Crypto Application

version 1.4

The Erlang/OTP SSL application includes software developed by the OpenSSL Project for use in the OpenSSL Toolkit (http://www.openssl.org/). Copyright (c) 1998-2002 The OpenSSL Project. All rights reserved.
This product includes cryptographic software written by Eric Young (eay@cryptsoft.com). This product includes software written by Tim Hudson (tjh@cryptsoft.com). Copyright (C) 1995-1998 Eric Young (eay@cryptsoft.com). All rights reserved.
For further OpenSSL and SSLeay license information se the chapter Licenses.

http://www.erlang.org

Typeset in LaTeX from SGML source using the DOCBUILDER 3.3.2 Document System.

Contents

1 Crypto User's Guide 1
1.1 Licenses . . . 1
1.1.1 OpenSSL License . . . 1
1.1.2 SSLeay License . . . 2

2 Crypto Release Notes 5
2.1 Crypto Release Notes . . . 5
2.1.1 Crypto 1.4 . . . 5
2.1.2 Crypto 1.3 . . . 5
2.1.3 Crypto 1.2.3 . . . 6
2.1.4 Crypto 1.2.2 . . . 6
2.1.5 Crypto 1.2.1 . . . 6
2.1.6 Crypto 1.2 . . . 6
2.1.7 Crypto 1.1.3 . . . 7
2.1.8 Crypto 1.1.2 . . . 7
2.1.9 Crypto 1.1.1 . . . 7
2.1.10 Crypto 1.1 . . . 7
2.1.11 Crypto 1.0 . . . 7

3 Crypto Reference Manual 9
3.1 crypto . . . 11
3.2 crypto . . . 13

Chapter 1

Crypto User's Guide

The *Crypto* application provides functions for computation of message digests, and functions for encryption and decryption.

This product includes software developed by the OpenSSL Project for use in the OpenSSL Toolkit (http://www.openssl.org/).

This product includes cryptographic software written by Eric Young (eay@cryptsoft.com).

This product includes software written by Tim Hudson (tjh@cryptsoft.com).

For full OpenSSL and SSLeay license texts, see Licenses [page 1].

1.1 Licenses

This chapter contains in extenso versions of the OpenSSL and SSLeay licenses.

1.1.1 OpenSSL License

```
/* ====================================================================
 * Copyright (c) 1998-2002 The OpenSSL Project.  All rights reserved.
 *
 * Redistribution and use in source and binary forms, with or without
 * modification, are permitted provided that the following conditions
 * are met:
 *
 * 1. Redistributions of source code must retain the above copyright
 *    notice, this list of conditions and the following disclaimer.
 *
 * 2. Redistributions in binary form must reproduce the above copyright
 *    notice, this list of conditions and the following disclaimer in
 *    the documentation and/or other materials provided with the
 *    distribution.
 *
 * 3. All advertising materials mentioning features or use of this
 *    software must display the following acknowledgment:
 *    "This product includes software developed by the OpenSSL Project
 *    for use in the OpenSSL Toolkit. (http://www.openssl.org/)"
 *
```

```
 * 4. The names "OpenSSL Toolkit" and "OpenSSL Project" must not be used to
 *    endorse or promote products derived from this software without
 *    prior written permission. For written permission, please contact
 *    openssl-core@openssl.org.
 *
 * 5. Products derived from this software may not be called "OpenSSL"
 *    nor may "OpenSSL" appear in their names without prior written
 *    permission of the OpenSSL Project.
 *
 * 6. Redistributions of any form whatsoever must retain the following
 *    acknowledgment:
 *    "This product includes software developed by the OpenSSL Project
 *    for use in the OpenSSL Toolkit (http://www.openssl.org/)"
 *
 * THIS SOFTWARE IS PROVIDED BY THE OpenSSL PROJECT ``AS IS'' AND ANY
 * EXPRESSED OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
 * IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR
 * PURPOSE ARE DISCLAIMED.  IN NO EVENT SHALL THE OpenSSL PROJECT OR
 * ITS CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
 * SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT
 * NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES;
 * LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION)
 * HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT,
 * STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE)
 * ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED
 * OF THE POSSIBILITY OF SUCH DAMAGE.
 * ====================================================================
 *
 * This product includes cryptographic software written by Eric Young
 * (eay@cryptsoft.com).  This product includes software written by Tim
 * Hudson (tjh@cryptsoft.com).
 *
 */
```

1.1.2 SSLeay License

```
/* Copyright (C) 1995-1998 Eric Young (eay@cryptsoft.com)
 * All rights reserved.
 *
 * This package is an SSL implementation written
 * by Eric Young (eay@cryptsoft.com).
 * The implementation was written so as to conform with Netscapes SSL.
 *
 * This library is free for commercial and non-commercial use as long as
 * the following conditions are aheared to.  The following conditions
 * apply to all code found in this distribution, be it the RC4, RSA,
 * lhash, DES, etc., code; not just the SSL code.  The SSL documentation
 * included with this distribution is covered by the same copyright terms
 * except that the holder is Tim Hudson (tjh@cryptsoft.com).
 *
 * Copyright remains Eric Young's, and as such any Copyright notices in
 * the code are not to be removed.
```

```
* If this package is used in a product, Eric Young should be given attribution
* as the author of the parts of the library used.
* This can be in the form of a textual message at program startup or
* in documentation (online or textual) provided with the package.
*
* Redistribution and use in source and binary forms, with or without
* modification, are permitted provided that the following conditions
* are met:
* 1. Redistributions of source code must retain the copyright
*    notice, this list of conditions and the following disclaimer.
* 2. Redistributions in binary form must reproduce the above copyright
*    notice, this list of conditions and the following disclaimer in the
*    documentation and/or other materials provided with the distribution.
* 3. All advertising materials mentioning features or use of this software
*    must display the following acknowledgement:
*    "This product includes cryptographic software written by
*     Eric Young (eay@cryptsoft.com)"
*    The word 'cryptographic' can be left out if the rouines from the library
*    being used are not cryptographic related :-).
* 4. If you include any Windows specific code (or a derivative thereof) from
*    the apps directory (application code) you must include an acknowledgement:
*    "This product includes software written by Tim Hudson (tjh@cryptsoft.com)"
*
* THIS SOFTWARE IS PROVIDED BY ERIC YOUNG ``AS IS'' AND
* ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
* IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
* ARE DISCLAIMED.  IN NO EVENT SHALL THE AUTHOR OR CONTRIBUTORS BE LIABLE
* FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL
* DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS
* OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION)
* HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT
* LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY
* OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF
* SUCH DAMAGE.
*
* The licence and distribution terms for any publically available version or
* derivative of this code cannot be changed.  i.e. this code cannot simply be
* copied and put under another distribution licence
* [including the GNU Public Licence.]
*/
```

Chapter 2

Crypto Release Notes

The Crypto Application provides functions for computation of message digests, and encryption and decryption functions.

This product includes software developed by the OpenSSL Project for use in the OpenSSL Toolkit (http://www.openssl.org/).

This product includes cryptographic software written by Eric Young (eay@cryptsoft.com).

This product includes software written by Tim Hudson (tjh@cryptsoft.com).

For full OpenSSL and SSLeay license texts, see Licenses [page 1].

2.1 Crypto Release Notes

This document describes the changes made to the Crypto application.

2.1.1 Crypto 1.4

Improvements and New Features

- The previously undocumented and UNSUPPORTED `ssh` application has been updated and documented. This release of the `ssh` application is still considered to be a beta release and (if necessary) there could still be changes in its API before it reaches 1.0.
  Also, more cryptographic algorithms have been added to the `crypto` application.
  *** POTENTIAL INCOMPATIBILITY ***
  Own Id: OTP-5631

2.1.2 Crypto 1.3

Improvements and New Features

- Added support for RFC 3826 - The Advanced Encryption Standard (AES) Cipher Algorithm in the SNMP User-based Security Model.
  Martin Bjrklund

2.1.3 Crypto 1.2.3

Fixed Bugs and Malfunctions

- Linked in drivers in the crypto, and asn1 applications are now compiled with the -D_THREAD_SAFE and -D_REENTRANT switches on unix when the emulator has thread support enabled.
  Linked in drivers on MacOSX are not compiled with the undocumented -lbundle1.o switch anymore. Thanks to Sean Hinde who sent us a patch.
  Linked in driver in crypto, and port programs in ssl, now compiles on OSF1.
  Minor makefile improvements in runtime_tools.
  Own Id: OTP-5346

2.1.4 Crypto 1.2.2

Improvements and New Features

- Corrected error handling. If the port to the driver that crypto uses is unexpectedly closed (which should not happen during normal operation of crypto), crypto will terminate immediately (rather than crashing the next time crypto is used). Also corrected build problems on Mac OS X.
  Own Id: OTP-5279

2.1.5 Crypto 1.2.1

Fixed Bugs and Malfunctions

- It was not possible in R9 to relink the crypto driver. The object file was missing as well as an example makefile. The crypto driver object file is now released with the application (installed in priv/obj). An example makefile has also been added to the priv/obj directory. The makefile serves as an example of how to relink the driver on Unix (crypto_drv.so) or Windows (crypto_drv.dll).
  Own Id: OTP-4828 Aux Id: seq8193

2.1.6 Crypto 1.2

Improvements and New Features

- Previous versions of Crypto where delivered with statically linked binaries based on SSLeay. That is not longer the case. The current version of Crypto requires dynamically linked OpenSSL libraries that the user has to install. The library needed is `libcrypto.so` (Unix) or `libeay32.[lib|dll]` (Win32). For further details see the crypto(6) application manual page.
- This version of Crypto uses the new DES interface of OpenSSL 0.9.7, which is not backward compatible with earlier versions of OpenSSL.
- The functions `des_ede3_cbc_encrypt/5` and `des_ede3_cbc_decrypt/5` have been renamed to `des3_cbc_encrypt/5` and `des3_cbc_decrypt/5`, respectively. The old functions have been retained (they are deprecated and not listed in the crypto(3) manual page).

Reported Fixed Bugs and Malfunctions

- The start of crypto failed on Windows, due to erroneous addition of a DES3 algorithm.
  Own Id: OTP-4684
  Aux Id: seq7864

2.1.7 Crypto 1.1.3

Reported Fixed Bugs and Malfunctions

- To obtain backward compatibility with the old SSLeay package, and with earlier versions of OpenSSL, the macro OPENSSL_DES_LIBDES_COMPATIBILITY has been added to `crypto_drv.c`. This is of importance only for the open source version of Crypto.

2.1.8 Crypto 1.1.2

Reported Fixed Bugs and Malfunctions

- In the manual page `crypto(3)` the function names `md5_finish` and `sha_finish` have been changed to `md5_final` and `sha_final` to correctly document the implementation. Own Id: OTP-3409

2.1.9 Crypto 1.1.1

Code replacement in runtime is supported. Upgrade can be done from from version 1.1 and downgrade to version 1.1.

Improvements and New Features

- The driver part of the Crypto application has been updated to use the erl_driver header file. Version 1.1.1 requires emulator version 4.9.1 or later.

2.1.10 Crypto 1.1

Reported Fixed Bugs and Malfunctions

- On Windows the crypto_drv was incorrectly linked to static run-time libraries instead of dynamic ones. Own Id: OTP-3240

2.1.11 Crypto 1.0

New application.

Crypto Reference Manual

Short Summaries

- Application crypto [page 11] – The Crypto Application
- Erlang Module crypto [page 13] – Crypto Functions

crypto

No functions are exported.

crypto

The following functions are exported:

- `start() -> ok`
  [page 13] Start the crypto server.
- `stop() -> ok`
  [page 13] Stop the crypto server.
- `info() -> [atom()]`
  [page 13] Provide a list of available crypto functions.
- `md5(Data) -> Digest`
  [page 13] Compute an `MD5` message digest from `Data`
- `md5_init() -> Context`
  [page 14] Creates an MD5 context
- `md5_update(Context, Data) -> NewContext`
  [page 14] Update an MD5 `Context` with `Data`, and return a `NewContext`
- `md5_final(Context) -> Digest`
  [page 14] Finish the update of an MD5 `Context` and return the computed `MD5` message digest
- `sha(Data) -> Digest`
  [page 14] Compute an `SHA` message digest from `Data`
- `sha_init() -> Context`
  [page 14] Create an SHA context
- `sha_update(Context, Data) -> NewContext`
  [page 14] Update an SHA context
- `sha_final(Context) -> Digest`
  [page 14] Finish the update of an SHA context

- `md5_mac(Key, Data) -> Mac`
  [page 15] Compute an MD5 MAC message authentification code
- `md5_mac_96(Key, Data) -> Mac`
  [page 15] Compute an MD5 MAC message authentification code
- `sha_mac(Key, Data) -> Mac`
  [page 15] Compute an MD5 MAC message authentification code
- `sha_mac_96(Key, Data) -> Mac`
  [page 15] Compute an MD5 MAC message authentification code
- `des_cbc_encrypt(Key, IVec, Text) -> Cipher`
  [page 15] Encrypt Text according to DES in CBC mode
- `des_cbc_decrypt(Key, IVec, Cipher) -> Text`
  [page 15] Decrypt Cipher according to DES in CBC mode
- `des3_cbc_encrypt(Key1, Key2, Key3, IVec, Text) -> Cipher`
  [page 16] Encrypt Text according to DES3 in CBC mode
- `des3_cbc_decrypt(Key1, Key2, Key3, IVec, Cipher) -> Text`
  [page 16] Decrypt Cipher according to DES in CBC mode
- `aes_cfb_128_encrypt(Key, IVec, Text) -> Cipher`
  [page 16] Encrypt Text according to AES in Cipher Feedback mode or Cipher Block Chaining mode
- `aes_cbc_128_encrypt(Key, IVec, Text) -> Cipher`
  [page 16] Encrypt Text according to AES in Cipher Feedback mode or Cipher Block Chaining mode
- `aes_cfb_128_decrypt(Key, IVec, Cipher) -> Text`
  [page 16] Decrypt Cipher according to AES in Cipher Feedback mode or Cipher Block Chaining mode
- `aes_cbc_128_decrypt(Key, IVec, Cipher) -> Text`
  [page 16] Decrypt Cipher according to AES in Cipher Feedback mode or Cipher Block Chaining mode
- `erlint(Mpint) ->`
  [page 16] Convert between binary multi-precision integer and erlang big integer
- `mpint(N) -> Mpint`
  [page 16] Convert between binary multi-precision integer and erlang big integer
- `rand_bytes(N) -> binary()`
  [page 17] Generate a binary of random bytes
- `rand_uniform(Lo, Hi) -> N`
  [page 17] Generate a random number
- `mod_exp(N, P, M) -> Result`
  [page 17] Perform N ^ P mod M
- `rsa_verify(Digest, Signature, Key) -> Verified`
  [page 17] Verify the digest and signature using rsa with given public key.
- `dss_verify(Digest, Signature, Key) -> Verified`
  [page 17] Verify the digest and signature using rsa with given public key.

crypto

Application

The purpose of the Crypto application is to provide message digest and DES encryption for SMNPv3. It provides computation of message digests MD5 and SHA, and CBC-DES encryption and decryption.

Configuration

The following environment configuration parameters are defined for the Crypto application. Refer to application(3) for more information about configuration parameters.

`debug = true | false` <optional> Causes debug information to be written to standard error or standard output. Default is `false`.

OpenSSL libraries

The current implementation of the Erlang Crypto application is based on the *OpenSSL* package version 0.9.7 or higher. There are source and binary releases on the web.

Source releases of OpenSSL can be downloaded from the OpenSSL[1] project home page, or mirror sites listed there.

The same URL also contains links to some compiled binaries and libraries of OpenSSL (see the `Related/Binaries` menu) of which the Shining Light Productions Win32 and OpenSSL[2] pages are of interest for the Win32 user.

For some Unix flavours there are binary packages available on the net.

If you cannot find a suitable binary OpenSSL package, you have to fetch an OpenSSL source release and compile it.

You then have to compile and install the library `libcrypto.so` (Unix), or the library `libeay32.dll` (Win32).

For Unix The `crypto_drv` dynamic driver is delivered linked to OpenSSL libraries in `/usr/local/lib`, but the default dynamic linking will also accept libraries in `/lib` and `/usr/lib`.

If that is not applicable to the particular Unix operating system used, the example `Makefile` in the Crypto `priv/obj` directory, should be used as a basis for relinking the final version of the port program.

For `Win32` it is only required that the library can be found from the `PATH` environment variable, or that they reside in the appropriate `SYSTEM32` directory; hence no particular relinking is need. Hence no example `Makefile` for Win32 is provided.

[1] URL: `http://www.openssl.org`
[2] URL: `http://www.shininglightpro.com/search.php?searchname=Win32+OpenSSL`

SEE ALSO

application(3)

crypto

Erlang Module

This module provides a set of cryptographic functions.

References:

- md5: The MD5 Message Digest Algorithm (RFC 1321)
- sha: Secure Hash Standard (FIPS 180-2)
- hmac: Keyed-Hashing for Message Authentication (RFC 2104)
- des: Data Encryption Standard (FIPS 46-3)
- aes: Advanced Encryption Standard (AES) (FIPS 197)
- ecb, cbc, cfb, ofb: Recommendation for Block Cipher Modes of Operation (NIST SP 800-38A).
- rsa: Recommendation for Block Cipher Modes of Operation (NIST 800-38A)
- dss: Digital Signature Standard (FIPS 186-2)

The above publications can be found at NIST publications[3], at IETF[4].

Types

```
byte() = 0 ... 255
ioelem() = byte() | binary() | iolist()
iolist() = [ioelem()]
```

Exports

`start() -> ok`

Starts the crypto server.

`stop() -> ok`

Stops the crypto server.

`info() -> [atom()]`

Provides the available crypto functions in terms of a list of atoms.

`md5(Data) -> Digest`

Types:

[3] URL: `http://csrc.nist.gov/publications`
[4] URL: `www.ietf.org`

- Data = iolist() | binary()
- Digest = binary()

Computes an MD5 message digest from Data, where the length of the digest is 128 bits (16 bytes).

`md5_init() -> Context`

Types:

- Context = binary()

Creates an MD5 context, to be used in subsequent calls to md5_update/2.

`md5_update(Context, Data) -> NewContext`

Types:

- Data = iolist() | binary()
- Context = NewContext = binary()

Updates an MD5 Context with Data, and returns a NewContext.

`md5_final(Context) -> Digest`

Types:

- Context = Digest = binary()

Finishes the update of an MD5 Context and returns the computed MD5 message digest.

`sha(Data) -> Digest`

Types:

- Data = iolist() | binary()
- Digest = binary()

Computes an SHA message digest from Data, where the length of the digest is 160 bits (20 bytes).

`sha_init() -> Context`

Types:

- Context = binary()

Creates an SHA context, to be used in subsequent calls to sha_update/2.

`sha_update(Context, Data) -> NewContext`

Types:

- Data = iolist() | binary()
- Context = NewContext = binary()

Updates an SHA Context with Data, and returns a NewContext.

`sha_final(Context) -> Digest`

Types:

- Context = Digest = binary()

Finishes the update of an SHA Context and returns the computed SHA message digest.

`md5_mac(Key, Data) -> Mac`

Types:

- Key = Data = iolist() | binary()
- Mac = binary()

Computes an MD5 MAC message authentification code from Key and Data, where the the length of the Mac is 128 bits (16 bytes).

`md5_mac_96(Key, Data) -> Mac`

Types:

- Key = Data = iolist() | binary()
- Mac = binary()

Computes an MD5 MAC message authentification code from Key and Data, where the length of the Mac is 96 bits (12 bytes).

`sha_mac(Key, Data) -> Mac`

Types:

- Key = Data = iolist() | binary()
- Mac = binary()

Computes an SHA MAC message authentification code from Key and Data, where the length of the Mac is 160 bits (20 bytes).

`sha_mac_96(Key, Data) -> Mac`

Types:

- Key = Data = iolist() | binary()
- Mac = binary()

Computes an SHA MAC message authentification code from Key and Data, where the length of the Mac is 96 bits (12 bytes).

`des_cbc_encrypt(Key, IVec, Text) -> Cipher`

Types:

- Key = Text = iolist() | binary()
- IVec = Cipher = binary()

Encrypts Text according to DES in CBC mode. Text must be a multiple of 64 bits (8 bytes). Key is the DES key, and IVec is an arbitrary initializing vector. The lengths of Key and IVec must be 64 bits (8 bytes).

`des_cbc_decrypt(Key, IVec, Cipher) -> Text`

Types:

- Key = Cipher = iolist() | binary()
- IVec = Text = binary()

Decrypts Cipher according to DES in CBC mode. Key is the DES key, and IVec is an arbitrary initializing vector. Key and IVec must have the same values as those used when encrypting. Cipher must be a multiple of 64 bits (8 bytes). The lengths of Key and IVec must be 64 bits (8 bytes).

```
des3_cbc_encrypt(Key1, Key2, Key3, IVec, Text) -> Cipher
```

Types:

- Key1 =Key2 = Key3 Text = iolist() | binary()
- IVec = Cipher = binary()

Encrypts Text according to DES3 in CBC mode. Text must be a multiple of 64 bits (8 bytes). Key1, Key2, Key3, are the DES keys, and IVec is an arbitrary initializing vector. The lengths of each of Key1, Key2, Key3 and IVec must be 64 bits (8 bytes).

```
des3_cbc_decrypt(Key1, Key2, Key3, IVec, Cipher) -> Text
```

Types:

- Key1 = Key2 = Key3 = Cipher = iolist() | binary()
- IVec = Text = binary()

Decrypts Cipher according to DES3 in CBC mode. Key1, Key2, Key3 are the DES key, and IVec is an arbitrary initializing vector. Key1, Key2, Key3 and IVec must and IVec must have the same values as those used when encrypting. Cipher must be a multiple of 64 bits (8 bytes). The lengths of Key1, Key2, Key3, and IVec must be 64 bits (8 bytes).

```
aes_cfb_128_encrypt(Key, IVec, Text) -> Cipher
aes_cbc_128_encrypt(Key, IVec, Text) -> Cipher
```

Types:

- Key = Text = iolist() | binary()
- IVec = Cipher = binary()

Encrypts Text according to AES in Cipher Feedback mode (CFB) or Cipher Block Chaining mode (CBC). Text must be a multiple of 128 bits (16 bytes). Key is the AES key, and IVec is an arbitrary initializing vector. The lengths of Key and IVec must be 128 bits (16 bytes).

```
aes_cfb_128_decrypt(Key, IVec, Cipher) -> Text
aes_cbc_128_decrypt(Key, IVec, Cipher) -> Text
```

Types:

- Key = Cipher = iolist() | binary()
- IVec = Text = binary()

Decrypts Cipher according to Cipher Feedback Mode (CFB) or Cipher Block Chaining mode (CBC). Key is the AES key, and IVec is an arbitrary initializing vector. Key and IVec must have the same values as those used when encrypting. Cipher must be a multiple of 128 bits (16 bytes). The lengths of Key and IVec must be 128 bits (16 bytes).

```
erlint(Mpint) ->
mpint(N) -> Mpint
```

Types:

- Mpint = binary()
- N = integer()

Convert a binary multi-precision integer `Mpint` to and from an erlang big integer. A multi-precision integer is a binary with the following form: `<<ByteLen:32/integer, Bytes:ByteLen/binary>>` where both `ByteLen` and `Bytes` are big-endian. Mpints are used in some of the functions in `crypto` and are not translated in the API for performance reasons.

`rand_bytes(N) -> binary()`

Types:

- N = integer()

Generates N bytes randomly uniform 0..255, and returns the result in a binary. Uses the `crypto` library pseudo-random number generator.

`rand_uniform(Lo, Hi) -> N`

Types:

- Lo, Hi, N = Mpint | integer()
- Mpint = binary()

Generate a random number `N, Lo =< N < Hi`. Uses the `crypto` library pseudo-random number generator. The arguments (and result) can be either erlang integers or binary multi-precision integers.

`mod_exp(N, P, M) -> Result`

Types:

- N, P, M, Result = Mpint
- Mpint = binary()

This function performs the exponentiation `N ^ P mod M`, using the `crypto` library.

`rsa_verify(Digest, Signature, Key) -> Verified`

Types:

- Verified = boolean()
- Digest, Signature = MPint
- Key = [E, N]
- E, N = MPint
- MPint = binary()

Verifies the digest and signature using the public key `Key`, using the `crypto` library function for RSA signature verification.

`dss_verify(Digest, Signature, Key) -> Verified`

Types:

- Verified = boolean()
- Digest, Signature = MPint
- Key = [P, Q, G, Y]

- P, Q, G, Y = MPint
- MPint = binary()

Verifies the digest and signature using the public key `Key`, using the `crypto` library function for DSS signature verification.

DES in CBC mode

The Data Encryption Standard (DES) defines an algoritm for encrypting and decrypting an 8 byte quantity using an 8 byte key (actually only 56 bits of the key is used).

When it comes to encrypting and decrypting blocks that are multiples of 8 bytes various modes are defined (NIST SP 800-38A). One of those modes is the Cipher Block Chaining (CBC) mode, where the encryption of an 8 byte segment depend not only of the contents of the segment itself, but also on the result of encrypting the previous segment: the encryption of the previous segment becomes the initializing vector of the encryption of the current segment.

Thus the encryption of every segment depends on the encryption key (which is secret) and the encryption of the previous segment, except the first segment which has to be provided with a first initializing vector. That vector could be chosen at random, or be counter of some kind. It does not have to be secret.

The following example is drawn from the old FIPS 81 standard (replaced by NIST SP 800-38A), where both the plain text and the resulting cipher text is settled. We use the Erlang bitsyntax to define binary literals. The following Erlang code fragment returns `true'.

```
Key = <<16#01,16#23,16#45,16#67,16#89,16#ab,16#cd,16#ef>>,
IVec = <<16#12,16#34,16#56,16#78,16#90,16#ab,16#cd,16#ef>>,
P = "Now is the time for all ",
C = crypto:des_cbc_encrypt(K, I, P),
C == <<16#e5,16#c7,16#cd,16#de,16#87,16#2b,16#f2,16#7c,
       16#43,16#e9,16#34,16#00,16#8c,16#38,16#9c,16#0f,
       16#68,16#37,16#88,16#49,16#9a,16#7c,16#05,16#f6>>,
<<"Now is the time for all ">> ==
                  crypto:des_cbc_decrypt(Key,IVec,C).
```

The following is true for the DES CBC mode. For all decompositions `P1 ++ P2 = P` of a plain text message `P` (where the length of all quantities are multiples of 8 bytes), the encryption `C` of `P` is equal to `C1 ++ C2`, where `C1` is obtained by encrypting `P1` with `Key` and the initializing vector `IVec`, and where `C2` is obtained by encrypting `P2` with `Key` and the initializing vector `l(C1)`, where `l(B)` denotes the last 8 bytes of the binary `B`.

Similarly, for all decompositions `C1 ++ C2 = C` of a cipher text message `C` (where the length of all quantities are multiples of 8 bytes), the decryption `P` of `C` is equal to `P1 ++ P2`, where `P1` is obtained by decrypting `C1` with `Key` and the initializing vector `IVec`, and where `P2` is obtained by decrypting `C2` with `Key` and the initializing vector `l(C1)`, where `l(.)` is as above.

For DES3 (which uses three 64 bit keys) the situation is the same.

Index of Modules and Functions

Modules are typed in *this way*.
Functions are typed in `this way`.

`aes_cbc_128_decrypt/3`
crypto , 16

`aes_cbc_128_encrypt/3`
crypto , 16

`aes_cfb_128_decrypt/3`
crypto , 16

`aes_cfb_128_encrypt/3`
crypto , 16

crypto
`aes_cbc_128_decrypt/3`, 16
`aes_cbc_128_encrypt/3`, 16
`aes_cfb_128_decrypt/3`, 16
`aes_cfb_128_encrypt/3`, 16
`des3_cbc_decrypt/5`, 16
`des3_cbc_encrypt/5`, 16
`des_cbc_decrypt/3`, 15
`des_cbc_encrypt/3`, 15
`dss_verify/3`, 17
`erlint/1`, 16
`info/0`, 13
`md5/1`, 13
`md5_final/1`, 14
`md5_init/0`, 14
`md5_mac/2`, 15
`md5_mac_96/2`, 15
`md5_update/2`, 14
`mod_exp/3`, 17
`mpint/1`, 16
`rand_bytes/1`, 17
`rand_uniform/2`, 17
`rsa_verify/3`, 17
`sha/1`, 14
`sha_final/1`, 14
`sha_init/0`, 14
`sha_mac/2`, 15
`sha_mac_96/2`, 15
`sha_update/2`, 14
`start/0`, 13
`stop/0`, 13

`des3_cbc_decrypt/5`
crypto , 16

`des3_cbc_encrypt/5`
crypto , 16

`des_cbc_decrypt/3`
crypto , 15

`des_cbc_encrypt/3`
crypto , 15

`dss_verify/3`
crypto , 17

`erlint/1`
crypto , 16

`info/0`
crypto , 13

`md5/1`
crypto , 13

`md5_final/1`
crypto , 14

`md5_init/0`
crypto , 14

`md5_mac/2`
crypto , 15

`md5_mac_96/2`
crypto , 15

`md5_update/2`
crypto , 14

`mod_exp/3`
crypto , 17

`mpint/1`
crypto , 16

rand_bytes/1
crypto , 17

rand_uniform/2
crypto , 17

rsa_verify/3
crypto , 17

sha/1
crypto , 14

sha_final/1
crypto , 14

sha_init/0
crypto , 14

sha_mac/2
crypto , 15

sha_mac_96/2
crypto , 15

sha_update/2
crypto , 14

start/0
crypto , 13

stop/0
crypto , 13