Pipelining

From Crypto++ Wiki

Pipelining is a paradigm in Crypto++ which allows data to flow from a source to a sink. As data flows, the data will encounter filters which transforms the data in some way in route to the sink. The motivation for the design was the Unix pipe system, so a Crypto++ pipeline has very close analogies to a Unix command line.

For example, consider a set of commands to Base64 encode a file and then saves the encoding to a second file:

```
cat filename | base64 > filename.b64
```

The Crypto++ pipeline to accomplish the same would be:

```
FileSource f(filename, new Base64Encoder(new FileSink(filename + ".b64")));
```

Generalizing the pipeline would be:

```
Source(source, new Filter(new Filter(new Sink(destination))));
```

The Annotated Class Reference (http://www.cryptopp.com/docs/ref/annotated.html) includes documentation for the Source interface (http://www.cryptopp.com/docs/ref/class_source.html) and Sink interface (http://www.cryptopp.com/docs/ref/class_sink.html) .

All filters inherit from `BufferedTransformation`. With respect to BufferedTransformations, intermediate objects of interest are:

- BlockTransformations
- StreamTransformations
- HashTransformations

With respect to Filters, objects of interest are:

- AuthenticatedEncryptionFilter and AuthenticatedDecryptionFilter
- StreamTransformationFilters
- HashFilters
- HashVerificationFilters
- SignerFilters
- SignatureVerificationFilters

Generally, a user defined filter will derive from `class Filter`. Wei Dai recommends examining `SignerFilter` in filters.h for a filter example.

Ownership

An important detail which needs to be examined is object ownership in a pipeline. According to "Important Usage Notes" in ReadMe.txt:

> If a constructor for A takes a pointer to an object B (except primitive types such as int and char), then A owns B and will delete B at A's destruction. If a constructor for A takes a reference to an object B, then the caller retains ownership of B and should not destroy it until A no longer needs it.

That means filters created with `new` are owned by the outer or encompassing object and will be destroyed by that object when no longer needed.

Sample Program

The following program transfers data from the first string to the second string. Though not very useful, it is the simplest demonstration of **pipelining**. An important reminder (from the **ReadMe**) is that the `StringSource`, which takes a pointer to the `StringSink`, owns the `StringSink`. So the `StringSource` will delete the `StringSink` when the `StringSource` destructor is invoked.

```
string s1 = "Pipeline", s2;

StringSource ss( s1, new StringSink( s2 ) );

cout << "s1: " << s1 << endl;
cout << "s2: " << s2 << endl;
```

The following program uses an `AutoSeededRandomPool` to generate an AES key. The key is hex encoded and then printed.

```
AutoSeededRandomPool prng;

SecByteBlock key(AES::DEFAULT_KEYLENGTH);
prng.GenerateBlock( key.data(), key.size() );

string encoded;

StringSource ss( key.data(), key.size(), true,
    new HexEncoder(
        new StringSink( encoded )
    ) // HexEncoder
); // StringSource

cout << "key: " << encoded << endl;
```

At times, a result is required from an intermediate object that is participating in a pipeline. Most notably is the `DecodingResult` from a `HashVerificationFilter`. In this situation, we **do not** want the filter to own the object and attempt to destroy it. To accomplish the goal, a `Redirector` would be used. Notice that the `Redirector` takes a reference to an object, and not a pointer to an object.

```
CCM< AES, TAG_SIZE >::Decryption d;
d.SetKeyWithIV( key, key.size(), iv, sizeof(iv) );
...

AuthenticatedDecryptionFilter df( d,
```

```
    new StringSink( recovered )
); // AuthenticatedDecryptionFilter

// Cipher text includes the MAC tag
StringSource ss( cipher, true,
    new Redirector( df )
); // StringSource

// If the object does not throw, here's the only
//  opportunity to check the data's integrity
bool b = df.GetLastResult();

if( true == b ) {
    cout << recovered << endl;
}
```


Categories: Sample | Source | Filter | Sink