Botan Tutorial

Jack Lloyd
lloyd@randombit.net

2010/08/07

Contents

1 Introduction

This document essentially sets up various simple scenarios and then shows how to solve the problems using botan. It's fairly simple, and doesn't cover many of the available APIs and algorithms, especially the more obscure or unusual ones. It is a supplement to the API documentation and the example applications, which are included in the distribution.

2 Initializing the Library

The first step to using botan is to create a `LibraryInitializer` object, which handles creating various internal structures, and also destroying them at shutdown.

```
#include <botan/botan.h>

int main()
   {
   Botan::LibraryInitializer init;
   return 0;
   }
```

If your application is multi-threaded, you need to tell botan this so that it will use locking where necessary. This is done by passing a string to the constructor of `LibraryInitializer`:

```
 Botan::LibraryInitializer init("thread_safe=yes");
```

3 Introduction to Pipe

Most operations in botan are specified in terms of transformations on streams. The class that handles the I/O and management for these streams is called `Pipe`. You can construct a `Pipe` with one or more `Filter`s, which sequentially process messages. You can only update a single message at a time, but you can leave the final output contents in a `Pipe` and read them out as desired.

Here is how you might hex encode two messages:

```
std::string message1 = "this is the first message";
const byte message2[] = "a second message";
Pipe pipe(new Hex_Encoder);

pipe.start_msg(); // must be called before writing to the pipe
pipe.write(message1);
pipe.end_msg(); // must be called to signal completion

/*
process_msg(x) is equivalent to calling
   start_msg(); write(x); end_msg();
*/
pipe.process_msg(message2);

Pipe::message_id n = pipe.message_count(); // returns 2

/* you can read a message as a string, here we read message 0 */
```

```
std::string first_result = pipe.read_all_as_string(0);

/* or a piece at a time using array/length, now we'll read the
   second message (message id 1)
*/

byte output[4096] = { 0 };
u32bit got = read(output, sizeof(output), 1);
if(got >= sizeof(output))
  // have to read again to get more of the message
```

You can also read output while the message is still active (before the call to **end_msg**), using the same interfaces. You can find out how much data is currently available for a particular `Pipe` by calling the member function **remaining**, which takes a message sequence number and returns the number of bytes that are currently available to read from that message.

4 Hashing a File

Hashing a file is done using a `Hash_Filter`, which takes a string which specifies which hash function you want to use:

```
Pipe pipe(new Hash_Filter("SHA-256"));
```

The output of a `Hash_Filter` is raw binary. The filter will not produce any output at all until you call **end_msg**.

5 Symmetric Cryptography

5.1 Authentication

5.2 User Authentication

6 Public Key Cryptography