USB Key Profile Manager for Mozilla

by

Yun Zhou

Advisor: Dr. Chris Pollett

Committee Members:
Dr. Melody Moh
Dr. Mark Stamp

Roadmap

1. Introduction
2. Background Information and Related Work
3. Design and Implementation
   - Event-driven structure
   - USB profile loader
   - User authentication
   - Bulk encryption
   - Sequence of operations
4. Performance
5. Conclusion and Future Work

“Cheat Sheet”

Model: XPCOM, Event-Driven

Mozilla Profile Manager

Profile Loader

- Linux mount table
- Mozilla registry

User Authentication

- Salted SHA-1 hash
- Mozilla’s *nsIHash*
- Mozilla window watcher
- Mozilla prompt service

Profile Encryption

- AES in CBC mode
- Mozilla’s AES implemetation
- Mozilla’s DOM window

Chapter I

Introduction

Introduction – Motivations

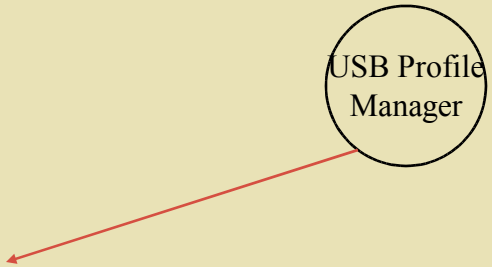
1. Mozilla is…
   - Large open-source software application.
   - Combines different advanced SW Dev concepts and design models (NSPR, XPCOM, DOM, XUL, etc.)
   - Provides a rich set of features (Layout engine, browser, mail/news, ChatZilla, network application and security protocols, etc.)
   - Provides customizable, modularized extensions.
2. Can save user profiles on USB keys.
3. But there are some weaknesses in the current profile manager.

Introduction – Goals

An XPCOM component will be implemented for managing user profiles stored on USB memory keys.

- Mobility.
- Security (user authentication and privacy protection).
- Easy to install.
- Without excessive performance overhead.

Mozilla's Architecture



Rapid Component Development with XPCOM

- Mozilla's own implementation of COM – Cross Platform Component Object Model.
- Cross-Platform
  XPCOM is Mozilla's component architecture and it provides a set of core libraries that facilitates software development cross different platforms.
- Modularity
  XPCOM structure helps to modularize the code, increase flexibility, scalability, maintainability and reusability of the code. The components can be implemented independently and installed separately according to the user's needs.

XPCOM - Interface

- An interface acts as a contractual agreement and it decides how other components can access the object that implements this interface.
- Each interface can be identified by its Contract ID.
- All interfaces are derived from a base interface *nsISupports* which tackles two major issues of interface-based programming: component lifetime and interface querying.
  - AddRef
  - Release
  - QueryInterface

Chapter II

Design and Implementation

Design and Implementation - Event Driven

- The USB Profile Manager acts upon events.

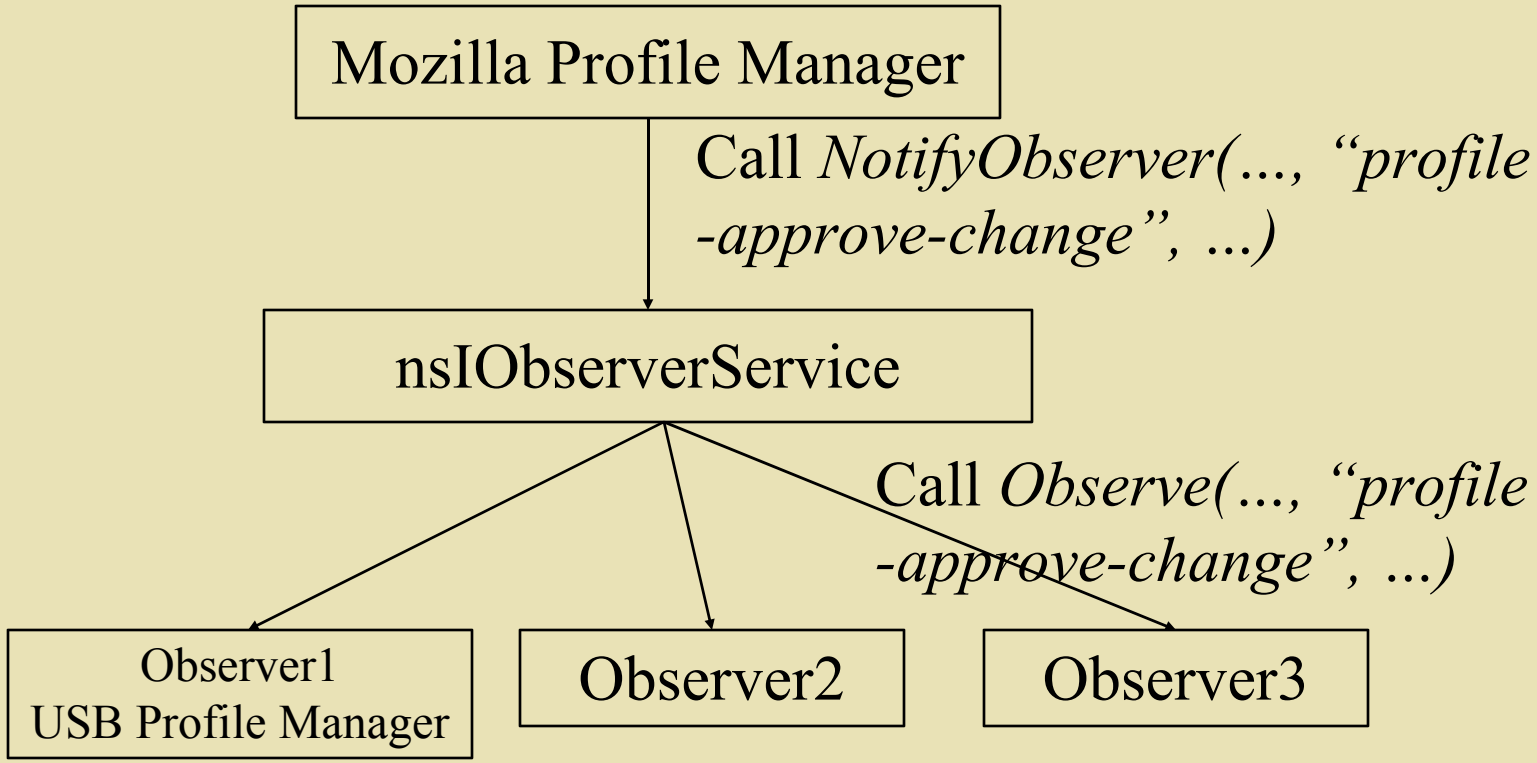
- At startup, it subscribes for the events of its interest, for example "*profile-approve-change*".

- When such event is generated, the component that initiates this event will notify all the subscribers by calling their *Observe()* function.

- Advantage: no change to the existing Mozilla code base. Loosely coupled the component. Better modularity.

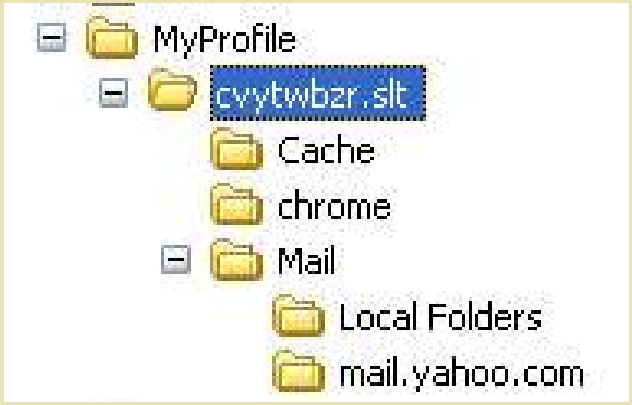
- Disadvantage: harder to implement. Hard to find appropriate events to subscribe for. Harder to control.

When the user switches between profiles…



Feature 1: USB Profile Loader

- Automatically detects mounted USB storage keys upon startup.
  Examine the mount table and search for a mounted element with the device name "*/dev/sda*".
  Call the *getmntent()* function to get the path name.
  On Linux, the device name "*/dev/sda*" stands for a mounted USB drive.
- Search for profiles.
- Load the profile to Mozilla's registry.
- Clean up at shutdown.



Feature 2: User Authentication

- User authentication is needed to prevent unauthorized users from gaining access to USB profiles. This protection is especially needed for portable devices such as USB keys.

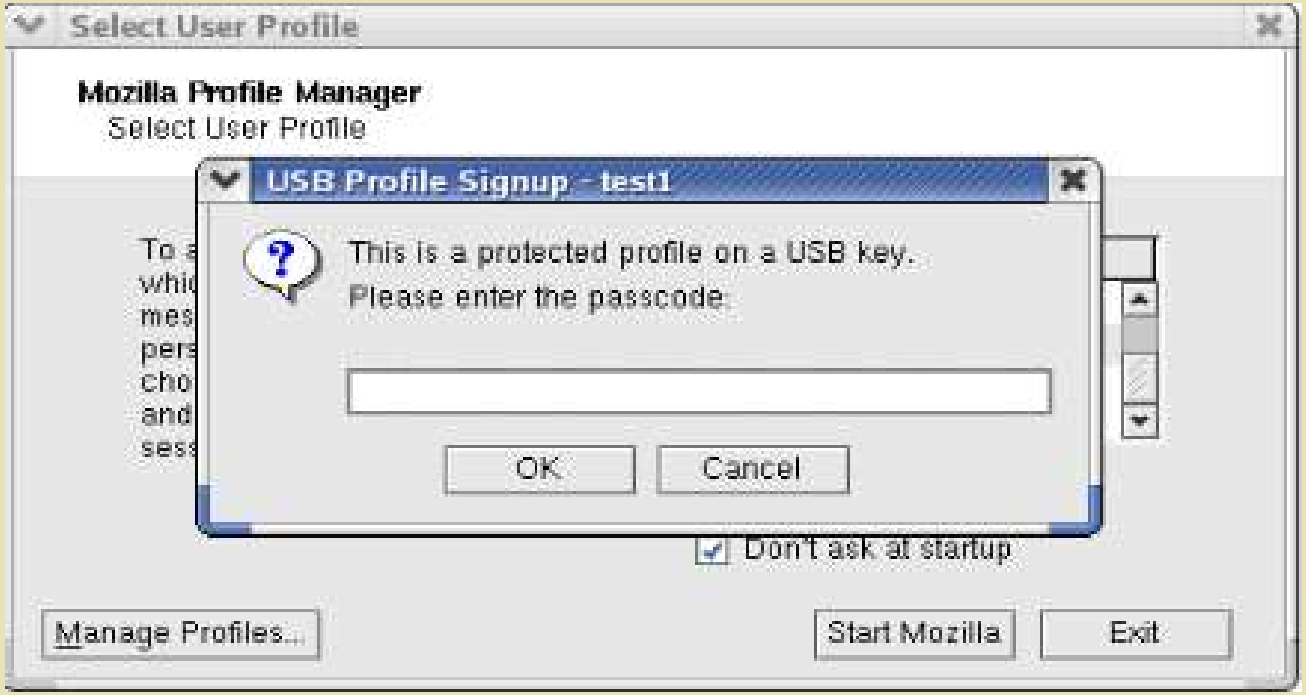
- But there is a tradeoff between usability and security. We let the user to decide.

User Authentication - UI

- Whenever the user starts an unprotected USB profile, she/he will be asked whether protection is needed.

User Authentication - UI

- When the user starts a protected USB profile, she/he is prompted for password.



- If unsuccessful, the user either has to select or create another profile, or stays with the current profile.

User Authentication – Salted SHA-1 Hash

- When the user enters a password for the first time, a SHA-1 hash is created for the password. Then a random integer, called “salt” is also generated and appended to the hash to generate another SHA-1 hash. Then the second hash is saved under the profile directory along with the salt.

- Original plan: use MD5 hash without salt.

SHA-1 Hash

- SHA-1 is a strong hash function, one-way, no collision has been found. The result has 20 bytes. The only possible attack is brute-force guessing of the plain text.
- SHA-1 works in stages. At each stage, a set of operations are applied to the previous message digest on the current block.

Dictionary Attack and Salted Hash

- The reason for appending a salt to the password before hashing is to make dictionary attack more time and resource consuming.
- Dictionary attack is a common attack on passwords. An attacker guesses all possible passwords, pre-computes their hashes and store them in a "dictionary". When the attacker captures a hash, she/he can compare the hash against the dictionary to see if any match exists. Only one round of computation is needed for all possible passwords, i.e., the dictionary is reusable.
- If the same attacker obtains a salted hash, he has to re-compute all the guessed passwords with the salt. And this calculation needs to be repeated for every different salt.

Generating the Hash

```
PRInt32 size = (PRInt32)strlen(thePassword);

nsresult rv;
nsCOMPtr<nsIHash> mDataHash = 0;

 // Initialize the crypto library. SHA1 hash has length 20
PRInt16 mHashType = nsIHash::HASH_AlgSHA1;
mDataHash = do_CreateInstance(NS_HASH_CONTRACTID, &rv);
if (NS_FAILED(rv)) return rv;

rv = mDataHash->Create(mHashType);
if (NS_FAILED(rv)) return rv;

rv = mDataHash->Begin();
```

- Create(mHashType): creates a computation context for a particular hash algorithm.
- Begin(): prepare for the hashing.

Generating the Hash (cont'd)

```
// Compute the hash...
int status;
if (mDataHash)
{
        mDataHash->Update(thePassword, size);
        status = PR_GetError();
        if (status < 0) return NS_ERROR_FAILURE;
}

PRUint32 hashlen;
mDataHash->ResultLen(mHashType, &hashlen);

(*computedHash) = (unsigned char *)PR_MALLOC(hashlen+1);
if (!(*computedHash)) return NS_ERROR_OUT_OF_MEMORY;


// Finish computing
mDataHash->End(*computedHash, &hashlen, hashlen);
status = PR_GetError();
if (status < 0) return NS_ERROR_FAILURE;
```

- Update(): updates the input and the hash length.
- End(): Finish hashing.

Feature 3: Encrypting the Profile with AES

- User authentication needs to go side-by-side with encryption.
- We use the AES (Advanced Encryption Standard) algorithm in CBC (Cipher Block Chaining) mode.
- AES is an iterated block cipher with a block size of 128 bits, allowing keys with variable-length, typically 128 bits, 192 bits and 256 bits.
- Depending upon the key length, AES runs through 10 to 14 rounds of encryption, each of which contains four operations :
  - –S-box;
  - –ShiftRow;
  - –MixColumn;
  - –MixAddRoundKey.

CBC Mode

Encryption	Decryption
C0 = E(IVxorP0,K)	P0 = IVxorD(C0,K)
C1 = E(C0xorP1,K)	P1 = C0xorD(C1,K)
C2 = E(C1xorP2,K)	P2 = C1xorD(C2,K)
…	…
Cn = E(C_{n-1}xorPn,K)	Pn = C_{n-1}xorD(Cn,K)

The biggest advantage of CBC is that the cipher texts are different even if the plain texts are the same, so it exposes much fewer hints to attackers than the ECB (Electronic Code Book) mode.

How to Use Mozilla's Rijndael Implementation

- Implemented in the NSS library
- *AES_CreateContext()*: Creates the context

```
struct AESContextStr
{
    unsigned int   Nb;
    unsigned int   Nr;
    PRUint32       *expandedKey;
    AESFunc        *worker;
    unsigned char iv[RIJNDAEL_MAX_BLOCKSIZE];
};
```

- *AES_Encrypt():* Executes the algorithm and returns the cipher text.
- *AES_DestroyContext():* destroys the context and frees up the space.

Encryption Steps

- Create the key and the IV (Initialization Vector) from the password and the salt.
- Pad the input as necessary.
- AES encryption.
- Encode the cipher text in base64 format.
- Performance improvement: only encrypt modified files.

When authentication fails…

- Two situations:
  1. At startup
  2. During switch

- During switch, we want to roll back the operation gracefully.

- Listen to the “profile-approve-change” event generated by the Mozilla profile manager before the switch takes place.

- Call VetoChange() of *nsIProfileChangeStatus*. No encryption or decryption will occur.

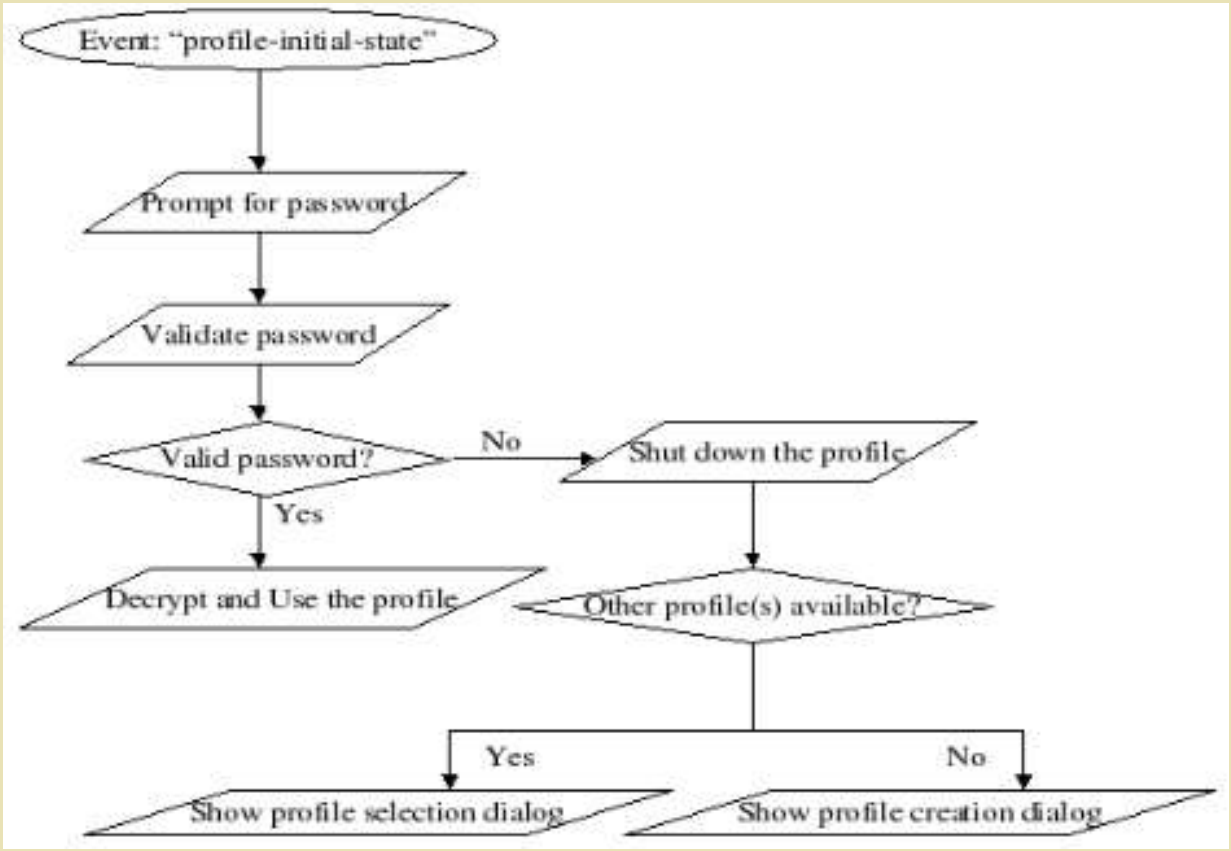
Sequence of Operations during Switch

- Upon the “profile-approve-change” event, authenticate the user.
- If successful, check whether the current profile is a protected USB profile. If yes, set the encryption flag.
- Decrypt the target profile.
- At the “profile-initial-state” event, the previous profile is updated by Mozilla Profile Manager. If the encryption flag is on, encrypt the previous profile.

Scenario 2: At startup…

- The USB Profile Manager finishes authentication and decryption before Mozilla Profile Manager does anything.
- If authentication succeeds, tell Mozilla Profile Manager to start with the selected profile without showing the dialog.

Sequence of Operations at Startup



Chapter III

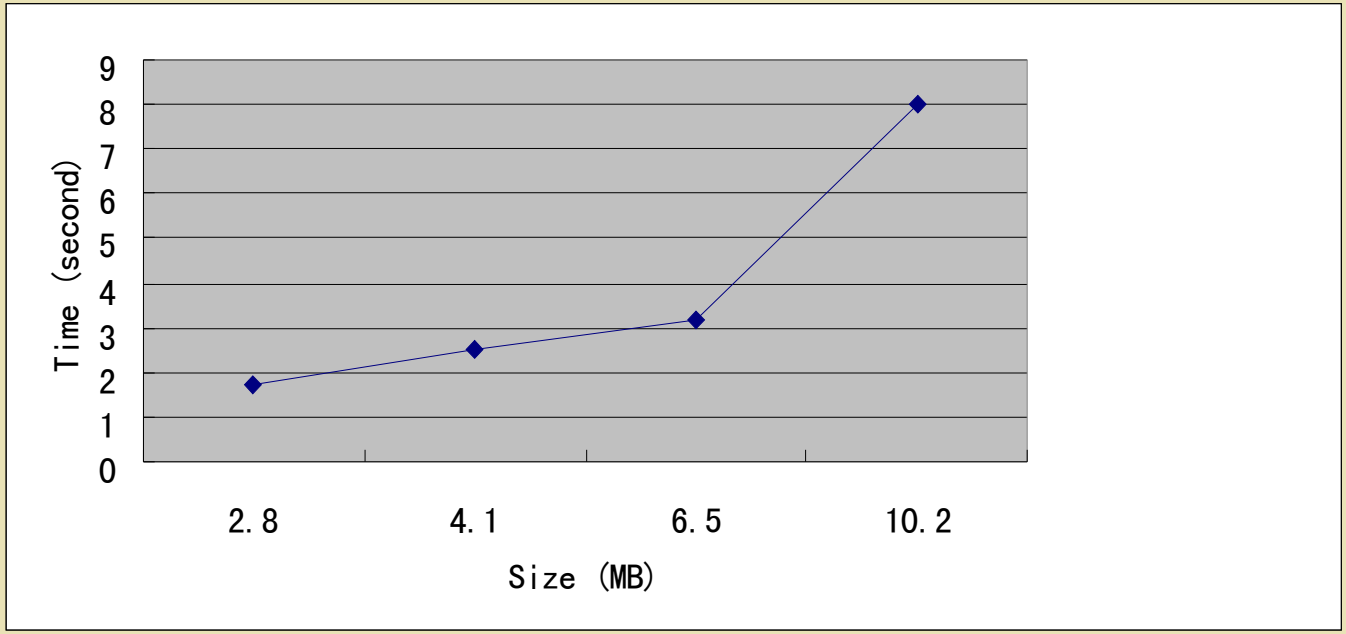
Performance

Performance Test

- Security protections add overheads to the existing functionality. Those overheads include computational cost and I/O costs during encryption and decryption. When the profile is large, for example over 10MB, this overhead can become significant.
- We assume that users do not switch between profiles frequently.

Performance Test (cont'd)

- Hardware configuration:
  Mobile Intel Celeron CPU 2.40 GHz with 192 MB of RAM
- The elapsed time in the figure includes the time for traversing the directory, AES encryption, Base64 encoding, deleting plaintext files.
- Also note the optimization feature mentioned earlier is turned off.



Conclusion

- In this project, we implemented the USB profile manager as an extension of the existing profile manager.
- The USB profile manager can automatically detect, load, and unload profiles stored on a USB key.
- Encryption, decryption and user authentication prevent unauthorized users from using or gathering information from protected profiles.
- So the USB profile achieves both mobility and security.
- The performance is acceptable with medium-sized profiles, assuming the user do not switch between profiles frequently.

Future Work

- Completely platform-independent by implementing the USBProfileLoader interface for each platform.
- A progress bar during encryption and decryption.
- Allow users to decide which part of the profile to encrypt. Can be an extension to the existing “Preferences” dialog.

Acknowledgement

I would like to take this opportunity to thank all the committee members
(Dr. Chris Pollett, Dr. Melody Moh and Dr. Mark Stamp)
for spending their time and effort on this
project and giving me precious advice.

Acronyms

AES – Advanced Encryption Standard
CBC – Cipher Block Chaining
DES - Data Encryption Standard
DOM – Document Object Model
ECB – Electronic Code Book
IV – Initialization Vector
NSS – Mozilla's Network Security Services
PSM – Mozilla's Personal Security Manager
USB – Universal Serial Bus
XPCOM – Cross Platform Component Object Model
XUL – XML User Interface Language

References

[BC03] D. P. Bovet, M. Cesati. Understanding the Linux Kernel. O'Reilly. 2003.

[CR04] W. Chang, B. Relyea. "Network Security Services (NSS)". Retrieved on 4/2/04, from http://www.mozilla.org/projects/security/pki/nss/.

[04] A. Flett. “Guide to the Mozilla string classes”. Retrieved on 8/12/04, from http://www.mozilla.org/projects/xpcom/string-guide.htm.

[HWW02] P. Le Hégaret, R. Whitmer and L. Wood. “W3C Document Object Model.” July 17, 2002. Retrieved on 8/2/04 from http://www.w3.org/DOM/.

[KPS02] C. Kaufman, R. Perlman and M. Speciner. Network Security: Private Communication in a Public World. Prentice Hall. 2002.

[KR02] V. Klíma and T. Rosa. “Strengthened Encryption in the CBC Mode.” May 24, 2002. Retrieved on 9/24/04 from http://eprint.iacr.org/2002/061.pdf.

[LDH04] B. Lord, J. Delgadillo, T. Hayes. "Personal Security Manager (NSS)". Retrieved on 4/2/04, from http://www.mozilla.org/projects/security/pki/psm/.

[M03] Nigel Mcfarlane. Rapid Application Development with Mozilla. Prentice Hall. 2003.

[M98] The Mozilla Organization. “Embedding API Reference”. Retrieved on 8/2/04, from http://www.mozilla.org/projects/embedding/embedapiref/embedapiTOC.html.

References (cont'd)

[M98] The Mozilla Organization. "NSPR Reference". Retrieved on 4/2/04, from http://www.mozilla.org/projects/nspr/reference/html/index.html.

[M01] The Mozilla Organization. "Posing Gecko dialogs in embedding applications". Retrieved on 8/2/04, from http://www.mozilla.org/projects/embedding/windowAPIs.html.

The Mozilla Organization. "Mozilla Cross-Reference". From http://lxr.mozilla.org/seamonkey/.

[N98] Netscape Communications. "Gecko DOM Reference". Retrieved on 8/2/04, from http://www.mozilla.org/docs/dom/domref/.

[P01] Rick Parrish. "XPCOM". Retrieved on 4/2/04, from http://www-106.ibm.com/developerworks/webservices/library/co-xpcom.html#h0.

[P99] B. Preneel. "State-of-the-art ciphers for commercial applications". Computers & Security. 1999.

[S95] B. Schneier. Applied Cryptography: Protocols, Algorithms and Source Code in C. Wiley. 1995.

[S99] W. R. Stanek. Mozilla Source Code Guide. Netscape Press. 1999.

[TO03] D. Turner, I. Oeschger. "Creating XPCOM Components". Retrieved on 1/12/04, from http://www.mozilla.org/projects/xpcom/book/cxc/.