# *flow* Documentation (experimental)

Manfred Morgner

14. Dezember 2010

# Inhaltsverzeichnis

**Data must *flow*!**

# Kapitel 1

# *flow*

*flow* will move your data in protected mode

## 1.1 What *flow* is

Flow is a client server application system which may move your data secure over insecure networks using untrusted service providers. The level of security you get out of *flow* is up to you! Meaning you have nearly unlimited control over your data while moving them.

YOUR data is data you own, which is data you send and data you receive. Receiving data may imply third entities. Other people for example, or data gathering devices.

As far as you'r in control of all the security credential at both end of a transportation chain, you are in control of your data. Other people, friends for example, will undermine your control over your data but only in limited ways.

### 1.1.1 What *flow* does

### 1.1.2 Why *flow* is developed

### 1.1.3 What *flow* is good for

## 1.2 Some unusual security basics

### 1.2.1 Two simple definitions

Assuming that data you send to others (autonom entities, not controlled by you) are not truly your data, we have to split the analyse in two parts:

YOUR data - meaning data over which you have full control and which will not change their ownership.

THEIR data - meaning data you share and therefore need to give some control in other hands.

### 1.2.2 Trustworthyness

Nearly no one knows about Software Certificates. Good willing people, business man, manager and lot of other people tried to send the message about Software Certificates (for our purpose simple 'Certificates') to the world, but the world was not listening. This is, because the mater is much too complicated.

Because of this, I will not try to do the next boring attempt to explain. If you'r not familiar with the matter and - as usual - not interested at all, forget the whole chapter about security. You don't need to care about it. Other ones care for you. (this was a joke!)

A key question in all matters of security is: Who is trustworthy?

The answer seems to be simple but it is not. You may take a most restrictive point of view and state, that the only person you are will to trust is you. But given the point of our discussion, an open network and (per definition) not trustworthy service providers building this network, your believe has to be wrong.

You'r only trustworthy against yourself if you understand the basics of data security. Even if you read this document, which means you'r interested in Data Security, probably you should not trust yourself - at least for the beginning.

### 1.2.3 To trust or not to trust

Should you trust yourself?

### 1.2.4 Identity in open networks

### 1.2.5 How to trust

### 1.2.6 Why should you care?

### 1.2.7 Where to go from here

# Kapitel 2

# The project

## 2.1 Concept

The concept of flow is, to move data securely through insecure networks using untrusted providers. To do this, *flow* has to fullfill three requirements

1. kick out the 'man in the middle'
2. ensure even the message server is unable know who is who
3. ensure only the receiver is able to read the data

The man in the middle is the easier part. He does not love Transport Layer Security - TLS, former SSL - and usually will focus on other means to break down your defences if he discovers such crud methods of securing your doings.

Ensuring that only the receiver will be able to understand your message is easy too, because asymetric data encryption is 'state of the art', even if no normal user understands how it works.

But how to provide the communication server with no information of who you are and whom to deliver the package without telling him who you are and whom to deliver the package?

Well, this is not only easy, but eases the rest of the boring transportation requirements too. This is, because you neither have to tell the server who your are nor who the receiver might be. A simple two step trick ensures this:

## 2.2 Secondary targets

### 2.2.1 Training

### 2.2.2 Education

### 2.2.3 Knowledge

## 2.3 Source Code

### 2.3.1 class Pulex

The Pulex class represents a neutral object containing client data to become moved to other clients using a server able to move Container objects. To be able to do so, a Container is able to move data wiht minimum amount of knowledge of what these data are. This works similar to a packet moved by a post office. The sender ist known, the receiver ist known but the content is not.

Regarding a Pulex, the sender ist mainly unknown, the receiver will become partially known if and when he calls is Container. The server will give the packet to every receiver who successfuly claims to be the receiver of it. To make things with a Pulex not as easy as with a Container, a Pulex sends the identity of its sender and its receiver in the form of a 40 character ascii representation of the fingerprint of their certificates.

The parent class to be used as 'inherited::methode()' for abstract usage of inherited methods. Mimimizes logical redundancy

```
    typedef CContainer inherited;
```

May be this is not a good idea at all, but enables compact code, which also may not be good idea. This operator appends data to the object as if it were a data sink which it is ;-)

```
            const std::string& operator << ( const std::string& rsData );
```

Friendly operators to pipe data to different types of stream

```
    friend std::ostream& operator << ( std::ostream&, CPulex& );
    friend      CSocket& operator << (      CSocket&, CPulex& );
```

A template to be used by output pipe operators to send the puley off

```
    template<typename T>
      T& Send( T& roStream );
```

```
#ifndef _PULEX_H
#define _PULEX_H

#include
#include

#include <list>
#include <iostream>


extern bool g_bVerbose;

class CPulex : public CContainer
  {
  private:
    typedef CContainer inherited;

    static const std::string s_sClassName;

  public:
             CPulex();
    virtual ~CPulex();

    virtual const std::string& ClassNameGet() const;

            const std::string& operator << ( const std::string& rsData );

    friend std::ostream& operator << ( std::ostream&, CPulex& );
    friend      CSocket& operator << (      CSocket&, CPulex& );
  protected:
    template<typename T>
      T& Send( T& roStream );

  protected:
    long ClientSideIDGet();

  }; // class CPulex


#endif // _PULEX_H
```

## 2.3.2 class Crypto

```
/***************************************************************************
 crypto.h  - description
 -----------------------
 begin                  : Thu Dec 02 2010
 copyright              : Copyright (C) 2010 by Manfred Morgner
 email                  : manfred@morgner.com
 ****************************************************************************/

#ifndef _CRYPTO_H
#define _CRYPTO_H

#include

#include <string>
#include <vector>

#include <openssl/rand.h> // RSA
#include <openssl/evp.h>  // EVP_MAX_KEY_LENGTH, ...


extern bool g_bVerbose;

#define RANDOM_BUFFER_SIZE 1024

typedef std::vector<unsigned char> CUCBuffer;

class CCrypto
  {
  protected:
    CUCBuffer m_oData;

    X509*     m_pX509;      // m_oEvpPkey frees this pointer
    CEvpPkey  m_oEvpPkey;
    CRsa      m_oRsa;

    const EVP_CIPHER* m_pfCipher;
    const EVP_MD*     m_pfDigest;

    unsigned char m_aucKey [EVP_MAX_KEY_LENGTH]; // 32
    unsigned char m_aucIv  [EVP_MAX_IV_LENGTH];  // 16
    unsigned char m_aucSalt[PKCS5_SALT_LEN];     //  8

    static const std::string s_sDelimiter;

             CCrypto(){}
  public:
             CCrypto( const std::string& rsInput );
    virtual ~CCrypto();

    void operator =    ( RSA* pRsa ) { m_oRsa = pRsa; }
         operator RSA* ()            { return (RSA*)m_oRsa; }

    void RsaKeyLoadPublic ( const std::string& rsFileRsaKey );
    void RsaKeyLoadPrivate( const std::string& rsFileRsaKey );
    void RsaKeyLoadFromCertificate( const std::string& rsFileCertificate );
    std::string       EncryptToBase64  ();
    const CUCBuffer& DecryptFromBase64( const std::string& rsBase64 );

  protected:
    void RandomSeed();
    void RsaKeyGenerate();

    void RandomGet( CUCBuffer& roBuffer );
    void RandomGet( unsigned char* pucBuffer, size_t nBufferSize );

    std::string       ConvertToBase64  ();
    const CUCBuffer& ConvertFromBase64( const std::string& rsBase64 );

    std::string ConvertToBase64( const unsigned char* pucData, size_t nSize );
    std::string EncryptToBase64( const unsigned char* pucData, size_t nSize );

    bool EncryptRsaPublic ( CUCBuffer& roBuffer );
    bool DecryptRsaPrivate( CUCBuffer& roBuffer );

    std::string SymetricKeyRsaPublicEncrypt();
    bool SymetricKeyRsaPrivateDecrypt( const std::string& rsEncrypted );

    int  SymetricKeyMake( const EVP_CIPHER* pfCipher = EVP_des_ede3_cfb(),
                          const EVP_MD*     pfDigest = EVP_sha1() );

  }; // class CCrypto

#endif // _CRYPTO_H
```

OpenSSL

GPL