RSA BSAFE®

Crypto-C

Cryptographic Components for C

Intel® Security Hardware User's Guide

Version 4.3

RSA SECURITY™

Copyright Notice

Contents

Chapter 1

Overview

RSA Security Inc. and Intel Corporation have teamed to provide C programmers access to the Intel Random Number Generator via the RSA BSAFE® Crypto-C interface.

Intel Hardware Security Features

The Intel® hardware security features are intended to provide a hardware infrastructure for cryptographic functions, such as random number generation. Version 1.0 of the hardware security features includes the Intel Random Number Generator (RNG), dedicated hardware that harnesses system thermal noise to generate random and indeterministic values. The generator is free-running, accumulating random bits of data until a 32-bit buffer is filled.

RSA BSAFE Crypto-C Interface

The RSA BSAFE Crypto-C software includes the RSA BSAFE Hardware API (BHAPI) interface, which allows manufacturers to provide an interface between their hardware and Crypto-C. Programmers who wish to take advantage of the hardware capabilities of a specific manufacturer, such as those provided by Intel, can simply incorporate certain features in their BSAFE Crypto-C software applications.

How This Book Is Organized

The audience for this document is application programmers who are familiar with Crypto-C and who wish to benefit from Intel's hardware security features in a Crypto-C application. The following topics are covered:

- Chapter 1, "Overview" (this chapter) gives an overview of the Intel hardware security features and the Crypto-C hardware interface.
- Chapter 2**, "**Using Intel Hardware With Crypto-C" describes the hardware chooser, and how to use it to specify the hardware features you wish to access.
- Chapter 3, "Using the Intel Random Number Generator" describes the Intel Random Number Generator (RNG) and presents sample code that shows how to use the RNG to collect random bits for your application.
- Appendix A, "Crypto-C Error Codes" lists hardware-specific error codes for the Intel hardware running under Crypto-C.
- Appendix B, "Intel Security Hardware Error Codes" lists the error codes returned by the underlying Intel hardware.
- Appendix C, "Redistributing the Intel Security Driver" describes how to resdistribute the Intel security driver and lists the locations of the installed driver files.

Additional Documentation

- *RSA BSAFE Crypto-C Library Reference Manual*: This manual contains the function prototypes and descriptions of the Crypto-C algorithm information types and key information types.
- *RSA BSAFE Crypto-C User's Manual*: This manual describes the Crypto-C six-step model, provides an overview of the cryptography used in Crypto-C, and gives extensive examples of how to use Crypto-C.

Overview of a Crypto-C Hardware Application

Creating a Crypto-C application that can use Intel's security hardware features is similar to creating any Crypto-C application. If you are not familiar with Crypto-C, you may wish to consult the introductory example in Chapter 1 of the *Crypto-C User's Guide*. For an application that will use hardware, the following differences should be noted:

Additional algorithm info types (AIs). Crypto-C provides some AIs that are designed to work only with hardware. These AIs extend the functionality of the Crypto-C application when the compatible hardware is present.

Additional algorithm methods (AMs). Recall that the AIs are merely the vessels that are used to set up the Crypto-C programming interface. The algorithm methods (AMs) do the actual work. In a software-only application, these methods are all part of the Crypto-C library. For a hardware-aware application, Crypto-C makes available additional methods that can be used to access the hardware.

Changes to the chooser structure. The chooser specifies the algorithm methods an application can use. In order to make your application as flexible as possible, and give you access to hardware from multiple vendors, Crypto-C extends the chooser model for hardware-aware applications. In this case, you create two choosers: a chooser that contains the methods built into Crypto-C, including any Crypto-C hardware-aware methods you need, and a second hardware chooser, that lists the methods supplied by the manufacturer, such as Intel, of the hardware you wish to use. Then you must call the function `B_CreateSessionChooser` to combine these two choosers and actively associate the manufacturer's method with the generic method supplied in Crypto-C. This created chooser can be used wherever you would have used the standard chooser in Crypto-C.

The Six-Step Sequence

The model for building a hardware-aware application with Crypto-C is similar to the six-step model described in Chapter 1 of the *Crypto-C User's Manual*. The differences are as follows:

1. Create: At this point you may want to create the session chooser that will be used in Step 3.
2. Set
3. Init: In this step, you must pass a modified chooser, the *session chooser,* to your initialization function. (The session chooser must be created earlier, but is passed in at this point.)
4. Update
5. Final
6. Destroy: In this step, in addition to calling the Destroy function and freeing any allocated memory, you should free the session chooser you created in Step 3.

If you are modifying an existing application to use hardware, you may also need to call different AIs and/or AMs in place of the software info types and methods you are currently using.

Chapter 2

Using Intel Hardware With Crypto-C

Crypto-C uses the RSA BSAFE Hardware API (BHAPI) to access the Intel hardware security features. In order to use this interface, you need to match the appropriate Crypto-C and Intel algorithm methods via a session chooser. In addition, your application must be linked with the `SEC32IPI.lib` library that comes with the security driver provided by Intel. See Appendix C, "Redistributing the Intel Security Driver" for instructions on installing the Intel security driver.

Algorithm Methods

The Intel features are supplied to BHAPI via application-specific algorithm methods (AMs), which are then used to set the appropriate algorithm object. Therefore, as the first step in setting up your BHAPI application, you must set up a session chooser that specifies which hardware methods to use.

Crypto-C Algorithm Methods

The Crypto-C interface specifies certain AMs that can be used for hardware. These AMs are generic and do not provide any hardware functionality; they are stubs that provide an interface between Crypto-C and the features of any hardware vendor. The vendor supplies a vendor-specific AM that can be used to access their hardware capabilities. The AMs required to support the Intel security hardware are included as part of Crypto-C.

As with all algorithm methods, the hardware-compatible AMs in Crypto-C are only available for certain algorithm info types (AIs). The AIs in Crypto-C offer differing levels of support for hardware, as follows:

- Crypto-C AIs that support hardware only, such as `AI_HW_Random`. For these AIs, the hardware-compatible AMs are listed in the *Crypto-C Library Reference Manual* as “Algorithm methods to include in application’s algorithm chooser.”
- Crypto-C AIs that support both hardware and software. For these AIs, the hardware-compatible algorithm methods are listed in the *Crypto-C Library Reference Manual* as “Token-based algorithm methods.”
- Crypto-C AIs that support only software and cannot be used with a hardware device. In this case, there are no AMs that can be used for hardware.

The hardware-compatible AMs supported by these AIs are part of the generic Crypto-C/BHAPI interface. To maintain flexibility in Crypto-C and allow support for hardware created by different manufacturers, BHAPI requires the manufacturer to supply a hardware method that is specific for the device.

Intel Hardware Algorithm Methods

To use the Intel hardware security features, you must create a special session chooser, which associates the Intel-specific hardware methods with the generic hardware methods built into the Crypto-C interface.

Table 2-1 **Intel hardware capabilities and corresponding Crypto-C functions**

Generic Hardware AM	Intel AM to Use in Session Chooser	Crypto-C AI	Primitive
AM_HW_RANDOM	HW_INTEL_RANDOM	AI_HW_Random	random number generation

The Session Chooser

Any Crypto-C application which uses hardware requires your application to declare two choosers:

- The baseline *software chooser*, such as the one that is used in any Crypto-C application. This chooser must be modified to include the generic Crypto-C hardware methods that support the desired hardware.
- The *hardware chooser*, that lists the manufacturer-specific hardware methods that can be used. This chooser has no analogue in a software-only application.

Once these choosers have been declared, you can combine them via a call to `B_CreateSessionChooser`. This creates the *session chooser*, which matches the hardware-compatible AMs in the software chooser with their actual instantiations in the hardware chooser. Specifically, for each item in the software chooser that references a hardware method, the hardware method replaces it.

Creating the Session Chooser

The following example shows how a simple session chooser can be created. This chooser is set up to access Intel's random number generator, which can then be used via the Crypto-C AI, `AI_HW_Random`.

Creating the Software Chooser

The software chooser is a conventional chooser, containing pointers to Crypto-C algorithm methods. In this case, since we are going to generate random numbers using the Intel hardware random number generator, we must include `AM_HW_RANDOM`. This algorithm method instructs Crypto-C to use a hardware method for generating random data. In an actual application, you would also list all the software methods, such as methods for encryption and decryption, that are required by your application. For random number generation, the software chooser can be set up as follows:

```
B_ALGORITHM_METHOD *SOFTWARE_CHOOSER[] = {
  &AM_HW_RANDOM,
  (B_ALGORITHM_METHOD *)NULL_PTR
};
```

Creating the Hardware Chooser

A hardware chooser is a list of manufacturer-supplied HW_TABLE_ENTRYs. Each entry defines the necessary code for accessing the specified piece of hardware. In the case of the Intel hardware random generator, use HW_INTEL_RANDOM.

If you wish, the hardware chooser can contain several HW_TABLE_ENTRYs, possibly supplied by different manufacturers, that all correspond to the same AM in the software chooser. If more than one hardware method can be matched to a single software method, then the hardware method listed first in the hardware chooser is associated with that software method. This association is normally created at link time. This offers applications the option to ensure a certain level of security by requiring specific hardware. However, the list can be modified at run time by creating the hardware chooser at run time, and specifying the order of the HW_TABLE_ENTRYs at that time.

If the hardware corresponding to the first method is not available, then Crypto-C continues down the list in order until a method corresponding to available hardware is found. If there is no hardware available for this method, then it defaults to the software method, if available, or returns an error if not.

```
HW_TABLE_ENTRY *HARDWARE_CHOOSER[] = {
  &HW_INTEL_RANDOM,
  (HW_TABLE_ENTRY *)NULL_PTR
};
```

Creating the Session Chooser

Once you have declared the software and hardware choosers, you can call B_CreateSessionChooser to associate a hardware method (for example, HW_INTEL_RANDOM) with a software method (AM_HW_RANDOM) so that when a software method is called, it turns to the hardware. In this example, HW_INTEL_RANDOM will be called when AM_HW_RANDOM is referenced.

```
if ((status = B_CreateSessionChooser
     (SOFTWARE_CHOOSER, &CHOOSER, (POINTER *)HARDWARE_CHOOSER,
      (ITEM *)NULL_PTR, NULL_PTR, &oemTagList)) != 0)
  break;
```

The session chooser you have created should be passed in as the chooser when you make the actual Crypto-C function call during the Crypto-C Init step, for example, as the chooser argument to `B_RandomInit`.

Hardware Availability

When you specify a specific hardware device via a manufacturer-specific AM, such as `HW_INTEL_RANDOM`, the application will verify that the hardware is present during the call to `B_CreateSessionChooser`. If the hardware is not present, Crypto-C will return an error. For some AMs, Crypto-C will default to a software implementation; for other AMs, such as `AM_HW_RANDOM`, there is no corresponding software AM and it is up to the application designer to decide how to proceed when the hardware is not present. Where relevant, the code examples for the specific Intel features give suggestions on how to proceed in the absence of the Intel hardware.

Hardware Errors

If the hardware fails, Crypto-C will return an error of BE_HARDWARE or BE_NOT_SUPPORTED. BE_HARDWARE indicates that the Intel primitive has returned an error. This error can be retrieved using `B_GetExtendedErrorInfo` (described in the *Crypto-C Library Reference Manual*), as shown below. In this example, ***randomAlgorithm*** is an algorithm that has been created to retrieve a seed from the Intel Random Number Generator. The data returned in the data field of ***errorData*** is a structure of type A_RSA_EXTENDED_ERROR (see A_RSA_EXTENDED_ERROR on page 11). The third parameter will return a pointer to the algorithm method that was in use when the error was encountered:

```
ITEM errorData;
POINTER am;

B_GetExtendedErrorInfo(randomAlgorithm, &errorData, &am);

/* Print out the error information. */
if (am == &HW_INTEL_RANDOM) {
  printf ("[Seed]  Code: %d\n",
          ((A_RSA_EXTENDED_ERROR *)errorData.data)->errorCode);
  printf ("[Seed]  Message: %s",
          ((A_RSA_EXTENDED_ERROR *)errorData.data)->errorMsg);
}
```

For information about the Intel error codes returned by `B_GetExtendedErrorInfo`, consult Appendix B.

An error of BE_NOT_SUPPORTED may mean that there is a problem accessing the BHAPI driver. See Appendix A for more information.

A_RSA_EXTENDED_ERROR

This Crypto-C structure is defined specifically for retrieving Intel error codes. It is defined as follows:

```
typedef struct {
    UINT4   errorCode;
    char    errorMsg[128];
} A_RSA_EXTENDED_ERROR
```

Definitions:

errorCode

The error code returned by the Intel hardware.

errorMsg

A `NULL`-terminated description of the error provided by Crypto-C.

Chapter 3

Using the Intel Random Number Generator

This chapter gives some background on random number generators and shows how to use the Intel Random Number Generator (RNG) with a Crypto-C application.

Random Numbers

All cryptosystems, whether secret-key systems like DES or public-key systems like RSA encryption, need a good source of cryptographic random numbers. The random numbers are used to generate input such as keys and initialization vectors. A good random number source should produce numbers that are unpredictable. Random numbers can be produced via hardware, such as the Intel RNG, or via a software pseudo-random number generator (PRNG), such as the PRNGs in Crypto-C, that has been seeded with true random input. A *seed* is unpredictable input, generated by hardware or manually by the user, that is used to set the initial state of the PRNG.

Note: An application with strong security requirements should use multiple sources of seeding and not rely on any single point of attack. For example, such an application might combine random bytes from the Intel Random Number Generator with user-generated input, such as gathering a seed through mouse movement and/or the keyboard. Random numbers generated by combining multiple sources of information should always be used as a seed to a PRNG, and never be used directly.

The Intel Random Number Generator

The Intel Random Number Generator is dedicated hardware that harnesses system thermal noise to generate random values. The generator is free-running, accumulating random bits of data until a 32-bit buffer is filled.

Whitening Hardware Results

The bits the Intel RNG supplies to the application have been *whitened* by the hardware; that is, a post-processing algorithm has been applied to reduce patterns in the hardware bits and make them less predictable. The advantage of performing whitening in software as well as hardware is that an attacker must modify the hardware and the software to make the HRNG leak secret information.

If you are seeding a pseudo-random number generator, you can use the random number without whitening for optimal performance. If you plan to use the random numbers directly, you may wish to apply additional whitening. Since the Intel RNG performs its own whitening, performing additional whitening may reduce the performance of your application.

Using the Intel RNG

The Intel RNG enables your application to get the seed bits that are needed to produce cryptographic keys and challenges that in turn can protect vast quantities of data. In a few milliseconds, the Intel RNG can produce all the random bits needed to seed an application. This is significantly faster than the software mechanisms for gathering unpredictable bits. Software mechanisms can take as long as ten seconds to gather a seed and often require user input (for example, via the mouse or keyboard).

Unavailability of Hardware

If the Intel RNG is unavailable, then the appropriate action depends on the security needs of the application. If the Intel RNG is not working at start-up, and thus there are no seed bits available from hardware randomness, then an application with exceptionally high security needs may want to inform the user and exit. Most applications can simply notify the user and request a user-supplied seed.

Pseudo-Random Number Generators (PRNGs)

Crypto-C provides several pseudo-random number generators that can be seeded via the Intel RNG and used to generate random numbers. The PRNGs in Crypto-C satisfy mathematical tests that measure randomness and are considered cryptographically secure. The Intel RNG can be used to provide a quick, secure seed to a PRNG. Once a PRNG has been seeded, it produces output up to ten thousand times faster than a hardware random number generator. In addition, a PRNG will not fail unless the CPU does. For most applications, using a PRNG that has been randomly seeded by the Intel Random Number Generator will provide the level of security needed, will be faster, and will avoid any potential problems due to hardware failure.

A PRNG should be reseeded at least every 2^{68} bytes of output.

Generating Random Numbers

This example demonstrates how to use the Intel Hardware Random Number Generator to seed a software-based pseudo-random number generator (PRNG). To generate random numbers, do the following:

1. Use the Intel Random Number Generator to generate a random seed. In general, you should use a seed that is at least 256 bits long.
2. Seed a pseudo-random number generator with the random value that you retrieved in the first step. Once you have provided a seed, you can use the PRNG to generate your random numbers.

If you are already using a Crypto-C PRNG in your applications, making the change to use the Intel Random Number Generator is easy. All you have to do is gather the seed as in 1 above, then make some minor changes to your existing implementation so that it can use the seed supplied by the Intel RNG.

Obtaining a Random Seed from Hardware

First, use the Intel Random Number Generator to acquire a random seed. To do this, you can write a function, `GenerateSeed`, that will retrieve random bytes from hardware.

Step 0: Create the Session Chooser

Before you can create an application that can access the Intel Random Number Generator, you need to create the session chooser that associates Intel's hardware method, `HW_INTEL_RANDOM`, with Crypto-C's generic method for hardware random number generation, `AM_HW_RANDOM`. First set up your software and hardware choosers, then call `B_CreateSessionChooser`. This call will combine the elements of the software chooser with those in the hardware chooser, associating Intel's hardware method (`HW_INTEL_RANDOM`) with `AM_HW_RANDOM`, so that when `AM_HW_RANDOM` is called, Crypto-C turns to the hardware. For more information see "The Session Chooser" on page 7.

```
B_ALGORITHM_METHOD *SOFTWARE_CHOOSER[] = {
  &AM_HW_RANDOM,
  (B_ALGORITHM_METHOD *)NULL_PTR
};
```

```
HW_TABLE_ENTRY *HARDWARE_CHOOSER[] = {
  &HW_INTEL_RANDOM,
  (HW_TABLE_ENTRY *)NULL_PTR
};

B_ALGORITHM_METHOD **CHOOSER = (B_ALGORITHM_METHOD **)NULL_PTR;

if ((status = B_CreateSessionChooser
     (SOFTWARE_CHOOSER, &CHOOSER, (POINTER *)HARDWARE_CHOOSER,
      (ITEM *)NULL_PTR, (POINTER *)NULL_PTR, &oemTagList)) != 0)
  break;
```

Step 1: Create an Algorithm Object

The next task is to create the algorithm object. This object will control the random byte generation. Creating the object only allocates the memory needed for the process. It does not initialize the object for random number generation.

```
  B_ALGORITHM_OBJ randomAlgorithm = (B_ALGORITHM_OBJ)NULL_PTR;

  if ((status = B_CreateAlgorithmObject (&randomAlgorithm)) != 0)
    break;
```

Step 2: Set the Algorithm Object

Set the algorithm info. We will specify `AI_HW_Random`, which will point to the hardware method that is associated to `AM_HW_RANDOM` via `B_CreateSessionChooser`.

```
  if ((status = B_SetAlgorithmInfo
        (randomAlgorithm, AI_HW_Random, NULL_PTR)) != 0)
    break;
```

Step 3: Initialize the Random Object

Initialize ***randomAlgorithm*** to generate random bytes. Here we pass the ***CHOOSER*** that was created via the call to `B_CreateSessionChooser` above. This chooser contains pointers to the hardware method that was associated with `AM_HW_RANDOM`.

```
    if ((status = B_RandomInit
         (randomAlgorithm, CHOOSER, (A_SURRENDER_CTX *)NULL_PTR)) != 0)
      break;
```

Step 4: Update the Random Object

Step 4 is not needed for random number seeding in hardware.

Step 5: Generate Random Bytes

Generate the random bytes for the seed. In this example, you will have the Crypto-C SDK generate ***seedMaxLength*** random bytes, storing the data in ***seedBytes***. The last parameter is a surrender context. In this case, generating random bytes should be very quick, so you can pass in a properly cast `NULL_PTR`.

```
    int seedBytes = 128;

    if ((status = B_GenerateRandomBytes
         (randomAlgorithm, seedBytes, seedMaxLength,
          (A_SURRENDER_CTX *)NULL_PTR)) != 0)
      break;
```

Note: If the Intel RNG is not present, or it returns an error, `B_GenerateRandomBytes` will return a non-zero value. (For more information, see Appendix A, “Error Codes”.) The appropriate action depends on the security needs of your application.

Step 6: Destroy All Objects

Step 6a:Destroy the Algorithm Object

Destroy the algorithm object. This step will free any allocated memory used by ***randomAlgorithm***. The memory is overwritten with zeros before it is deallocated, so that any potentially sensitive information is not left in memory.

```
    B_DestroyAlgorithmObject (&randomAlgorithm);
```

Step 6b: Free the Session Chooser

Free the session chooser. It is important to free the session chooser, so that any handles to hardware and allocated memory are released.

```
    if ((status = B_FreeSessionChooser (&CHOOSER, &oemTagList)) != 0)
      break;
```

Retrieving Hardware Error Codes

If the hardware fails or cannot return a seed, Crypto-C will return an error of BE_HARDWARE or BE_NOT_SUPPORTED. BE_HARDWARE indicates that the Intel Random Number Generator has returned an error. This error can be retrieved using `B_GetExtendedErrorInfo`, as shown below:

```
      ITEM errorData;
      POINTER am;

      /* Call B_GetExtendedErrorInfo to retrieve the error information.
         The data returned in the data field of errorData is a structure
         of A_RSA_EXTENDED_ERROR type.  The third parameter
         will return a pointer to the algorithm method that was in
         use when the error was encountered.  */
      B_GetExtendedErrorInfo(randomAlgorithm, &errorData, &am);

      /* Print out the error information. */
      if (am == &HW_INTEL_RANDOM) {
        printf ("[Seed]  Code: %d\n",
                ((A_RSA_EXTENDED_ERROR *)errorData.data)->errorCode);
        printf ("[Seed]  Message: %s",
                ((A_RSA_EXTENDED_ERROR *)errorData.data)->errorMsg);
      }
```

For information about the Intel error codes returned by `B_GetExtendedErrorInfo`, consult Appendix B. An error of BE_NOT_SUPPORTED may mean that there is a problem accessing the BHAPI driver. See Appendix A for more information.

Generating Random Numbers in Software

Once you have a random seed, you can generate pseudo-random numbers in software. After the seed has been passed to the software algorithm info type, this is similar to any Crypto-C PRNG implementation. The only difference is the fast, truly random, seed operation. For this example, you will use Crypto-C's SHA1 PRNG to generate random numbers.

Note: This example will work whether the seed was gathered from the Intel RNG or via another, backup method.

The example in this section is almost identical to the example in the *Crypto-C User's Manual*, "Generating Random Numbers." Steps 1, 2, 3, and 6 are identical; the only difference is in the seeding of the PRNG in Step 4 and the random number generation in Step 5.

Note: For this software call, you do not need to create a special session chooser. A standard Crypto-C software chooser is sufficient.

Step 1: Create an Algorithm Object

As before, you need to start by creating an algorithm object. This is identical to the software implementation

```
if ((status = B_CreateAlgorithmObject (&randomAlgorithm)) != 0)
  break;
```

Step 2: Set the Algorithm Object

To set the random algorithm object to use Crypto-C's SHA1 random number generator, you need to supply the appropriate algorithm info type. For SHA1, this is `AI_X962Random_V0`. Again, this is identical to a software implementation.

Note: This algorithm info type is named after the standard where the pseudo-random number generator is defined. Because SHA1 is considered one of the most secure implementations for creating pseudo-random numbers, there are a number of SHA1 random number generators in the literature. All of them use SHA1, but may differ in certain implementation details. Therefore, the AI is named after the standard for clarity and precision.

```
if ((status = B_SetAlgorithmInfo
     (randomAlgorithm, AI_X962Random_V0, NULL_PTR)) != 0)
  break;
```

Step 3: Initialize the Random Algorithm

To initialize the random algorithm, you must pass the algorithm object, the algorithm chooser, and a surrender context. As mentioned before, the algorithm chooser does not need to be a session chooser; a simple software chooser will suffice, so this call is also identical to a software implementation.

```
B_ALGORITHM_METHOD *RANDOM_CHOOSER[] = {
  &AM_SHA_RANDOM,
  (B_ALGORITHM_METHOD *)NULL_PTR
};

  if ((status = B_RandomInit
       (randomAlgorithm, RANDOM_CHOOSER,
        (A_SURRENDER_CTX *)NULL_PTR)) != 0)
    break;
```

Step 4: Seed the Random Object

In this step, you will seed the random object using the seed generated by the Intel RNG. If the RNG cannot be found, or fails during processing, you can ask the user to enter a seed value.

Note: If the Intel RNG is not working at start-up, and there are no seed bits available from hardware randomness, then a very high-security application might want to inform the user and exit.

First, acquire the random seed. To do this you must supply a function, `GenerateSeed`, to retrieve the random seed from hardware. An example of the `GenerateSeed` function is shown in the previous section, “Obtaining a Random Seed from Hardware” on page 16:

```
  #define BYTES_TO_GENERATE = 128

  randomSeedLen = BYTES_TO_GENERATE;
  randomSeed = (unsigned char *)T_malloc (randomSeedLen);
  GenerateSeed (randomSeed, randomSeedLen);
```

Once you have the random seed and its length, pass both into `B_RandomUpdate`. This call would be identical in a software implementation:

```
    if ((status = B_RandomUpdate
         (randomAlgorithm, randomSeed, randomSeedLen,
          (A_SURRENDER_CTX *)NULL_PTR)) != 0)
      break;
```

Step 5: Generate Random Numbers

Before calling `B_GenerateRandomBytes`, prepare a buffer for receiving the random bytes. This is a little different than the software implementation.

```
    randomByteBuffer = T_malloc (BYTES_TO_GENERATE);
    if ((status = (randomByteBuffer == NULL_PTR)) != 0)
          break;

    T_memset (randomByteBuffer, 0, BYTES_TO_GENERATE);
```

Now you can generate the random bytes. Since generating 128 bytes is quick, you can use a `NULL_PTR` for the surrender context:

```
    if ((status = B_GenerateRandomBytes
         (randomAlgorithm, randomByteBuffer, BYTES_TO_GENERATE,
          (A_SURRENDER_CTX *)NULL_PTR)) != 0)
      break;
```

Step 6: Destroy all Objects

Remember to destroy all objects when you are done with them, and free all memory. Again, this is identical to the software implementation:

```
  B_DestroyAlgorithmObject (&randomAlgorithm);
  T_memset (randomSeed, 0, randomSeedLen);
  T_free (randomSeed);
  T_free (randomByteBuffer);
```

Appendix A

Crypto-C Error Codes

Table A-1 lists the hardware-related error values returned by Crypto-C. If Crypto-C receives a hardware-level error from the Intel hardware, Crypto-C will return `BE_HARDWARE`. The underlying Intel error code can be retrieved using the Crypto-C `B_GetExtendedErrorInfo` function. See Appendix B for a description of the Intel error codes.

Table A-1 **Crypto-C hardware-related error return values**

Return Value	Description
BE_NOT_SUPPORTED	The user attempted to call a BHAPI AI after no matching AM was found. Probable causes include: • The vendor's BHAPI driver is not properly installed in the Windows registry • The vendor's BHAPI driver failed to load • The hardware is not present.
BE_HARDWARE	The hardware is present, but has returned an error. Further information can be obtained by calling B_GetExtendedErrorInfo.

Appendix B

Intel Security Hardware Error Codes

Table B-1 lists the error values returned by the underlying Intel hardware. If Crypto-C returns an error of `BE_HARDWARE`, the underlying Intel error code can be retrieved using the Crypto-C function `B_GetExtendedErrorInfo`.

Table B-1 **Intel Security Hardware Error Codes**

Value	Description
ISD_EDISABLED	The hardware device has been disabled and can no longer be used.
ISD_EINPUT	The hardware device is not currently available.
ISD_ENOTAVAIL	This service is not supported by this implementation.
ISD_EOK	There was no error. The function executed successfully.
ISD_ETESTFAIL	The hardware device has failed internal tests and is no longer available.
ISD_EUNKNOWN	There is an unknown error in hardware.

Appendix C

Redistributing the Intel Security Driver

Determining That the Firmware Hub Is Installed on the Target System

Before installing the Intel Security Driver, you should verify that the firmware hub is installed on the target system, as follows:

Operating System	Firmware Hub Installation Check
Microsoft Windows 95 Microsoft Windows 98	Check the following registry key: HKEY_LOCAL_MACHINE\Enum\BIOS for *INT0800
Microsoft Windows NT 4.0	There is no way to detect the presence of the firmware hub.

Redistributing the Driver

The Intel Security Driver can be redistributed in two ways: via a silent install (using InstallShield) or via `.inf` files.

Redistributing via a Silent Install

To redistribute the Intel Security Driver in your security-based applications, add the following steps to your installation script:

1. Copy all files from the `\REDISTRIB` folder, located at the root of the Crypto-C CD, to the appropriate folder within your application build tree.
2. In your application installation script, include the following line to install the driver files on the destination (user's) machine.

   setup –s

   This command line runs an InstallShield* silent install.
3. Handle any error codes returned by the **setup –s** command line. (For more information, see the InstallShield documentation.)

Error Code	Meaning
0	The silent installation finished successfully.
-1	A general error occurred.
-3	Data that were requested during the silent installation were not found or were undefined.
-4	There is not enough memory available to continue.
-7	InstallShield could not create the log file (most likely InstallShield is trying to write the log file to CD-ROM or other read-only media).
-11	Unknown error during setup — the generic error message.
-51	InstallShield was unable to create the specified folder.
-52	InstallShield cannot access the specified folder.
-53	Invalid option selected.

4. Reboot the destination (user's) machine to activate the security driver.

Files Installed

The silent install places the driver files in the following locations:

File	O/S	Location	Description
ISECDRV.SYS	Microsoft Windows NT 4.0	\windows\system32\drivers	Legacy Microsoft Windows NT 4.0 driver
ISECDRV.SYS	Microsoft Windows 98	\windows\system32\drivers	Microsoft wdm style driver
ISECDRV.VXD	Microsoft Windows	\windows\system	Microsoft Windows 95 Plug and Play driver
SEC32IPI.lib	All three operating systems	<*drive*>:<*path*>\Sample Applications\Source	Intel® Security Driver IPI library file
UNINST.ISU	All three operating systems	<*drive*>:<*path*>	InstallShield uninstall file
ICSP.DLL	Microsoft Windows 95 Microsoft Windows 98	\windows\system	CSP DLL file
	Microsoft Windows NT 4.0	\WINNT\system32	
		where:	
		<*drive*> is the drive specified during installation (C:, by default)	
		<*path*> is the path specified during installation (\Program Files\Intel\Intel Security Driver, by default)	

Redistributing the Driver via .inf Files

Instead of running the InstallShield silent install, described above, you can have the user install the driver on the target system by doing the following.

User Instructions for Installing the Intel Security Driver

To install the Intel Security Driver, do one of the following, depending on whether you are installing on Microsoft Windows NT 4.0, Microsoft Windows 95, or Microsoft Windows 98.

Microsoft Windows NT 4.0

1. Copy the following files from the Crypto-C CD to a floppy disk.
```
\REDISTRIB
   \INF
      \WinNT4
         NTDriver.reg
         ISECDRV.SYS
```
2. Log on to the target system as the administrator.
3. Insert the floppy disk you created in step 1.
4. Copy `ISECDRV.SYS` from the floppy disk to:
`\WINNT\SYSTEM32\DRIVERS`
5. Edit the `NTDriver.REG` file to verify that the Imagepath setting specifies the correct path to the `WINNT` directory.
6. Run the `NTDriver.REG` file.
7. Restart the computer.
8. To determine if the driver is loaded and working properly:
Click **Start | Settings | Control Panel**, then double-click the Devices icon.
9. In the list of Devices, locate the `ISECDRV` device. Verify that its Status is "Started" then set its Startup to "Automatic" (via the Startup button).

Microsoft Windows 95

1. Copy the following files from the Crypto-C CD to a floppy disk.

```
\REDISTRIB
   \INF
      \Win95
         ISD_95.INF
         ISECDRV.VXD
```

2. Log on to the target system.
3. Insert the floppy disk you created in step 1.
4. Click **Start|Settings|Control Panel**.
5. Double-click the **System** Icon.
6. Select the **Device Manager** Tab on the System Properties dialog box.
7. Double-click the **Intel Firmware Hub** option in the System Devices section.
8. When the Intel Firmware Hub Properties page appears, select the **Driver** Tab.
9. Select the **Update Driver** Button.
10. Select the **Search for Driver** Option.

    Since the ISD_95.INF file and ISECDRV.VXD files are located on the floppy disk, Windows should automatically find the driver and prompt you to finish the installation.
11. After Windows has copied the driver you will be prompted to restart the machine. Select **Yes** at this prompt.
12. After Windows is restarted you should now find the Firmware Hub listed in the System Devices section of the Device Manager.

Microsoft Windows 98

1. Copy the following files from the Crypto-C CD to a floppy disk.

```
        \REDISTRIB
          \INF
             \WDM
                ISD_WDM.INF
                ISECDRV.SYS
                ISD_CAT.CAT
```

2. Log on to the target system.
3. Insert the floppy disk you created in step 1.
4. Click **Start | Settings | Control Panel**.
5. Double-click the **System** Icon.
6. Select the **Device Manager** Tab on the System Properties dialog box.
7. Double-click the **Intel Firmware Hub** option in the System Devices section.
8. Select the **Reinstall Driver** Button.
9. When the Update Device Driver Wizard appears, select the **Next** Button.
10. Select the "Search for a better driver ..." option, then click **Next**.
11. Select the Location of the ISD_WDM.INF File and Driver, then click **Next**.
12. After a few seconds Windows should find the Driver. When it does, click **Next** to finish loading the driver.
13. After the file copy is complete you should see a message reporting that Windows has finished installing an updated driver.
14. Click **Finish**.
15. You should now find the Firmware Hub listed in the System Devices section of the Device Manager.

Index