RESEARCH ARTICLE OPEN ACCESS

Cryptographic Processor in The Implementation Of Ipsec Protocols

Sridevi, Dr.Manjaiah D.H
Assistant Professor, Department of Computer Science, Karnatak University, Dharwad
Professor, Department of Computer Science, Mangalore University, Mangalore

Abstract
A compact cryptographic processor with custom integrated cryptographic coprocessors is designed and implemented. The processor is mainly aimed for IPSec applications, which require intense processing power for cryptographic operations. In the present design, this processing power is achieved via the custom cryptographic coprocessors. These are an AES engine, a SHA-1 engine and a Montgomery modular multiplier, which are connected to the main processor core through a generic flexible interface. The processor core is fully compatible with Zylin Processor Unit (ZPU) instruction set, allowing the use of ZPU tool chain. A minimum set of required instructions is implemented in hardware, while the rest of the instructions are emulated in software.

Keywords: Cryptography, Processor, GCC, IPSec, Zylin Processor Unit.

I. Introduction

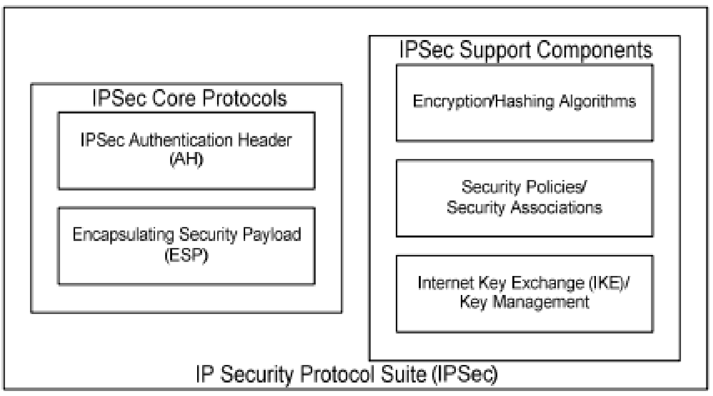
The Internet Protocol (IP) is the protocol that is used for data communication over the Internet. It is also referred to as the Transmission Control Protocol/Internet Protocol (TCP/IP). IP delivers distinguished protocol packets, which are usually referred to as datagrams, from the source host to the destination host based on their addresses, by means of addressing methods and structures for datagram encapsulation. The first version of addressing structure is referred to as Internet Protocol Version 4 (IPv4), which is still the dominant protocol of the Internet. However, its successor, Internet Protocol Version 6 (IPv6) is nowadays being deployed actively worldwide. The main disadvantage of IP is its lack of a general-purpose mechanism for ensuring the authenticity and privacy of data. IP datagrams are usually routed between devices over unknown networks; hence, any information in the datagrams can easily be intercepted and even changed. As a result of the inherent security weaknesses of IP and the increased utilization of Internet services for critical applications, IP Security (IPSec) protocols were developed [1]. At first, IPSec was developed for IPv6, but then it has been engineered to cover the security needs of both IPv4 and IPv6 networks. Its operation in both versions differs only in the datagram formats used for authentication header (AH) and encapsulating security payload (ESP).

1.1 IPSec Operation and Core Protocols

When two devices want to communicate securely, they set up a secure path that may traverse across many insecure intermediate systems. To perform this engagement, these devices must satisfy certain rules:

- They must agree on a set of security protocols to use so that each one sends data in a format the other can understand.
- They must decide on a specific encryption algorithm to use in encoding data.
- They must exchange keys that are used to decode the data that has been cryptographically encoded.
- After background work is completed, each device must use the protocols, methods, and keys previously agreed upon to encode data and send it across the network.

In the realization of its operation, IPSec uses many different components and core protocols as shown in Figure 1. Because of this multi-technique and multi-protocol characteristic of IPSec, its main architecture and behavior of all the core components and protocols are not defined in a single Internet standard. Instead, a collection of continuously evolving Request for Comments (RFCs) [4] defines the architecture, services and specific protocols which are used in IPSec. Most important of these standards are listed in Table .1.


IPSec Support Components
Encryption/Hashing Algorithms
Security Policies/ Security Associations
Internet Key Exchange (IKE)/ Key Management
IPSec Core Protocols
IPSec Authentication Header (AH)
Encapsulating Security Payload (ESP)
IP Security Protocol Suite (IPSec)
Figure 1 Overview of IPSec protocols and components

1.2 Important IPSec standards.

RFC Number	Name	Description
4301	Security Architecture for the Internet Protocol	The main IPSec document, describing the architecture and general operation of the technology, and showing how the different components fit together.
4302	IP Authentication Header	Defines the IPSec Authentication Header (AH) protocol, which is used for ensuring data integrity and origin verification.
4835	Cryptographic Algorithm Implementation Requirements for ESP and AH	Describes encryption and authentication algorithms for use by ESP and AH.
4303	IP Encapsulating Security Payload (ESP)	Describes the IPSec ESP protocol, which provides data encryption for confidentiality.
4306	The Internet Key Exchange (IKE)	Describes the IKE protocol that's used to negotiate security associations and exchange keys between devices for secure communications.

Table 1: IPSec Standards

Two main pieces of IPSec, which actually manage information encoding to ensure security, are the authentication header (AH) and the encapsulating security payload (ESP) [3]. They are known as the core protocols of IPSec.

II. Cryptographic Coprocessors

The three coprocessors implement the AES encryption, SHA-1 hashing and Montgomery modular multiplication. Of these, AES encryption and SHA-1 hashing are must algorithms for IPSec, while Montgomery modular multiplication (MMM) is the computational component of the RSA algorithm used in Internet Key Exchange (IKE) protocol of IPSec protocol suite. The flexible coprocessor interface explained before requires the coprocessors to behave as RAMs from the ZPU based main processor's point of view. Therefore AES and SHA-1 coprocessors are embedded into wrappers, which imitate single-port RAM behavior. This is not necessary for the MMM coprocessor, as it actually uses RAMs for operand and result storage.

III. IPSEC protocol implementation

In order to present the operation and use of the cryptographic processor in the implementation of IPSec protocols and components. Each example demonstrates the use of one of the coprocessors. The first example uses the AES coprocessor in order to implement the Counter with Cipher Block Chaining-Message Authentication (CCM) mode, which is an optional combined encryption and authentication scheme of the IPSec protocol suite. It is followed by the Hash based Message Authentication Code (HMAC), which utilizes the SHA-1 coprocessor. The last example uses the Montgomery modular multiplier coprocessor in order to implement the RSA encryption/decryption algorithm, which is an important component of the Internet Key Exchange (IKE) protocol of IPSec.

3.1 Counter with Cipher Block Chaining-Message Authentication Code (CCM)

Counter with Cipher Block Chaining-Message Authentication Code (CCM) [2] is used to provide assurance of the privacy and the authenticity of data by combining the techniques of the Counter (CTR) mode [7] and the Cipher Block Chaining-Message Authentication Code (CBC-MAC) algorithm. CCM is based on an approved symmetric key block cipher algorithm whose block size is 128 bits, such as the Advanced Encryption Standard (AES) algorithm. CCM can be considered as a mode of operation of the block cipher algorithm. A single key to the block cipher must be established beforehand among the parties to the data. So, CCM should be implemented within a well-designed key management structure. The security properties of CCM depend on the secrecy of this key. CCM is intended for use in a packet environment. All of the data should be available in storage before CCM is applied. CCM is not designed to support partial processing or stream processing. Three inputs to CCM are:

- data that will be both authenticated and encrypted which is called the payload,
- associated data that will be authenticated but not encrypted,
- a unique value called nonce, which is assigned to the payload and the associated data.

CCM consists of two related processes: generation-encryption and decryption-verification. Only the forward cipher function of the block cipher algorithm is used within these primitives. In generation-encryption, cipher block chaining is applied to the payload, the associated data, and the nonce to generate a message authentication code (MAC). Then, counter mode encryption is applied to the MAC and the payload, to transform them into an unreadable form which is called the cipher text. Therefore, it can be seen that CCM generation-encryption expands the size of the payload by the size of the MAC. In decryption-verification, counter mode decryption is applied to the supposed cipher text to recover the MAC and the corresponding payload.

Then, cipher block chaining is applied to the payload, which is the received data, and the received nonce to verify the correctness of the MAC. A successful verification provides assurance that the payload and the associated data originated from a source with access to the key. A MAC provides stronger assurance of authenticity 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 the verification of a MAC is designed to detect intentional, unauthorized modifications of the data, as well as accidental modifications.

3.2 Description of CCM

As mentioned before, two CCM processes are called generation-encryption and decryption-verification. The order of the steps of these two processes is a little bit flexible. For example, the generation of the counter blocks may occur at any time before they are used. In fact, the counter blocks may be generated in advance to be considered as inputs to the processes. The below algorithm explains the generation-encryption process. The input data to the generation-encryption process is a valid nonce, a valid payload string and a valid associated data string, which are formatted according to the formatting function. The CBC-MAC mechanism is applied to the formatted data to generate a MAC, whose length is a prerequisite. Counter mode encryption, which requires a sufficiently long sequence of counter blocks as input, is applied to the payload string and separately to the MAC. The resulting data which is called the ciphertext (denoted *C)* is the output of the generation-encryption process.

Prerequisites
Block cipher algorithm,
Key K,
Counter generation function,
MAC length Tlen,
Inputs:
Valid nonce N (salt + initialization vector(IV)),
Valid payload P of length plen bits (= M blocks – each block is 128 bits long),
Valid associated data A of length Alen bits (= D blocks – each block is 128 bits long),
Outputs:
Ciphertext C.
Steps:

```
1.Apply the formatting faction to (M, A, P) to produce D+M blocks $B_1, B_2, …., B_{D+M}$. IV is added at the beginning of the block as $B_0 = IV$.
2. Set $X_i = E_k(B_0)$.
3. for i= 1 to D+M, do $X_{i+1} = E_k (X_i \oplus B_i)$.
4. Set $T = MSB_{plew}(X_{D+M+1})$.
5. Apply the counter generation function to generate the counter blocks $Ctr_0, Ctr_1,…., Ctr_M$, where M= | Plen/128|.
6. for j=0 to M, do $S_j = E_k(Ctr_j)$.
7. Set $S = S_1 \| S_2 \| … \| S_M$.
8. Return $C = (P \oplus MSB_{plew}(S) \| T \oplus MSB_{plew}(S_0))$.
```

The input to the decryption-verification process, which is described in the below pseudo-code, is a supposed ciphertext, an associated data string and the nonce that is believed to be used in the generation of the supposed ciphertext. Counter mode decryption is applied to the supposed ciphertext to produce the corresponding MAC and payload. If the nonce, the associated data string and the payload are valid, then these strings are formatted into blocks according to the formatting function and the CBC-MAC mechanism is applied to verify the MAC. If the verification succeeds, then the decryption-verification process returns the payload as output. Otherwise, only the error message INVALID is returned. When the error message INVALID is returned, the payload *P* and the MAC *T* should not be displayed. Moreover, the implementation should ensure that an unauthorized party cannot distinguish if the error message results from Step 7 or from Step 10, for example from the timing of the error message.

Block cipher algorithm,
Key K,
Counter generation function,
Valid MAC length Tlen,
Inputs:
Valid nonce N (salt + initialization vector(IV)),
Associated data A of length Alen bits (= D blocks – each block is 128 bits long),
Supposed ciphertext C of length Clen bits (= R blocks – each block is 128 bits long),

Outputs:
Either the payload P of length plen bits (= M blocks – each block is 128 bits long), or INVALID
Steps:

```
  1. If clen $\leq$ Tlen, then return INVALID.
  2. Apply the counter generation function to generation the counter blocks
     $Ctr_0, Ctr_1,…., Ctr_M$, where M= | Clen- Tlen /128|.
  3. for j=0 to M, do $S_j = E_k(Ctr_j)$.
  4. Set $S = S_1 \| S_2 \| … \| S_M$.
  5. Set $P = MSBc_{lew-Tlen}(C) \oplus MSB_{Clen-Tlen}(S)$).
6. SET $T = MSB_{Tlen}(C) \oplus MSB_{Tlen}(S_0)$).
7. If N, A or P is not valid, then return INVALID. Else, Apply the formatting function to (N, A, P) to produce the blocks $B_1, B_2, | … B_{D+M}$.
8. Set $X_i = E_k(B_0)$
9. For I = 1 to D+M, do $X_{i+1} = E_k(X_i \oplus B_i)$.
10. If $T \neq MSB_{Tlen}(X_{D+M+1})$, then return INVALID. Else, return P.
```

3.3 AES-CCM

AES-CCM is performed on 16-byte (128-bit) blocks. However, since the processor data bus is 32-bits wide, AES input is not directly sent to the core in 128-bit format. At first, necessary data (such as flags, nonce, payload, AAD) is read from corresponding

RAM addresses and then the 128-bit input to the AES core is formed and stored in four 32-bit temporary variables in the software. These temporary variables are mapped to 32-bit wide RAM locations (words) in the actual hardware. 4-word AES input read from the temporary memory locations is sent to the four AES input registers to be processed. The AES input registers are also mapped to specific locations in the processor's address space. Once the AES inputs are transferred, a write is issues to the virtual AES command/status register signaling the AES to start its encryption. This sets the physical "busy" signal in the processor and halts its program execution, until the AES coprocessor completes its run and clears the "busy" signal by issuing the "encrypted block ready" signal. The encrypted block is then read from four (4x32=128-bit) AES output addresses. AES is performed for blocks of cipher block chaining and counter mode parts. In the end, the cipher text is formed from the encrypted counter blocks and MAC (MAC length is given as input - *Tlen).* For IPSec purposes, certain inputs are fixed in the standard, such as the length of AAD and the length of nonce. For example, AAD length is stated as 8 or 12 bytes in the standard [8]. Therefore, simplifications can be done on the software code to cover only these IPSec properties.
The decryption-verification is the reverse process of the above operation.

3.4 Hash-based Message Authentication Code (HMAC)

In communications, providing a way to check the integrity of information transmitted over or stored in an unknown medium is a major necessity. Mechanisms that provide such integrity checks based on a secret key are called message authentication codes (MACs), as mentioned in previous section. A MAC that uses an approved cryptographic hash function in conjunction with a secret key is called hash-based message authentication code (HMAC) [5]. The main goals behind the HMAC construction [9] are:

- to use available hash functions without modifications,
- to preserve the original performance of the hash function,
- to use and handle keys in a simple way,
- to have a good cryptographic analysis of the strength of the authentication mechanism on the underlying hash function,
- to allow easy replace ability of the underlying hash function, in case that faster or more secure hash functions are available in the future.

Any iterative cryptographic hash function, such as SHA-1, SHA-224 ... etc., may be used in the calculation of an HMAC. So, the resulting MAC algorithm is termed as HMAC-SHA-1, HMAC-SHA-224 . etc., accordingly. The size of the output of HMAC is the same as that of the underlying hash function (160, 256 or 512 bits in case of SHA-1, SHA-256 and SHA-512, respectively), although it can be truncated if desired.

3.5 HMAC

In the definition of HMAC, the cryptographic hash function is denoted by *H* and the secret key is denoted by K. The byte-length of blocks on which *H* operates iteratively, is denoted by *B* (*B* = 64 for SHA-1, SHA-224, SHA-256 and *B* = 128 for SHA-384, SHA-512). The byte-length of hash function outputs is denoted by *L* (L=20, 28, 32, 48 and 64 for SHA-1, SHA-224, SHA-256, SHA-384 and SHA-512, respectively). The authentication key *K* can be of any length up to *B*, the block length of the hash function. Two fixed and different strings *ipad* and *opad* are defined as follows (the 'i' and 'o' are mnemonics for inner and outer):
ipad = the byte 0x36 repeated B times,
opad = the byte 0x5C repeated B times.
To compute a MAC over the data *'text* using the HMAC function, the following operation is performed:
$\mathrm{MAC}(\mathrm{text})_t = \mathrm{HMAC}(K, \mathrm{text})_t = H((K_0 \oplus \mathrm{opad}) \,||\, H((K_0 \oplus \mathrm{opad}) \,||\, \mathrm{text}))_t$
Step by step process of the HMAC algorithm is as follows:

1. If the length of K=B: set K_0 = K. go to step 4.
2. If the length of K > B: hash K to obtain an L byte string, then append (B-L) zeros to create a B-byte string K_0 (i.e . $K_0 = H(K) \,||\, 00 \ldots 00$). Go to step 4.
3. If the length of K< B : append zeros to the end of K to create a B-byte string K_0 (i. e if K is 20 bytes in length and B = 64, then K will be append with 44 zero bytes 0X00).
4. XOR K_0 with ipad to produced a B-byte string : $K_0 \oplus$ ipad.
5. Append the stream of data 'text' to the string resulting from step 4: $((K_0 \oplus \mathrm{ipad}) \,||\, \mathrm{text})$.
6. Apply H to the stream generated in step 5: $H((K_0 \oplus \mathrm{ipad} \,||\, \mathrm{text})$.
7. XOR K_0 with opad: $K_0 \oplus$ ipad.
8. Append the result from step 6 to step 7: $(K_0 \oplus \mathrm{opad}) \,||\, H((K_0 \oplus \mathrm{ipad}) \,||\, \mathrm{text})$.
9. Apply H to the result from step 8: $H(K_0 \oplus \mathrm{opad}) \,||\, H((K_0 \oplus \mathrm{ipad}) \,||\, \mathrm{text}))$.
10. Select the leftmost t bytes of the result of step 9 as the MAC.

3.6 HMAC-SHA-1-96

HMAC-SHA-1-96 is performed on 64-byte (512-bit) blocks. However, a SHA-1 input is not directly sent to the core in 512-bit format. At first, necessary data (such as key and payload) is read from corresponding RAM addresses and then the 512-bit input to the SHA-1 core is formed and stored in sixteen 32-bit temporary variables. As in the case of AES, once the SHA-1 coprocessor inputs are ready, a

write is issued to the virtual SHA-1 command/status register and the coprocessor starts its operations halting the main processor's program execution via the "busy" signal. After SHA-1 completes processing the 512-bit input block, it releases the busy and the processor continues program execution. At this step, the "hashed data" can either be read from the SHA-1 output registers, or hashing can be continued with new inputs. For IPSec, certain inputs are fixed in the standard. For example, the length of payload is fixed to multiples of 512 bits and the key length is fixed to 20 bytes (160 bits – 5 RAM locations) [10]. Therefore, simplifications can be done on the software code to cover only these IPSec properties.
The algorithm is rewritten according to this, as follows:

```
$B_0[0:4] = (K\oplus_i pad)$; $B_0[5:15]$= ipad) bytes
$B_1$ = text[0:15]
$B_2$ = text[16:31]
-
-
-
$B_N$ = text[16x(N-1):16x(N-1))+15]

Clear=1;
H[0:4] = IV;
SHA_in = $B_0$;
Start=1;
For(i=1 to N)
        H[0:4] = SHA_out;
        SHA_in =$B_i$;
        Start=1;
End for
C= SHA_out;
$A_0[0:4] = (K\oplus opad)$;$A_0[5:15]$ = opad bytes
$A_0[0:4]$ = c; $A_1[5:15]$= SHA_padding
Clear = 1;
h[0:4]= IV;
SHA_in = $A_0$;
Start = 1;
h[0:4] = SHA_out;
SHA_in = $A_1$;
Start = 1;
Y = SHA_out[0:2]
```

In the algorithm pseudo-code given above, the "clear" signal is given in addition to the "start" in order to identify the first 512-bit block input of the hashing operation. The purpose of this identification is to select initialization vector as h[0:4]. Unlike the "start" signal, it does not activate the busy and halt program execution.

3.4 RSA Encryption and Decryption for Internet Key Exchange

The public key algorithm RSA (which stands for Ron Rivest, Adi Shamir and Leonard Adleman, who first publicly described it at MIT, in 1977) is the first algorithm known to be suitable for both asymmetric encryption/decryption and signature generation/verification purposes, and it was one of the first great advances in public key cryptography. RSA, which became patent free and released to the public domain in 2000, is the most widely used public key algorithm in electronic commerce protocols and believed to be secure given sufficiently long keys and the use of up-to-date implementations. RSA involves a public key and a private key. The public key can be known to everyone and it is used for encrypting messages. Messages encrypted with the public key can only be decrypted using the private key. RSA gets its security from integer factorization problem. Difficulty of factoring large numbers is the basis of security of RSA (512, 1024 or 2048 bits long, generally).

3.6 RSA

RSA encryption algorithm has different modes, such as RSA-512, RSA-1024, RSA-2048, depending on the length of the inputs. Therefore, RSA is performed on 64, 128 or 256 byte (512, 1024 or 2048 bits) blocks. However, an RSA input is not directly sent to the MMM coprocessor in 512, 1024 or 2048 bits format. At first, necessary data (such as message, keys, modulus and K constant) is read from corresponding RAM addresses. As memory locations are 32-bit wide, MMM inputs are sent to corresponding 16(x32=512-bit), 32(x32=1024-bit) or 64(x32=2048-bit) input addresses to be processed. The rest is similar to AES and SHA-1 coprocessor operation. A write into the virtual MMM command/status register starts its operation, activates the busy signal, and halts the processor program execution. When MMM output is ready, busy signal is released and processor program continues its run by transferring the multiplication result from the MMM output registers to target addresses inside the memory.
RSA algorithm can directly be implemented in software. Recall the algorithm in original format:

```
// r = MMM(1,K,n) //
Mem(MMM_A)$\leftarrow$ mem(1);
Mem(MMM_B)$\leftarrow$ mem(K);
Start = 1;
Mem(MMM_C)$\leftarrow$ mem(n);
mem(m); $\leftarrow$ $m$em(MMM_Y);
for i=0 to E-1
    Mem(MMM_B)$\leftarrow$ mem(m);
If e(i)
        // r = MMM(r,m,n) //

        Mem(MMM_A)$\leftarrow$ mem(r);
        start = 1;
        mem(r); $\leftarrow$ $m$em(MMM_Y);
end if
        // m = MMM(m,m,n) //
        Mem(MMM_A)$\leftarrow$ mem(m);
        start = 1;
        mem(m); $\leftarrow$ $m$em(MMM_Y);
end for
```

```
// r = MMM(r,1,n) //
Mem(MMM_A)← mem(r);
Mem(MMM_B)← mem(1);
start = 1;
mem(r); ← mem(MMM_Y);
c← mem;
```

IV. Conclusion

The processor is mainly targeted for IPSec applications, and is composed of a ZPU instruction set compatible microcontroller core, and cryptographic coprocessors connected to this core via a simple and generic plug-in interface. There are three coprocessors capable of implementing the AES encryption and SHA-1 hashing in full compliant with the standards, as well as Montgomery modular multiplication up to 2048-bits. These coprocessors are accessed by the main controller core like regular RAMs, which forms the basis idea for the flexible interface. The interface is generic in the sense that it allows any module to be connected to the main core regardless of the input/output definition or the function of the module with the addition of a simple wrapper around the module. The cryptographic processor is intended as a proof-of-concept for the flexible interface and a development platform for a commercial IPSec product. It will be possible to evaluate performance of the complete IPSec protocol suite on this processor on either simulation or FPGA development boards.

Reference

[1] IPSec, http://en.wikipedia.org/wiki/IPsec
[2] Counter with CBC-MAC (CCM), RFC 3610, 2003.
[3] Kozierok, C. M., The TCP/IP Guide, 2005.
[4] RFC Index, http://tools.ietf.org/rfc/index
[5] Thull, D., Sannino, R., Performance considerations for an embedded implementation of OMA DRM 2, Design, Automation and Test in Europe, 2005
[6] Osvik, D. A., Bos, J. W., Stefan, D., Canright, D., Fast software AES encryption, In International Workshop on Fast Software Encryption, LNCS Springer, 2010
[7] Using Advanced Encryption Standard (AES) Counter Mode, RFC3686, 2004
[8] Using Advanced Encryption Standard (AES) CCM Mode with IPsec Encapsulating Security Payload (ESP), RFC4309, 2005
[9] Krawczyk, H., Bellare, M., Canetti, R., HMAC: Keyed-Hashing for Message Authentication, Internet Engineering Task Force, Request for Comments (RFC) 2104, February 1997
[10] The Use of HMAC-SHA-1-96 within ESP and AH, RFC2404, 1998