Programming with GNU Crypto

Version 2.0.0, 20 October 2003

Casey Marshall
Raif S. Naffah

This manual is for the GNU Crypto library, version 2.0.0.

In Hoc Signo Tectis

Table of Contents

1 Introduction

GNU Crypto is a free, high-quality, versatile, and provably correct implementation of a wide array of cryptographic primitives and tools written in the Java programming language. It provides an application programmer's interface (API) to a number of cryptographic algorithms, a variety of end-user tools, and a full Java cryptography architecture (JCA) provider.

The algorithms implemented by GNU Crypto include symmetric key ciphers for protecting data, message digests and message authentication codes for proving the integrity of data, digital signature schemes for proving the authenticity of data, and algorithms for generating unguessable pseudo-random numbers. The API is deliberately designed to be low-level, with access to the bare innards of the cryptographic algorithms involved, so more complex libraries and programs can be built.

GNU Crypto does not implement any algorithms that are encumbered by patents, and does not rely on any non-free code or documentation. GNU Crypto is designed to run in any Java environment that is compatible with Sun's Java runtime version 1.2 or later. This includes GNU Classpath, a free software implementation of the Java class libraries, and free virtual machines such as Kissme, Japhar, Kaffe, and the Jikes RVM.

This manual covers the basics for using the GNU Crypto API in new Java programs. It describes the public API for all the implemented algorithms, describes which algorithms are implemented, and provides simple examples of each. The reader is assumed to have some knowledge about cryptography and the Java programming language.

This is not a reference about cryptography, the Java programming language, or the Java cryptography architecture API. For an introduction to cryptography, we recommend the following books:

- Bruce Schneier, *Applied Cryptography: Protocols, Algorithms, and Source Code in C, Second Edition* [Sch95].
- Alfred J. Menezes, Paul C. Van Oorschot, and Scott A. Vanstone, *Handbook of Applied Cryptography* [MOV96].

The JCA API documentation is available on-line from either Sun Microsystems (`http://java.sun.com/products/jce/doc/apidoc/`) or the Legion of the Bouncy Castle (`http://www.bouncycastle.org/docs/index.html`). There are copious references about the Java programming language available (although, as far as the author is aware, no free manuals are available at the time of writing).

GNU Crypto is always available on the web from `http://www.gnu.org/software/gnu-crypto/`, via anonymous FTP from `ftp://ftp.gnupg.org/gcrypt/gnu-crypto/`. The mailing list for bugs, help, and discussion is `gnu-crypto-discuss@gnu.org`, and additional information about the project is available on Savannah at `http://savannah.gnu.org/projects/gnu-crypto/`.

"Java" is a registered trademark of Sun Microsystems.

Part 1: The GNU Crypto API

2 Ciphers

This chapter describes the symmetric ciphers implemented by GNU Crypto, and how to create and use them. The package name for all GNU Crypto ciphers is `gnu.crypto.cipher`. The ciphers implemented by GNU Crypto are:

- The **Advanced Encryption Standard**, or the **AES**. The AES is a symmetric block cipher with a 128 bit block size and a key size of 128, 192, or 256 bits. The AES was adopted as US FIPS PUB 197 [NIST01] by the National Institute of Standards and Technology (NIST) in November 2001 after a five-year process of standarization and public comment. The AES was written by Joan Daemen and Vincent Rijmen for the AES process, and is derived from the Rijndael cipher.
- **Anubis**. The Anubis cipher is a symmetric block cipher with a 128 bit block size and a key size from 128 to 320 bits, with increments of 32 bits. Anubis was designed by Paulo Barreto and Vincent Rijmen, and has been submitted as a candidate cipher to the New European Schemes for Signatures, Integrity, and Encryption (NESSIE) process.
- **Blowfish**. The Blowfish symmetric block cipher was designed by Bruce Schneier. It has a 64 bit block size and a key size of up to 448 bits. Blowfish encryption and decryption are very fast in software, especially on 32 bit microprocessor architectures.
- **DES**. DES is the Data encryption standard, a 64-bit cipher with a 56-bit key. DES was developed by IBM in the 1970's for a standardization process begun by the National Bureau of Standards (now NIST). DES should not be used in new applications in favor of the new standard, AES, except for compatibility.
- **Identity cipher**. The identity, or null cipher, is not a true cipher as it does not transform the data input, but rather copies it directly to the output.
- **Khazad**. The Khazad cipher is a symmetric block cipher with a 64 bit block size and a 128 bit key size. Khazad was designed by Paulo Barreto and Vincent Rijmen, and has been submitted as a candidate cipher to the New European Schemes for Signatures, Integrity, and Encryption (NESSIE) process.
- **Rijndael**. Rijndael is a symmetric block cipher written by Joan Daemen and Vincent Rijmen as a candidate to the Advanced Encryption Standard process, and was adopted as the AES. Rijndael additionally has a 192 and 256 bit block size.
- **Serpent**. The Serpent cipher was designed by Ross Anderson, Eli Biham, and Lars Knudsen as a proposed cipher for the Advanced Encryption Standard. Serpent has a 128 bit block size, and a key size of 128, 192, or 256 bits.
- **Square**. The Square cipher was designed by Joan Daemen and Vincent Rijmen and was cryptanalyzed by Lars Knudsen. It has a 128 bit block size and a 128 bit key size.
- **Triple-DES**, or DESede, is a combined cipher based on the Data Encryption Standard. It is the iteration of three seperate instances of DES with three independent keys, and therefore has a 64 bit block size and a key size of 168 bits.
- **Twofish**. The Twofish cipher was designed by Bruce Schneier, John Kelsey, Doug Whiting, David Wagner, Chris Hall, and Niels Ferguson as a proposed cipher for the Advanced Encryption Standard. Twofish has a 128 bit block size, and a key size of 128, 192, or 256 bits.

2.1 The IBlockCipher Interface

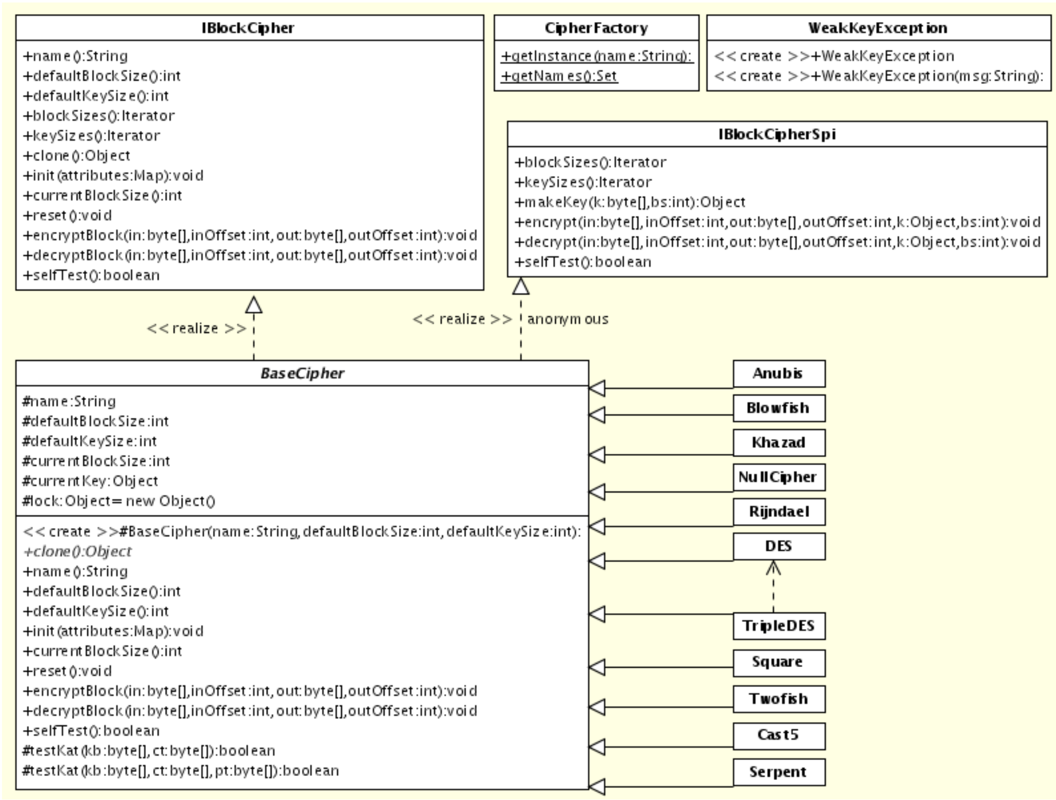

Figure 1: Ciphers class diagram

All ciphers in GNU Crypto implement the `IBlockCipher` interface, and support all the methods listed in this section.

`java.lang.String` **CIPHER_BLOCK_SIZE** [Variable]

A property name in the attributes map that is passed to the `init` method, representing the cipher's desired block size. The mapped value should be a `java.lang.Integer` of the cipher's block size, in bytes. If this attribute is omitted, the cipher's default block size is used.

`java.lang.String` **KEY_MATERIAL** [Variable]

A property name in the attributes map that is passed to the `init` method, representing the bytes that are to compose the cipher's key. The mapped value must be a byte array, and its length must be one of the cipher's supported key sizes.

`void` **init** (`java.util.Map` *attributes*) `throws java.security.InvalidKeyException, java.lang.IllegalStateException` [Function]

Initializes the cipher for transforming data. The *attributes* parameter must be a `java.util.Map` that has, at least, a mapping between the `KEY_MATERIAL` property name to a byte array containing the key. Ciphers *may* define other property names. If the supplied byte array is not an acceptable key, this method throws a `java.security.InvalidKeyException`. If this instance has already been initialized, this method throws a `java.lang.IllegalStateException`.

`java.lang.String` **name** ( ) [Function]

Returns the cipher's canonical name.

`int` **defaultBlockSize** ( ) [Function]

Returns the default block size, in bytes.

`int` **defaultKeySize** ( ) [Function]

Returns the default key size, in bytes.

`java.util.Iterator` **blockSizes** ( ) [Function]

Returns a `java.util.Iterator` of the cipher's supported block sizes. Each element of the iterator is a `java.lang.Integer`.

`java.util.Iterator` **keySizes** ( ) [Function]

Returns a `java.util.Iterator` of the cipher's supported key sizes. Each element of the iterator is a `java.lang.Integer`.

`java.lang.Object` **clone** ( ) [Function]

Returns a clone of this cipher. The cloned instance must be initialized, as this method will not clone the cipher's internal key.

`int` **currentBlockSize** ( ) `throws java.lang.IllegalStateException` [Function]

Returns the cipher's current block size, in bytes, or will throw a `java.lang.IllegalStateException` if this instance has not been initialized.

`void` **reset** ( ) [Function]

Resets this instance, which may then be re-initialized.

`void` **encryptBlock** (`byte[]` *`plaintext`*, `int` *`inOffset`*, `byte[]` *`ciphertext`*, `int` *`outOffset`*) `throws java.lang.IllegalStateException` [Function]

Encrypts a block of bytes from *plaintext* starting at *inOffset*, storing the encrypted bytes in *ciphertext*, starting at *outOffset*. It is up to the programmer to ensure that there is at least one full block in *plaintext* from *inOffset* and space for one full block in *ciphertext* from *outOffset*. A `java.lang.IllegalStateException` will be thrown if the cipher has not been initialized.

`void` **decryptBlock** (`byte[]` *`ciphertext`*, `int` *`inOffset`*, `byte[]` *`plaintext`*, `int` *`outOffset`*) `throws java.lang.IllegalStateException` [Function]

Decrypts a block of bytes from *ciphertext* starting at *inOffset*, storing the encrypted bytes in *plaintext*, starting at *outOffset*. It is up to the programmer to ensure that there is at least one full block in *ciphertext* from *inOffset* and space for one full block in *plaintext* from *outOffset*. A `java.lang.IllegalStateException` will be thrown if the cipher has not been initialized.

`boolean` **selfTest** ( ) [Function]

Performs a simple test of conformance, to ensure that there are no implementation or system errors. This method returns `true` if the test succeeds; `false` otherwise.

2.2 The CipherFactory Class

The ciphers in GNU Crypto can usually be initiallized directly through their constructors, but the preferred way is to use the `CipherFactory` class, with the following method:

`static IBlockCipher` **getInstance** (`java.lang.String` *`name`*) [Function]

Returns a new cipher instance for the cipher named *name*, or `null` if no such cipher exists. This method will throw a `java.lang.InternalError` if the new instance's self-test fails.

The class also defines this method:

`static java.util.Set` **getNames** ( ) [Function]

This method returns a `java.util.Set` of the names (each element of type `java.lang.String`) of all supported ciphers.

2.3 Example

The following example transforms the plaintext to the ciphertext, and the ciphertext back to the plaintext, using the AES in electronic codebook mode with no padding. Note also the classes for cipher modes and padding schemes for more complex constructions.

```
IBlockCipher cipher = CipherFactory.getInstance("AES");
Map attributes = new HashMap();
attributes.put(IBlockCipher.CIPHER_BLOCK_SIZE, new Integer(16));
attributes.put(IBlockCipher.KEY_MATERIAL, key_bytes);
cipher.init(attributes);
int bs = cipher.currentBlockSize();

for (int i = 0; i + bs < pt.length; i += bs)
   {
      cipher.encryptBlock(pt, i, ct, i);
   }

for (int i = 0; i + bs < cpt.length; i += bs)
   {
      cipher.decryptBlock(ct, i, cpt, i);
   }
```

3 Modes

Cipher modes operate on the next level up from the underlying block cipher. They transform the blocks going in and out of the cipher in ways to give them desirable properties in certain circumstances. The cipher modes implemented by GNU Crypto, which is contained in the `gnu.crypto.mode` package and are referenced herein by their three-letter abbreviations described below, are:

- **Cipher block chaining mode**. The "CBC" mode makes every block of the ciphertext depend upon all previous blocks by adding *feedback* to the transformation. This is done by XORing the plaintext with the previous ciphertext (or, with the first block, an initialization vector) before it is transformed. That is, encryption looks like: $C_i = E_k(P_i \oplus C_{i-1})$; and decryption is $P_i = C_{i-1} \oplus E_k^{-1}(C_i)$.
- **Counter mode**. Counter mode, referred to as "CTR" mode, is one of a class of sequenced cipher modes that turn the underlying cipher into a *keystream*. Counter mode relys on a simple counter register that is updated for every block processed. For plaintexts $P_1 \ldots P_n$, ciphertexts $C_1 \ldots C_n$, counter elements $T_1 \ldots T_n$, and an encryption function E_k, encryption is defined as $C_i = P_i \oplus E_k(T_i)$ and decryption as $P_i = C_i \oplus E_k(T_i)$.
- **Electronic codebook mode**. Or "ECB" mode, is the most obvious cipher mode: the cipher block is the direct output of the forward function, and the plain block is the direct output of the inverse function. That is, encryption is $C_i = E_k(P_i)$ and decryption is $P_i = E_k^{-1}(C_i)$.
- **Integer counter mode**. "ICM" mode has features in common with counter mode described above. The counter, T_i, is computed by $T_i = (T_0 + i) \bmod 256^b$, where b is the cipher's block size. T_0 is initialized to the integer representation of some initialization vector. The keystream bytes are then $E_k(T_i)$. Encryption and decryption are then $C_i = P_i \oplus E_k(T_i)$ and $P_i = C_i \oplus E_k(T_i)$, respectively.
- **Output feeback mode**. "OFB" mode creates a keystream by repeatedly iterating the underlying block cipher over an initialization vector. That is, the ith keystream block is $X_i = E(X_{i-1})$ for $1 < i \leq n$, and $X_1 = IV$. Like the other stream modes, the input block i is transformed by the exclusive-or of the block with X_i.

3.1 The IMode Interface

The `IMode` interface is similar to the `IBlockCipher` interface, except modes have a *state* associated with them, e.g. whether the instance is used for encryption or decryption. The `IMode` interface is usually the one that is used when encrypting or decrypting; `IBlockCipher` is used when the lowest level—the cipher function itself—needs to be accessed. `IMode` extends `IBlockCipher` interface, and thus all methods specified in that interface are implemented in modes, and have the same meaning. The properties passed to the `init` method of `IBlockCipher` may also be passed to the `init` mehtod of `IMode`, along with the following property names.

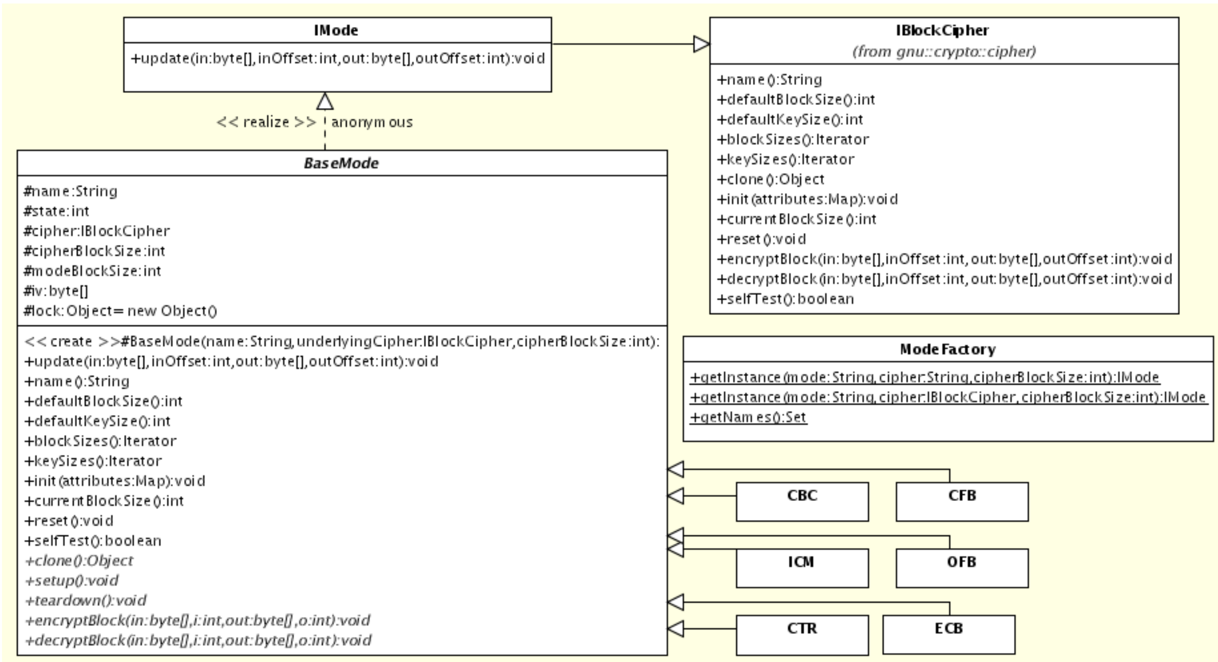

Figure 2: Modes class diagram

`java.lang.String` **STATE** [Variable]

The property name for the mode's state, as passed to the `init` method. Values for this property are an `java.lang.Integer` containing either the `ENCRYPTION` constant or the `DECRYPTION` constant.

`int` **ENCRYPTION** [Variable]

The value passed for the `STATE` property, wrapped in a `java.lang.Integer`, which indicates that the instance is to be used for encryption.

`int` **DECRYPTION** [Variable]

The value passed for the `STATE` property, wrapped in a `java.lang.Integer`, which indicates that the instance is to be used for decryption.

`java.lang.String` **MODE_BLOCK_SIZE** [Variable]

The property name for the block size of this mode. The value for this propery should be a `java.lang.Integer` of the block size. If omitted, the underlying cipher's block size is used.

`java.lang.String` **IV** [Variable]

The property name for the initialization vector to initialize this mode with, if required. The value should be a byte array equal in size to the `MODE_BLOCK_SIZE` property. If omitted a byte array consisting of zeros is used.

void **update** (byte[] *in*, int *inOffset*, byte[] *out*, int *outOffset*) [Function]
throws java.lang.IllegalStateException

Transforms the block in *in* starting at *inOffset* into the block in *out* starting at *outOffset*. Encryption or decryption is performed depending upon the value passed along with the `state` property given to the `init` method. A `java.lang.IllegalStateException` is thrown if this instance has not been initialized, and it is up to the programmer to ensure that there is one full block in *in* starting at *inOffset*, and enough space for one full block in *out* starting at *outOffset*. Since modes can have states, and may require that the be used in a particular sequence, using this method is preferred over the `encryptBlock` and `decryptBlock` methods of `IBlockCipher`.

3.2 The ModeFactory Class

The preferred way to get mode instances is through the `ModeFactory` class, from one of the following methods:

static IMode **getInstance** (java.lang.String *mode*, java.lang.String *cipher*, int *cipherBlockSize*) [Function]

Returns an instance of *cipher* wrapped in an instance of *mode*, initialized to a block size of *cipherBlockSize*, or returns `null` if no appropriate cipher or mode is available. The *mode* argument is one of the names described above, and *cipher* is one of the names described in the Ciphers chapter.

static IMode **getInstance** (java.lang.String *mode*, IBlockCipher *cipher*, int *cipherBlockSize*) [Function]

Returns an instance of *mode* using the already-initialized *cipher*, initializing the mode with a block size of *cipherBlockSize*, or returns `null` if no appropriate mode is available.

Additionally the following method is defined:

static java.util.Set **getNames** () [Function]

This method returns a `java.util.Set` of the names (each element of type `java.lang.String`) of all supported modes.

3.3 Example

The following example encrypts and decrypts a byte array with the AES in CFB mode. See the next chapter on padding for instances where the input is not a multiple of the cipher or mode's block size.

```
IMode mode = ModeFactory.getInstance("CFB", "AES", 16);
Map attributes = new HashMap();

// These attributes are defined in gnu.crypto.cipher.IBlockCipher.
attributes.put(IMode.KEY_MATERIAL, key_bytes);
attributes.put(IMode.CIPHER_BLOCK_SIZE, new Integer(16));

// These attributes are defined in IMode.
attributes.put(IMode.STATE, new Integer(IMode.ENCRYPTION));
attributes.put(IMode.IV, iv_bytes);
mode.init(attributes);
int bs = mode.currentBlockSize();

for (int i = 0; i + bs < pt.length; i += bs)
   {
      mode.update(pt, i, ct, i);
   }

mode.reset();
attributes.put(IMode.STATE, new Integer(IMode.DECRYPTION);
mode.init(attributes);

for (int i = 0; i + bs < ct.length; i += bs)
   {
      mode.update(ct, i, cpt, i);
   }
```

4 Padding

A padding scheme is merely a standard method of ensuring that the input to be encrypted is a multiple of the cipher's block size. The padding schemes of GNU Crypto are in package `gnu.crypto.pad` and include:

- **PKCS #7**. PKCS #7 (referred to as "PKCS7" in GNU Crypto) pads the input P with the quantity $w = b - (|P| \bmod b)$, where b is the cipher's block size, encoded as w bytes. That is, if the input is 5 bytes shorter than the required length, then the input is padded with the byte equal to 5 five times. This padding scheme supports block sizes of $2 \leq b \leq 256$ bytes.
- **Trailing bit complement**. The "TBC" pad appends the complement of the last bit in the input until the input is the desired length. That is, if the last bit is `1`, then the input is padded with `0`, and if the last bit is `0`, then the input is padded with `1`. This padding scheme supports block sizes up to 256 bytes.

4.1 The IPad Interface

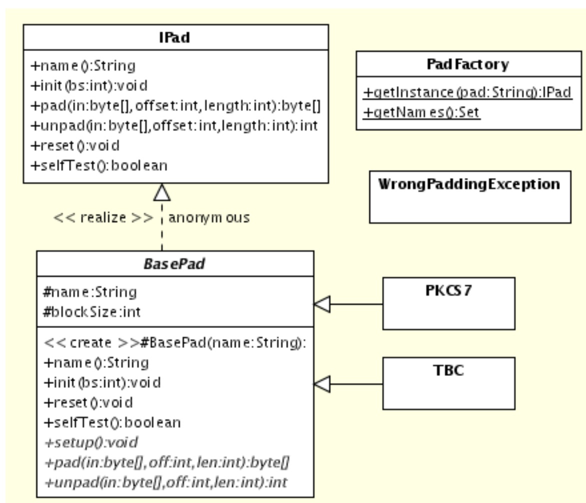

Figure 3: Padding class diagram

The `IPad` interface is used seperately from ciphers and modes. The methods defined by padding schemes are:

`void` **init** (`int` *bs*) `throws java.lang.IllegalStateException, java.lang.IllegalArgumentException` [Function]

Initializes this padding scheme for the specified block size. This method throws a `java.lang.IllegalStateException` if this instance has already been initialized but not reset, and will throw a `java.lang.IllegalArgumentException` if *bs* is not a supported block size.

`void` **reset** ( ) [Function]

Resets this instance, which may then be re-initialized later.

`byte[]` **pad** (`byte[]` *input*, `int` *offset*, `int` *length*) [Function]

Examines the bytes in *input* as the plaintext, starting at *offset* and considering *length* bytes, and returns the appropriate, possibly empty, byte array containing the padding.

`int` **unpad** (`byte[]` *input*, `int` *offset*, `int` *length*) `throws WrongPaddingException` [Function]

Examines the bytes in *input* as the plaintext, starting at *offset* and considering *length* bytes, and returns the number of bytes that should be trimmed off the end of *input* to unpad the plaintext. Throws a `WrongPaddingException` if the padding bytes to not correspond to the bytes expected by this padding scheme.

`java.lang.String` **name** ( ) [Function]

Returns the canonical name of this instance.

`boolean` **selfTest** ( ) [Function]

Performs a simple conformance test on the padding scheme, to avoid implementation or run time errors.

4.2 The PadFactory Class

Padding instances are created with the following method in the `PadFactory` class:

`static IPad` **getInstance** (`String` *pad*) [Function]

Gets an instance of the padding scheme with name *pad*, or `null` if no such padding scheme is available.

This class also defines this method:

`static java.util.Set` **getNames** () [Function]

Returns a set of strings with the names of all padding schemes implemented by GNU Crypto.

4.3 Example

The following example pads an input buffer, transforms the padded buffer with already-initialized `IMode` instances, then unpads the output buffer.

```
IPad padding = IPad.getInstance("PKCS7");
padding.init(blockSize);
byte[] pad = padding.pad(input, 0, input.length);
byte[] pt = new byte[input.length + pad.length];
byte[] ct = new byte[pt.length];
byte[] cpt = new byte[pt.length];
System.arraycopy(input, 0, pt, 0, input.length);
System.arraycopy(pad, 0, pt, input.length, pad.length);

for (int i = 0; i + blockSize < pt.length; i += blockSize)
   {
      enc.update(pt, i, ct, i);
   }

for (int i = 0; i + blockSize < ct.length; i += blockSize)
   {
      dec.update(ct, i, cpt, i);
   }

int unpad = padding.unpad(cpt, 0, cpt.length);
byte[] output = new byte[cpt.length - unpad];
System.arraycopy(cpt, 0, output, 0, output.length);
```

5 Cascades and Assemblies

This chapter describes two patterns implemneted by the GNU Crypto library that allow users to combine the basic cipher (and other) primitives into higher level components in order to offer more flexible functionalities. These two patterns are: Cascade and Assembly.

The **Cascade** is a means of assembling block cipher Modes of Operations into an ordered sequence of *stages*. A *stage* is a representation of a Mode (of Operations) wired in a designated *direction*: FORWARD or REVERSED. A Mode staged in the FORWARD direction would encrypt input blocks, producing ciphertext, while the same Mode, wired in the REVERSED direction would do the opposite; i.e. decrypt an input text producing a plaintext.

In the simplest case, all stages in a Cascade have k-bit keys, and the stage inputs and outputs are all n-bit quantities. The stage ciphers may differ (general cascade of ciphers), or all be identical (cascade of identical ciphers).

An **Assembly** is a construction of an ordered set of **Transformer** objects. Each **Transformer** is wired to operate in PRE_PROCESSING or POST_PROCESSING mode –the Transformer's **Operation**. In PRE_PROCESSING, the input is first processed by the Transformer before being passed to the rest of the chain, while in POST_PROCESSING state, the Transformer first passes the input to the rest of the chain and only processes the output of the returned data.

5.4 Cascades

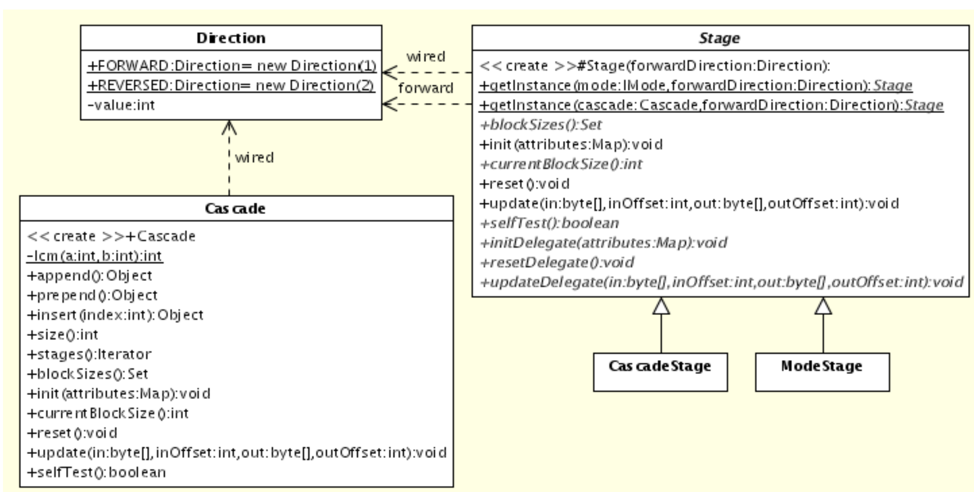

Figure 4: Cascade class diagram

5.5 Direction

An enumeration type for wiring **Stage** instances into **Cascade** chains, as well as for operating a **Cascade** in a given direction.

This class cannot be instantiated; but its (only) two possible values can be used for constructing **Stage** elements, and initializing **Cascade** instances:

- **FORWARD**: equivalent to `gnu.crypto.mode.IMode#ENCRYPTION`; and its inverse value
- **REVERSED**: equivalent to `gnu.crypto.mode.IMode#DECRYPTION`.

This class offers a *Factory* method to return the inverse of a designated **Direction** instance:

`Direction` **reverse** (`Direction` *d*) [Function]

5.6 Stage

This class represents a **Stage** in a **Cascade** cipher.

Each stage may be either an implementation of a Block Cipher Mode of Operation (an instance of `gnu.crypto.mode.IMode`) or another **Cascade** cipher (an instance of `Cascade`). Each **Stage** has also a *natural* operational direction when constructed for inclusion within a **Cascade**. This *natural* direction dictates how data flows from one *Stage* into another when stages are chained together in a **Cascade**. One can think of a **Stage** and its natural direction as the specification of how to wire the **Stage** into the chain.

The following diagrams may help understand the paradigm. The first shows two stages chained together, each wired in the same direction (`Direction#FORWARD`).

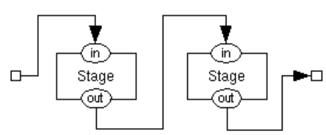

Figure 5: Stages wired in same direction

The second diagram shows two stages, one in a `Direction#FORWARD` direction, while the other is wired in a `Direction#REVERSED` direction.

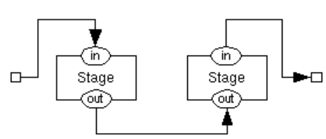

Figure 6: Stages wired in different directions

`gnu.crypto.assembly.Stage` **DIRECTION** [Variable]

> A property name in the attributes map that is passed to the `init` method, representing the stage's desired wiring direction. The mapped value should be a valid `gnu.crypto.assembly.Direction` value. If this attribute is omitted, `Direction.FORWARD` is used.

The following *Factory* methods, allow instantiation of concrete **Stage** class instances that adapt instances of either `gnu.crypto.mode.IMode` or (other) `Cascade` classes to operate as a **Stage** in a **Cascade**:

`Stage` **getInstance** (`IMode` *mode*, `Direction` *forwardDirection*) [Function]

Given a designated *mode* (an instance of `gnu.crypto.mode.IMode`, and a **Direction**, this method returns a **Stage** instance that adapts this designated *mode* to operate as a **Stage** in a **Cascade**.

`Stage` **getInstance** (`Cascade` *cascade*, `Direction` *forwardDirection*) [Function]

Given a designated *cascade* (an instance of `gnu.crypto.assembly.Cascade`, and a **Direction**, this method returns a **Stage** instance that adapts this designated *cascade* to operate as a **Stage** in another **Cascade**.

The following instance methods are also available:

`java.util.Set` **blockSizes** () [Function]

Returns the `Set` of supported block sizes for this **Stage**. Each element in the returned `Set` is an instance of `Integer`.

`void` **init** (`java.util.Map` *attributes*) `throws java.security.InvalidKeyException` [Function]

Initializes the stage for operation with specific characteristics. Those characteristics are defined in *attributes*: a set of name-value pairs that describes the desired future behavior of this instance. This method throws an `IllegalStateException` if the instance is already initialized. It throws an `java.security.InvalidKeyException` if the key data (used to initialize the underlying Mode or Cascade) is invalid.

`int` **currentBlockSize** () `throws IllegalStateException` [Function]

Returns the current block size for this stage. This method throws an `IllegalStateException` if the instance is not yet initialized.

`void` **reset** () [Function]

Resets the stage for re-initialization and use with other characteristics. This method always succeeds.

`void` **update** (`byte[]` *in*, `int` *inOffset*, `byte[]` *out*, `int` *outOffset*) [Function]

Processes exactly one block of *plaintext* (if wired in the `Direction#FORWARD` direction) or *ciphertext* (if wired in the `Direction#REVERSED` direction), from *in* starting at *inOffset*, and storing the resulting bytes in *out*, starting at *outOffset*. An `IllegalStateException` will be thrown if the stage has not yet been initialized.

`boolean` **selfTest** `()` [Function]

Conducts a simple *correctness* test that consists of basic symmetric encryption / decryption test(s) for all supported block and key sizes of underlying block cipher(s) wrapped by Mode leafs. The test also includes one (1) variable key Known Answer Test (KAT) for each block cipher. It returns `true` if the tests succeed, and `false` otherwise.

5.7 Cascade

A **Cascade** Cipher is the concatenation of two or more block ciphers each with independent keys. Plaintext is input to the first stage; the output stage `i` is input to stage `i + 1`; and the output of the last stage is the **Cascade**'s ciphertext output.

In the simplest case, all stages in a **Cascade** have *k*-bit keys, and the stage inputs and outputs are all n-bit quantities. The stage ciphers may differ (general cascade of ciphers), or all be identical (cascade of identical ciphers).

The term *block ciphers* used above refers to implementations of `gnu.crypto.mode.IMode`, including the `gnu.crypto.mode.ECB` mode which basically exposes a symmetric-key block cipher algorithm as a *Mode* of Operations.

`String` **DIRECTION** [Variable]

The name of a property in the attributes map that is passed to the `init` method, representing the cascade's desired wiring direction. The mapped value should be a valid `gnu.crypto.assembly.Direction` value. If this attribute is omitted, `gnu.crypto.assembly.Direction.FORWARD` is used.

`Object` **append** `(Stage` *`stage`*`)` `throws IllegalArgumentException` [Function]

Adds to the end of the current chain, a designated *stage*. Returns a unique identifier for this added stage, within this cascade. An `IllegalArgumentException` is thrown if *stage* is already in the chain, or it has incompatible characteristics with the current elements already in the chain. On the other hand, an `IllegalStateException` will be thrown if the cascade has already been initialized, or if the designated *stage* is null.

`Object` **prepend** `(Stage` *`stage`*`)` `throws IllegalArgumentException` [Function]

Adds to the beginning of the current chain, a designated *stage*. Returns a unique identifier for this added stage, within this cascade. An `IllegalArgumentException` is thrown if *stage* is already in the chain, or it has incompatible characteristics with the current elements already in the chain. On the other hand, an `IllegalStateException` will be thrown if the cascade has already been initialized, or if the designated *stage* is null.

`Object` **insert** (`int` *index*, `Stage` *stage*) `throws IllegalArgumentException, IndexOutOfBoundsException` [Function]

Inserts a designate *stage* **Stage** into the current **Cascade**, at the specified *index* (zero-based) position. Returns a unique identifier for this added stage, within this cascade. Throws an `IllegalArgumentException` if *stage* is already in the chain, or it has incompatible characteristics with the current elements already in the chain. Throws an `IllegalStateException` if the instance is already initialized. Finally, this method throws an `IndexOutOfBoundsException` if *index* is less than `0` or greater than the current size of this cascade.

`int` **size** () [Function]

Returns the current number of stages in this chain.

`java.util.Iterator` **stages** () [Function]

Returns an `java.util.Iterator` over the stages contained in this instance. Each element of this iterator is a concrete implementation of a `gnu.crypto.assembly.Stage`.

`java.util.Set` **blockSizes** () [Function]

Returns a `java.util.Set` of supported block sizes for this **Cascade** that are common to all of its chained stages. Each element in the returned set is an instance of `Integer`.

`void` **init** (`java.util.Map` *attributes*) `throws InvalidKeyException` [Function]

Initializes the chain for operation with specific characteristics, as specified by the contents of *attributes* –a set of name-value pairs that describes the desired future behavior of this instance. Throws an `IllegalStateException` if the chain, or any of its stages, is already initialized. Throws an `InvalidKeyException` if the initialization data provided with the stage is incorrect or causes an invalid key to be generated.

`int` **currentBlockSize** () [Function]

Returns the currently set block size for the chain. Throws an `IllegalStateException` if the instance is not yet initialized.

`void` **reset** () [Function]

Resets the chain for re-initialization and use with other characteristics. This method always succeeds.

`void` **update** (`byte[]` *in*, `int` *inOffset*, `byte[]` *out*, `int` *outOffset*) [Function]

Processes exactly one block of *plaintext* (if initialized in the `gnu.crypto.assembly.Direction#FORWARD` direction) or *ciphertext* (if initialised in the `gnu.crypto.assembly.Direction#REVERSED` direction), from *in*, starting at index position *inOffset*, returning the result in *out*, starting at index position *outOffset*. Throws an `IllegalStateException` if the instance is not yet initialized.

`boolean` **selfTest** () [Function]

Conducts a simple *correctness* test that consists of basic symmetric encryption / decryption test(s) for all supported block and key sizes of underlying block cipher(s) wrapped by Mode leafs. The test also includes one (1) variable key Known Answer Test (KAT) for each block cipher. Returns `true` if the implementation passes the tests. Returns `false` otherwise.

5.8 Example

The following example demonstrates how a DES-EDE block cipher can be constructed as a Cascade of three DES Stages.

```
HashMap map = new HashMap();
HashMap map1 = new HashMap();
HashMap map2 = new HashMap();
HashMap map3 = new HashMap();

Cascade new3DES = new Cascade();
Object des1 = new3DES.append(
    Stage.getInstance(
        ModeFactory.getInstance(Registry.ECB_MODE, new DES(), 8),
        Direction.FORWARD));
Object des2 = new3DES.append(
    Stage.getInstance(
        ModeFactory.getInstance(Registry.ECB_MODE, new DES(), 8),
        Direction.REVERSED));
Object des3 = new3DES.append(
    Stage.getInstance(
        ModeFactory.getInstance(Registry.ECB_MODE, new DES(), 8),
        Direction.FORWARD));

map.put(des1, map1);
map.put(des2, map2);
map.put(des3, map3);

map1.put(IBlockCipher.KEY_MATERIAL, key1material);
map2.put(IBlockCipher.KEY_MATERIAL, key2material);
map3.put(IBlockCipher.KEY_MATERIAL, key3material);

// encryption
map.put(Cascade.DIRECTION, Direction.FORWARD);
byte[] pt = ...; // some plaintext to encrypt
byte[] ct = new byte[pt.length]; // where ciphertext is returned

try
  {
    new3DES.init(map);
    new3DES.update(pt, 0, ct, 0);
```

```
    }
  catch (InvalidKeyException x)
    {
      x.printStackTrace(System.err);
    }
```

5.9 Assemblies

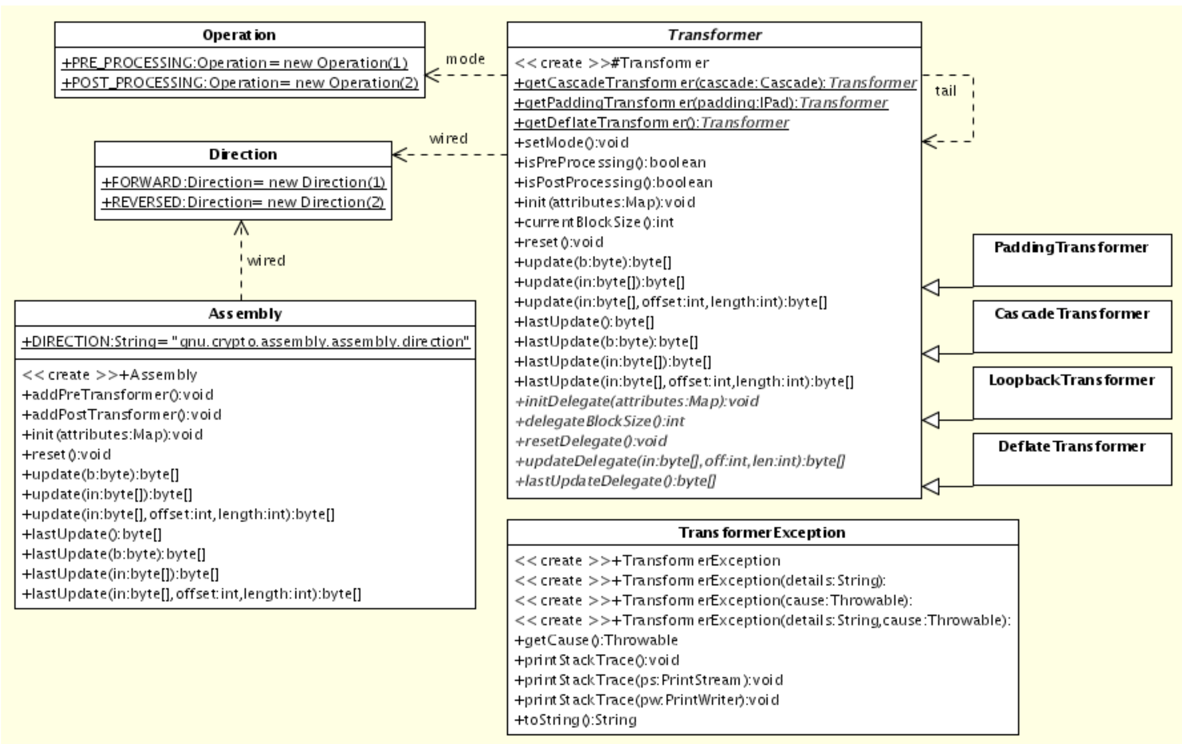

Figure 7: Assembly class diagram

5.10 Operation

An enumeration type for specifying the mode of operation of a **Transformer** instance, when wired into an **Assembly**.

This class cannot be instantiated; but its (only) two possible values can be used for constructing **Transformer** elements:

- **PRE_PROCESSING**: to mean that the input data is first processed by the current **Transformer** before being passed to the rest of the chain; and
- **POST_PROCESSING**: to mean that the input data is first passed to the rest of the chain, and the resulting bytes are then processed by the current **Transformer**.

5.11 Transformer

A **Transformer** is an abstract representation of a two-way *transformation* that can be chained together with other instances of this type. Examples of such transformations in this library are:

- **CascadeTransformer** that adapts an instance of a `gnu.crypto.assembly.Cascade`,
- **PaddingTransformer** that adapts an instance of `gnu.crypto.pad.IPad`, and finally
- **DeflateTransformer** that adapts a ZLib-based deflater/inflater algorithm implementation.

The special type `LoopbackTransformer` is also available and is implicitly (and silently) added to each instance of an **Assembly**.

A **Transformer** is characterized by the followings:

- It can be chained to other instances, to form an **Assembly**.
- When configured in an **Assembly**, it can be set to apply its internal transformation on the input data stream before (pre-processing) or after (post-processing) passing the input data to the next element in the chain. Note that the same type **Transformer** can be used in either pre-processing, or post-processing modes.
- A special transformer –`LoopbackTransformer`– is used to close the chain.
- A useful type of **Transformer** –one we're interested in– has internal buffers. The distinction between a casual push (update) operation, and the last one, allows to correctly flush any intermediate bytes that may exist in those buffers.

To allow wiring **Transformer** instances together, a *minimal output size* in bytes is necessary. The trivial case of a value of `1` for such attribute practically means that no output buffering, from the previous element, is needed –which is independent of buffering the input if the **Transformer** implementation itself is block-based.

This class exposes one class attribute and three Factory methods. They are:

`String` **DIRECTION** [Variable]

The name of a property in the attributes map that is passed to the `init` method, representing the transformation's desired wiring direction. The mapped value should be a valid `gnu.crypto.assembly.Direction` value. If this attribute is omitted, `gnu.crypto.assembly.Direction.FORWARD` is used.

`Transformer` **getCascadeTransformer** (`Cascade` *`cascade`*) [Function]

Returns the designated *cascade* instance wrapped in an Adapter for use as a **Transformer**.

`Transformer` **getPaddingTransformer** (`IPad` *`padding`*) [Function]

Returns the designated *padding* instance wrapped in an Adapter for use as a **Transformer**.

`Transformer` **getDeflateTransformer** () [Function]

Returns a **Transformer** that underlies an implementation of the ZLib algorithm, able to deflate (compress) and inflate (decompress) data.

Concrete class instances of this abstract class, also expose the following instance methods:

`void` **setMode** (`final Operation` *mode*) [Function]

Sets the operational mode of this **Transformer** to the designated *mode* value. Throws `IllegalStateException` if this instance has already been assigned an operational mode.

`boolean` **isPreProcessing** () [Function]

Returns `true` if this **Transformer** has been wired in pre-processing mode; returns `false` otherwise. Throws an `IllegalStateException` if this instance has not yet been assigned an operational mode.

`boolean` **isPostProcessing** () [Function]

Returns `true` if this **Transformer** has been wired in post-processing mode; returns `false` otherwise. Throws an `IllegalStateException` if this instance has not yet been assigned an operational mode.

`void` **init** (`java.util.Map` *attributes*) `throws TransformerException` [Function]

Initializes the **Transformer** for operation with specific characteristics, indicated by the designated *attributes*. The latter being a set of name-value pairs that describes the desired future behavior of this instance. Throws an `IllegalStateException` if the instance is already initialized.

`int` **currentBlockSize** () [Function]

Returns the block-size of this **Transformer**. A value of `1` indicates that this instance is block-agnostic.

`void` **reset** () [Function]

Resets the **Transformer** for re-initialization and use with other characteristics. This method always succeeds.

`byte[]` **update** (`byte` *b*) `throws TransformerException` [Function]

Convenience method that calls the method with same name and three arguments, using a byte array of length `1` whose contents are the designated byte *b*. Returns the result of transformation. Throws an `IllegalStateException` if the instance is not yet initialized. Throws an `TransformerException` if a transformation-related exception occurs during the operation.

`byte[]` **update** (`byte[]` *`in`*) `throws TransformerException` [Function]

Convenience method that calls the same method with three arguments. All bytes in *in*, starting from index position *0* are considered. Returns the result of transformation. Throws an `IllegalStateException` if the instance is not yet initialized. Throws a `TransformerException` if a transformation-related exception occurs during the operation.

`byte[]` **update** (`byte[]` *`in`*, `int` *`offset`*, `int` *`length`*) `throws TransformerException` [Function]

Returns the result of processing a designated *length* bytes from a given *in* byte array, starting at position *offset*. Throws an `IllegalStateException` if the instance is not yet initialized. Throws an `TransformerException` if a transformation-related exception occurs during the operation.

`byte[]` **lastUpdate** () `throws TransformerException` [Function]

Convenience method that calls the same method with three arguments. A zero-long byte array is used. Returns the result of transformation. Throws an `IllegalStateException` if the instance is not yet initialized. Throws an `TransformerException` if a transformation-related exception occurs during the operation.

`byte[]` **lastUpdate** (`byte` *`b`*) `throws TransformerException` [Function]

Convenience method that calls the method with same name and three arguments, using a byte array of length `1` whose contents are the designated byte *b*. Returns the result of transformation. Throws an `IllegalStateException` if the instance is not yet initialized. Throws an `TransformerException` if a transformation-related exception occurs during the operation.

`byte[]` **lastUpdate** (`byte[]` *`in`*) `throws TransformerException` [Function]

Convenience method that calls the same method with three arguments. All bytes in *in*, starting from index position `0` are considered. Returns the result of transformation. Throws an `IllegalStateException` if the instance is not yet initialized. Throws an `TransformerException` if a transformation-related exception occurs during the operation.

`byte[]` **lastUpdate** (`byte[]` *`in`*, `int` *`offset`*, `int` *`length`*) `throws TransformerException` [Function]

Returns the result of processing a designated *length* bytes from the given *in* byte array, starting at index position *offset* and signals, at the same time, that this is the last *push* operation on this **Transformer**. Throws an `IllegalStateException` if the instance is not yet initialized. Throws an `TransformerException` if a transformation-related exception occurs during the operation.

5.12 Assembly

An **Assembly** is a construction consisting of a chain of **Transformer** elements; each wired in pre- or post- operational mode. This chain is (always) terminated by one `LoopbackTransformer` element.

Once constructed, and correctly initialized, the bulk of the methods available on the **Assembly** are delegated to the *head* of the **Transformer** chain of the **Assembly**.

`String` **DIRECTION** [Variable]

The name of a property in the attributes map that is passed to the `init` method, representing the assembly's desired wiring direction. The mapped value should be a valid `gnu.crypto.assembly.Direction` value. If this attribute is omitted, `gnu.crypto.assembly.Direction.FORWARD` is used.

`boolean` **addPreTransformer** (`Transformer` *t*) [Function]

Adds the designated **Transformer** *t*, to the head of the current chain, and signals that it should operate in pre-processing mode; i.e. it should apply its internal transformation algorithm on the input data stream, **before** it passes that stream to the next element in the *chain*. Throws an `IllegalArgumentException` if the designated **Transformer** has a non-null tail; i.e. it is already an element of a chain.

`boolean` **addPostTransformer** (`Transformer` *t*) [Function]

Adds the designated **Transformer** *t*, to the head of the current chain, and signals that it should operate in post-processing mode; i.e. it should apply its internal transformation algorithm on the input data stream, **after** it passes that stream to the next element in the *chain*. Throws an `IllegalArgumentException` if the designated **Transformer** has a non-null tail; i.e. it is already an element of a chain.

`void` **init** (`java.util.Map` *attributes*) `throws TransformerException` [Function]

Initializes the **Assembly** for operation with specific characteristics, indicated by the designated *attributes*. The latter being a set of name-value pairs that describes the desired future behavior of this instance. Throws an `IllegalStateException` if the instance is already initialized.

`void` **reset** () [Function]

Resets the **Assembly** for re-initialization and use with other characteristics. This method always succeeds.

`byte[]` **update** (`byte` *b*) `throws TransformerException` [Function]

Convenience method that calls the method with same name and three arguments, using a byte array of length `1` whose contents are the designated byte *b*. Returns the result of transformation. Throws an `IllegalStateException` if the instance is not yet initialized. Throws an `TransformerException` if a transformation-related exception occurs during the operation.

`byte[]` **update** (`byte[]` *`in`*) `throws TransformerException` [Function]

Convenience method that calls the same method with three arguments. All bytes in *in*, starting from index position *0* are considered. Returns the result of transformation. Throws an `IllegalStateException` if the instance is not yet initialized. Throws a `TransformerException` if a transformation-related exception occurs during the operation.

`byte[]` **update** (`byte[]` *`in`*, `int` *`offset`*, `int` *`length`*) `throws TransformerException` [Function]

Returns the result of processing a designated *length* bytes from a given *in* byte array, starting at position *offset*. Throws an `IllegalStateException` if the instance is not yet initialized. Throws an `TransformerException` if a transformation-related exception occurs during the operation.

`byte[]` **lastUpdate** () `throws TransformerException` [Function]

Convenience method that calls the same method with three arguments. A zero-long byte array is used. Returns the result of transformation. Throws an `IllegalStateException` if the instance is not yet initialized. Throws an `TransformerException` if a transformation-related exception occurs during the operation.

`byte[]` **lastUpdate** (`byte` *`b`*) `throws TransformerException` [Function]

Convenience method that calls the method with same name and three arguments, using a byte array of length `1` whose contents are the designated byte *b*. Returns the result of transformation. Throws an `IllegalStateException` if the instance is not yet initialized. Throws an `TransformerException` if a transformation-related exception occurs during the operation.

`byte[]` **lastUpdate** (`byte[]` *`in`*) `throws TransformerException` [Function]

Convenience method that calls the same method with three arguments. All bytes in *in*, starting from index position `0` are considered. Returns the result of transformation. Throws an `IllegalStateException` if the instance is not yet initialized. Throws an `TransformerException` if a transformation-related exception occurs during the operation.

`byte[]` **lastUpdate** (`byte[]` *`in`*, `int` *`offset`*, `int` *`length`*) `throws TransformerException` [Function]

Returns the result of processing a designated *length* bytes from the given *in* byte array, starting at index position *offset* and signals, at the same time, that this is the last *push* operation on this **Transformer**. Throws an `IllegalStateException` if the instance is not yet initialized. Throws an `TransformerException` if a transformation-related exception occurs during the operation.

5.13 Example

The following example shows an **Assembly** that compresses its input data, before encrypting it with a Blowfish algorithm, in OFB mode, with PKCS7 padding.

```
import gnu.crypto.Registry;
import gnu.crypto.util.Util;
import gnu.crypto.assembly.Assembly;
import gnu.crypto.assembly.Cascade;
import gnu.crypto.assembly.Direction;
import gnu.crypto.assembly.Stage;
import gnu.crypto.assembly.Transformer;
import gnu.crypto.assembly.TransformerException;
import gnu.crypto.cipher.Blowfish;
import gnu.crypto.cipher.IBlockCipher;
import gnu.crypto.mode.IMode;
import gnu.crypto.mode.ModeFactory;
import gnu.crypto.pad.IPad;
import gnu.crypto.pad.PadFactory;


HashMap attributes = new HashMap();
HashMap modeAttributes = new HashMap();

Cascade ofbBlowfish = new Cascade();
Object modeNdx = ofbBlowfish.append(
    Stage.getInstance(
        ModeFactory.getInstance(Registry.OFB_MODE, new Blowfish(), 8),
        Direction.FORWARD));

attributes.put(modeNdx, modeAttributes);
IPad pkcs7 = PadFactory.getInstance(Registry.PKCS7_PAD);

Assembly asm = new Assembly();
asm.addPreTransformer(Transformer.getCascadeTransformer(ofbBlowfish));
asm.addPreTransformer(Transformer.getPaddingTransformer(pkcs7));
asm.addPreTransformer(Transformer.getDeflateTransformer());

// plaintext and key material
byte[] km = new byte[] { 0,  1,  2,  3,  4,  5,  6,  7,  8};
byte[] iv = new byte[] {-1, -2, -3, -4, -5, -6, -7, -8, -9};
byte[] pt = new byte[] { 0,  1,  2,  3,  4,  5,  6,  7,  8,  9,  10,  11};
byte[] tpt = new byte[11 * pt.length];

// forward transformation
modeAttributes.put(IBlockCipher.KEY_MATERIAL, km);
modeAttributes.put(IMode.IV, iv);
attributes.put(Assembly.DIRECTION, Direction.FORWARD);
```

```
try
  {
    asm.init(attributes);
  }
catch (TransformerException x)
  {
    x.printStackTrace(System.err);
  }

byte[] ct = null;
ByteArrayOutputStream baos = new ByteArrayOutputStream();
try
  {
    for (int i = 0; i < 10; i++)
      { // transform in parts of 12-byte a time
        System.arraycopy(pt, 0, tpt, i * pt.length, pt.length);
        ct = asm.update(pt);
        baos.write(ct, 0, ct.length);
      }
  }
catch (TransformerException x)
  {
    x.printStackTrace(System.err);
  }

try
  {
    System.arraycopy(pt, 0, tpt, 10 * pt.length, pt.length);
    ct = asm.lastUpdate(pt);
  }
catch (TransformerException x)
  {
    x.printStackTrace(System.err);
  }

baos.write(ct, 0, ct.length);
ct = baos.toByteArray();

// reversed transformation
attributes.put(Assembly.DIRECTION, Direction.REVERSED);
try
  {
    asm.init(attributes);
  }
catch (TransformerException x)
  {
    x.printStackTrace(System.err);
  }
```

```
byte[] ot = null;
try
  {
    ot = asm.lastUpdate(ct); // transform the lot in one go
  }
catch (TransformerException x)
  {
    x.printStackTrace(System.err);
  }
```

6 Message Digests

Message digests, or one-way hash functions, generate fixed-sized signatures from variable-sized texts, in such a way that it is computationally infeasible to determine the source text from the signature or to find a different text that hashes to the same signature. Hash functions in GNU Crypto are in the `gnu.crypto.hash` package, and are:

- **MD2**. MD2 is an early-generation hash function with an 128 bit output size, developed by Ron Rivest at RSA Data Security, Inc., and described by Burton Kaliski in RFC 1319 [Kal92]. No significant cryptanalysis has been published about MD2, but it is still recommended that new applications use a different message digest algorithm.
- **MD4**. MD4 was also developed by Ron Rivest at RSA Data Security, Inc. and is described by Rivest in RFC 1320 [Riv92a]. MD4 has a 128 bit output size. It is not recommended that MD4 be used in new applications.
- **MD5**. MD5 is a successor to MD4, developed by Ron Rivest and described in RFC 1321 [Riv92b], and has a 128 bit output size. MD5 is not widely considered secure any longer, and using other message digests with longer output sizes is recommended.
- **RIPEMD**. RIPEMD-128 and RIPEMD-160 have 128 bit and 160 bit output sizes, and were developed by Hans Dobbertin, Antoon Bosselaers, and Bart Preneel as successors to the RIPEMD hash.
- The **Secure Hash Algorithm**, **SHA-1**. The secure hash algorithm was developed by the National Institute for Standards and Technology, published in FIPS 180-1. SHA-1 has a 160 bit output length. FIPS 180-2, dated August 2002, added the specifications for three additional SHA implementations for output sizes of 256-, 384- and 512-bit respectively. These three algorithms are referred to as **SHA-256**, **SHA-384** and **SHA-512**.
- **Tiger** is a hash function created by Lars Anderson and Eli Biham, optimized for 64-bit architectures. It can produce a 192, 160, or 128 bit hash. [AnB96]
- **Whirlpool**. Whirlpool was designed by Paulo S. L. M. Barreto and Vincent Rijmen, and has a 512 bit output length.

6.1 IMessageDigest Interface

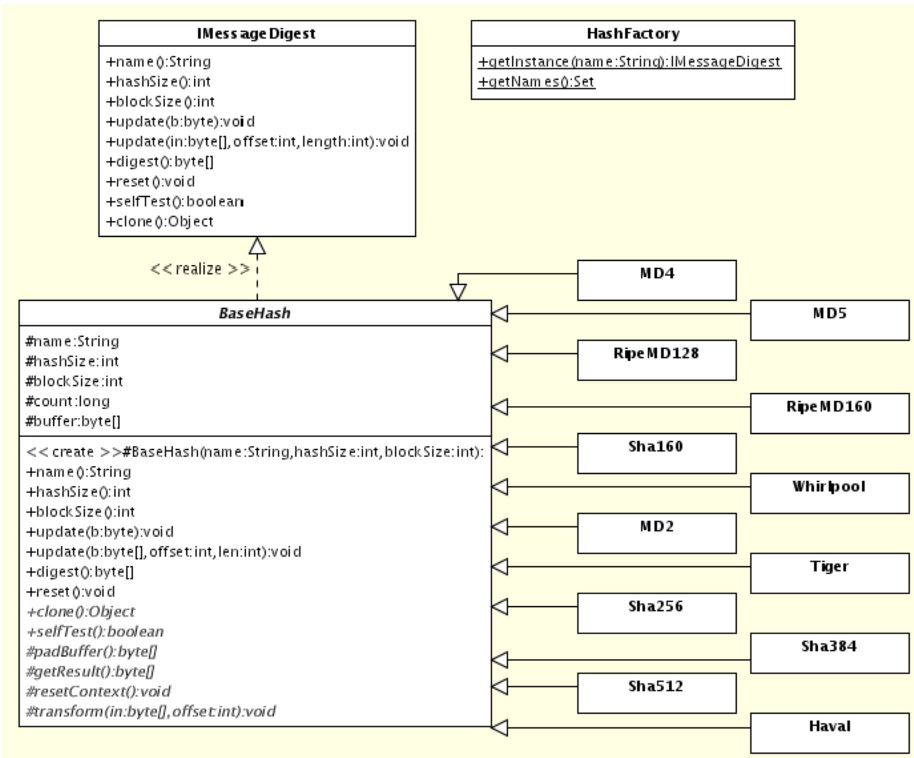

Figure 8: Message Digest class diagram

`void` **update** (`byte` *b*) [Function]

Updates the hash being computed with a single byte.

`void` **update** (`byte[] buf, int offset, int length`) [Function]

Update the hash being computed with *length* of the bytes in *buf* starting at *offset*. The programmer should ensure that *buf* is at least $offset + length$ bytes long.

`byte[]` **digest** ( ) [Function]

Finishes the computation of the hash and returns the result as a byte array. The input read thusfar may be padded first (depending on the algorithm), and the instance is reset.

`java.lang.String` **name** ( ) [Function]

Returns the canonical name of this message digest.

`int` **hashSize** ( ) [Function]

Returns the size of the final hash (the byte array returned by `digest()`) in bytes.

`int` **blockSize** ( ) [Function]

Returns the algorithm's internal block size, in bytes.

`void` **reset** ( ) [Function]

Resets the internal state of the hash, making its state equivalent to that of a newly-created instance.

`boolean` **selfTest** ( ) [Function]

Performs a simple conformance test of the underlying implementation, to guard against implementation or environment errors. Returns `true` if the test succeeds, `false` if it fails.

`java.lang.Object` **clone** ( ) [Function]

Copies the state of this instance into a new instance, returning the copy. This copy can then be used in the same way as the original instance.

6.2 HashFactory Class

Message digest instances are created with the static factory method:

`IMessageDigest` **getInstance** (`java.lang.String` *name*) [Function]

Creates a message digest instance for the algorithm *name*, or `null` if there is no such algorithm.

The `HashFactory` class also defines the method:

`java.util.Set` **getNames** ( ) [Function]

Returns a set of the names (strings) of all available message digest implementations.

6.3 Example

```
IMessageDigest md = HashFactory.getInstance("SHA-1");
md.update(input, 0, input.length);

byte[] digest = md.digest();
```

7 Message Authentication Codes

A message authentication code, or MAC, is akin to a *keyed hash function*, in that it produces a fixed-length identifier for variable-length data along with a key. The purpose of a MAC is to guarantee the integrity and authenticity of data, as it is computationally infesible to fake a MAC without knowledge of the key. MAC algorithms in GNU Crypto are in the `gnu.crypto.mac` package, and include:

- **Hash-based MAC**. Hash-based MACs, also called HMACs, use a normal message digest algorithm to compute the code based on input data and the key. GNU Crypto therefore implements an HMAC for every message digest it supports, and the name of a HMAC is usually "HMAC-" concatenated with the message digest's name; see the previous chapter on message digests for further discussion.
- The **Truncated Multi-Modular Hash** function, **TMMH**. TMMH/16 and TMMH/32 are universal hash functions; GNU Crypto implements TMMH/16. TMMH/16 has a variety of parameters, which are described later in this chapter. TMMH is described in [McG02].
- **UHASH-32**. UHASH-32 is a keyed hash function that outputs a hash of 8 bytes. The key supplied to this MAC must be 16 bytes long. UHASH is described in [Kro00].
- **UMAC-32**. The UMAC family of algorithms are parameterized, meaning that low-level choices such as endianness and the underlying cryptographic primitive are not fixed. The UMAC algorithms are described in [Kro00]. GNU Crypto implements UMAC-32, which performs well on 32- and 64-bit architectures, and has a key length of 16 bytes and an output length of 8 bytes. See the section on UMAC-32 for further discussion.

7.1 IMac Interface

`java.lang.String` **MAC_KEY_MATERIAL** [Variable]

A key in the attributes map passed to the `init` method. The value is taken to be a byte array, which contains the key as raw bytes. The length of the key must be at least the length of the computed hash in the case of hash-based MACs.

`java.lang.String` **TRUNCATED_SIZE** [Variable]

The actual size of the returned hash, taken from the first bytes of the raw result. The value must be a `java.lang.Integer` containing the desired length, which should not be smaller than 80 bits or one half the MAC's usual output length, whichever is larger.

`void` **init** (`java.util.Map` *attributes*) `throws java.security.InvalidKeyException, java.lang.IllegalStateException` [Function]

Initializes this MAC instance with a specified attributes map, which maps keys (such as `MAC_KEY_MATERIAL`) to parameters (such as the key bytes). Throws a `java.security.InvalidKeyException` if the key is unacceptable or omitted, and trows a `java.lang.IllegalStateException` if this instance has already been initialized.

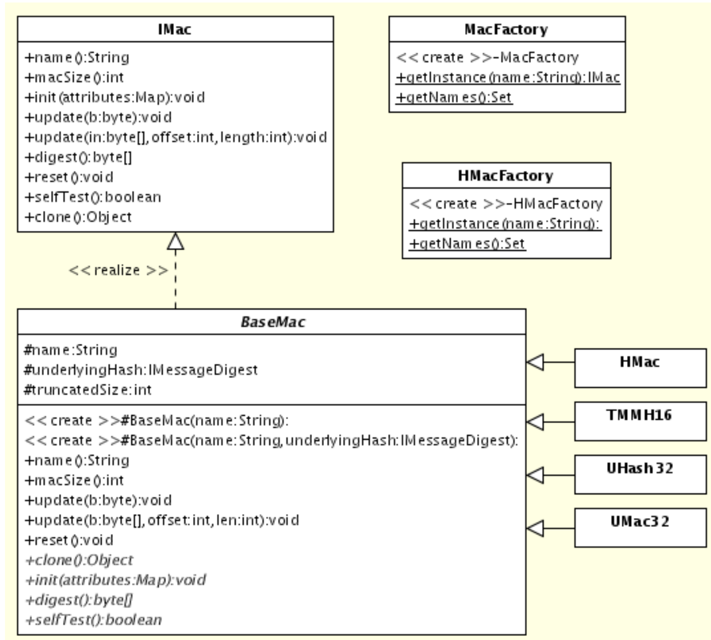

Figure 9: Message Authentication Code (MAC) class diagram

`void` **update** (`byte` *b*) [Function]

Continues the computation of the MAC with a single input byte, *b*.

`void` **update** (`byte[]` *in*, `int` *offset*, `int` *length*) [Function]

Continues the computation of the MAC with a portion of the byte array *in*, starting at *offset* and considering *length* bytes.

`byte[]` **digest** ( ) [Function]

Finishes the computation of the MAC and returns it in a new byte array. The instance is reset after this method returns.

`void` **reset** ( ) [Function]

Resets the internal state of this instance, which may then be re-initialized.

`int` **macSize** ( ) [Function]

Returns the size of the final MAC, in bytes.

`java.lang.String` **name** ( ) [Function]

Returns the canonical name of this algorithm.

`java.lang.Object` **clone** ( ) [Function]

Returns a copy of this instance, which may be used the same way as the original.

`boolean` **selfTest** ( ) [Function]

Performs a simple conformance test on this implementation; returns `true` if the test is successful, `false` if not.

7.2 MacFactory Class

MAC instances are created with the following factory method:

`IMac` **getInstance** (`java.lang.String` *name*) [Function]

Returns an instance of the MAC algorithm named *name*, or `null` if no such algorithm exists.

Additionally the `MacFactory` class defines the following method:

`java.util.Set` **getNames** ( ) [Function]

Returns a `java.util.Set` of the names of all available MAC algorithms.

7.3 TMMH/16

In addition to the key, the TMMH/16 requires three more parameters passed to its `init` method, using the following three keys:

`java.lang.String` **TAG_LENGTH** [Variable]

The output length, in bytes, represented as a `java.lang.Integer`. This value must be an even integer between 2 and 64.

`java.lang.String` **KEYSTREAM** [Variable]

An instance of `gnu.crypto.prng.IRandom`, which is to serve as the source of random bytes for this instance.

`java.lang.String` **PREFIX** [Variable]

A byte array of `TAG_LENGTH` bytes. If this parameter is omitted an all-zero byte array will be used. This value is XORed with the digest just before it is returned.

7.4 UMAC-32

The UMAC-32 algorithm requires, in addition to the key, a *nonce* byte array. The byte array must be 1–16 bytes of random data, which is passed to the `init` method of `IMac` in the attributes map. `UMac32` defined an additional key for this map:

`java.lang.String` **NONCE_MATERIAL** [Variable]

The key for the *nonce* material for the attributes map. The value mapped must be a byte array of size 1–16 bytes.

7.5 Example

```
IMac mac = MacFactory.getInstance("HMAC-SHA-160");
HashMap attributes = new HashMap();
attributes.put(IMac.MAC_KEY_MATERIAL, key_bytes);
attributes.put(IMac.TRUNCATED_SIZE, new Integer(12));
mac.init(attributes);

mac.update(input, 0, input.length);

byte[] result = mac.digest();
```

8 Keypairs and Key Agreements

This chapter is about keypairs. In the first section, keypair generation and keypair encoding and decoding concepts and API are described. The second section deals with key agreement protocols.

The code is organised into subpackages, each pertaining to a keypair algorithm. Four such algorithms are covered in this version of the library. They are:

- **dh**: Diffie-Hellman. The apparent intractability of this algorithm forms the basis for the security of many cryptographic schemes.
- **dss**: Digital Signature Standard.
- **rsa**: Named after its inventors Ron Rivest, Adi Shamir, and Leonard Adleman. Its security is based on the intractibility of the integer factorization problem.
- **srp6**: As described in Thomas Wu's paper "SRP-6: Improvements and Refinements to the Secure Remote Password Protocol," dated October 29, 2002. [Wu02]

8.6 Keypairs

The following class diagram shows the most important classes in the library that collaborate to implement the keypair generation functionality:

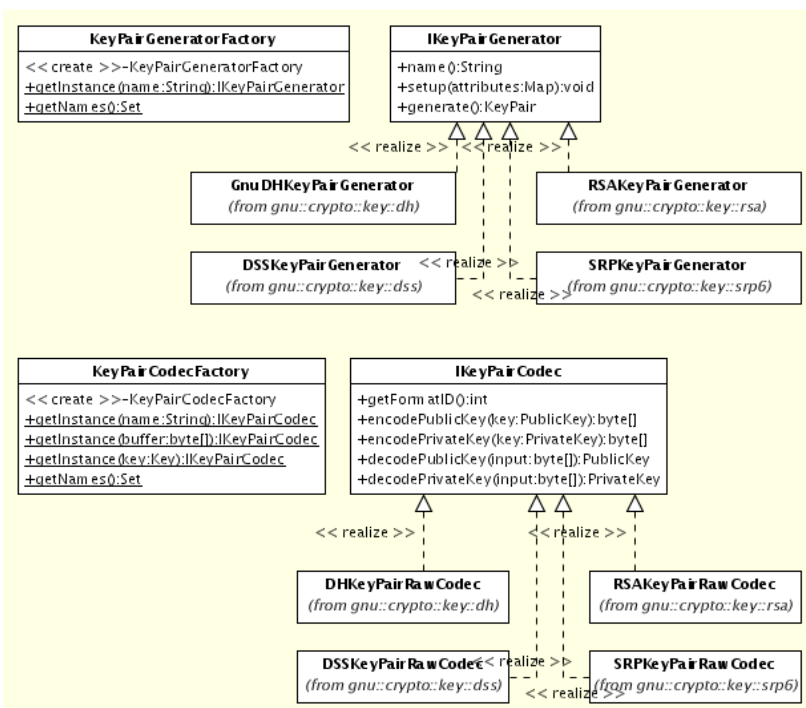

Figure 10: Keypair generation class diagram

The next figure is a sequence diagram showing the entities and messages involved in using those classes:

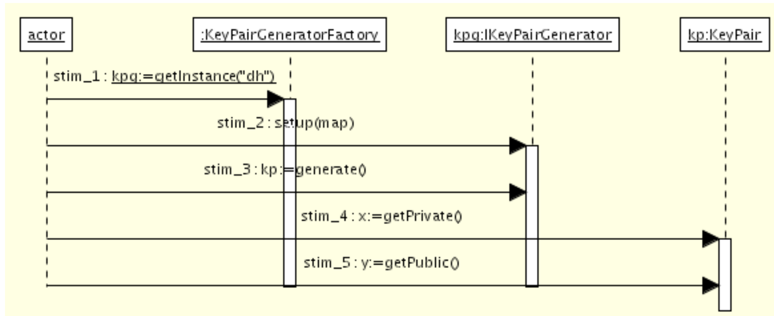

Figure 11: Keypair generation sequence diagram

8.7 Algorithm-Specific Attributes

8.7.1 Diffie-Hellman

Each of these constants are defined in the `gnu.crypto.key.dh.GnuDHKeyPairGenerator` class.

`java.lang.String` **SOURCE_OF_RANDOMNESS** [Variable]

Property name for the source of random bits to use when generating keys. The value mapped by this property must be of type `gnu.crypto.prng.IRandom` which must have been previously initialized. If undefined, then a default PRNG is used.

`java.lang.String` **DH_PARAMETERS** [Variable]

Property name for an optional `javax.crypto.spec.DHGenParameterSpec` instance to use for this generator.

`java.lang.String` **PRIME_SIZE** [Variable]

Property name of the size in bits (an instance of `java.lang.Integer`) of the public prime p.

`java.lang.String` **EXPONENT_SIZE** [Variable]

Property name of the size in bits (an instance of `java.lang.Integer`) of the private exponent x.

8.7.2 DSS

Each of these constants are defined in the `gnu.crypto.key.dss.DSSKeyPairGenerator` class.

`java.lang.String` **SOURCE_OF_RANDOMNESS** [Variable]

Property name for the source of random bits to use when generating keys. The value mapped by this property must be of type `gnu.crypto.prng.IRandom` which must have been previously initialized. If undefined, then a default PRNG is used.

`java.lang.String` **DSS_PARAMETERS** [Variable]

Property name of an optional `java.security.spec.DSAParameterSpec` instance to use for this generator's p, q, and g values. The default is to generate these values or use pre-computed ones, depending on the value of the *USE_DEFAULTS* attribute.

`java.lang.String` **MODULUS_LENGTH** [Variable]

Property name for the modulus length, in bits. The value mapped by this property must be of type `java.lang.Integer`.

`java.lang.String` **USE_DEFAULTS** [Variable]

Property name of an instance of `java.lang.Boolean` indicating wether or not to use pre-computed default values for the algorithm parameters. Three sets of such parameters are also provided covering 512-bit (*KEY_PARAMS_512*, 768-bit (*KEY_PARAMS_768*) and 1024-bit (*KEY_PARAMS_512*) keylength.

8.7.3 RSA

Each of these constants are defined in the `gnu.crypto.key.rsa.RSAPSSKeyPairGenerator` class.

`java.lang.String` **SOURCE_OF_RANDOMNESS** [Variable]

Property name for the source of random bits to use. The value mapped by this property must be of type `gnu.crypto.prng.IRandom`, which must have been previously initialized. If undefined, then a default PRNG is used.

`java.lang.String` **MODULUS_LENGTH** [Variable]

Property name for the length, in bits, of the modulus. The value mapped by this property must be of type `java.lang.Integer`.

`java.lang.String` **RSA_PARAMETERS** [Variable]

Property name for the optional values of e and n. The value mapped by this property must be of type `java.security.spec.RSAKeyGenParameterSpec` Random or default values will be used instead if this parameter is not specified.

8.7.4 SRP6

Each of these constants are defined in the `gnu.crypto.key.srp6.SRPKeyPairGenerator` class.

`java.lang.String` **SOURCE_OF_RANDOMNESS** [Variable]

Property name for the source of random bits to use. The value mapped by this property must be of type `gnu.crypto.prng.IRandom`, which must have been previously initialized. If undefined, then a default PRNG is used.

`java.lang.String` **MODULUS_LENGTH** [Variable]

Property name of the length (an instance of `java.lang.Integer`) of the modulus N of an SRP key.

`java.lang.String` **SHARED_MODULUS** [Variable]

Property name of the value of the modulus N of an SRP key. The value mapped by this property, if/when defined, must be of type `java.math.BigInteger`. It is an optional parameter. If undefined, then a new value is generated, unless *USE_DEFAULTS* is set to TRUE.

`java.lang.String` **GENERATOR** [Variable]

Property name of the value of the generator g of an SRP key. The value mapped by this property, if/when defined, must be of type `java.math.BigInteger`. It is an optional parameter. If undefined, then a new value is generated, unless *USE_DEFAULTS* is set to TRUE.

`java.lang.String` **USE_DEFAULTS** [Variable]

Property name of an instance of `java.lang.Boolean` indicating wether or not to use pre-computed default values for the algorithm parameters. Seven sets of such parameters are also provided covering 512-bit (*N_512*, 640-bit (*N_640*), 768-bit (*N_768*), 1024-bit (*N_1024*), 1280-bit (*N_1280*), 1536-bit (*N_1536*) and 2048-bit (*N_2048*) shared modulus length.

8.8 The IKeyPairGenerator Interface

All signature algorithms in GNU Crypto have their corresponding key pair generators, which implement this interface and provide the following methods:

`void` **setup** (`java.util.Map` *`attributes`*) `throws java.lang.IllegalArgumentException` [Function]

Initializes this key pair generator with the given attrubutes. The property names used are algorithm-dependent, and are described in the next section. This method throws a `java.lang.IllegalArgumentException` if the given attributes are incorrect or incomplete.

`java.security.KeyPair` **generate** () [Function]

Generates and returns a new key pair based on the attributes used to configure this instance.

`java.lang.String` **name** () [Function]

Returns the canonical name of the algorithm this class generates key pairs for.

8.9 The KeyPairGeneratorFactory Class

`IKeyPairGenerator` **getInstance** (`java.lang.String` *`algorithm`*) [Function]

Returns an instance of a key pair generator for *algorithm*, or `null` if no such generator is available.

`java.util.Set` **getNames** () [Function]

Returns an unmodifiable set of all available key pair generator algorithms, each entry a `java.lang.String`.

8.10 The IKeyPairCodec Interface

A key pair codec is used to externalize and de-externalize the key pairs used in GNU Crypto. There is no factory class, but rather the implementations have public, zero-argument constructors. The available codecs are:

- `gnu.crypto.key.dh.DHKeyPairRawCodec`, for encoding and decoding Diffie-Hellman key pairs.
- `gnu.crypto.key.dss.DSSKeyPairRawCodec`, for encoding and decoding DSS key pairs.
- `gnu.crypto.key.rsa.RSAKeyPairRawCodec`, for encoding and decoding RSA key pairs.
- `gnu.crypto.key.srp6.SRPKeyPairRawCodec`, for encoding and decoding SRP key pairs.

`int` **RAW_FORMAT** [Variable]

Constant identifying the “raw” format used by GNU Crypto.

`java.security.PrivateKey` **decodePrivateKey** (`byte[]` *encoded*) [Function]

Decodes a private key from its external representation, returning it as an appropriate instance of `java.security.PrivateKey`. This function will throw a `java.lang.IllegalArgumentException` if the encoded bytes cannot be decoded or are incorrect.

`java.security.PublicKey` **decodePublicKey** (`byte[]` *encoded*) [Function]

Decodes a public key from its external representation, returning it as an appropriate instance of `java.security.PublicKey`. This function will throw a `java.lang.IllegalArgumentException` if the encoded bytes cannot be decoded or are incorrect.

`byte[]` **encodePrivateKey** (`java.security.PrivateKey` *key*) [Function]

Encodes a private key to its external representation, returning the encoded bytes. This function will throw a `java.lang.IllegalArgumentException` if the key cannot be encoded by this instance.

`byte[]` **encodePublicKey** (`java.security.PublicKey` *key*) [Function]

Encodes a public key to its external representation, returning the encoded bytes. This function will throw a `java.lang.IllegalArgumentException` if the key cannot be encoded by this instance.

`int` **getFormatID** () [Function]

Returns the format identifier of this codec, such as `RAW_FORMAT`.

8.11 Example

The following example demonstrates how to generate a DSS keypair.

```
IKeyPairGenerator kpg = KeyPairGeneratorFactory.getInstance(Registry.DSS_KPG);
HashMap map = new HashMap();
map.put(DSSKeyPairGenerator.MODULUS_LENGTH, new Integer(512));
map.put(DSSKeyPairGenerator.USE_DEFAULTS, new Boolean(false));
kpg.setup(map);
KeyPair kp = kpg.generate();

BigInteger p1 = ((DSAPublicKey) kp.getPublic()).getParams().getP();
BigInteger p2 = ((DSAPrivateKey) kp.getPrivate()).getParams().getP();

BigInteger q1 = ((DSAPublicKey) kp.getPublic()).getParams().getQ();
BigInteger q2 = ((DSAPrivateKey) kp.getPrivate()).getParams().getQ();

BigInteger g1 = ((DSAPublicKey) kp.getPublic()).getParams().getG();
BigInteger g2 = ((DSAPrivateKey) kp.getPrivate()).getParams().getG();
```

8.12 Key Agreements

8.13 Protocols

A key agreement protocol is a means by which two parties engage in an exchange of incoming/outgoing messages, at the end of which, both participants would share a common secret. Such a shared secret can then be used to provide different security services such as replay detection, integrity protection, and confidentiality protection.

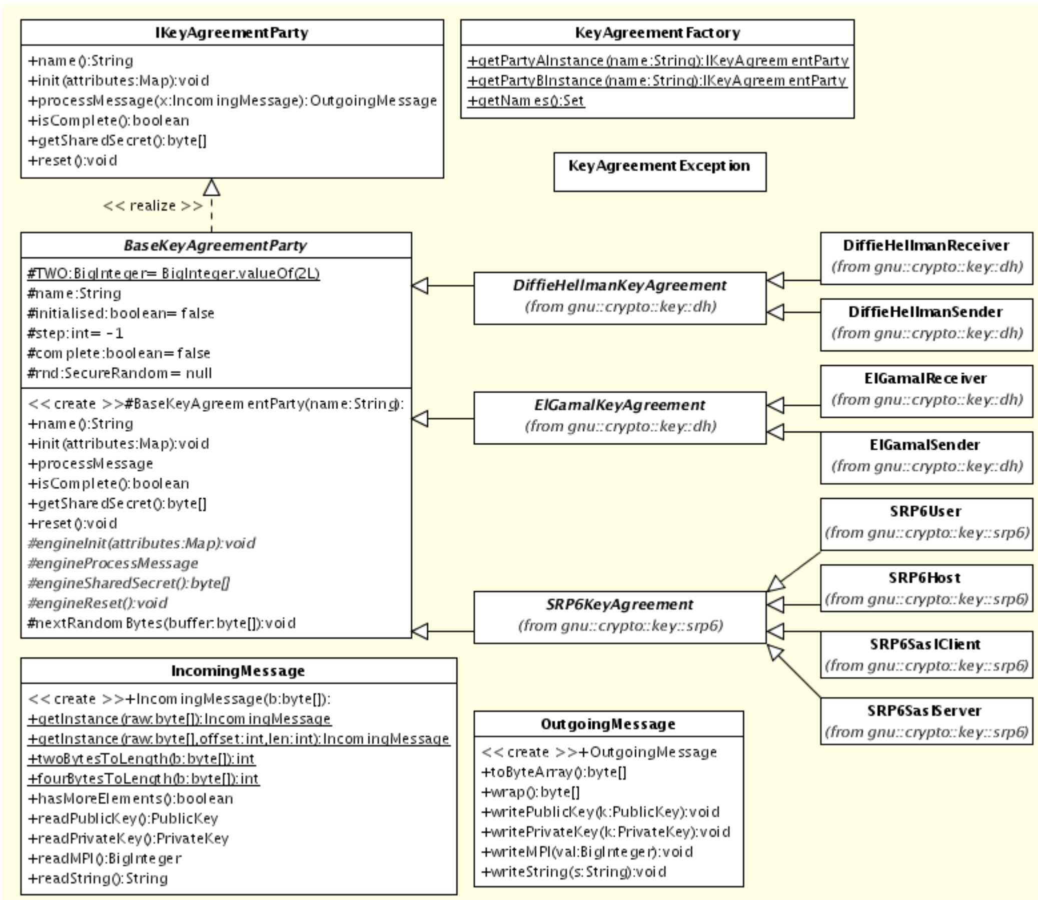

Figure 12: Key agreement class diagram

Four key agreement protocols are implemented in this library. They are:

- Diffie-Hellman **basic version**, also known as the **Static-Static Mode** in RFC-2631. [RFC2631]
- ElGamal version, knwon as **half-certified** Diffie-Hellman key agreement, as well as **Ephemeral-Static Mode** in RFC-2631. [RFC2631]
- Secure Remote Password protocol known as SRP-6. [Wu02]
- The version of SRP-6 as used in the SASL-SRP proposed mechanism.

The following sequence diagram shows a possible use of the key agreement API classes to negotiate a Diffie-Hellman protocol:

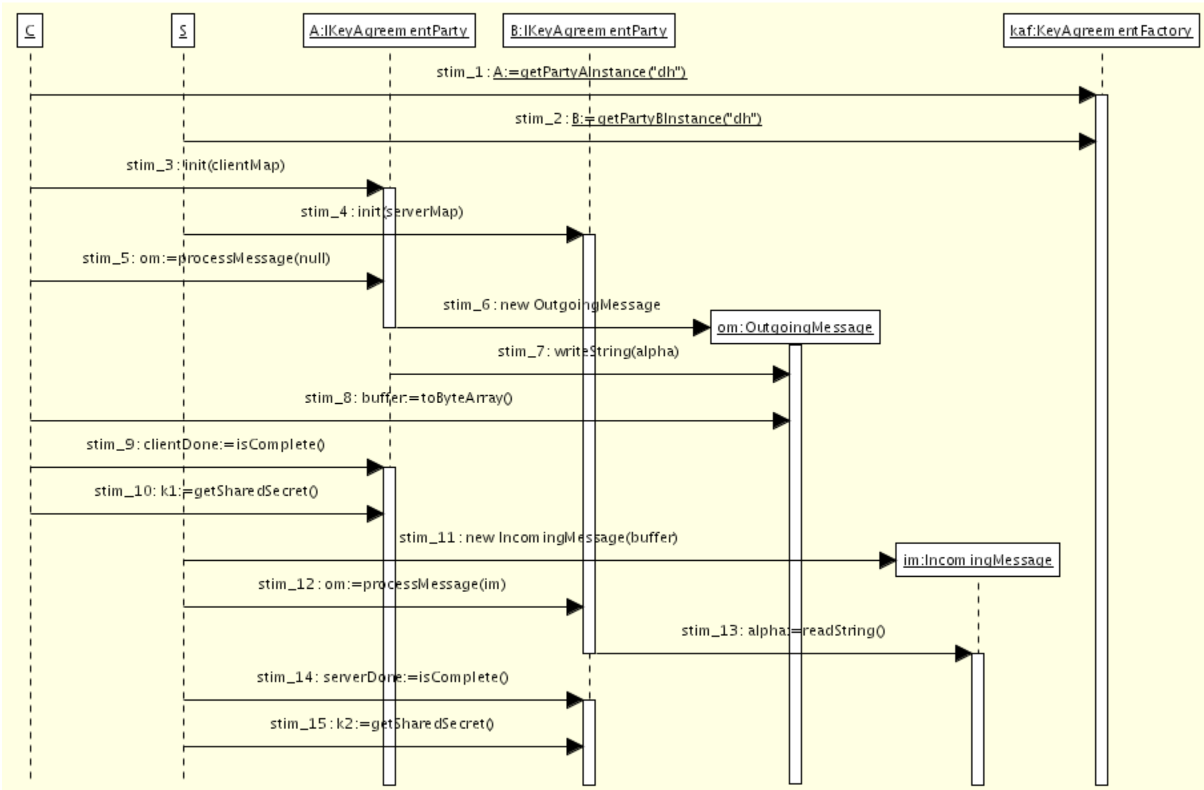

Figure 13: Key agreement sequence diagram

8.14 The IKeyAgreementParty Interface

`java.lang.String` **name** () [Function]

Returns the canonical name of the key agreement protocol.

`void` **init** (`java.util.Map` *attributes*) `throws` `gnu.crypto.key.KeyAgreementException` [Function]

Initializes this instance. The *attributes* parameter must be a `java.util.Map` that has the required name-value pairs needed for this instance. An instance of `gnu.crypto.key.KeyAgreementException` is thrown if an exception occurs during this process.

`gnu.crypto.key.OutgoingMessage` **processMessage** (`gnu.crypto.key.IncomingMessage` *in*) `throws` `gnu.crypto.key.KeyAgreementException` [Function]

Processes an incoming message (*in*) at one end, generating a message (the returned object which may be null) that will be processed by the other party(ies). A `gnu.crypto.key.KeyAgreementException` may be thrown if an exception occurs during this process.

`boolean` **isComplete** () [Function]

Returns `true` if the party in the key agreement protocol exchange has completed its part of the exchange; and `false` otherwise. If this method returns `false`, then an `java.lang.IllegalStateException` is thrown for any method invocation except `init`.

`byte[]` **getSharedSecret** () `throws gnu.crypto.key.KeyAgreementException` [Function]

Returns the byte array containing the shared secret as generated by this party. A `gnu.crypto.key.KeyAgreementException` is thrown if the key agreement is not yet initialised, or is initialised but the exchange is still in progress.

`void` **reset** () [Function]

Resets this instance for re-use with another set of attributes.

8.15 The KeyAgreementFactory class

Instances for two-party key agreement protocols can be instantiated with the *Factory* methods of this class:

`gnu.crypto.key.IKeyAgreementParty` **getPartyAInstance** (`java.lang.String` *name*) [Function]

Creates an instance of an *initiator* of a key agreement protocol given the *name* of this protocol. A `null` if there is no such protocol implementation.

`gnu.crypto.key.IKeyAgreementParty` **getPartyBInstance** (`java.lang.String` *name*) [Function]

Creates an instance of a *recipient* of a key agreement protocol given the *name* of this protocol. A `null` if there is no such protocol implementation.

`java.util.Set` **getNames** () [Function]

Returns a set of the names (`java.lang.String`) of all available key agreement protocols.

8.16 Example, Key agreement

The following example shows ...

```
IKeyPairGenerator kpg = KeyPairGeneratorFactory.getInstance(Registry.DH_KPG);
kpg.setup(new HashMap()); // use default values
KeyPair kpA = kpg.generate();
KeyPair kpB = kpg.generate();
IKeyAgreementParty A = new DiffieHellmanSender();
IKeyAgreementParty B = new DiffieHellmanReceiver();

Map mapA = new HashMap();
mapA.put(DiffieHellmanKeyAgreement.KA_DIFFIE_HELLMAN_OWNER_PRIVATE_KEY,
         kpA.getPrivate());
Map mapB = new HashMap();
mapB.put(DiffieHellmanKeyAgreement.KA_DIFFIE_HELLMAN_OWNER_PRIVATE_KEY,
         kpB.getPrivate());

A.init(mapA);
B.init(mapB);

// (1) A -> B: g**x mod p
OutgoingMessage out = A.processMessage(null);

// (2) B -> A: g^^y mod p
out = B.processMessage(new IncomingMessage(out.toByteArray()));

byte[] k2 = B.getSharedSecret();

// A computes the shared secret
out = A.processMessage(new IncomingMessage(out.toByteArray()));

byte[] k1 = A.getSharedSecret();
```

9 Signatures

This chapter describes the digital signature schemes implemented in GNU Crypto. The package for all signature and related classes is `gnu.crypto.sig`. The following signature schemes are implemented:

- **DSS**, the Digital Signature Standard, was standardized in 1994 by the National Institute of Standards and Technology in the Federal Information Processing Standards (FIPS) Publication 186 [FIPS186]. DSS uses the secure hash algorithm (SHA-1) internally, and produces a 160 bit signature.
- **RSA-PSS**. This is a digital signature scheme based on the combination of the RSA algorithm with the Probabilistic Signature Scheme (PSS) encoding scheme. RSA was invented by Ron Rivest, Adi Shamir, and Leonard Adleman; the PSS encoding was developed by Mihir Bellare and Phillip Rogaway. During efforts to adopt RSA-PSS into the IEEE P1363a standards effort, certain adaptations to the original version of RSA-PSS were made by Mihir Bellare and Phillip Rogaway and also by Burt Kaliski (the editor of IEEE P1363a) to facilitate implementation and integration into existing protocols. [JoK00]

9.1 The ISignature Interface

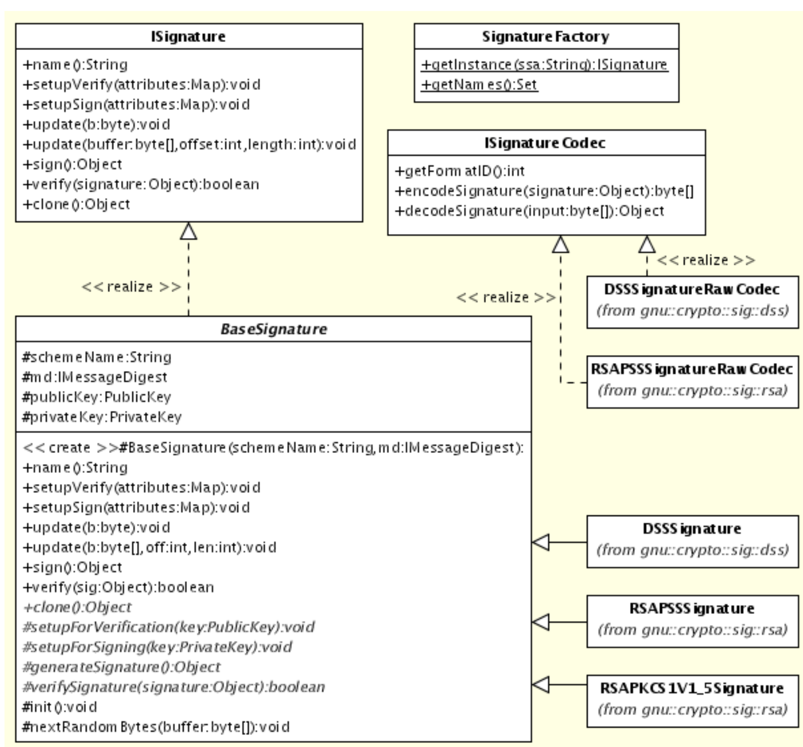

Figure 14: Signature class diagram

All digital signature schemes implement the `ISignature` interface, and support the following methods:

`java.lang.String` **SIGNER_KEY** [Variable]

A property name in the attributes map that is passed to instances being prepared for signing. The value mapped by this key must be a `java.security.PrivateKey` that is appropriate for the instance's algorithm (e.g. an instance of DSS would require a subclass of `java.security.interfaces.DSAPrivateKey`).

`java.lang.String` **VERIFIER_KEY** [Variable]

A property name in the attributes map that is passed to instances being prepared for verifying a signature. The value mapped by this key must be a `java.security.PublicKey` that is appropriate for the instance's algorithm, just as is the case with the signing key.

`java.lang.String` **SOURCE_OF_RANDOMNESS** [Variable]

A property name in the attributes map that is passed to instances being prepared for use as either signers or verifiers. The value mapped must be an already-initialized instance of `gnu.crypto.prng.IRandom`.

`void` **setupSign** (`java.util.Map` *attributes*) `throws java.lang.IllegalArgumentException` [Function]

Initializes this instance for signing. The *attributes* parameter must be a `java.util.Map` that has, at least, a mapping between the `SIGNER_KEY` property and the appropriate private key.

`void` **setupVerify** (`java.util.Map` *attributes*) `throws java.lang.IllegalArgumentException` [Function]

Initializes this instance for verifying a signature. The *attributes* parameter must be a `java.util.Map` that has, at least, a mapping between the `VERIFIER_KEY` property and the appropriate public key.

`void` **update** (`byte` *b*) `throws java.lang.IllegalStateException` [Function]

Update either the signing or verifying operation with the next byte in the message. This method will throw a `java.lang.IllegalStateException` if this instance has not been initialized for either signing or verifying.

`void` **update** (`byte[]` *buf*, `int` *off*, `int` *len*) `throws java.lang.IllegalStateException` [Function]

Update either the signing or verifying operation with the next *len* bytes of *buf*, starting at *offset*. This method will throw a `java.lang.IllegalStateException` if this instance has not been initialized for either signing or verifying.

`java.lang.Object` **sign** () `throws java.lang.IllegalStateException` [Function]

Finishes a signing operation and returns the final signature. This method will throw a `java.lang.IllegalStateException` if this instance has not been initialized for signing.

`boolean` **verify** (`java.lang.Object` *signature*) `throws java.lang.IllegalStateException` [Function]

Finishes a verifying operation by checking if the argument, a native signature object, matches the expected signature. This methods returns `true` if the signature is valid, `false` otherwise. This method will throw a `java.lang.IllegalStateException` if this instance has not been initialized for verifying.

`java.lang.String` **name** () [Function]

Returns the canonical name of this instance's signature algorithm.

`java.lang.Object` **clone** () [Function]

Returns a copy of this signature object.

9.2 The SignatureFactory Class

Instances of `ISignature` can be retrieved with the class methods of the `SignatureFactory` class:

`ISignature` **getInstance** (`java.lang.String` *name*) [Function]

Creates an instance of the signature scheme for *name*, or `null` if there is no such algorithm.

`java.util.Set` **getNames** () [Function]

Returns a set of the names (`java.lang.String`) of all available signature schemes.

9.3 The ISignatureCodec Interface

The `ISignatureCodec` interface defines methods for externalizing and de-externalizing native signature results, as would be returned by the `ISignature.sign()` method, or passed to `ISignature.verify()` method. The only format currently supported is the "RAW" codec, which is specific to GNU Crypto.

Each signature scheme implements its own raw codec. There is no factory for codecs, but rather you should create instances of

- `gnu.crypto.sig.dss.DSSSignatureRawCodec` if you are reading or writing DSS signatures, or
- `gnu.crypto.sig.rsa.RSAPSSSignatureRawCodec` if you are reading or writing RSA-PSS signatures.

Each of these classes has a zero-argument constructor, needs no initialization, and defines these methods:

`java.lang.Object` **decodeSignature** (`byte[]` *encoded*) [Function]

Decodes a signature from the byte represention *encoded*, and returns the signature in the signature algorithm's native form. Implementations may throw an unchecked exception (such as `java.lang.IlligalArgumentException`) if the argument is improperly formatted.

`byte[]` **encodeSignature** (`java.lang.Object` *signature*) [Function]

Encodes a native signature to an external byte representation. Implementations may throw an unchecked exception (such as `java.lang.IlligalArgumentException`) if the argument is not of the algorithm's native signature type.

`int` **getFormatID** () [Function]

Returns the format identifier for this codec, such as `RAW_FORMAT`.

`int` **RAW_FORMAT** [Variable]

Format identifier for GNU's "raw" codec.

9.4 Signature Example

```
ISignature dss = SignatureFactory.getInstance("DSS");
Map attrib = new HashMap();
attrib.put(ISignature.SIGNER_KEY, privateDsaKey);
dss.setupSign(attrib);

dss.update(message, 0, message.length);
Object sig = dss.sign();

ISignatureCodec codec = new DSSSignatureRawCodec();
byte[] encoded = codec.encodeSignature(sig);

Object sig2 = codec.decodeSignature(encoded);

attrib.clear();
attrib.put(ISignature.VERIFIER_KEY, publicDsaKey);
dss.setupVerify(attrib);

dss.update(message, 0, message.length);
boolean valid = dss.verify(sig);
```

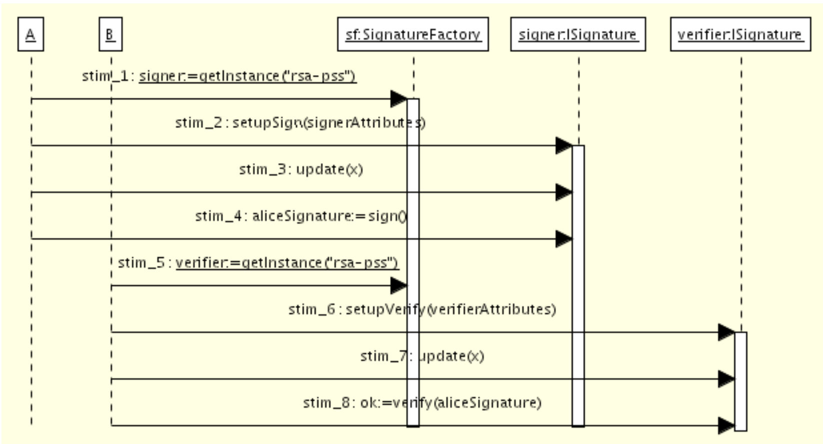

Figure 15: Signature sequence diagram

10 Random Numbers

The pseudo-random number generator (PRNG) classes of GNU Crypto are used to generate streams of cryptographically secure pseudo-random bytes.

- **ARCFOUR** is an implementation of the ARCFOUR stream cipher's keystream generator. ARCFOUR is the name of a stream cipher that is believed to be compatible with RSA Data Security, Inc.'s RC4 stream cipher, and is a decendent of an algorithm that was posted anonymously to a mailing list in 1994.
- **ICM**, or the Integer Counter Mode PRNG, is an algorithm that creates a PRNG around a block cipher. The default cipher used in this implementation is Rijndael, the AES. ICM is described in [McG01].
- **MD**, or PRNGs based around a cryptographic hash function.
- **UMAC-KDF** is a PRNG based on the UMAC key derivation function.

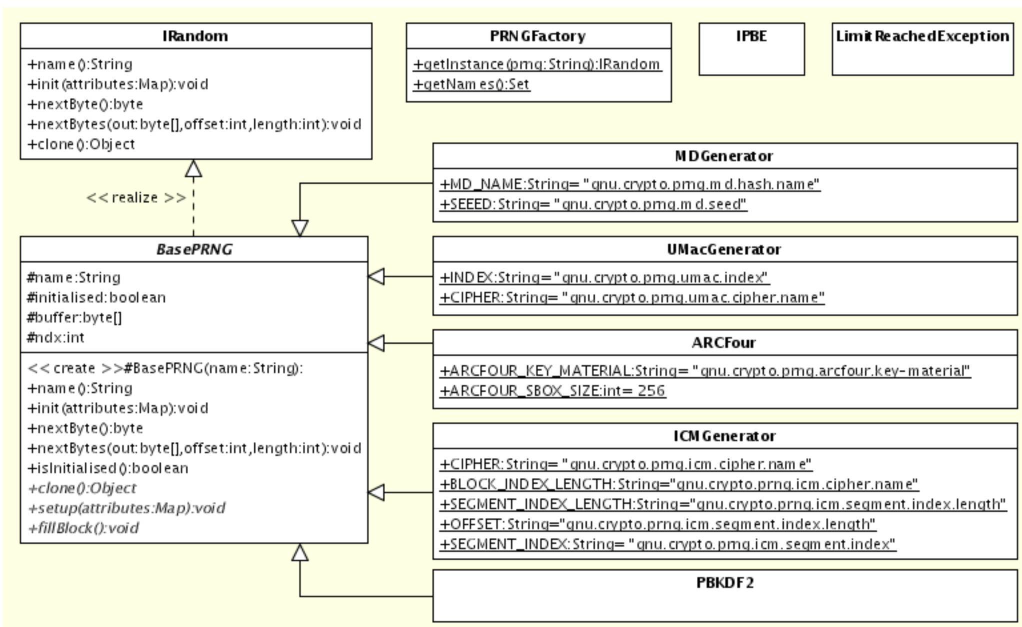

Figure 16: PRNG class diagram

10.1 The IRandom Interface

`void` **init** (`java.util.Map` *attributes*) [Function]

Initializes this PRNG, preparing it for use. Throws an `IllegalArgumentException` if the given attributes are not appropriate for this PRNG algorithm.

`byte` **nextByte** () `throws LimitReachedException` [Function]

Returns the next pseudo-random byte in this generator's sequence. Throws a `LimitReachedException` if this generator cannot produce any more bytes of any quality.

`void` **nextBytes** (`byte[]` *out*, `int` *off*, `int` *len*) `throws LimitReachedException` [Function]

Fills the buffer *out* with the next *len* bytes in this generator's sequence, storing the bytes beginning at *off*. Throws a `LimitReachedException` if this generator cannot produce any more bytes of any quality.

`java.lang.String` **name** () [Function]

Returns the canonical name of this PRNG algorithm.

`java.lang.Object` **clone** () [Function]

Returns a copy of this instance. The copy will be in the exact same state as this instance, and will be independent of this instance.

10.2 The PRNGFactory Class

`IRandom` **getInstance** (`java.lang.String` *name*) [Function]

Returns an instance of the named PRNG algorithm, or `null` if no such named algorithm exists.

`java.util.Set` **names** () [Function]

Returns a `java.util.Set` of the names (`java.lang.String`) of all available PRNG algorithms.

10.3 ARCFour

The ARCFour keystream is implemented in the class `ARCFour`, which defines the following additional constant:

`java.lang.String` **ARCFOUR_KEY_MATERIAL** [Variable]

A property name in the attributes map used to initialize instances of `ARCFour`. The value mapped must be a byte array of the secret key, which can be up to 256 bytes long.

Also note that using the ARCFour PRNG as a stream cipher is as simple as:

```
IRandom arcfour; // initialized elsewhere.
byte in, out;

out = in ^ arcfour.next();
```

10.4 MDGenerator

Generic message digest-based PRNGs are implemented via the `MDGenerator` class, which defines the following additional constants:

`java.lang.String` **MD_NAME** [Variable]

A property name in the attributes map used to initialize instances of `MDGenerator`. The value mapped must be a `String` representing the name of the hash function to use, such as “MD5”. If this attribute is omitted the secure hash algorithm, SHA-1, is used.

`java.lang.String` **SEEED** [Variable]

A property name in the attributes map used to initialize instances of `MDGenerator`. The value mapped must be a byte array carrying the seed, with which to seed the PRNG. This attribute is optional.

10.5 ICMGenerator

The ICM generator accepts a number of additional parameters, all contained in the following constants of the `ICMGenerator` class. The appropriate values, including the limits of the integral types, are specific to the ICM generator algorithm.

`java.lang.String` **BLOCK_INDEX_LENGTH** [Variable]

A property name in the attributes map used to initialize instances of `ICMGenerator`. The value mapped must be a `java.lang.Integer`.

`java.lang.String` **CIPHER** [Variable]

A property name in the attributes map used to initialize instances of `ICMGenerator`. The value mapped must be a `gnu.crypto.cipher.IBlockCipher`, and is the underlying cipher used in the algorithm.

`java.lang.String` **OFFSET** [Variable]

A property name in the attributes map used to initialize instances of `ICMGenerator`. The value mapped must be a `java.math.BigInteger` or a byte array of the same length of the underlying cipher's block size.

`java.lang.String` **SEGMENT_INDEX** [Variable]

A property name in the attributes map used to initialize instances of `ICMGenerator`. The value mapped must be a `java.math.BigInteger`.

`java.lang.String` **SEGMENT_INDEX_LENGTH** [Variable]

A property name in the attributes map used to initialize instances of `ICMGenerator`. The value mapped must be a `java.lang.Integer`.

10.6 UMacGenerator

The UMac KDF generator accepts the following additional parameters, which are contained in the `UMacGenerator` class.

`java.lang.String` **CIPHER** [Variable]

A property name in the attributes map used to initialize instances of `UMacGenerator`. The value mapped must be of type `gnu.crypto.cipher.IBlockCipher`.

`java.lang.String` **INDEX** [Variable]

A property name in the attributes map used to initialize instances of `UMacGenerator`. The value mapped must be of type `java.lang.Integer`.

10.7 PRNG Example

```
Map attrib = ...;
IRandom rand = PRNGFactory.getInstance("MD");

attrib.put(MDGenerator.MD_NAME, "MD5");
attrib.put(MDGenerator.SEEED, seedBytes);

random.init(attrib);

for (int i = 0; i < bytes.length; i++)
   {
      in[i] ^= random.nextByte();
   }

random.nextBytes(bytes, 0, bytes.length);
```

Part 2: External API Support

11 JCE Support

GNU Crypto provides a full JCE (Java Cryptography Environment) provider for all its algorithms. This chapter breifly describes these classes and how to use them.

11.1 Installing the JCE Classes

Java runtimes such as those based around Classpath, Kaffe, and JREs from Sun and IBM up to version 1.4 do not include the JCE classes, encompassed by the `javax.crypto` package and its subpackages. Furthermore, many commercial Java 1.4 and later runtime environments do not allow providers to be installed if they are not digitally signed by an authority. The GNU Crypto developers do not agree with this practice and are not seeking to have GNU Crypto's provider signed.

To overcome this GNU Crypto includes a clean-room implementation of the `javax.crypto` packages, which is a modified version of the clean-room JCE distributed by the Legion of the Bouncy Castle `http://bouncycastle.org/`. If building these classes is enabled at compile-time, a Java archive file `javax-crypto.jar` will be built, along with the appropriate shared native libraries if you are using GCJ. Simply adding it to your system classpath should suffice, possibly replacing or superceding the `jce.jar` file that came with your virtual machine.

The JCE included mirrors most of the features of the reference JCE, except the `ExemptionMechanism` classes are omitted. U.S. export rules as of January 2000 no longer apply to open source software that is freely available on the Internet, so these classes have no practical use in GNU Crypto.

11.2 Installing the GNU Crypto Provider

The GNU Crypto provider is implemented in the class `gnu.crypto.jce.GnuCrypto`, and is available by the name "GNU Crypto". You can install this provider at run-time by including in your program a statement such as:

```
java.security.Security.addProvider(new gnu.crypto.jce.GnuCrypto());
```

Or by putting the following in your security properties file, usually located at `${JRE_HOME}/lib/security/${VM_NAME}.security`:

```
security.provider.N=gnu.crypto.jce.GnuCrypto
```

Where 'N' is the appropriate preference number. Doing this, and asserting that the `gnu-crypto.jar` file is in your classpath, will complete the installation of the provider.

11.3 List of Available Algorithms

The algorithms available through the GNU Crypto provider are, grouped by type, with alternate names in parentheses:

Cipher: AES, ANUBIS, ARCFOUR (RC4), BLOWFISH, DES, KHAZAD, RIJNDAEL, SERPENT, SQUARE, TRIPLEDES, TWOFISH.

Ciphers may, of course, be appended with any of the modes and paddings available in GNU Crypto, such as “AES/CBC/TBC”.

KeyPairGenerator: DSS (DSA), RSA.

MAC: HMAC-MD2, HMAC-MD4, HMAC-MD5, HMAC-RIPEMD128 (HMAC-RIPEMD-128), HMAC-RIPEMD160 (HMAC-RIPEMD-160), HMAC-SHA160 (HMAC-SHA, HMAC-SHA1, HMAC-SHA-160, HMAC-SHS), HMAC-TIGER, HMAC-WHIRLPOOL, TMMH16, UHASH32, UMAC32.

MessageDigest: MD2, MD4, MD5, RIPEMD128 (RIPEMD-128), RIPEMD-160 (RIPEMD-160), SHA-160 (SHA, SHA1, SHA-1, SHS), TIGER, WHIRLPOOL.

SecureRandom: ARCFOUR (RC4), ICM, MD2PRNG, MD4PRNG, MD5PRNG, RIPEMD128PRNG, RIPEMD160PRNG, SHA-160PRNG (SHAPRNG, SHA-1PRNG, SHA1PRNG), TIGERPRNG, WHIRLPOOLPRNG, UMAC-KDF.

Signature: DSS/RAW (SHA/DSA, SHA1/DSA, SHA-1/DSA, SHA-160/DSA, DSAwithSHA, DSAwithSHA1, DSAwithSHA160), RSA-PSS/RAW (RSA-PSS, RSAPSS).

GNU Free Documentation License

0. PREAMBLE

   The purpose of this License is to make a manual, textbook, or other written document *free* in the sense of freedom: to assure everyone the effective freedom to copy and redistribute it, with or without modifying it, either commercially or noncommercially. Secondarily, this License preserves for the author and publisher a way to get credit for their work, while not being considered responsible for modifications made by others.

   This License is a kind of "copyleft", which means that derivative works of the document must themselves be free in the same sense. It complements the GNU General Public License, which is a copyleft license designed for free software.

   We have designed this License in order to use it for manuals for free software, because free software needs free documentation: a free program should come with manuals providing the same freedoms that the software does. But this License is not limited to software manuals; it can be used for any textual work, regardless of subject matter or whether it is published as a printed book. We recommend this License principally for works whose purpose is instruction or reference.

1. APPLICABILITY AND DEFINITIONS

   This License applies to any manual or other work that contains a notice placed by the copyright holder saying it can be distributed under the terms of this License. The "Document", below, refers to any such manual or work. Any member of the public is a licensee, and is addressed as "you".

   A "Modified Version" of the Document means any work containing the Document or a portion of it, either copied verbatim, or with modifications and/or translated into another language.

   A "Secondary Section" is a named appendix or a front-matter section of the Document that deals exclusively with the relationship of the publishers or authors of the Document to the Document's overall subject (or to related matters) and contains nothing that could fall directly within that overall subject. (For example, if the Document is in part a textbook of mathematics, a Secondary Section may not explain any mathematics.) The relationship could be a matter of historical connection with the subject or with related matters, or of legal, commercial, philosophical, ethical or political position regarding them.

   The "Invariant Sections" are certain Secondary Sections whose titles are designated, as being those of Invariant Sections, in the notice that says that the Document is released under this License.

   The "Cover Texts" are certain short passages of text that are listed, as Front-Cover Texts or Back-Cover Texts, in the notice that says that the Document is released under this License.

A “Transparent” copy of the Document means a machine-readable copy, represented in a format whose specification is available to the general public, whose contents can be viewed and edited directly and straightforwardly with generic text editors or (for images composed of pixels) generic paint programs or (for drawings) some widely available drawing editor, and that is suitable for input to text formatters or for automatic translation to a variety of formats suitable for input to text formatters. A copy made in an otherwise Transparent file format whose markup has been designed to thwart or discourage subsequent modification by readers is not Transparent. A copy that is not “Transparent” is called “Opaque”.

Examples of suitable formats for Transparent copies include plain ASCII without markup, Texinfo input format, LaTeX input format, SGML or XML using a publicly available DTD, and standard-conforming simple HTML designed for human modification. Opaque formats include PostScript, PDF, proprietary formats that can be read and edited only by proprietary word processors, SGML or XML for which the DTD and/or processing tools are not generally available, and the machine-generated HTML produced by some word processors for output purposes only.

The “Title Page” means, for a printed book, the title page itself, plus such following pages as are needed to hold, legibly, the material this License requires to appear in the title page. For works in formats which do not have any title page as such, “Title Page” means the text near the most prominent appearance of the work’s title, preceding the beginning of the body of the text.

2. VERBATIM COPYING

   You may copy and distribute the Document in any medium, either commercially or noncommercially, provided that this License, the copyright notices, and the license notice saying this License applies to the Document are reproduced in all copies, and that you add no other conditions whatsoever to those of this License. You may not use technical measures to obstruct or control the reading or further copying of the copies you make or distribute. However, you may accept compensation in exchange for copies. If you distribute a large enough number of copies you must also follow the conditions in section 3.

   You may also lend copies, under the same conditions stated above, and you may publicly display copies.

3. COPYING IN QUANTITY

   If you publish printed copies of the Document numbering more than 100, and the Document’s license notice requires Cover Texts, you must enclose the copies in covers that carry, clearly and legibly, all these Cover Texts: Front-Cover Texts on the front cover, and Back-Cover Texts on the back cover. Both covers must also clearly and legibly identify you as the publisher of these copies. The front cover must present the full title with all words of the title equally prominent and visible. You may add other material on the covers in addition. Copying with changes limited to the covers, as long as they preserve the title of the Document and satisfy these conditions, can be treated as verbatim copying in other respects.

   If the required texts for either cover are too voluminous to fit legibly, you should put the first ones listed (as many as fit reasonably) on the actual cover, and continue the rest onto adjacent pages.

   If you publish or distribute Opaque copies of the Document numbering more than 100, you must either include a machine-readable Transparent copy along with each Opaque copy, or state in or with each Opaque copy a publicly-accessible computer-network location containing a complete Transparent copy of the Document, free of added material, which the general network-using public has access to download anonymously at no charge using public-standard network protocols. If you use the latter option, you must take reasonably prudent steps, when you begin distribution of Opaque copies in quantity, to ensure that this Transparent copy will remain thus accessible at the stated location until at least one year after the last time you distribute an Opaque copy (directly or through your agents or retailers) of that edition to the public.

It is requested, but not required, that you contact the authors of the Document well before redistributing any large number of copies, to give them a chance to provide you with an updated version of the Document.

4. MODIFICATIONS

You may copy and distribute a Modified Version of the Document under the conditions of sections 2 and 3 above, provided that you release the Modified Version under precisely this License, with the Modified Version filling the role of the Document, thus licensing distribution and modification of the Modified Version to whoever possesses a copy of it. In addition, you must do these things in the Modified Version:

A. Use in the Title Page (and on the covers, if any) a title distinct from that of the Document, and from those of previous versions (which should, if there were any, be listed in the History section of the Document). You may use the same title as a previous version if the original publisher of that version gives permission.

B. List on the Title Page, as authors, one or more persons or entities responsible for authorship of the modifications in the Modified Version, together with at least five of the principal authors of the Document (all of its principal authors, if it has less than five).

C. State on the Title page the name of the publisher of the Modified Version, as the publisher.

D. Preserve all the copyright notices of the Document.

E. Add an appropriate copyright notice for your modifications adjacent to the other copyright notices.

F. Include, immediately after the copyright notices, a license notice giving the public permission to use the Modified Version under the terms of this License, in the form shown in the Addendum below.

G. Preserve in that license notice the full lists of Invariant Sections and required Cover Texts given in the Document's license notice.

H. Include an unaltered copy of this License.

I. Preserve the section entitled "History", and its title, and add to it an item stating at least the title, year, new authors, and publisher of the Modified Version as given on the Title Page. If there is no section entitled "History" in the Document, create one stating the title, year, authors, and publisher of the Document as given on its Title Page, then add an item describing the Modified Version as stated in the previous sentence.

J. Preserve the network location, if any, given in the Document for public access to a Transparent copy of the Document, and likewise the network locations given in the Document for previous versions it was based on. These may be placed in the "History" section. You may omit a network location for a work that was published at least four years before the Document itself, or if the original publisher of the version it refers to gives permission.

K. In any section entitled "Acknowledgments" or "Dedications", preserve the section's title, and preserve in the section all the substance and tone of each of the contributor acknowledgments and/or dedications given therein.

L. Preserve all the Invariant Sections of the Document, unaltered in their text and in their titles. Section numbers or the equivalent are not considered part of the section titles.

M. Delete any section entitled "Endorsements". Such a section may not be included in the Modified Version.

N. Do not retitle any existing section as "Endorsements" or to conflict in title with any Invariant Section.

If the Modified Version includes new front-matter sections or appendices that qualify as Secondary Sections and contain no material copied from the Document, you may at your option designate some or all of these sections as invariant. To do this, add their titles to the list of Invariant Sections in the Modified Version's license notice. These titles must be distinct from any other section titles.

You may add a section entitled "Endorsements", provided it contains nothing but endorsements of your Modified Version by various parties—for example, statements of peer review or that the text has been approved by an organization as the authoritative definition of a standard.

You may add a passage of up to five words as a Front-Cover Text, and a passage of up to 25 words as a Back-Cover Text, to the end of the list of Cover Texts in the Modified Version. Only one passage of Front-Cover Text and one of Back-Cover Text may be added by (or through arrangements made by) any one entity. If the Document already includes a cover text for the same cover, previously added by you or by arrangement made by the same entity you are acting on behalf of, you may not add another; but you may replace the old one, on explicit permission from the previous publisher that added the old one.

The author(s) and publisher(s) of the Document do not by this License give permission to use their names for publicity for or to assert or imply endorsement of any Modified Version.

5. COMBINING DOCUMENTS

You may combine the Document with other documents released under this License, under the terms defined in section 4 above for modified versions, provided that you include in the combination all of the Invariant Sections of all of the original documents, unmodified, and list them all as Invariant Sections of your combined work in its license notice.

The combined work need only contain one copy of this License, and multiple identical Invariant Sections may be replaced with a single copy. If there are multiple Invariant Sections with the same name but different contents, make the title of each such section unique by adding at the end of it, in parentheses, the name of the original author or publisher of that section if known, or else a unique number. Make the same adjustment to the section titles in the list of Invariant Sections in the license notice of the combined work.

In the combination, you must combine any sections entitled "History" in the various original documents, forming one section entitled "History"; likewise combine any sections entitled "Acknowledgments", and any sections entitled "Dedications". You must delete all sections entitled "Endorsements."

6. COLLECTIONS OF DOCUMENTS

   You may make a collection consisting of the Document and other documents released under this License, and replace the individual copies of this License in the various documents with a single copy that is included in the collection, provided that you follow the rules of this License for verbatim copying of each of the documents in all other respects.

   You may extract a single document from such a collection, and distribute it individually under this License, provided you insert a copy of this License into the extracted document, and follow this License in all other respects regarding verbatim copying of that document.

7. AGGREGATION WITH INDEPENDENT WORKS

   A compilation of the Document or its derivatives with other separate and independent documents or works, in or on a volume of a storage or distribution medium, does not as a whole count as a Modified Version of the Document, provided no compilation copyright is claimed for the compilation. Such a compilation is called an "aggregate", and this License does not apply to the other self-contained works thus compiled with the Document, on account of their being thus compiled, if they are not themselves derivative works of the Document.

   If the Cover Text requirement of section 3 is applicable to these copies of the Document, then if the Document is less than one quarter of the entire aggregate, the Document's Cover Texts may be placed on covers that surround only the Document within the aggregate. Otherwise they must appear on covers around the whole aggregate.

8. TRANSLATION

   Translation is considered a kind of modification, so you may distribute translations of the Document under the terms of section 4. Replacing Invariant Sections with translations requires special permission from their copyright holders, but you may include translations of some or all Invariant Sections in addition to the original versions of these Invariant Sections. You may include a translation of this License provided that you also include the original English version of this License. In case of a disagreement between the translation and the original English version of this License, the original English version will prevail.

9. TERMINATION

   You may not copy, modify, sublicense, or distribute the Document except as expressly provided for under this License. Any other attempt to copy, modify, sublicense or distribute the Document is void, and will automatically terminate your rights under this License. However, parties who have received copies, or rights, from you under this License will not have their licenses terminated so long as such parties remain in full compliance.

10. FUTURE REVISIONS OF THIS LICENSE

    The Free Software Foundation may publish new, revised versions of the GNU Free Documentation License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns. See http://www.gnu.org/copyleft/.

    Each version of the License is given a distinguishing version number. If the Document specifies that a particular numbered version of this License "or any later version" applies to it, you have the option of following the terms and conditions either of that specified version or of any later version that has been published (not as a draft) by the Free Software Foundation. If the Document does not specify a version number of this License, you may choose any version ever published (not as a draft) by the Free Software Foundation.

ADDENDUM: How to use this License for your documents

To use this License in a document you have written, include a copy of the License in the document and put the following copyright and license notices just after the title page:

```
Copyright (C)  year  your name.
Permission is granted to copy, distribute and/or modify this document
under the terms of the GNU Free Documentation License, Version 1.1
or any later version published by the Free Software Foundation;
with the Invariant Sections being list their titles, with the
Front-Cover Texts being list, and with the Back-Cover Texts being list.
A copy of the license is included in the section entitled ``GNU
Free Documentation License''.
```

If you have no Invariant Sections, write "with no Invariant Sections" instead of saying which ones are invariant. If you have no Front-Cover Texts, write "no Front-Cover Texts" instead of "Front-Cover Texts being *list*"; likewise for Back-Cover Texts.

If your document contains nontrivial examples of program code, we recommend releasing these examples in parallel under your choice of free software license, such as the GNU General Public License, to permit their use in free software.

Copying GNU Crypto

GNU General Public License

Version 2, June 1991

Preamble

The licenses for most software are designed to take away your freedom to share and change it. By contrast, the GNU General Public License is intended to guarantee your freedom to share and change free software—to make sure the software is free for all its users. This General Public License applies to most of the Free Software Foundation's software and to any other program whose authors commit to using it. (Some other Free Software Foundation software is covered by the GNU Library General Public License instead.) You can apply it to your programs, too.

When we speak of free software, we are referring to freedom, not price. Our General Public Licenses are designed to make sure that you have the freedom to distribute copies of free software (and charge for this service if you wish), that you receive source code or can get it if you want it, that you can change the software or use pieces of it in new free programs; and that you know you can do these things.

To protect your rights, we need to make restrictions that forbid anyone to deny you these rights or to ask you to surrender the rights. These restrictions translate to certain responsibilities for you if you distribute copies of the software, or if you modify it.

For example, if you distribute copies of such a program, whether gratis or for a fee, you must give the recipients all the rights that you have. You must make sure that they, too, receive or can get the source code. And you must show them these terms so they know their rights.

We protect your rights with two steps: (1) copyright the software, and (2) offer you this license which gives you legal permission to copy, distribute and/or modify the software.

Also, for each author's protection and ours, we want to make certain that everyone understands that there is no warranty for this free software. If the software is modified by someone else and passed on, we want its recipients to know that what they have is not the original, so that any problems introduced by others will not reflect on the original authors' reputations.

Finally, any free program is threatened constantly by software patents. We wish to avoid the danger that redistributors of a free program will individually obtain patent licenses, in effect making the program proprietary. To prevent this, we have made it clear that any patent must be licensed for everyone's free use or not licensed at all.

The precise terms and conditions for copying, distribution and modification follow.

TERMS AND CONDITIONS FOR COPYING, DISTRIBUTION AND MODIFICATION

0. This License applies to any program or other work which contains a notice placed by the copyright holder saying it may be distributed under the terms of this General Public License. The "Program", below, refers to any such program or work, and a "work based on the Program" means either the Program or any derivative work under copyright law: that is to say, a work containing the Program or a portion of it, either verbatim or with modifications and/or translated into another language. (Hereinafter, translation is included without limitation in the term "modification".) Each licensee is addressed as "you".

   Activities other than copying, distribution and modification are not covered by this License; they are outside its scope. The act of running the Program is not restricted, and the output from the Program is covered only if its contents constitute a work based on the Program (independent of having been made by running the Program). Whether that is true depends on what the Program does.

1. You may copy and distribute verbatim copies of the Program's source code as you receive it, in any medium, provided that you conspicuously and appropriately publish on each copy an appropriate copyright notice and disclaimer of warranty; keep intact all the notices that refer to this License and to the absence of any warranty; and give any other recipients of the Program a copy of this License along with the Program.

   You may charge a fee for the physical act of transferring a copy, and you may at your option offer warranty protection in exchange for a fee.

2. You may modify your copy or copies of the Program or any portion of it, thus forming a work based on the Program, and copy and distribute such modifications or work under the terms of Section 1 above, provided that you also meet all of these conditions:

   a. You must cause the modified files to carry prominent notices stating that you changed the files and the date of any change.

   b. You must cause any work that you distribute or publish, that in whole or in part contains or is derived from the Program or any part thereof, to be licensed as a whole at no charge to all third parties under the terms of this License.

   c. If the modified program normally reads commands interactively when run, you must cause it, when started running for such interactive use in the most ordinary way, to print or display an announcement including an appropriate copyright notice and a notice that there is no warranty (or else, saying that you provide a warranty) and that users may redistribute the program under these conditions, and telling the user how to view a copy of this License. (Exception: if the Program itself is interactive but does not normally print such an announcement, your work based on the Program is not required to print an announcement.)

   These requirements apply to the modified work as a whole. If identifiable sections of that work are not derived from the Program, and can be reasonably considered independent and separate works in themselves, then this License, and its terms, do not apply to those sections when you distribute them as separate works. But when you distribute the same sections as part of a whole which is a work based on the Program, the distribution of the whole must be on the terms of this License, whose permissions for other licensees extend to the entire whole, and thus to each and every part regardless of who wrote it.

Thus, it is not the intent of this section to claim rights or contest your rights to work written entirely by you; rather, the intent is to exercise the right to control the distribution of derivative or collective works based on the Program.

In addition, mere aggregation of another work not based on the Program with the Program (or with a work based on the Program) on a volume of a storage or distribution medium does not bring the other work under the scope of this License.

3. You may copy and distribute the Program (or a work based on it, under Section 2) in object code or executable form under the terms of Sections 1 and 2 above provided that you also do one of the following:

   a. Accompany it with the complete corresponding machine-readable source code, which must be distributed under the terms of Sections 1 and 2 above on a medium customarily used for software interchange; or,

   b. Accompany it with a written offer, valid for at least three years, to give any third party, for a charge no more than your cost of physically performing source distribution, a complete machine-readable copy of the corresponding source code, to be distributed under the terms of Sections 1 and 2 above on a medium customarily used for software interchange; or,

   c. Accompany it with the information you received as to the offer to distribute corresponding source code. (This alternative is allowed only for noncommercial distribution and only if you received the program in object code or executable form with such an offer, in accord with Subsection b above.)

   The source code for a work means the preferred form of the work for making modifications to it. For an executable work, complete source code means all the source code for all modules it contains, plus any associated interface definition files, plus the scripts used to control compilation and installation of the executable. However, as a special exception, the source code distributed need not include anything that is normally distributed (in either source or binary form) with the major components (compiler, kernel, and so on) of the operating system on which the executable runs, unless that component itself accompanies the executable.

   If distribution of executable or object code is made by offering access to copy from a designated place, then offering equivalent access to copy the source code from the same place counts as distribution of the source code, even though third parties are not compelled to copy the source along with the object code.

4. You may not copy, modify, sublicense, or distribute the Program except as expressly provided under this License. Any attempt otherwise to copy, modify, sublicense or distribute the Program is void, and will automatically terminate your rights under this License. However, parties who have received copies, or rights, from you under this License will not have their licenses terminated so long as such parties remain in full compliance.

5. You are not required to accept this License, since you have not signed it. However, nothing else grants you permission to modify or distribute the Program or its derivative works. These actions are prohibited by law if you do not accept this License. Therefore, by modifying or distributing the Program (or any work based on the Program), you indicate your acceptance of this License to do so, and all its terms and conditions for copying, distributing or modifying the Program or works based on it.

6. Each time you redistribute the Program (or any work based on the Program), the recipient automatically receives a license from the original licensor to copy, distribute or modify the Program subject to these terms and conditions. You may not impose any further restrictions on the recipients' exercise of the rights granted herein. You are not responsible for enforcing compliance by third parties to this License.
7. If, as a consequence of a court judgment or allegation of patent infringement or for any other reason (not limited to patent issues), conditions are imposed on you (whether by court order, agreement or otherwise) that contradict the conditions of this License, they do not excuse you from the conditions of this License. If you cannot distribute so as to satisfy simultaneously your obligations under this License and any other pertinent obligations, then as a consequence you may not distribute the Program at all. For example, if a patent license would not permit royalty-free redistribution of the Program by all those who receive copies directly or indirectly through you, then the only way you could satisfy both it and this License would be to refrain entirely from distribution of the Program.

   If any portion of this section is held invalid or unenforceable under any particular circumstance, the balance of the section is intended to apply and the section as a whole is intended to apply in other circumstances.

   It is not the purpose of this section to induce you to infringe any patents or other property right claims or to contest validity of any such claims; this section has the sole purpose of protecting the integrity of the free software distribution system, which is implemented by public license practices. Many people have made generous contributions to the wide range of software distributed through that system in reliance on consistent application of that system; it is up to the author/donor to decide if he or she is willing to distribute software through any other system and a licensee cannot impose that choice.

   This section is intended to make thoroughly clear what is believed to be a consequence of the rest of this License.
8. If the distribution and/or use of the Program is restricted in certain countries either by patents or by copyrighted interfaces, the original copyright holder who places the Program under this License may add an explicit geographical distribution limitation excluding those countries, so that distribution is permitted only in or among countries not thus excluded. In such case, this License incorporates the limitation as if written in the body of this License.
9. The Free Software Foundation may publish revised and/or new versions of the General Public License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns.

   Each version is given a distinguishing version number. If the Program specifies a version number of this License which applies to it and "any later version", you have the option of following the terms and conditions either of that version or of any later version published by the Free Software Foundation. If the Program does not specify a version number of this License, you may choose any version ever published by the Free Software Foundation.
10. If you wish to incorporate parts of the Program into other free programs whose distribution conditions are different, write to the author to ask for permission. For software which is copyrighted by the Free Software Foundation, write to the Free Software Foundation; we sometimes make exceptions for this. Our decision will be guided by the two

goals of preserving the free status of all derivatives of our free software and of promoting the sharing and reuse of software generally.

NO WARRANTY

11. BECAUSE THE PROGRAM IS LICENSED FREE OF CHARGE, THERE IS NO WARRANTY FOR THE PROGRAM, TO THE EXTENT PERMITTED BY APPLICABLE LAW. EXCEPT WHEN OTHERWISE STATED IN WRITING THE COPYRIGHT HOLDERS AND/OR OTHER PARTIES PROVIDE THE PROGRAM "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. THE ENTIRE RISK AS TO THE QUALITY AND PERFORMANCE OF THE PROGRAM IS WITH YOU. SHOULD THE PROGRAM PROVE DEFECTIVE, YOU ASSUME THE COST OF ALL NECESSARY SERVICING, REPAIR OR CORRECTION.
12. IN NO EVENT UNLESS REQUIRED BY APPLICABLE LAW OR AGREED TO IN WRITING WILL ANY COPYRIGHT HOLDER, OR ANY OTHER PARTY WHO MAY MODIFY AND/OR REDISTRIBUTE THE PROGRAM AS PERMITTED ABOVE, BE LIABLE TO YOU FOR DAMAGES, INCLUDING ANY GENERAL, SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE USE OR INABILITY TO USE THE PROGRAM (INCLUDING BUT NOT LIMITED TO LOSS OF DATA OR DATA BEING RENDERED INACCURATE OR LOSSES SUSTAINED BY YOU OR THIRD PARTIES OR A FAILURE OF THE PROGRAM TO OPERATE WITH ANY OTHER PROGRAMS), EVEN IF SUCH HOLDER OR OTHER PARTY HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

END OF TERMS AND CONDITIONS

Appendix: How to Apply These Terms to Your New Programs

If you develop a new program, and you want it to be of the greatest possible use to the public, the best way to achieve this is to make it free software which everyone can redistribute and change under these terms.

To do so, attach the following notices to the program. It is safest to attach them to the start of each source file to most effectively convey the exclusion of warranty; and each file should have at least the "copyright" line and a pointer to where the full notice is found.

```
one line to give the program's name and a brief idea of what it does.
Copyright (C) yyyy  name of author

This program is free software; you can redistribute it and/or modify
it under the terms of the GNU General Public License as published by
the Free Software Foundation; either version 2 of the License, or
(at your option) any later version.

This program is distributed in the hope that it will be useful,
but WITHOUT ANY WARRANTY; without even the implied warranty of
MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
GNU General Public License for more details.

You should have received a copy of the GNU General Public License
along with this program; if not, write to the

   Free Software Foundation, Inc.
   59 Temple Place - Suite 330
   Boston, MA  02111-1307
   USA
```

Also add information on how to contact you by electronic and paper mail.

If the program is interactive, make it output a short notice like this when it starts in an interactive mode:

```
Gnomovision version 69, Copyright (C) 19yy name of author
Gnomovision comes with ABSOLUTELY NO WARRANTY; for details type
'show w'. This is free software, and you are welcome to redistribute
it under certain conditions; type 'show c' for details.
```

The hypothetical commands '`show w`' and '`show c`' should show the appropriate parts of the General Public License. Of course, the commands you use may be called something other than '`show w`' and '`show c`'; they could even be mouse-clicks or menu items—whatever suits your program.

You should also get your employer (if you work as a programmer) or your school, if any, to sign a "copyright disclaimer" for the program, if necessary. Here is a sample; alter the names:

```
Yoyodyne, Inc., hereby disclaims all copyright interest in the program
‘Gnomovision’ (which makes passes at compilers) written by James Hacker.

signature of Ty Coon, 1 April 1989
Ty Coon, President of Vice
```

This General Public License does not permit incorporating your program into proprietary programs. If your program is a subroutine library, you may consider it more useful to permit linking proprietary applications with the library. If this is what you want to do, use the GNU Library General Public License instead of this License.

Acknowledgements

Many people contribute to the GNU Crypto project, and in many different capacities. Any omission to this list is accidental. Feel free to contact raif@fl.net.au if you have been left out.

Barreto, Paulo S. L. M.	paulo.barreto@terra.com.br
Ferrier, Nic	nferrier@tapsellferrier.co.uk
Kmett, Edward	ekmett@cxss.com
Koch, Werner	wk@gnupg.org
Louchart-Fletcher, Olivier	olivier@zipworld.com.au
Marshall, Casey	rsdio@metastatic.org
Naffah, Raif S.	raif@fl.net.au
Osvik, Dag Arne	osvik@ii.uib.no
Selensminde, Gisle	gselens@broadpark.no
Wielaard, Mark	mark@klomp.org
Wu, Thomas J.	tom@arcot.com

Figure Index

A

C

K

M

P

S

Index

H

I

K

L

M

N

O

P

R

S

T

U

V

W

References

[Kal92] Burton Kaliski, *The MD2 Message-Digest Algorithm*, RFC 1319.
See `http://www.ietf.org/rfc/rfc1319.txt`.

[Kro00] Ted Krovetz, John Black, Shai Halevi, Alejandro Hevia, Hugo Krawczyk, and Phillip Rogaway, *UMAC: Message Authentication Code using Universal Hashing*, Internet-Draft, October 2000.
See `http://www.cs.ucdavis.edu/~rogaway/umac/draft-krovetz-umac-01.txt`.

[McG02] David A. McGrew, *The Truncated Multi-Modular Hash Function (TMMH), Version Two*, Internet-Draft, October 2002.
See `http://www.ietf.org/internet-drafts/draft-irtf-cfrg-tmmh-00.txt`.

[MOV96] Alfred J. Menezes, Paul C. Van Oorschot, Scott A. Vanstone (Editor); *Handbook of Applied Cryptography* (1992 CRC Press); ISBN 0849385237.

[NIST95] Federal Information Processing Standards Publication 180-1: Secure Hash Standard. 17 April 1995, National Institute for Standards and Technology.
See `http://www.itl.nist.gov/fipspubs/fip180-1.htm`.

[NIST01] Federal Information Processing Standards Publication 197: Advanced Encryption Standard (AES). 26 November 2001, National Institute for Standards and Technology.
See `http://csrc.nist.gov/publications/fips/fips197/fips-197.pdf`.

[RFC2631]
Eric Rescorla. *Diffie-Hellman Key Agreement Method.*
See `http://www.ietf.org/rfc/rfc2631.txt`.

[Riv92a] Ron Rivest, *The MD4 Message-Digest Algorithm*, RFC 1320.
See `http://www.ietf.org/rfc/rfc1320.txt`.

[Riv92b] Ron Rivest, *The MD5 Message-Digest Algorithm*, RFC 1321.
See `http://www.ietf.org/rfc/rfc1321.txt`.

[Sch95] Bruce Schneier, *Applied Cryptography: Protocols, Algorithms, and Source Code in C, Second Edition* (1995 John Wiley & Sons); ISBN 0471117099.

[Wu02] Thomas J. Wu, *SRP-6: Improvements and Refinements to the Secure Remote Password Protocol* (29 October 2002).
See `http://srp.stanford.edu/srp6.ps`.