Using the Cryptography Interface (BCrypto)

STB_BCrypto-AN100-D1

REVISION HISTORY

Revision	*Date*	*Change Description*
STB_BCrypto-AN100-D1	01/30/09	Initial release.

Broadcom Corporation
5300 California Avenue
Irvine, CA 92617

TABLE OF CONTENTS

Section 1: Introduction

This document describes how to access the cryptography interface (abbreviated “BCrypto”) for the Settop API. It is part of the Broadcom Settop Reference Software Release. Refer to the document *Settop Reference Software User Guide* (document number STB_RefSw-SWUM10x-R) for a high-level overview of the Reference Software.

Access to cryptographic functionality in the Settop API is controlled by two interfaces, BCrypto and bstream_set_encryption. This document will address BCrypto.

BCrypto is intended to provide generic cryptographic functionality. Although there are controls for certain special hardware features (for example, performing decryption only on the payload portion of an MPEG packet), the interface is designed as a general-purpose cryptographic engine.

Other Settop API functionality may be built on top of BCrypto, or an application can take advantage of the cryptographic features for its own purposes. Encrypted PVR on the BCM74xx family of chips uses BCrypto.

GLOSSARY

- **3DES**: See TDES
- **AES**: Advanced Encryption Standard, the successor to DES for US Government cryptography; it is also widely used as a secure cipher and is replacing DES and TDES
- **Asymmetric algorithm**: a cryptographic algorithm in which an asymmetric key is used
- **Asymmetric key**: The key consists of at least two parts: a public key which can be shared with anyone, and a private key which only the owner possesses. In a given operation pair, the public key is used for one half of the operation, and the private key is used for the other half. For instance, a private key is used to sign an e-mail, and the matching public key is used to verify the signature.
- **Block cipher**: A block cipher uses a symmetric key to encrypt or decrypt a fixed-size block of data. Typically, encryption and decryption are not interchangeable, so that an encrypt-encrypt cycle on a block will not result in the original plaintext. Block ciphers often require a padding scheme to deal with data that does not fit into the block cipher’s block size.
- **CBC**: Cipher Block Chaining mode, a method by which the previous block is XOR’d with the plaintext prior to encryption (and an IV is used to start the process)
- **Cipher mode**: A means of feeding the previous result into the current block to prevent the key from being leaked in subsequent operations.
- **DEA**: See DES
- **DES**: Defense Encryption Standard, a US Government approved algorithm which was once widely used; it is considered relatively insecure today, and has been replaced in US Government use by AES
- **ECB**: Electronic Code Book mode, a method by which each block is treated discretely; this block mode can still leak patterns in the underlying data
- **Decrypt**: Reversing the encryption operation and applying it to ciphertext, resulting in the original plaintext
- **Encrypt**: Mathematically changing plaintext into ciphertext, resulting in obscured data; in modern cryptography, a decrypt operation must be used to restore the original plaintext
- **Hash**: A means of fingerprinting a large block of data as a single, fixed-size value; used to verify file integrity, and also used (in combination with asymmetric algorithms) when digitally signing files, messages, and e-mail
- **MD5**: A non-governmental hashing algorithm, commonly used for quick file verification

- **RSA**: A commonly used asymmetric algorithm, consisting of a public key, private key, and exponent; RSA stands for the initials of the inventors of the algorithm.
- **SHA-1**: A U.S. Government hashing algorithm, commonly used as a message digest for digitally signing e-mail (among other uses).
- **Sign**: Using a hash and an asymmetric algorithm private key to stamp a message such that it can be later shown to be signed by this particular key
- **Streaming cipher**: A streaming cipher uses a key and some operations to encrypt an arbitrarily-sized amount of data. Streaming ciphers do not require a padding scheme on the data being encrypted.
- **Symmetric algorithm**: A cryptographic algorithm in which a symmetric key is used
- **Symmetric key**: The same key value is used to perform both the encryption and decryption operation. When two different agents are used, both agents must possess the key (one to encrypt, one to decrypt).
- **TDES**: Triple DES, a means of extending the effective life of DES by using it three times with either two or three keys
- **Triple DES**: See TDES
- **Verify**: Using a hash and an asymmetric algorithm's public key to verify that the message is intact and that the private key linked to the public key was used to sign the message

Section 2: Introduction to Cryptography

Block Ciphers

Block ciphers are used to secure large amounts of data. They are generally performed on blocks of data, and the data needs to be padded to match the block size. Most common ciphers work on 8-byte blocks. Different block modes are used to chain blocks together (and not all declared block modes are supported by all algorithms). DES, Triple DES, and AES are block ciphers.

Streaming Ciphers

A streaming cipher, like a block cipher, works on large amounts of data. However, unlike a block cipher, a streaming cipher will operate on data one byte at a time. RC4 is a streaming cipher.

Public/Private Key

Asymmetric operations rely on two separate keys, a public key and a private key. The public key can be shared, and the two sides will each use their own private key with the opposite agent’s public key to perform operations. This is most often used with sign/verify operations. RSA is an example.

Hashing

A hash algorithm turns a large amount of data into a small fingerprint. This allows an operation such as sign to work, and relies on the hash preventing the data from being modified in a way that causes a hash collision. MD5 and SHA-1 are hashes.

Conditional Access

Conditional access is used to decrypt protected streams (either off-air or IP Settop). It may be built in terms of block ciphers (AES, DES, Triple DES) or proprietary algorithms. CA is accessed via bstream_set_encryption and is not supported via BCrypto.

Data Format

The AES, DES, and Triple DES hardware implementations can be set to encrypt only portions of the data. This allows the encryption of MPEG or similar video streams to operate only on the data payloads. This is not supported in software.

Section 3: Using BCrypto

BCrypto has three primary functions and four helper functions. The primary functions setup an operation, perform the operation (possibly called multiple times), and then close the operation. The helper functions guarantee proper allocation for hardware-based cryptography, initialize structures, and provide feedback.

The three primary functions are: `bcrypto_open()`, `bcrypto_process()`, and `bcrypto_close()`.

The four helper functions are: `bcrypto_settings_init()`, `bcrypto_status()`, `bcrypto_alloc()`, and `bcrypto_free()`.

Basic documentation is available in BSEAV/api/include/bsettop_crypto.h. This document addresses the BCrypto details.

A normal mode of operation would look something like this:

```
bcrypto_settings_init();
bcrypto_open();
while (!done) {
    bcrypto_process();
}
bcrypto_close();
```

Section 4: Parameters Guide

BCRYPTO SETTINGS

The following code lists the parameters for the BCrypto API:

```
typedef struct bcrypto_settings {
    bencryption_params encryption;
    bcrypto_operation operation;
    bcrypto_data_format data_format;
    bcryptp_hardware_ctrl hardware_ctrl;
    bool multiple_hashes;
} bcrypto_settings;
```

- `encryption` will be covered below.
- `operation` controls the operation being performed. With block ciphers, this function will typically be `bcrypto_operation_encrypt()` or `bcrypto_operation_decrypt()`. For public/private key algorithms such as RSA, this will usually be `bcrypto_operation_sign` or `bcrypto_operation_verify` (although there are certificate management functions which depend on encrypt/decrypt). For hashing algorithms, this will be set to `bcrypto_operation_hash`.
- `data_format` only applies to select algorithms when operating with hardware m2m, and it can be set to `bcrypto_data_format_raw` to process all data, `bcrypto_data_format_mpeg` to touch only mpeg packet payloads, or `bcrypto_data_format_dss` to touch only DSS packet payloads.
- `hardware_control` should be set to default (where `bcrypto_process` will choose whether to use the m2m or the software implementation), none for software-only, and m2m for hardware cryptography.
- `multiple_hashes` is only used for certain algorithms, and it allows *`bcrypto_process`* to handle cumulative hashes and to deal with underlying implementations which require a closure step. If this is set, there must be an additional call to `bcrypto_process` at the end of the entire process (before `bcrypto_close`) with an input buffer of NULL and an input length of 0 to allow the final calculation. At this point, the final hash will be returned in the output buffer. Hashes may or may not provide an intermediate hash when used in this mode.

BENCRYPTION PARAMETERS

The following code lists the parameters for the bcrypto API:

```
typedef struct bencryption_params {
 bencryption_type type;
 bcrypto_blockmode blockmode;
 bencryption_residue residue;
 bool odd;
 bool even;
 bool key_ladder;
 uint16_t pid;
 unsigned key_length;
 uint8_t key[32];
 uint8_t iv[32];
 const void *long_key;
} bencryption_params;
```

- `type` controls the encryption algorithm to use.
- `blockmode` indicates the blockmode to use. At this time, only CBC and ECB are implemented, and not all symmetric algorithms support both modes in all cases. Public key algorithms, RC4, and hashes do not use a blockmode.
- `residue, odd, even`, and `pid` are ignored by bcrypto at this time. This structure is shared with Conditional Access, and these variables are only used for CA.
- `key_length` is the length of the key in bits. If `long_key` is being used, this is the length of the structure pointed to by `long_key` in bits (for example, `sizeof(long_key_structure)*8`).
- `key` contains the key or keys used, provided the value(s) can fit into 256 bits. If the key is longer, use long_key. Hashes do not set a key.
- `iv` contains the initial vector required for CBC for supported symmetric algorithms. This variable is otherwise ignored.
- `long_key` is a pointer to either an array or a structure, depending on the algorithm. For instance, if a 40 byte key is used for RC4, `long_key` should point to an array with the key and `key_length` should be 320 (40*8). For RSA, there are special structures, documented in the section on RSA. This variable is also used to control key derivation via keyladder.

Section 5: Individual Algorithm Details

DES, THE DEFENSE ENCRIPTION STANDARD

DES, also known as DEA (the Defense Encryption Algorithm), is a Feistel-box algorithm developed by IBM and submitted to NSA. It was the first U.S. Government-approved algorithm to be published, and in many ways formed the basis of the modern cryptography community. DES is defined in the U.S. Government publication FIPS 46-3.

DES PARAMETERS

Using DES via BCrypto requires that the following parameters be set:

- `encryption.type` must be set to `bencryption_type_des`
- `encryption.key` must contain the 64 bit DES key
- `encryption.key_length` must be 64
- `operation` should be set to `bcrypto_operation_encrypt` or `bcrypto_operation_decrypt`
- `encryption.hardware_ctrl` may be `bcrypto_hardware_default, bcrypto_hardware_none`, or `bcrypto_hardware_m2m`
- `encryption.blockmode` should be set to `bcrypto_blockmode_ecb` or `bcrypto_blockmode_cbc`

If `encryption.blockmode` is `bcrypto_blockmode_cbc`, then `encryption.iv` needs to contain the 8-byte initial vector

DES PLATFORM-SPECIFIC NOTES

The BCM97400, BCM97401, and B97403 platforms do not support non-zero IV values at this time.

The BCM97400 A0 and BCM97403 A0 platforms do not support DES CBC.

The m2m implementation of DES does support the `data_format` values, allowing the encryption of only payload data. This is used for PVR encryption in Settop API.

The DES key for m2m may be derived via keyladder.

TRIPLE DES (TDES OR 3DES), THE DEFENSE ENCRIPTION STANDARD

Triple DES is based on DES and was developed to extend the life of DES. It is generally done as three operations with either two or three keys (encrypt-decrypt-encrypt or decrypt-encrypt-decrypt). With two-key TDES, the first key (key A) is used for the first and third operation, and the second key (key B) is used for the middle operation. TDES is defined in the U.S. Government Special Publication 800-67.

TDES Parameters

Using TDES via BCrypto requires that the following parameters be set:

- `encryption.type` must be set to `bencryption_type_3des`
- `encryption.key` must contain the 128-bit DES keys A and B, with the first eight bytes containing key A and the second eight bytes containing key B. Three-key TDES is not currently supported.
- `encryption.key_length` must be 128
- `operation` should be set to either `bcrypto_operation_encrypt` or `bcrypto_operation_decrypt`.
- `encryption.hardware_ctrl` may be `bcrypto_hardware_default, bcrypto_hardware_none,` or `bcrypto_hardware_m2m`
- `encryption.blockmode` should be set to `bcrypto_blockmode_ecb` or `bcrypto_blockmode_cbc`.

If `encryption.blockmode` is `bcrypto_blockmode_cbc`, then `encryption.iv` needs to contain the 8-byte initial vector

TDES Platform-Specific Notes

The BCM 97400, BCM97401, and BCM97403 platforms do not support non-zero IV values at this time. The BCM97400 A0 and BCM 97403 A0 platforms do not support TDES CBC.

The m2m implementation of TDES does support the `data_format` values, allowing the encryption of only payload data. This is used for PVR encryption in the Settop API.

The TDES key for m2m may be derived via keyladder.

AES, the advanced Encryption Standard

With DES and TDES reaching a point of ineffectiveness, a competition was held (sponsored by NIST) to find a replacement algorithm. Rijndael, one of the entries, was selected and became the new standard. AES is the common term for this algorithm, and it is in widespread use today. AES is defined by the U.S. Government publication FIPS 197.

AES Parameters

Using AES via BCrypto requires that the following parameters be set:

- `encryption.type` must be set to `bencryption_type_aes`
- `encryption.key` must contain the 128 bit AES key
- `encryption.key_length` must be 128
- `operation` should be set to either `bcrypto_operation_encrypt` or `bcrypto_operation_decrypt`
- `encryption.hardware_ctrl` may be `bcrypto_hardware_default`, `bcrypto_hardware_none`, or `bcrypto_hardware_m2m`
- `encryption.blockmode` should be set to `bcrypto_blockmode_ecb` or `bcrypto_blockmode_cbc`

If `encryption.blockmode` is `bcrypto_blockmode_cbc`, then `encryption.iv` needs to contain the 8-byte initial vector

AES Platform-Specific Notes

The BCM97400, BCM97401, and BCM97403 platforms do not support non-zero IV values at this time.

The m2m implementation of AES does support the `data_format` values, allowing the encryption of only payload data. This is used for PVR encryption in the Settop API.

The AES key for m2m may be derived via keyladder.

RC4

RC4 is a commonly used streaming cipher with currency in SSL communications and certificate-based e-mail. It is considered secure enough for some operations, but cryptographers generally discourage its use in new applications. WEP and WPA, two wireless encryption protocols, are based on RC4. WEP was broken fairly easily due to errors in implementation.

RC4 Parameters

Using RC4 via bcrypto requires that the following parameters be set:

- `encryption.type` must be set to `bencryption_type_rc4`
- `encryption.key` should contain the key if the key is less than or equal to 32 bytes (256 bits) in length
- `encryption.key_length` must be the length of the key in bits
- `operation` should be set to either `bcrypto_operation_encrypt` or `bcrypto_operation_decrypt`.
- `encryption.hardware_ctrl` may be `bcrypto_hardware_default` or `bcrypto_hardware_none`.
- `encryption.blockmode` is ignored.
- `encryption.long_key` should be NULL if the key length is 256 or less. For longer keys, it should point to an array of bytes containing the key.

RC4 Platform-Specific Notes

The BCM97400, BCM97401, and BCM97403 platforms do not support RC4 in hardware at this time.

RC4-Specific Notes

RC4 is currently only implemented in software. By default, software support for algorithms is not compiled in, so Settop API will need to be built with `BCRYPTO_SW_SUPPORT=y` to access this. It possible that future hardware or chip revisions may add RC4 support.

RSA

RSA is a public-key cryptographic algorithm suitable for general use. It is generally considered secure given sufficiently long keys. RSA was first described in 1977, and the name is derived from the names of the inventors (Ron Rivest, Adi Shamir, and Leonard Adleman). It is commonly used for SSL certificates, secure e-mail, and file verification.

RSA PARAMETERS

Using RSA via bcrypto requires the following parameters to be set:

- `encryption.type` must be set to `bencryption_type_rsa`
- `encryption.key` is ignored, as the supported key sizes require the use of `long_key`
- `encryption.key_length` must be 8*sizeof(struct referenced by `long_key`)
- *`operation`* should be set to `bcrypto_operation_encrypt`, `bcrypto_operation_decrypt`, `bcrypto_operation_sign`, or `bcrypto_operation_verify`
- `encryption.hardware_ctrl` may be `bcrypto_hardware_default` or `bcrypto_hardware_m2m`
- `long_key` should be a pointer to either a `bcrypto_rsa_1024` or a `bcrypto_rsa_2048` struct containing the key

RSA PLATFORM-SPECIFIC NOTES

At this time, RSA is only supported in hardware on the BCM97401 platform. When using RSA, `key_length` is used to determine which structure to use when de-referencing `long_key` (and how to copy the pointer when running in kernel mode).

Note: There is a `key_length` in the struct itself which also needs to have the appropriate value (1024 or 2048).

The hardware implementation only requires `d`, `e`, and `n`. In all cases, the keys should be stored as big-endian large numbers, with leading zero-padding as necessary. The `bcrypto_process` will trim values (such as `e`) if needed by the hardware.

RSA MISCELLANEOUS NOTES

BCrypto does not currently support RSA in software.

BCrypto does not provide a way to parse an RSA key blob from a certificate or similar package into the constituent parts for the structure.

SHA-1, THE SECURE HASH ALGORITHM

There are several Secure Hash Algorithms defined, with SHA-1 being the most common in current deployment. SHA-1 is the common hashing method used for signing e-mail and is in common use with SSL, PGP, SSH, S/MIME, and IPsec. SHA-1 is defined in the U.S. Government publication FIPS 180-2 (along with the higher-security SHA-224, SHA-256, SHA-384, and SHA-512).

SHA-1 PARAMETERS

Using SHA-1 via bcrypto requires that the following parameters be set:

- `encryption.type` must be set to `bencryption_type_sha1`
- *`operation`* should be set to `bcrypto_operation_hash`.
- `encryption.hardware_ctrl` may be `bcrypto_hardware_default` or `bcrypto_hardware_none`.
- `multiple_hashes` must be true if more than one call to `bcrypto_process` will be made.

SHA-1 NOTES

SHA-1 is currently only implemented in software. By default, software support for algorithms is not compiled in, so the Settop API will need to be built with `BCRYPTO_SW_SUPPORT=y` to access this. Future hardware or chip revisions may add SHA-1 support.

SHA-1 maintains a cumulative hash, so it does not require a final call to *bcrypto_process* with a 0 length and NULL input buffer to complete the hash.

SHA-1 requires the output buffer to be 20 bytes in length to hold the hash.

MD5, THE MESSAGE-DIGEST ALGORITHM

MD5 is a non-governmental hash function designed by the same cryptographer who designed RC4. It is frequently used for file integrity checks. There are known weaknesses in the algorithm which has resulted in it not being recommended for new applications. MD5 is specified in the Internet standards document RFC 1321.

MD5 PARAMETERS

Using MD5 via bcrypto requires the following parameters to be set:

- `encryption.type` must be set to `bencryption_type_md5`.
- `operation` should be set to `bcrypto_operation_hash`.
- `encryption.hardware_ctrl` may be `bcrypto_hardware_default` or `bcrypto_hardware_none`.
- `multiple_hashes` is ignored.

MD5 Notes

MD5 is currently only implemented in software. By default, software support for algorithms is not compiled in, so Settop API will need to be built with `BCRYPTO_SW_SUPPORT=y` to access this. Future hardware or chip revisions may add MD5 support.

MD5 is not implemented as a cumulative hash, so if a block of data needs to be hashed, the entire hash needs to be calculated in a single `bcrypto_process` call. If a second call is made, it will start over and not consider the previously hashed data. This may change in the future.

MD5 requires the output buffer to be 16 bytes in length to hold the hash.

DVB, C2, CSS

The DVB, C2 and CSS algorithms are not supported via BCrypto at this time.

Broadcom Corporation

5300 California Avenue
Irvine, CA 92617
Phone: 949-926-5000
Fax: 949-926-5203