ADSP-BF70x Blackfin+ Processors Cryptographic Algorithm Validation

Contributed by G. Yi

Rev 1 – December 11, 2014

Introduction

The ADSP-BF70x Blackfin+ Processors series (hereafter referred to as ADSP-BF707) has successfully passed the Cryptographic Algorithm Validation Program (CAVP) for select cryptographic functions. This EE-Note provides an overview of the validation process and how passing the CAVP helps toward full FIPS 140-2[1] conformance.

Most functions are fully performed by the ADSP-BF707 on-chip cryptographic hardware accelerator. Two of the functions use a combination of software running on the processor core and the accelerator engine. For this hybrid approach, the code is provided with this EE-Note in the associated `.ZIP` file.

Cryptographic Module Validation Program (CMVP) Overview

In order for a security module to be accepted for use by a federal agency in the United States of America and Canada, they must be validated conforming to FIPS 140-2. This validation program was created jointly between the National Institute of Standards and Technology (NIST) and Communications Security Establishment Canada (CSEC). It is used to validate the security modules to FIPS 140-2, which defines different levels of qualitative security.

The Cryptographic Algorithm Validation Program (CAVP) is a prerequisite for CMVP. It validates that the implemented cryptographic functions are functionally correct. ADSP-BF707 processors have successfully passed CAVP validation for select cryptographic functions. Therefore, a user or developer can easily proceed with module validation through CMVP of their system or device using ADSP-BF707 processors.

ADSP-BF707 Cryptographic Processing Functionality

ADSP-BF707 processors contain several cryptographic processing units that help offload the computations that would otherwise be performed on the processors core. These are the Security Packet Engine (PKTE), the True Random Number Generator (TRNG), and the Public Key Accelerator (PKA).

The Security Packet Engine contains a collection of cryptographic functions which include symmetric ciphers, hash functions, keyed-hash message authentication code (HMAC) functions and a pseudo-random number generator. It also supports different programming models with different levels of intervention from the host processor. For complete details on the PKTE, please refer to the *ADSP-BF70x Blackfin+ Processor Hardware Reference*[2].

The functions that have been successfully tested and validated in the CAVP (Table 1) are all supported by the PKTE.

Cryptographic Function	**Validation Number**
TDES (also known as 3DES) using 1, 2 or 3 different 56-bit keys and also in ECB and CBC modes	1771
AES -128bit, -192bit and -256bit in ECB, CBC and ICM modes	3026
SHA-1, SHA-224 and SHA-256	2528
HMAC based on SHA-1, SHA-224 and SHA-256	1912
Deterministic Random Number Generator defined by the X9.31[3] standard and based on the AES-128 cipher	1309

Table 1. Validated Cryptographic Functions on ADSP-BF707 Blackfin+ Processors

Hybrid Processing

For the TDES, AES and SHA functions, the user can just program in the necessary parameters and the PKTE can process the data autonomously. For the other functions, HMAC and RNG, the PKTE needs intervention and intermediate processing from the Blackfin+ core. This is known as hybrid processing.

This section describes the algorithms and how they were implemented on the ADSP-BF707 processor to go through CAVP validation. For full details on programming the PKTE, please refer to the hardware reference[2].

Random Number Generator (RNG)

A Random Number Generator defined by X9.31, based on the AES-128 cipher is available in the PKTE. A user can normally configure it and let it run and obtain random numbers. The implementation though prevents it from passing one of the tests from the CAVP. The date/time value is incremented autonomously in step with the system clock and not with the algorithm. Also, due to buffer size constraints, it's not possible to obtain the 10,000th value for the Monte Carlo test.

As such, the RNG was implemented as a combination of software running ADSP-BF707 Blackfin+ core and using the PKTE. This allowed the successful testing for the RNG.

X9.31 RNG Algorithm Description

The RNG algorithm, as defined by the X9.31[3] standard is:

Let `e x K(Y)` represent the AES encryption of `Y` using key `K`. Also, let the following be the three inputs used by the RNG function:

- `K`, a 128-bit AES key
- `V`, a 128-bit seed value
- `DT`, a 128-bit date/time vector which is updated on each iteration

First, an intermediate value, `I`, is calculated by encrypting the date/time (`DT`) value with key `K`.

$$I = e\ x\ K(DT)$$

This intermediate value, `I`, is then XOR'ed with the seed value, `V`, and encrypted using key, `K`. The result is the random number output value.

$$R = e\ x\ K(I \oplus V)$$

To prepare for the next output, the seed value, `V`, is updated by XOR'ing the current output, `R`, with the previously calculated intermediate value, `I`, and then encrypted with key, `K`.

$$V = e\ x\ K(R \oplus I)$$

As mentioned before, the date/time value, `DT` is also updated for the next iteration.

X9.31 RNG Implementation on ADSP-BF707 Processors

The RNG algorithm described above can be implemented as a combination of software running on the processor core and using the PKTE. The `BF70xPrngSim` project contained in the associated `.ZIP` file shows how this is accomplished.

The PKTE is set up to run three times to perform the encryptions. Besides configuring the PKTE, the processor core also calculates the XOR of the random value output and the intermediate value, *`I`*. Finally, the *`DT`* value is also updated for the next iteration. This is basically incrementing a 128-bit value by one.

Hash-Based Message Authentication Code (HMAC)

The HMAC algorithm is specified in the FIPS 198-1 publication[4]. The MAC function helps to verify the integrity and authentication of the input data simultaneously.

HMAC Algorithm Description

The HMAC algorithm is described next. The HMAC can be calculated with any cryptographic hash function. On ADSP-BF707 processors, HMAC can be performed using SHA-1, SHA-224 or SHA-256. The following are parameters used in the calculation of an HMAC output.

- *`K`*, input key
- *`K0`*, key used in HMAC calculation
- *`B`*, block size in bytes
- *`ipad`*, inner pad value of the byte 0x36 repeated B times
- *`opad`*, outer pad value of the byte 0x5c repeated B times
- *`L`*, size of the output hash digest in bytes

The formula below is used for calculating the HMAC result.

`HMAC(K, input) = H((K0 ⊕ opad) || H((K0 ⊕ ipad) || input))`

The first step is obtaining *`K0`*. If the length of the input key, *`K`* is the same length as the input data, *`B`*, then *`K0 = K`*. If the length of the input key `K`, is greater than `B`, calculate the hash of `K`, `H(K)` of length *`L`*. Append bytes of 0's to the output so that the length is equal to *`B`*. The same is done if an input key is less than *`B`*; bytes of zeros are appended until the size matches *`B`*.

The next step is to XOR *`K0`* with a string of 0x36's of length *`B`* bytes. The input data is then appended to this result and the hash applied to it.

`H((K0 ⊕ ipad) || input)`

This hash digest is then appended to a separately calculated output of the XOR'ing of *`K0`* and *`opad`*. Finally, the output of this is also hashed, producing the final HMAC result.

HMAC Implementation on ADSP-BF707 Processors

On ADSP-BF707 processors, HMAC is accomplished via a hybrid approach of software running on the processor core and the hardware accelerator, the PKTE. The core handles the padding, the XOR'ing and the setting up of the PKTE to run the chosen hash function used for the HMAC. The PKTE is mainly used to calculate the hash digest.

The PKTE can be used in two ways to calculate the HMAC result. One way is shown in Figure 1. The PKTE can be used to calculate hash digests up to five times potentially, depending on the length of the input key.

The hash function can be broken up into pieces. For example, in the formula, `K0` is XOR'ed with `ipad` and then the input is appended. This result is fed into a hash function. With the PKTE, this can be broken up into multiple hash operations because the PKTE can perform the calculation in multiple steps, not needing all the input at once. As such, in the aforementioned formula, `K0` XOR'ed with `ipad` and be calculated with one operation of the PKTE and then continued with another PKTE operation with the input data.

With this methodology, the final hash can also be separated into two hash functions.

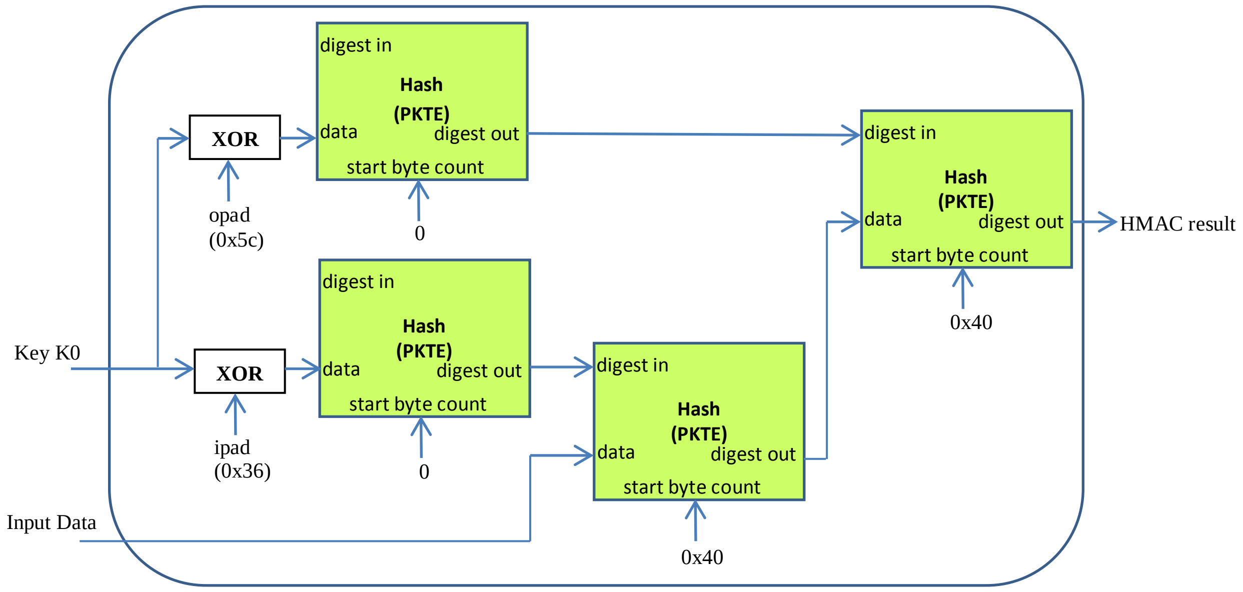

Figure 1. HMAC Calculation Block Diagram (Method 1)

Another way of performing the HMAC calculation, which is the method used in the CAVP, is to have the PKTE perform three hash operations (assuming *K0* was already obtained).

As shown in Figure 2, the inner hash and outer pad values XOR'ed with K0 can be hashed using the PKTE using two operations, and then the outputs, along with the input data can be fed into a third hash operation as a continuation of a previously started hash operation. The PKTE can be loaded with previously calculated inner and outer digest values and finish the HMAC calculation.

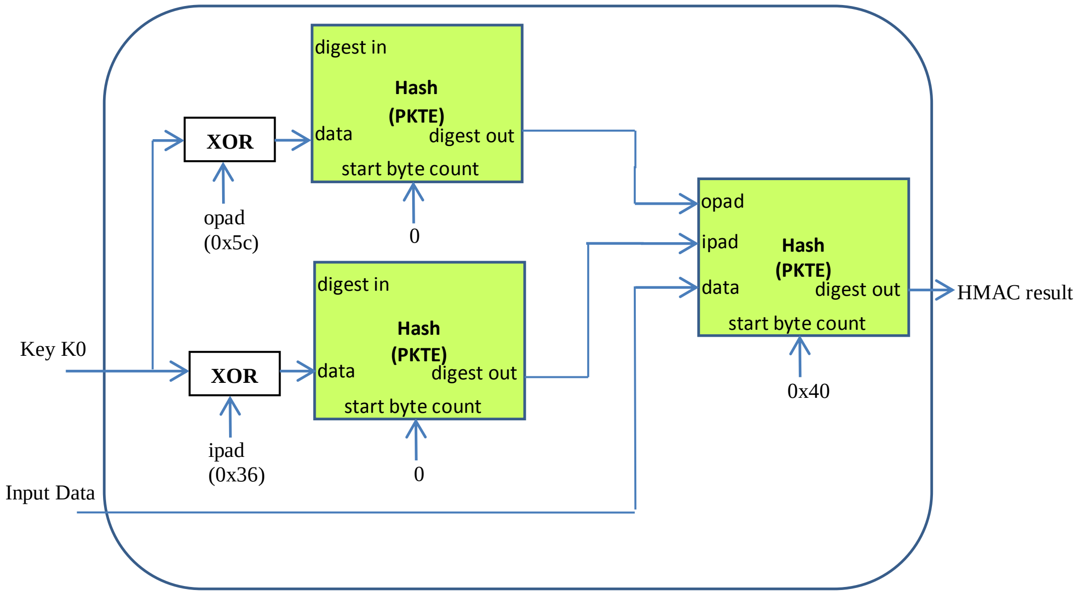

Figure 2. HMAC Calculation Block Diagram (Method 2)

Associated .ZIP File

This EE-Note is accompanied by a `.ZIP` file containing two CrossCore® Embedded Studio (CCES) projects. One is the implementation of the RNG algorithm (`BF70xPrngSim`), and the other one is the implementation of the HMAC function (`BF70xHMAC`), both described in the earlier sections.

Developed using CCES release 1.1.0, each of these two projects reads in an input file containing the test data, and outputs a single result file.

Conclusion

The Cryptographic Algorithm Validation Program (CAVP) is a pre-requisite test program to the Cryptographic Module Validation Program (CMVP) for FIPS 140-2 conformance.

To assist customers, the CAVP validation has already been performed and validated for select cryptographic functions. As previously discussed, most functions can be directly performed by the ADSP-BF707 PKTE hardware accelerator engine. The others are implemented as a combination of software running on the Blackfin+ processor core and the PKTE hardware accelerator, as illustrated in this EE-Note and the accompanying code provided in the associated `.ZIP` file.

References

[1] *Federal Information Processing Standards Publication 140-2 (`http://csrc.nist.gov/publications/fips/fips140-2/fips1402.pdf`).* May 25, 2001. National Institute of Standards and Technology.

[2] *ADSP-BF70x Blackfin+ Hardware Reference.* Rev 0.2. May 2014. Analog Devices, Inc.

[3] *Digital Signatures Using Reversible Public Key Cryptography for the Financial Services Industry (rDSA).* Rev X9.31-1998, September 9, 1998. American National Standards Institute.

[4] *The Keyed-Hash Message Authentication Code (HMAC). Rev FIPS PUB 198-1*, July 2008. National Institute of Standards and Technology.

Readings

[5] *Frequently Asked Questions for the Cryptographic Module Validation Program* (`http://csrc.nist.gov/groups/STM/cmvp/documents/CMVPFAQ.pdf`). June 5th, 2014. National Institute of Standards and Technology.

Document History

Revision	Description
Rev 1 – December 11, 2014 by G. Yi	Initial Release