter process performs AES

with a 128-bits counter block that is generated with nonce and counter value as mentioned above. Except first encrypted counter block, all the encrypted counter block XOR-ed with formatted block. First encrypted counter block is XOR-ed with final data of CBC-MAC that is MAC data for the plaintext.

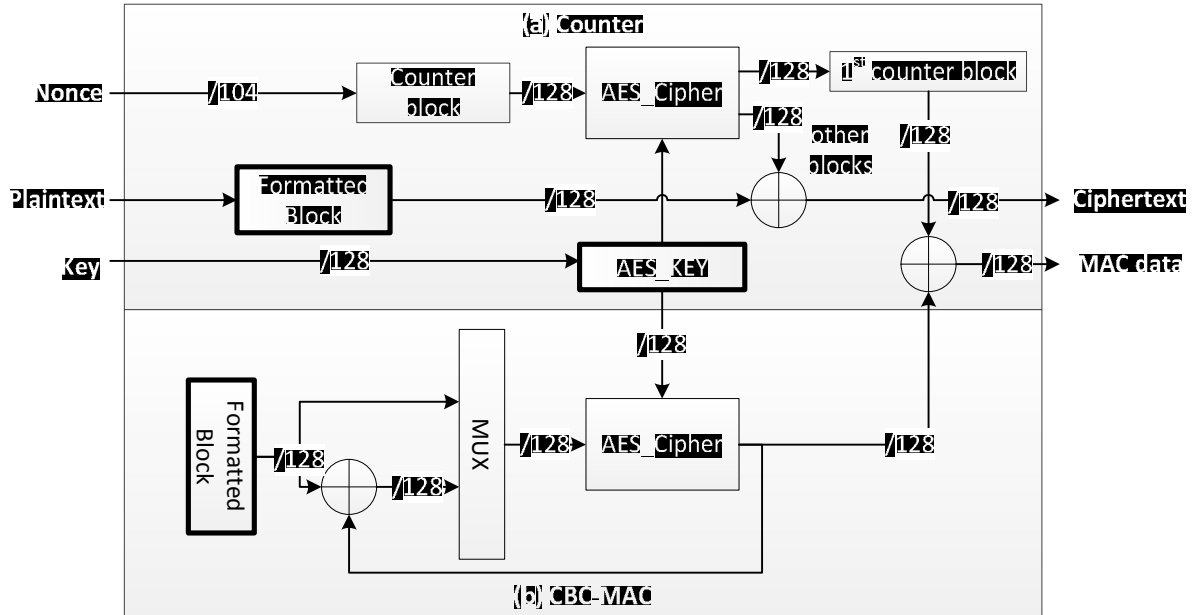


Figure 4 - 13. Proposed architecture of AES-CCM

Figure 4-5. (b) is architecture of CBC-MAC. It perform chaining to previous AES encrypted chaining block and current formatted block using XOR operation. Therefore, it has strong data dependency current and previous block to generated chaining block. However, it is performed with counter process, simultaneously. Using this method, we didn't wait CBC-MAC result to MAC data. That means, we can save huge clock cycles in our system.

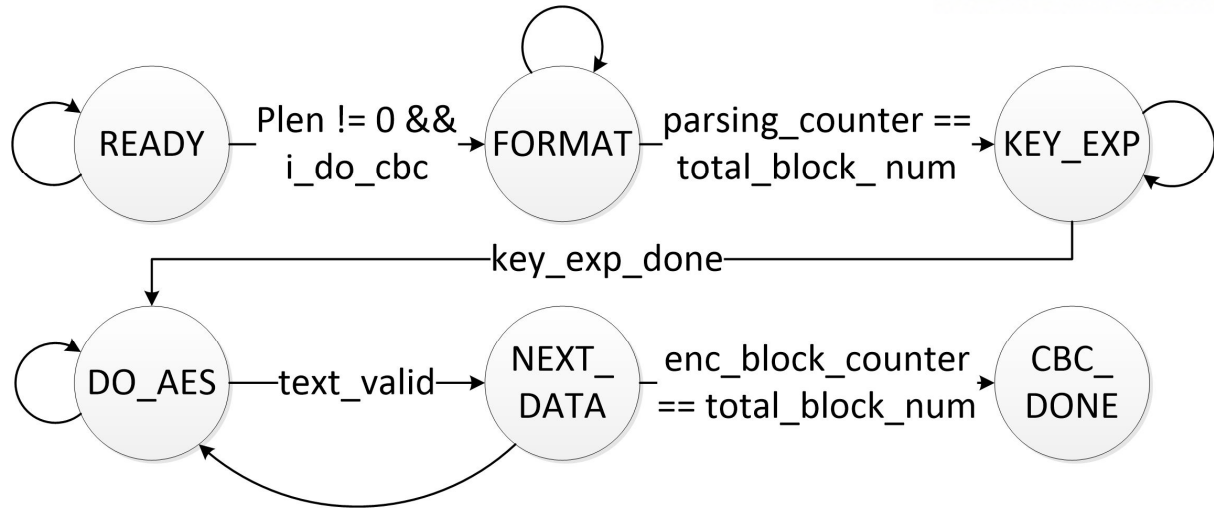


Figure 4 - 14. Finite state machine of AES-CCM

To control AES-CCM, we need to generate control signal to control each module in AES-CCM such as figure 4-6. This FSM start from READY state. If enable signal (i_do_cbc) is entered into the system and the length of plaintext is not zero than state transits to FORMAT state. In FORMAT state, system make formatted block data using formatted block process. FORMAT state is remained until $parsing_counter$ is not equal to $total_block_num$. If two values are equal to each other that means all plaintext is formatted in to the block. Next state as KEY_EXP is performed key expansion process in AES encryption engine until finish of key expansion (key_exp_done). In the DO_AES state, AES engine encrypt the counter block on counter process or the chaining block in the CBC-MAC. This operation can controlled just one state of FSM because encryption timing of both counter and CBC-MAC process is same. If DO_AES state is done than state transited to NEXT_DATA to read next formatted data for the next encryption process. If $enc_block_counter$ and $total_block_num$ are same than state is transited to CBC_DONE and system is terminated. That means, there are no remaining block need to encrypt because of the number of encrypted block number is same to the number of formatted block.

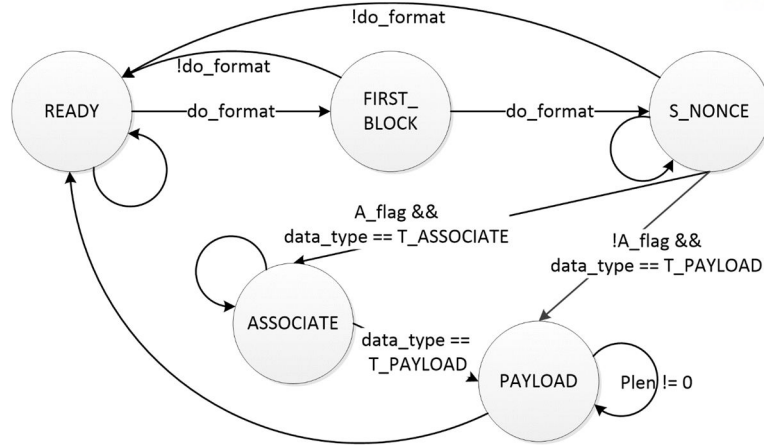


Figure 4 - 15. Finite state machine of formatting function

Figure 4-7 is FSM of formatting function that is sub FSM of AES-CCM FSM. First state READY is transited to FIRST_BLOCK when start signal (do_format) is entered in the system. In FIRST_BLOCK state generate flag byte of first formatted block using the length of plaintext and the length of MAC. After that, state is transited to S_NONCE. Until these transition, do_format must be high. S_NONCE state formatting nonce state into the formatted block and transited to ASSOCIATE state if it A_flag is true and data_type is equal to T_ASSOCIATE. If not, state is transited to PAYLOAD state. In ASSOCIATE and PAYLOAD data perform formatting function according to the data_type. PAYLOAD state is transited to READY state, if Plen is equal to zero.

4.2 SHA-256

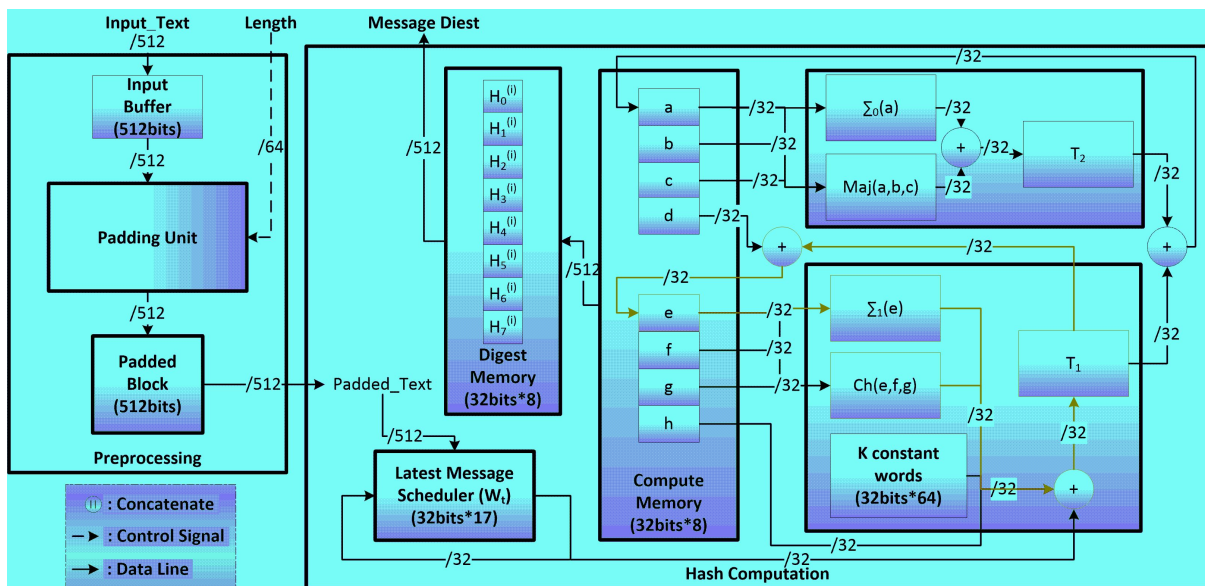


Figure 4 - 16. Proposed architecture of SHA-256

To make fast and efficiency SHA-256, we are consisted parallel preprocessing and hash computation structure like the figure 1 that is proposed structure of SHA-256. In preprocessing stage, the input data is parsed in the 512 padded blocks. The last padded block is consisted 448 bits message that is concatenated 1bit 1 value and sequential 0 values, and length. The padded blocks in preprocessing stage are used sequentially. So, this stage must be completed before the hash computation stage.

When preprocessing is completed than block scheduler prepare the message schedule blocks (W_t). W_t are used to calculate the eight working variables (Compute Memory) and the two temporary variables. Therefore, they are must be prepared before the computation of a compute memory and the two temporary variables to generate the message digest blocks.

To prepare W_t , we are employed a Latest Message Block Schedule module. Actually, W_t from 0 to 15 are equal to corresponding the padded blocks, but W_t from 16 to 63 are needed calculation using previous W_t . So, the Latest Message Block Schedule module make latest W_t . And this operation is one step ahead because the other hash computation operations are required this latest W_t .

In figure 1, we are drawn critical path of hash computation using red path. This path has data dependency to calculated variable itself. And, many 32-bit add operations are also affect delay of the path. To reduce delay of this critical path, we are used adder/substracter IP logic that is provided Xilinx ISE tool [20]. That can be reduced delay by the add operations with some latency.

4.3 Hash-DRBG

As mentioned above, hash-DRBG is generated to the pseudo-random bits (RB) using hash function. Figure 4-9 shows proposed hash-DRBG architecture for crypto engine of IEEE 1609.2. Hash-DRBG has 3 main modules (Instantiation_state, Hash_DRBG_Generate, and Hash_DRBG_Reseed) and 3 sub modules (Hash_df, SHA-256, and Hashgen) to generate random value.

First main module is the Instantiation_state that perform initialization to seed value as V, constant value as C, and reseed counter (reseed_counter) these are output signal of the Instantiation_state. The Instantiation_state has 5 inputs these are Entropy_input, nonce, Personalization_string, Seedlen and Security_strength to initiate V, C and Reseed_counter. First 3 inputs are concatenated and entered to Hash_df that initiate V. At this time, the length of seed (Seedlen) is entered to Hash_df to determine length of V. Seed value V is concatenated with 0x00 and entered Hash_df to initiate C. Reseed_counter is simply initialized with 1.

Second main module is the Hash_DRBG_Generate. The RB is generated in this module. The

Hash_DRBG_Generate check that the Reseed_counter is bigger than the Reseed_interval. If the Reseed_counter is bigger than the Reseed_interval then this module generate Reseed_required signal and terminate the module. If not, module generate the RB. The additional input (Addi_input) is optional input to generate the RB that is performed to SHA-256 with 0x02 and V to w. w is added with V and performed modular operation with Seedlen. This modular operation is required huge arithmetic logic but we implement modular operation using simple bus wire syntax in the Verilog-HDL. Because, divider is the multiplier of two. Using bus wire syntax, we can implement shift arithmetic operation. In this step, remainder of modular operation alter the V. The next process is performing the Hashgen sub module with V and requested number of bits. The Hashgen make RB using SHA-256. After that, module regenerate V for the next RB. To regenerate V, SHA-256 module is used also three 256-bit adder (red circle in Figure 4-9) and right shift operation is required. Right shift is same to previous one. But, 256-bit adder yield critical path of this module. If, we use '+' operation in Verilog-HDL. It make huge delay. To solve this problem, we use Xilinx Adder/Subtractor IP and reduce delay of critical path in this module. All process in this module is done, Reseed_counter is increased to check how many times RB is generated.

Third main module is Hash_DRBG_Reseed to reinitialize the V, C and reseed. It has similar step of Instantiation_state module. But, to reinitialize V, data is concatenated with V, 0x01, Entropy_input, and Additional_input. At this step, Entropy_input and Additional_input can be changed to security reason.

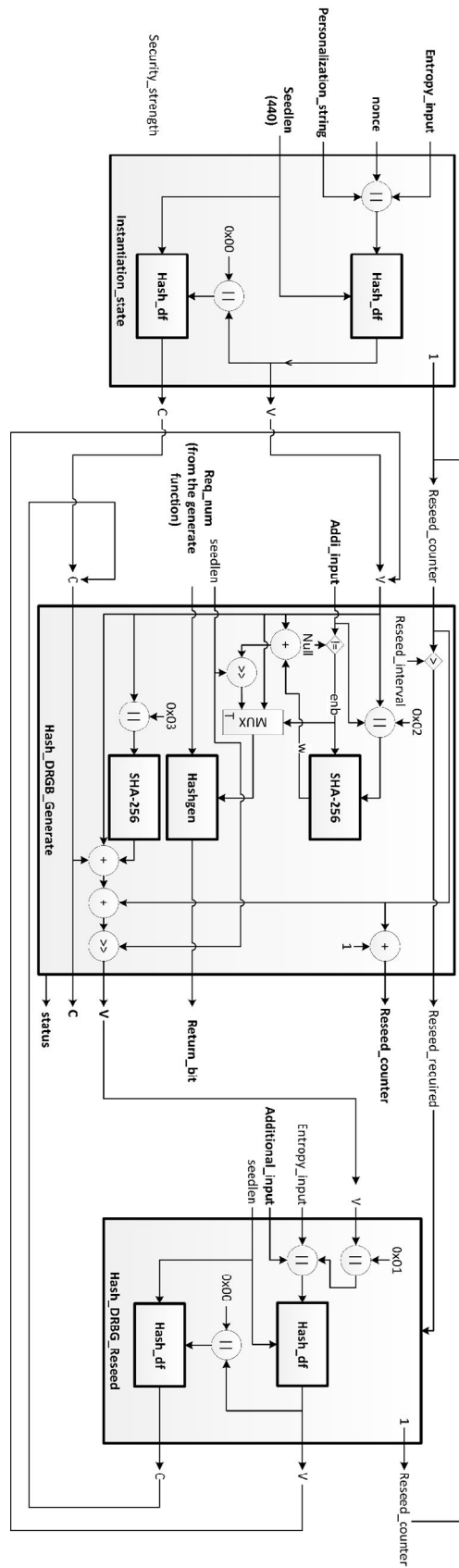


Figure 4 - 17. Proposed architecture of Hash_DRBG

Now, we describe the implementation of the sub modules in this algorithm these are Hash_df, Hashgen and SHA-256. SHA-256 is same in section 4.2 SHA-256. We can show the structure of the Hash_df in figure 4-10. It is permutation Input_string using SHA-256 until the length of Requested_bits is equal to No_of_bits_to_return (seedlen), frequently. To calculate the length of Requested_bits, module perform division and ceiling also counter. This arithmetic operation is also implemented bus wire syntax. We can also confirm structure of the Hashgen sub module in figure 4-11. This module similar to the Hash_df. It is permutation the V to generate the RB.

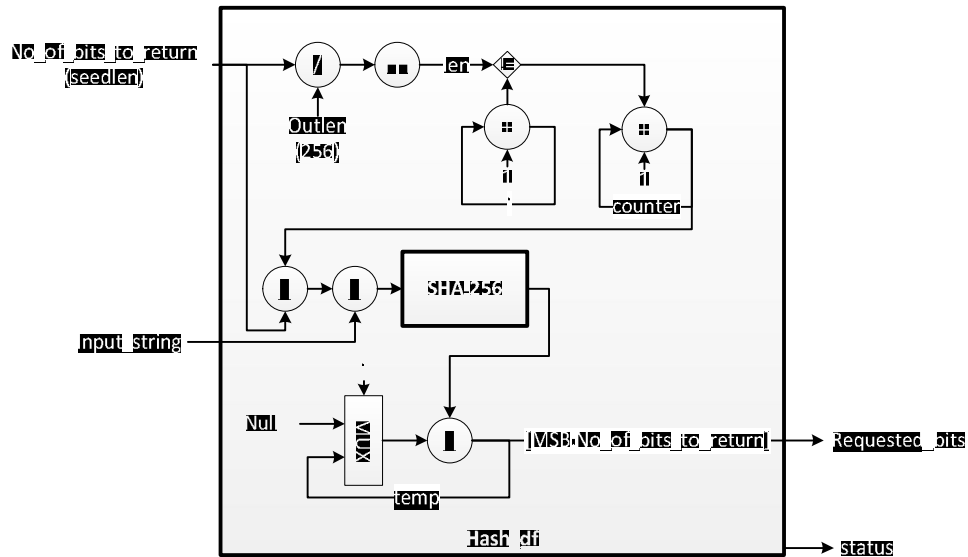


Figure 4 - 18. Proposed architecture of Hash_df

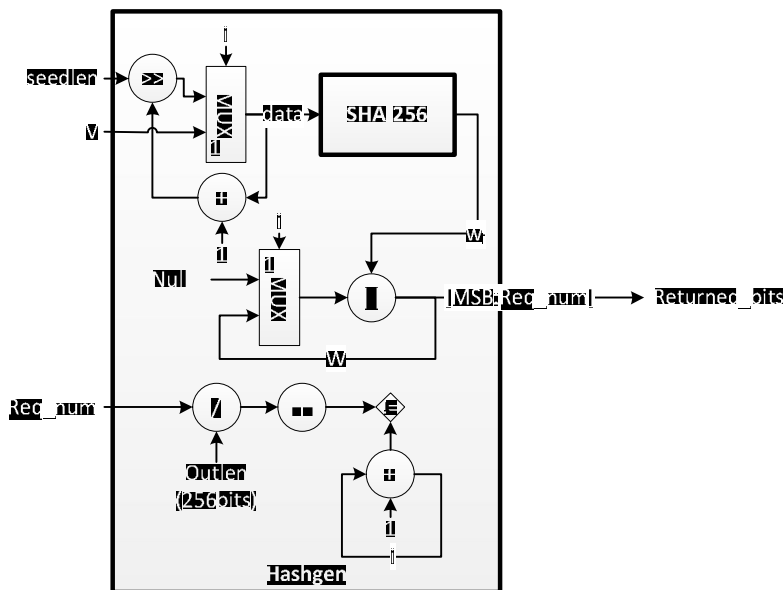


Figure 4 - 19. Proposed architecture of Hashgen

Hash_DRBG module is controlled using FSMs below. Because, each module has relationship for the operation sequence to generate RM. Figure 4-12 shows the FSM of instantiation algorithm. In this FSM, instantiation is done than generate i_do_generate signal to call the FSM of generate in Figure 4-14. The FSM of generate is control the Hash_DRBG_Generate module to generate RM. If, reseed is required than call the FSM of reseed in figure 4-13 to regenerate V, C, and Reseed_counter. If not, FSM is iterated when i_do_generate is high. The FSM of reseed is called than FSM make control signal to handle Hash_DRBG_Reseed module. FSM is done than the FSM of generate is operated to generate RB.

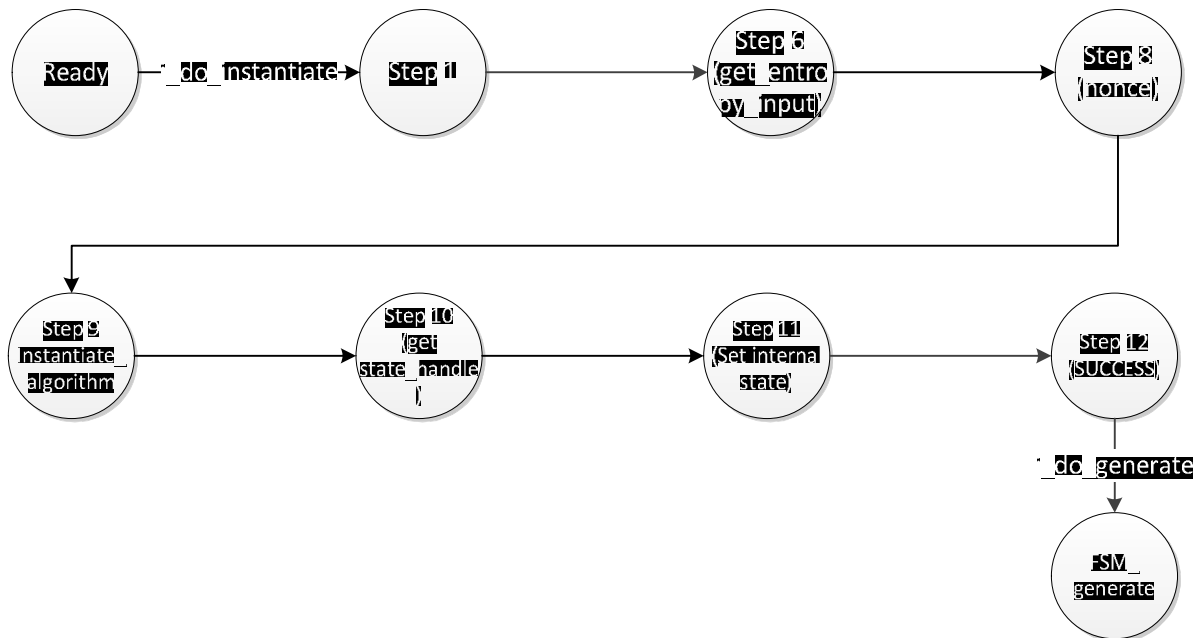


Figure 4 - 20. Finite state machine of instantiation

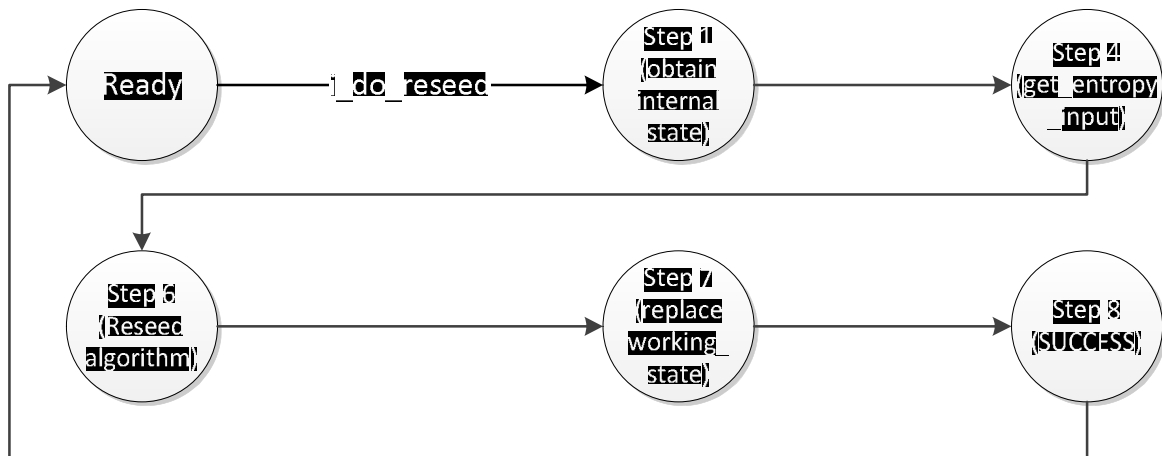


Figure 4 - 21. Finite state machine of reseed

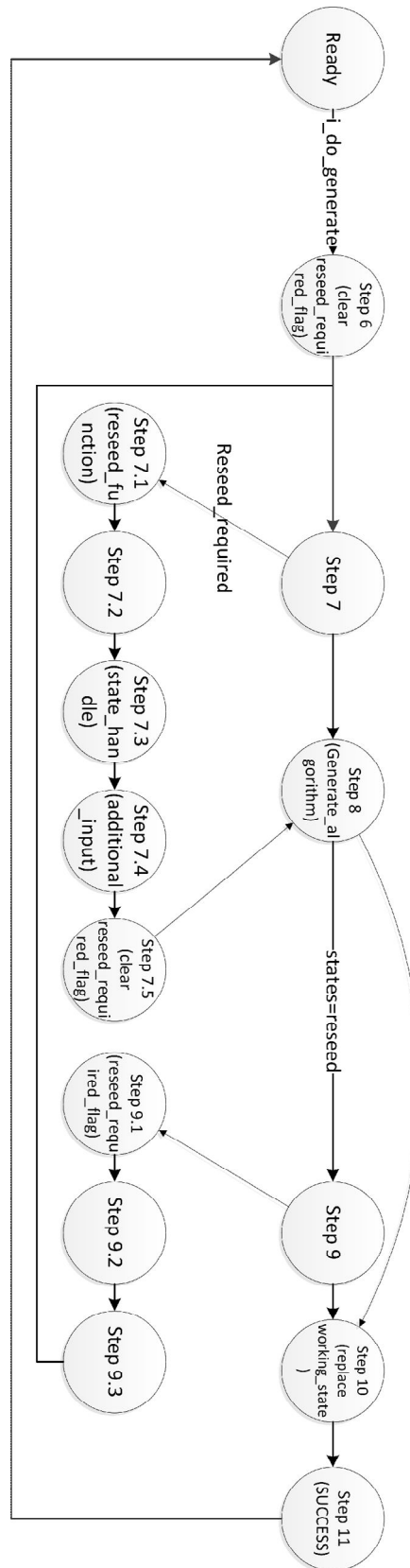


Figure 4 - 22. Finite state machine of generate

4.4 ECDSA

As mentioned previously, ECDSA use multiplication arithmetic operation with wide operands. It is required huge amount of clock cycles. To reduce clock cycles of multiplication, we propose binary_multiplication logic in figure 4-15. This module add the left shifted multipliers when each bit in multiplicand is true. That means, to calculate multiplication, we accumulate the left shifted multipliers.

Proposed binary multiplication use simple shift operation, counters and parallel_adder in figure 4-16. Binary multiplication is controlled by a bit_count counter. The bit_count is increased at the positive edge of clock signal or parallel_adder_done is equal to 1 when num_of_op is equal to 9. That means, the Operand register file is full than bit_count is wait until processing of parallel_adder. Otherwise, the Operand register file is charged with left shifted up to bit_count i_a when least significant bit (LSB) of shifted i_b is equal to 1. And, LSB of shifted i_b is 1 than the num_of_op is increased up to 9. The num_of_op is count how many elements are occupied in the Operand register file and take a role the select signal of DEMUX to select the element of the Operand register file. If num_of_op is equat to 9 than parallel_adder is operated to add the Operand register file and num_of_op is initialized to 1 when parallel_adder_done signal is true. And, parallel_adder_done is true than output signal of the parallel_adder (o_output) is saved to Operand[0] to accumulate previous addition result.

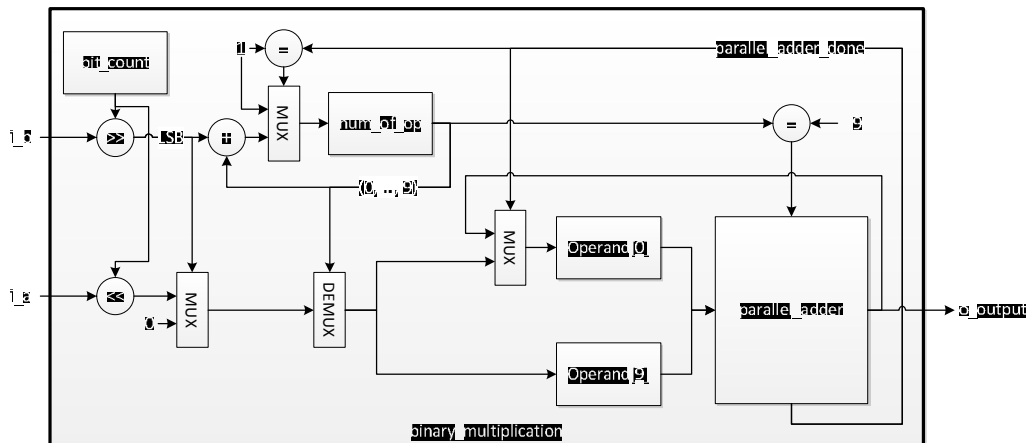


Figure 4 - 23. Proposed architecture of binary_multiplication

As mentioned above, the parallel_adder is used for the binary_multiplication to accumulate the multiplier. In figure 4-16, we can show the structure of the parallel_adder. It has two counter adder_count and clk_count. adder_count control number of iteration of addition to accumulate value. It is increased up to 4 when clk_count is equal to 24. And, control the index of the a and the b register

files to accumulate value into the Operand[0] register. clk_count is control the 256_adders by increasing up to 24. The 256_adder is Xilinx adder IP and has 22 delays to generate output. But, we make delay up to 25 for the reliability of function when module is synthesized.

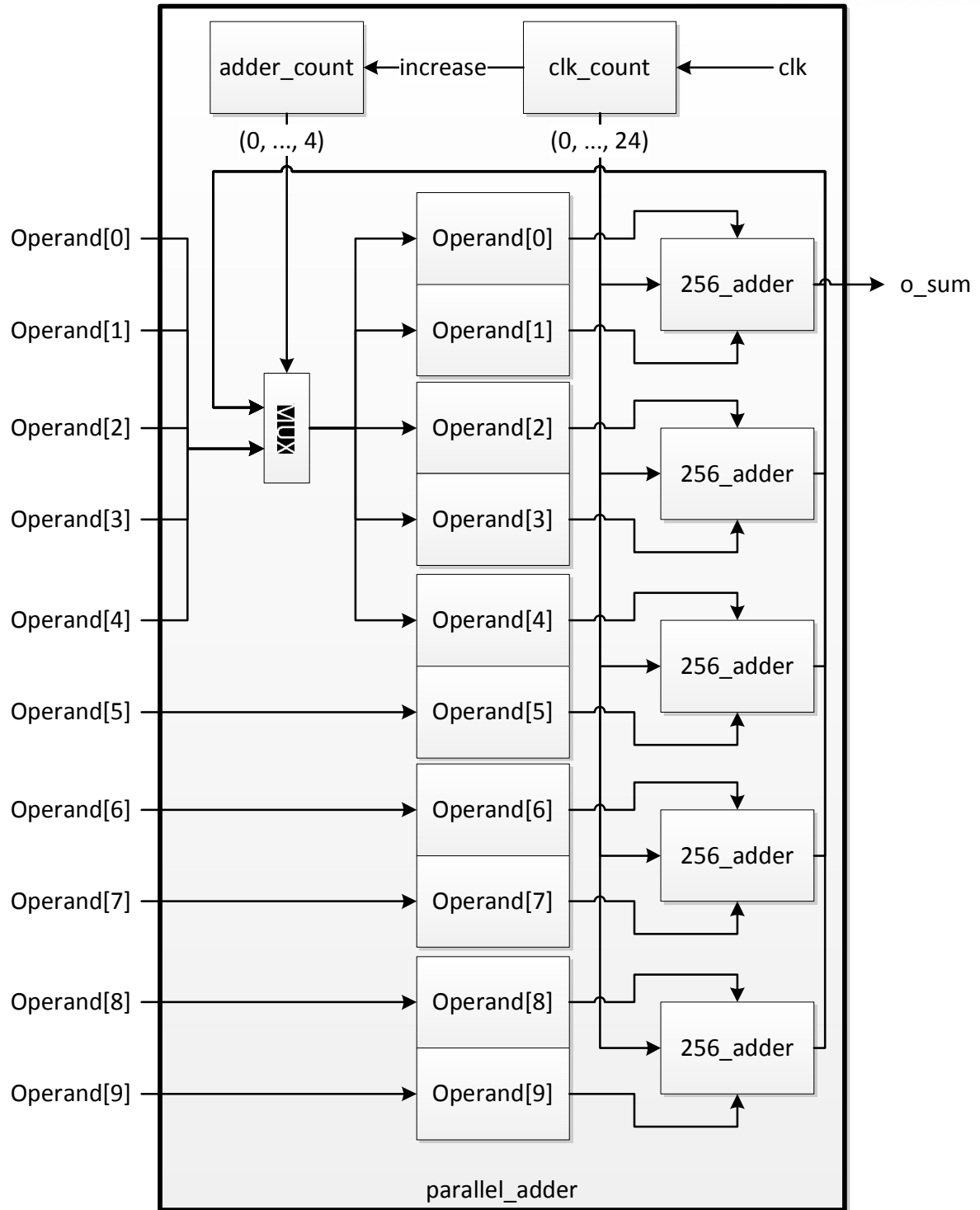


Figure 4 - 24. Proposed architecture of parallel_adder

Chapter V

Experiment Results

In this study, Xilinx Virtex-5 FPGA chip and ISE 14.5 synthesis tools are used to implement and synthesize the proposed systems with Verilog-HDL [22, 23]. However, ECDSA isn't synthesized and made the experimental result. The total number of register slice and LUT of Virtex-5 is 58880 and 58880, respectively.

5.1 AES-CCM

Proposed AES-CCM is implemented and synthesized with Virtex-5 FPGA library. As a result, 11913 FPGA slices and 24062 LUTs are used with 166.20 MHz clock frequency. To encrypt the 640 bit plaintext, AES-CCM spends 433 clock cycles. And, AES spends 76 clock cycles for 128 bits encryption data. The result of implementation is summarized in table 5-1. In this implementation, we use a lot of resource for AES-CCM because of internal register for protect processed data from attack of outside. Table 5-2 shows the usage of register files that is protect processed data.

Table 5 - 5. FPGA implementation result and comparison with previous works of AES-CCM

Device	FPGA Slice	Clock Frequency (MHz)	LUT	Power (mW)
Spartan-3 [21]	523	63.7	-	-
Virtex-2 [25]	3474	80.3	-	-
Virtex-2 [9]	1609	117.88	2511	618
Virtex-4-LX [9]	1921	149	3186	1023
Virtex-5-LX [26]	1809	213	-	-
Virtex-5-SX [This work]	11913	166.20	24062	1332.8

Table 5 - 6. Usage of register files

Register File Name	Width (bit)	Depth
input_data	128	16
input_register	8	256
parser_memory	128	256
key_box	128	22
ctr_memory	128	256

Figure 5-1 shows the RTL synthesis result of AES-CCM. We can show AES, Count, CBC-MAC,

and interface modules in this figure, respectively. Figure 5-2 is the functional result of AES-CCM.
 o_mic_t is MAC data of plaintext.

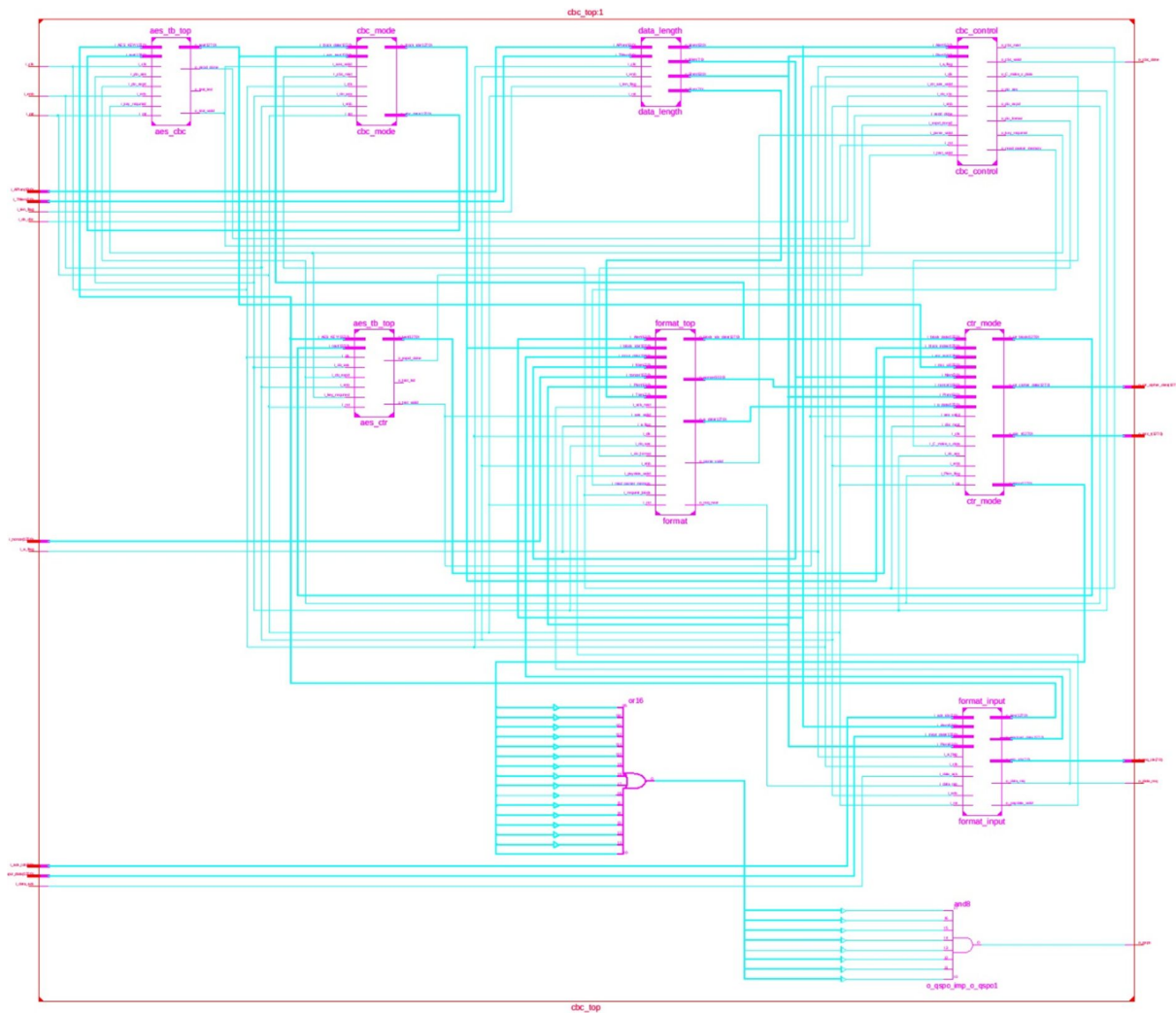


Figure 5 - 7. RTL synthesis result of AES-CCM

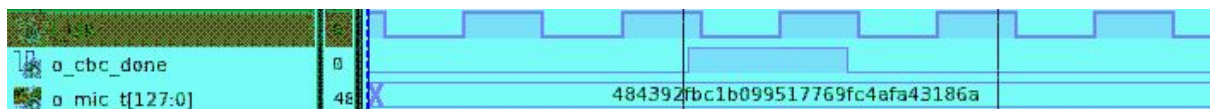


Figure 5 - 8. Functional simulation result of AES-CCM

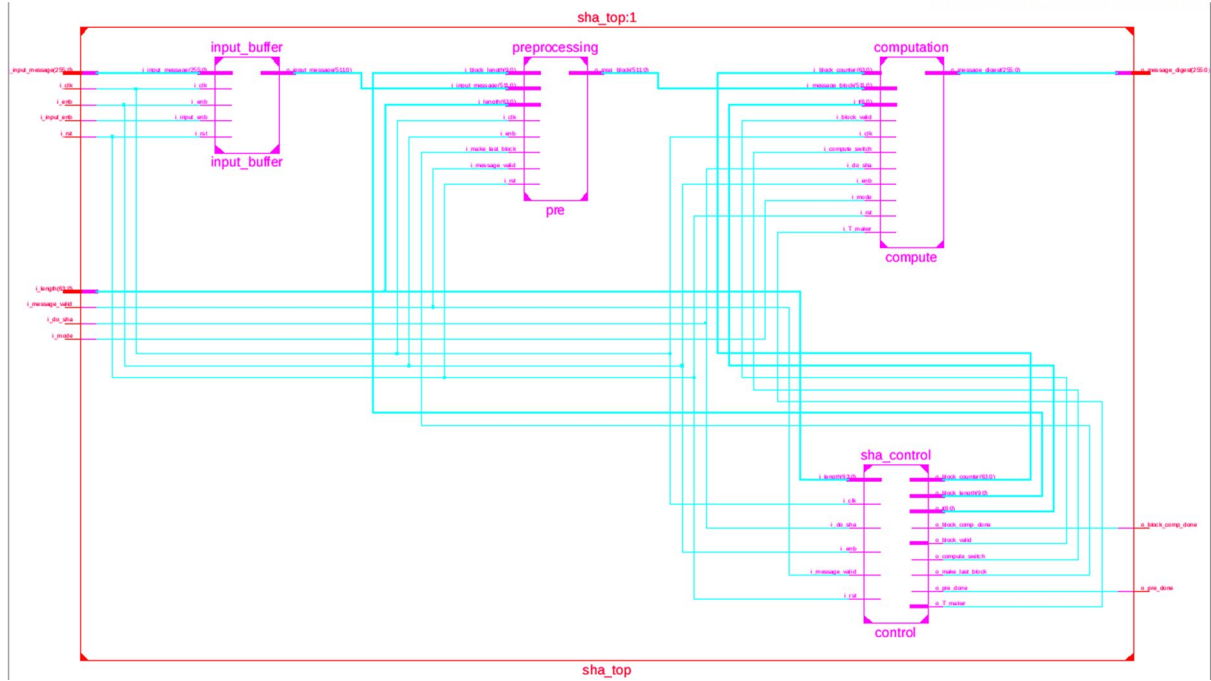


Figure 5 - 10. RTL view of the SHA-256

5.3 Hash-DRBG

We also perform implementation and synthesis of proposed hash-DRBG architecture and result is summarized in table 5-4. In this table, 16704 FPGA slices and 27055 LUTs are used for hash-DRBG with 64.263 MHz and 1330.41 mW consuming power. Unfortunately, we can't find the reference to compare our result. The implemented Hash-DRBG spend 828 clock cycles to generate pseudo-random bits.

Table 5 - 8. Synthesis result of proposed hash-DRBG architecture

Device	FPGA Slice	Clock Frequency (MHz)	LUT	Power (mW)
Virtex-5	16704	64.263	27055	1330.41

Figure 5-5 is RTL synthesis result of our proposed hash-DRBG. We can confirm that all process in DRBG are allocated and connected each other.

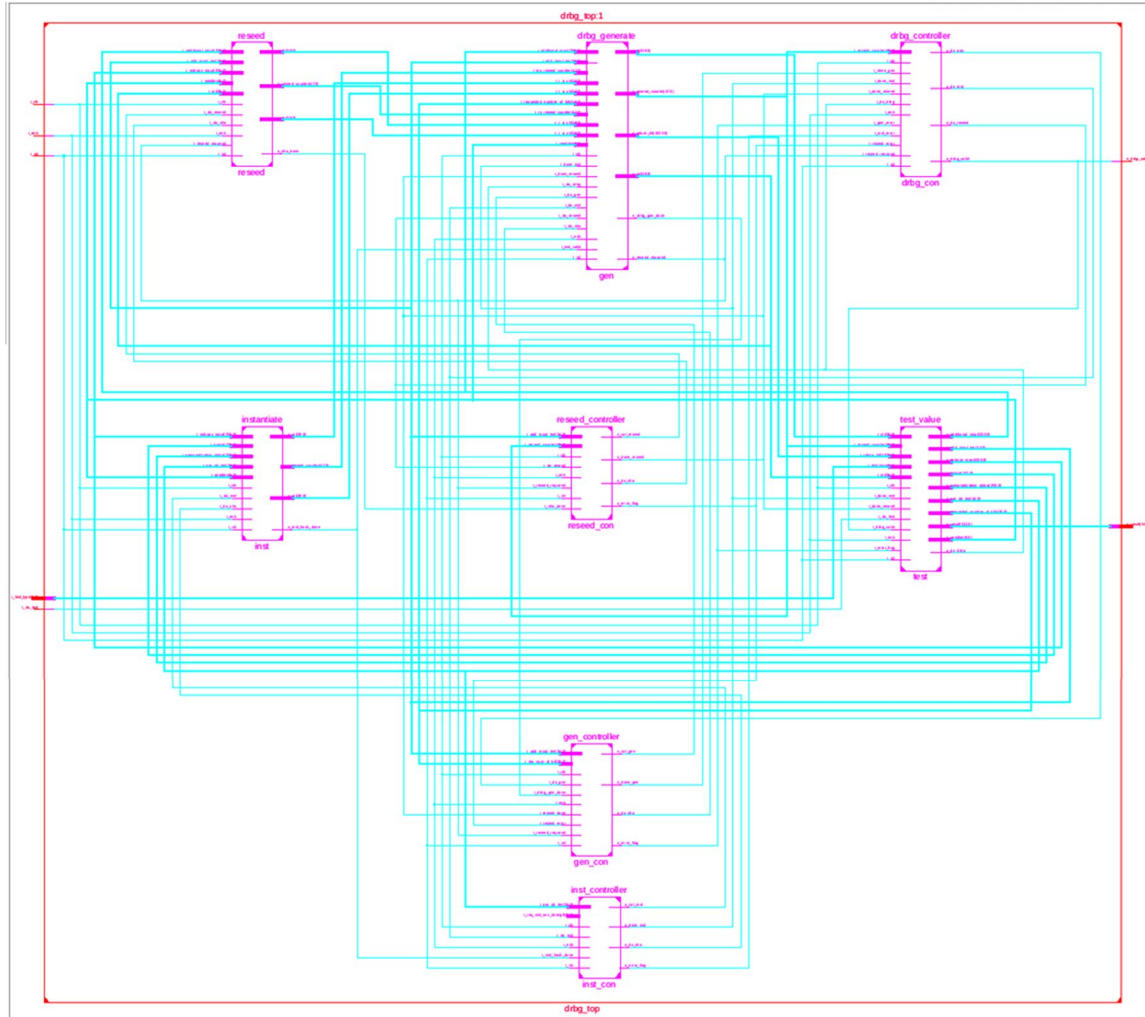


Figure 5 - 11. RTL view of proposed hash-DRBG

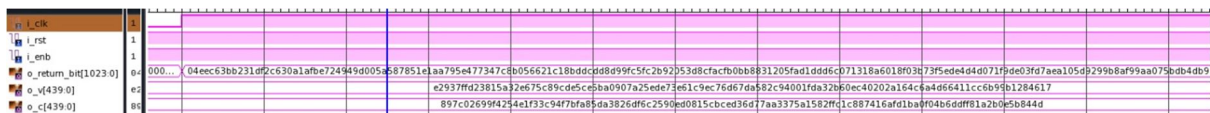


Figure 5 - 12. Timing simulation result of hash-DRBG

Figure 5-6 is timing simulation result of our proposed hash-DRBG. o_return_bit is generated random bits with o_v (seed). o_c is the constant value of current state.

Chapter VI

Conclusions

In this paper, we implement secure algorithm in IEEE 1609.2 WAVE using FPGA hardware for VC security service. VC has a lot of communication load, because a vehicle object communicate many other vehicle objects. Also, VC use wireless communication that has serious security problem like other wireless communication methods. This security problem has critical affect to not only vehicle passengers but also entire traffic. Therefore, implementation of fast and efficient crypto engine for VC is essential security component in VC.

To do this implementation, we modify the 8 bits data path of AES to 32 bits with 32 bits S-Box. And, Xilinx Adder IP is used for SHA-256, hash_DRBG, and ECDSA to support arithmetic operations. In ECDSA, we develop binary_multiplication module for the scalar multiply, modular and inversion. Also, parallel architecture are used to increase operation speed and we analyze the data dependency of the algorithm to convert sequential operation to parallel.

As a result, we can confirm timing simulation about the AES-CCM, SHA-256, Hash-DRBG. And, we can confirm the all logic is placed in FPGA via RTL view of synthesis. Each synthesis results can be high speed operation for the VC. But, we need to improve the period of critical path and reduce resource utilization for the embedded system.

REFERENCES

1. IEEE Std 1609.2-2013. (April, 2013.) IEEE Standard for Wireless Access in Vehicular Environments Security Services for Applications and Management Messages. [Online]. Available: <http://standards.ieee.org/ndstds/standard/1609.2-2013.html>
2. T. Schütze, "Automotive Security: Cryptography for Car2X Communication", tech. rep., Rodhe & Schwarz, Germany, pp. 1-16, March, 2011.
3. NSA, (April, 2010) Mathematical routines for the NIST prime elliptic curves [Online]. Available: https://www.nsa.gov/ia/_files/nist-routines.pdf
4. Roberto A. Uzcátegui, Guillermo Acosta-Marum, "WAVE: A Tutorial," IEEE Communications Magazine, Vol. 47, Issue. 5, pp.126–133, May 2009.
5. K. Koscher, A. Czeskis, F. Roesner, S. Patel, and T. Kohno, "Experimental security analysis of a modern automobile" Security and Privacy (SP), 2010 IEEE Symposium, pp. 447-462, 2010.
6. C. Bryant, "Cars could be next victim of cyber attacks", <http://www.ft.com/intl/cms/s/0/59ccfbbe-90b9-11e2-a456-00144feabdc0.html#axzz2S154hemO>, Financial Times, March 22, 2013.
7. Y. Wang, X. Duan, D. Tian, G. Lu, H. Yu, "Throughput and Delay Limits of 802.11p and its Influence on Highway Capacity", Procedia-Social and Behavioral Sciences, vol. 96, pp. 2096-2104, Nov. 2013.
8. Jonathan Petit and Zoubir Mammeri, "Authentication and consensus overhead in vehicular ad hoc networks", Telecommunication Systems, Vol. 52, Issue 4, pp. 2699-2712, 2013.
9. I. Algreto-Badillo, et al, "Efficient hardware architecture for the AES-CCM protocol of the IEEE 802.11i standard", Computers & Electrical Engineering, pp. 565-577, 2010.
10. H. E. Michail, et al., "On the Exploitation of a High-Throughput SHA-256 FPGA Design for HMAC," ACM Trans. on Reconfigurable Tech. and Sys., vol. 5 no. 1, pp. 1–28, 2012.
11. R. García, et al., "A compact FPGA-based processor for the Secure Hash Algorithm SHA-256," Computers & Electrical Engineering, vol. 40, no. 1, pp. 194–202, 2014

12. Miguel Morales-Sandoval and Claudia Feregrino-Urbe, "On the Hardware Design of an Elliptic Curve Cryptosystem," ENC'04, pp.64-70, Sept. 2004.
13. W. Stallings, Cryptography and Network Security Principles and Practices, 4th ed. Prentice Hall, 2005.
14. NIST FIPS-197. (November, 2001.) "Advanced Encryption Standard," [Online]. Available: <http://csrc.nist.gov/publications/PubsFIPS.html>
15. NIST SP 800-38C. (May, 2004.) Recommendation for Block Cipher Modes of Operation: The CCM Mode for Authentication and Confidentiality. [Online]. Available: <http://csrc.nist.gov/publications/PubsSPs.html>
16. NIST FIPS PUB 180-3. (October, 2008), "Secure Hash Standard(SHS)," [Online]. Available : http://csrc.nist.gov/publications/fips/fips180-3/fips180-3_final.pdf
17. Chanbok Jeong and Youngmin Kim, "Implementation of Efficient SHA-256 Hash Algorithm for Secure Vehicle Communication using FPGA", ISOCC 2014 conf., pp. 224-225, Nov. 2014.
18. NIST SP 800-90A. (January, 2012), "Recommendation for Random Number Generation Using Deterministic Random Bit Generators," Available : [Online]. <http://csrc.nist.gov/publications/nistpubs/800-90A/SP800-90A.pdf>
19. Tao Zhang, Luca Delgrossi, Vehicle Safety Communications: Protocols, Security, and Privacy, Hoboken, New Jersey : John Wiley Sons, Inc., 2012.
20. FIPS PUB 186-3. (June, 2009.) "Digital Signature Standard (DSS)," [Online]. Available: <http://csrc.nist.gov/publications/PubsFIPS.html>
21. Xilinx LogiCORE IP Adder/Subtractor v11.0. (March, 2011), [Online]. Available: http://www.xilinx.com/support/documentation/ip_documentation/addsub_ds214.pdf
22. Xilinx. (March, 2013.) XST User Guide for Virtex-4, Virtex-5, Spartan-3, and Newer CPLD Devices. [Online]. Available: http://www.xilinx.com/support/documentation/sw_manuals/xilinx14_5/xst.pdf
23. Xilinx. (March, 2013.) ChipScope Pro Software and Cores User Guide. [Online]. Available: http://www.xilinx.com/support/documentation/sw_manuals/xilinx14_5/chipscope_pro_sw_cores_ug029.pdf

24. A. Aziz, A. Samiah, N. Ikram, "A secure framework for robust secure wireless network (RSN) using AES-CCMP", Proceedings of the fourth international Bhurban conference on applied sciences and technology, 2005.
25. N. Smyth, M. McLoone, J.V. McCanny, "WLAN security processor", IEEE Trans Circ Syst I: Fund Theory, pp. 1506-1520, 2006.
26. H. Rha, H. Choi, "Efficient Pipelined Multistream AES CCMP Architecture for Wireless LAN", Information Science and Applications (ICISA), 2012 International Conference, pp. 1-5, May 2012.

Acknowledgement

I would like to appreciate completion my master thesis to Prof. Youngmin Kim. His advice, teaching, and encouragement guided to make better research result and quality of my life. And, also, research members in NanoDA Lab. thank for help adapting for UNIST and researching about my thesis. Committee members of my thesis defense are also thank you for the advice about my thesis. It is very helpful for my further research plan.

저의 사랑하는 부모님의 헌신과 동생 덕분에 연구에 전념할 수 있었습니다. 이렇게 잘 연구를 끝내고 졸업할 수 있게 도와 주셔서 감사합니다.

그리고 나의 고등학교 친구들, 대학교 친구들과 연구실 분들, 그리고 UNIST에 정섭이, 은지, BICDL 식구들에게도 졸업을 축하해 주어 감사하다고 전하고 싶습니다.

I also would like to express appreciation to some other people, well, I'll cut this short for now.