Secure Jam Resistant Key Transfer: Using the DOD CAC Card to secure a radio link by employing the BBC jam resistant algorithm

Major Stephen S. Hamilton
United States Military Academy
West Point, NY
and
Dr. John A. Hamilton Jr.
Auburn, University
Auburn, AL

ABSTRACT

The demand for reliable secure wireless communication is constantly increasing especially within the Department of Defense. As wireless communications become more critical, the need to secure them increases as well. A theoretical approach to jam resistance without a pre-shared secret (BBC algorithm) was developed at the United States Air Force Academy that can enable radios to transfer keys in a jam resistant manner. This algorithm is theoretically proven to provide significant jam resistance, however it does not provide security. The DOD Common Access Card (CAC) can enable security without a pre-shared secret using public and private key pairs. This application combines the CAC card and BBC algorithm to provide a jam resistant secure method of generating and transporting a temporal symmetric key without the use of pre-shared keys. The application specifically contains a user friendly GUI for CAC card operations on the Linux platform to include reading Public Key Infrastructure (PKI) Certificates, user authentication, and basic key management. It also contains the functions to transmit and receive public keys and encrypted symmetric keys using the BBC codec and the GNU Software Radio.

INTRODUCTION

The current method of providing jam-resistant secure communications in the military is through the use of the Single Channel Ground and Airborne Radio System (SINCGARS). This radio provides "...commanders with a highly reliable, secure, easily maintained Combat Net Radio (CNR) with voice and data handling capability, in support of command and control operations."[1] Although this radio is widely used throughout the military, in order for it to operate in secure jam resistant (frequency hopping) modes, it must be loaded with a symmetric key and a frequency hopping set or hop set. Once these two keys are loaded and timing is synced between two radios, they can transmit and receive data or voice in an encrypted jam resistant manner. The problem exists when one or more users of the radio either loses the key or cannot obtain the key necessary to communicate.

In the field, military personnel can easily forfeit security on communications systems in order to communicate. This includes either switching a SINCGARS radio to plain text mode or even using commercially bought family service radios. This results in soldiers using Frequency Modulated (FM) voice in the clear that is easily jammed and monitored. Although this is very dangerous, the reason why this could occur is that the immediate risk of not having communications is much higher than having communications that is not jam resistant and unsecure. However, it can easily be argued that the overall risk is much higher if the communications are intercepted giving the enemy potentially devastating intelligence.

Commanders in the military may opt to say "we will just enforce standards" so that soldiers *must* use secured radios thus solving the problem. Although this may brief well, it does not consider a combat scenario where a convoy is hit and one lone SINCGARS radio is functional, however its backup battery dies and the soldier has no pre-shared keys loaded. In this scenario, the soldier has *no* other choice than to transmit in the clear in order to call for help. The problem is not that the soldier does not want to follow the Standard Operating Procedures (SOPs), he or she simply does not have any other method of communication. Therefore the problem lies with the equipment, not the soldiers or Tactics, Techniques, and Procedures (TTPs). The question then is how can radios issued to soldiers provide immediate jam resistant secure communications without the need to distribute and load shared keys?

978-1-4244-2677-5/08/$25.00 ©2008 IEEE

Currently there is no immediate solution to this particular problem. There has been work on securing software radios, to include securing the download of SDR configuration files. In this security scheme, X.509 certificates are used, however no jam resistance is employed during the transfer of these certificates.[2] In addition, numerous military communication systems to include the SINCGARS, ARC-220, RF-5000, and EPLRS use anti-jamming techniques; however they all require a pre-shared secret in order to enable these communications.[3]

Although the solution does not exist, a new radio system that is being developed provides a potential platform for a solution. The Joint Tactical Radio System or JTRS, is a "family of interoperable, affordable software defined radios...which provide secure, wireless networking communications capabilities for joint forces"[4] This platform could implement the BBC[5] (Baird, Bahn, and Collins) jam resistance algorithm in its software, making it possible to add the capability without hardware modification. Since this new radio platform is being built and fielded, the problem left is to implement the BBC on a software radio, and perform a public key exchange for transport of symmetric keys.

BBC BACKGROUND

The BBC Algorithm consists of two parts: encoding and decoding. The encoding algorithm encodes the message in BBC format, and the decoding algorithm assumes multiple messages were sent, and decodes all messages found. These multiple messages are assumed to be combined using a bitwise OR. Therefore, a 0 can become a 1, however the reverse is not true. This is the heart of the jam resistance in the algorithm: it works off of the presence of data rather than the modulation of data. In jamming a radio, the attacker cannot take away radio frequency waves that are transmitted. Since the attacker cannot take the radio waves away, his or her intent is to add noise to the data to the extent that the receiver cannot receive or receives a modification of the original message. In order to help explain how the BBC Algorithm works, a toy example will be used on a small string of data. In this example, the message to be sent is 011. The hash is a simple predefined lookup table in figure 1. The process of building the message to be sent is as follows:

1. Read first bit in message, and hash: H(0) = 20
2. Read next bit in message, and hash message: H(01) = 5
3. Read next bit (entire message at this point): H(011) = 19

Msg	**Hash**	**Msg**	**Hash**	**Msg**	**Hash**
0	20	11	2	100	14
1	15	000		101	9
00	10	001	12	110	4
01	5	010	17	111	3
10	1	011	19		

Figure 1: Hash table

1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
				X					
11	**12**	**13**	**14**	**15**	**16**	**17**	**18**	**19**	**20**
								X	X

Figure 2: Message BBC encoded, and ready for transmission.

The receiver performs slightly more work, since it must anticipate the message. The procedure for receiving the message is as follows:

1. Listen at timeslots for either a 1 or a 0. H(0) = 20 H(1) = 15. Given the message above, 20 is marked, and 15 is not, therefore the message begins with 0.
2. Listen for H(00)=10 and H(01)=5. A mark is on 5, so the message is now 01.
3. Listen for H(010)= 17 and H(011)=19. A mark is on 19, so the message is confirmed to be 011.

This example shows so far the ability to decode the message. In order to show jam resistance, the example will be shown with 25% of the domain filled with jam marks. Figure 3 shows this example with jamming displayed as a Z mark.

1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
	Z			Z			Z		Z
11	**12**	**13**	**14**	**15**	**16**	**17**	**18**	**19**	**20**
			Z					X	X

Figure 3: Message with random Zs to represent jamming marks.

In this jamming example, the receiver would decode the following:
H(0) = 20, true. H(1) = 15, false Message = 0
H(01) = 5, true. H(00) = 10, true Message = 01 and/or 00
H(000)= 7, false. H(001) = 12, false. This means the

mark at H(00) is a jam/hallucination.
H(010) = 17, false. H(011) = 19, true. Message = 011.

MOZILLA NETWORK SECURITY SERVICES (NSS) BACKGROUND

The Mozilla Network Security Services "...is a set of libraries designed to support cross-platform development of security-enabled client and server applications."[6] The services NSS provide include PKCS #11, which "...governs communication with cryptographic tokens (such as hardware accelerators and smart cards) and permits application independence from specific algorithms and implementations." Not only these services provide communication with smart cards, but the NSS modules have received completed FIPS 140 validation four times over the past 11 years.[7]

Since the final target of this project is for a Federal or more specifically Department of Defense application, it is important to only use cryptographic modules that are federally certified. FIPS 140-1 is titled Security Requirements for Cryptographic Modules. The standard specifies the security elements related to the secure design and implementation of cryptographic modules.[8] FIPS 140-2 is an update to FIPS 140-1, since FIPS 140-1 is no longer used to test cryptographic modules. Although FIPS 140-1 is no longer used, modules that were certified against it are still recognized as FIPS 140-1 compliant.[9] As of July 13th, 2007, FIPS 140-3 is only in draft form, and is not currently being used.[10] Therefore the NSS modules are in compliance with the latest FIPS standard, which supports its use as the cryptographic module in this project.

In addition to Federal compliance with cryptography, the module provides a high level API for interfacing with the CAC card. The basic cryptographic platform in NSS is showed in Figure 4.

Netscape Security Services	
Coolkey Library	NSS Software Key Database
PCSC-Lite	NSS Database Files
Vendor Specific Smartcard Driver	

Figure 4: NSS Cryptographic platform depicting linkage to smart card interface.

The NSS module is a generic security module that interacts with the NSS database of software cryptographic tokens. It also supports hardware tokens through the use of libraries. In this project, the Coolkey library is used. The concept behind Coolkey is to enable a "...PKI solution that provides smart card login, single sign-on, secure messaging, and secure email access."[11] This is basically a part of the Fedora Directory Services, analogous to Active Directory in the Microsoft Windows Environment. The specific module that interacts with the CAC card is the Coolkey PKCS #11 module. PKCS #11 is the Cryptographic Token Interface Standard that specifies an open API called Cryptoki, which provides an object-based approach to cryptography. This standard allows applications to interact with PKCS #11 compliant devices without modification of cryptographic programming calls.[12]

In addition to the PKCS #11 Coolkey module, another layer of interoperability exists that connects to the smart card reader driver. This layer is Personal Computer/Smart Card or PC/SC. PC/SC builds on the ISO 7816 standard to provide low-level device interfaces and device independent application APIs in order to allow multiple applications to share multiple smart card devices attached to a system. The specification is platform independent, which allows it to be used on any operating system.[13] In this project, PCSC-Lite is used to connect to the smart card driver and provide the low level interface to the Coolkey library. PCSC-Lite is an API toolkit that was created by the Movement for the Use of Smart Cards in a Linux Environment (MUSCLE) group. Using this interface, the applications are abstracted from driver details so that multiple different smart card readers can be used in a homogeneous fashion.[14]

JAM RESISTANT KEY TRANSFER PROCESS

The process to share a secret key using BBC jam resistance begins with the user selecting a Public Key Certificate to be transferred. It is important to note that the entire certificate must be transferred as opposed to the modulus of the public key. The reason for the transmitting the entire certificate is to validate that the certificate was in fact issued by a recognized Certificate Authority. Certificates are signed by hashing the user's certificate, encrypting the hash with the Certificate Authority's private key. To validate the certificate, the receiver hashes the message, and decrypts the signature with the Certificate Authority's public key. Once this is performed, the hashes are compared, and if they match, the certificate is validated as being issued by the Certificate Authority. If the certificate is not validated once it is received, the communication should not continue, since it could mean an attacker is trying to transmit a fake certificate in order to establish communication. If this is not the case, then the root certificate authorities must be verified to ensure the certificate is being checked with the appropriate authority.

Once the user has chosen the certificate to be used for key exchange, the certificate is prepared for transmission. Initially, the entire certificate is retrieved from the card. Once it is retrieved, it is encoded in Distinguished Encoding Rules (DER) format. This format is a common method of saving a certificate to a file, and is defined by the International Telecommunication Union Recommendation X.690. The rules were approved on 14 July 2002.[15] After the certificate is encoded using DER, it is saved as a file in preparation to be encoded using the BBC Algorithm.

Although the toy example presented above makes the BBC algorithm appear simple, the actual implementation is slightly more complex. Specifically, the toy example did not address how a message begins to be received, and how the receiver determines when a message is complete. In the example, an attacker could potentially transmit at the right time to produce an identical message to the transmitted message except the last bit is flipped. The receiver would then concurrently decode two messages (ex. 011 and 010). Although the original message was not jammed, the receiver must make a decision on which message is the original, and which is not. It is important to note that although this can be done, it is more likely it happens by chance. Since the message is hashed, the time when the jam must occur to flip the last bit may have been in the past, making it impossible for the jammer to do this in realtime. However, if it is done by chance or the attacker already knows the message to be sent, two different messages would be decoded at the receiver. In order to help prevent this problem in the implementation, checksum bits are added to the message. These checksum bits are simple 0s appended to the message, which when hashed with the original message, greatly reduces the attacker's chance of inserting a different message.

Another addition beyond the toy example is the formatting and padding of the message. In the toy example, it was already known that the message length was 3 bits, making the decoding straightforward. However, if the message is 1024 bits or potentially longer, it makes more sense to begin breaking up the message and encoding in blocks. Once these blocks or packets are encoded, they can be formatted so that the receiver knows when a new packet is arriving. In the implementation, the default size of a packet is 512 bits. Therefore the message after padding is broken up into 512 bit packets that are encoded, and marked with bookend bits. These bookend bits are used to signal the decoder that a packet is present. After the message is prepared for transmission, it is modulated. In the current code, it is a very simple time domain modulation. This means the packet marks are spaced out, and the timing between marks is what defines the message that is being transmitted. This modulation is saved to a sink file that will be transmitted by the GNU Radio. A python script is then executed with the file sink as the source of data to be transmitted at a user specified frequency. The USRP must be able to support the requested frequency, and this is determined by the population of the daughterboards. Ettus research currently provides daughterboards that cover part of the frequency ranges from 0 to 2.9 GHz. The daughterboards used for this project are the RFX1200 Transceiver cards, which have a frequency range from 1150-1400MHz. If the python script receives an acceptable frequency, it reads the modulated file and creates a block that transmits this modulated file.

On the receiver, the platform and setup is the same; however a different python file must be running to receive the data from the USRP. It reads straight raw data, and writes raw data to a file continuously or for a specified amount of time. Once it is finished, the reverse of the message preparation described above must occur. The file is read and demodulated, and the BBC decode algorithm is invoked. After demodulation, the decode algorithm first searches for the bookend marks of a packet. This must be a standard already predefined, so that interference transmitted before the BBC packet can be discarded.

Once bookend marks are found, a potential BBC message might exist, so decoding begins. The decoding is similar to encoding, however it is slightly more complex due to the fact it could receive multiple messages.

The decoder uses a depth-first-search to begin extracting the messages from the packet. The decoder then sets up a message as if it were to be transmitted. This is essentially a padded message with no data. At this point, a zero is added to the message, and the message is hashed. The result of this hash is then checked against the modulation file to determine if a mark was set at that location. If a mark is found at the location, and if the padded message is not complete then the padded message is added to the output. If the padded message is not complete, then it adds just the zero to the output. If the maximum number of messages that can be decoded from a single packet is reached, the process is over, which is a predefined user value. Next, the codec backtracks to the last message bit that is a zero, and if no message bit is zero the decoding ends. If not, the last zero bit gets changed to a one, and the entire process is repeated.

MAIN PROGRAM

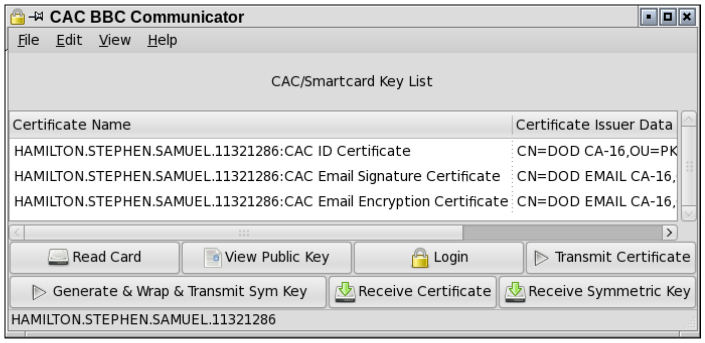
The BBC Communicator application is written entirely in C and compiled for the Linux platform. Although the GNU Radio can be run on platforms other than Linux, the primary platform it was designed for is Linux. Linux is also a common platform for embedded systems, since the kernel and OS can be tailored to meet specific hardware needs. The flavor of Linux chosen for the application is Fedora. The version is currently Fedora 8, however the application has been compiled and proven to work on Fedora Core 6 and Fedora 9. Theoretically, the application will work on any Linux distribution that can run the GNU Radio, NSS Libraries, Coolkey library, and is able to connect to a smart card using the pcscd daemon.

The initial design of the BBC Communicator is founded upon the NSS libraries. NSS requires the Netscape Portable Runtime library in order to operate. This runtime "...provides a platform-neutral API for system level and libc like functions. The API is used in the Mozilla client, many of Netscape/AOL/iPlanet's and other software offerings."[16] Although these libraries can be compiled and installed from source code, they are also available through the Fedora Yellowdog Updater Modified (yum). Therefore the command "yum install nss nspr" will install the latest available nss and nspr libraries. In order to utilize these libraries for development (i.e. compiling this project), the development libraries must be installed. This can also be done with yum by typing "yum install nss-devel nspr-devel." Once these libraries are installed, an application can be developed using any of the public functions available.

In addition to using the NSS module for cryptographic operations, the Gnome Toolkit or GTK+ libraries are utilized in order to create a user friendly Graphical User Interface (GUI) for the application. The project utilizes a GUI to perform all CAC card operations and GNU Radio transmission with the intent of providing a more visual and hands on demonstration for decision makers within the Department of Defense.

In the background, the program begins by initializing the NSS database. This initialization gives the application the ability to utilize all NSS functions, in addition to accessing certificates within the NSS database.

If the database initializes successfully, and a CAC card is present when the user presses Read Card, the three certificates installed on the CAC will be listed in the main window as shown in figure 5. The first column shows the certificate name, while the second shows the details of the certificate issuer. Although the CAC card typically contains exactly three certificates, any smartcard will work as long as at least one certificate is loaded on it. The window is populated by the list of all certificates that NSS is able to read from the card. Once this list is populated, the user can choose one of the certificates to perform an operation. The operations available at this point are to view the modulus of the public key (in hexadecimal) associated with the certificate and transmit the public key.



Certificate Name	Certificate Issuer Data
HAMILTON.STEPHEN.SAMUEL.11321286:CAC ID Certificate	CN=DOD CA-16,OU=PK
HAMILTON.STEPHEN.SAMUEL.11321286:CAC Email Signature Certificate	CN=DOD EMAIL CA-16,
HAMILTON.STEPHEN.SAMUEL.11321286:CAC Email Encryption Certificate	CN=DOD EMAIL CA-16,

HAMILTON.STEPHEN.SAMUEL.11321286

Figure 5: A CAC card is loaded, and the certificates on the CAC are shown in the window.

Once the transmit key is pressed, if a certificate is selected, the program begins by saving the selected certificate to a file in the DER encoded format. Then, the BBC encoder is invoked, and the .DER file is BBC encoded, and modulated for transmission. This is saved to a sink file that is read by the python transmission program. This python program reads the file, and transmits the marks using the USRP.

The receiving computer must also have a USRP and the CAC Communicator running on it. The user should press receive certificate before the other user transmits it to ensure the entire certificate is received. The receiver times out after a pre-defined time, and then BBC decoding is performed. Although this application requires the two users to coordinate the certificate transfer, an implementation in the field may be designed in such a way so that one user (ex. a base station at a Tactical Operation Center) is always set to receive keys, and sounds an alert when a certificate is received.

When the receiver has obtained the public key, the key can be validated as issued by one of the DOD Root Certificate Authorities (root certificates must be pre-loaded in the NSS database), and then used to encrypt a symmetric key. The CAC card has the capability of generating various types of symmetric keys on the card, and returning them to the application requesting it. In this case, the key created is an AES 256 bit key. Once the key is generated, the DER certificate that was received is loaded and used to encrypt the newly generated symmetric key. This new encoded package is called a wrapped symmetric key. The wrapped key is then saved to a data file. This file is then sent to the BBC encoder and modulator, and transmitted across the USRP. The original sender then must be ready to receive this wrapped key.

In order to receive a key, the user must log into the CAC card. This is because the private key is protected on the card by PIN login. The key never leaves the card, however the user must log into the card before data can be decoded by the card using the private key on the card. If the user tries to decode the key without being logged into the card first, the status bar will inform the user that they must log in to perform that operation. Once the user logs into the card, and presses the "Receive Symmetric Key" button, the receiver is enabled for a brief period of time, in which data is captured. Once the time period has ended, the data is searched for a BBC encoded message. If a message is found, it is decoded, and the resultant wrapped key is saved to a file. In order to reduce complexity of the program and make it more user friendly, each private key on the card is tried, and the key that unwraps the symmetric key is displayed in the status bar. The symmetric key can be used at this point on either station for secure jam resistant communications to include data, voice, or multimedia transmissions.

CONCLUSION

The presented application successfully performed secure jam-resistant transfer of a secret symmetric key. Initial tests using a signal generator to jam during a key transfer showed that a constant wave transmitted on over 25% of the key transmission resulted in a successful key transfer. Once more than 50% of the transfer was jammed, the BBC decoder consumed the CPU and decoding could not complete, which was expected based on the implementation. Since jam resistance did occur, the CAC BBC Communicator can be easily used for a demonstration of using the CAC card to secure radio communications in a jam resistant manner without the use of pre-loaded or distributed COMSEC.

The BBC Algorithm proves to be a very effective method for jam resistance, and this project helps display its capabilities. It complements the DOD Public Key Infrastructure since it can perform jam resistant public key and wrapped key delivery without a shared secret. The user friendly GUI allows users to employ the BBC Algorithm on key transfer from any DOD CAC card across the GNU Software Radio with the click of a button. This application has the potential to be implemented in the Joint Tactical Radio System, and fielded throughout the armed services. All unsecured communications that occur in the field can be easily secured with a jam resistant key transfer initiated by a service member's CAC card. In addition to instant secure communications, the ability to perform reliable secure symmetric key transfer could revolutionize how military units perform key distribution in the field.

COMSEC key management in the Army is a very time consuming and resource intensive process. Each month, a key changeover occurs, and they must be physically distributed throughout the tactical units to various COMSEC key holding devices. Most radios like the SINCGARS have a battery that stores the key when the unit is off, however these batteries eventually lose their charge, and the radio must be re-keyed. This project effectively shows that these problems can be overcome using the DOD issued CAC card. In fact, a very small easy to use hand held radio could be created that uses the CAC card and BBC algorithm to perform voice communications with an on demand CAC generated temporal or session key for point to point communications. This radio could be used in tactical situations when key transfer is difficult or impossible and secure jam resistant communications are necessary. In addition to radio communications, jam resistant key transfer could be used to securely transfer a key set and frequency hopping sets over the air for later distribution to legacy radios.

The project was built using the GNU Software Radio, which allows modulation types to be changed using different software code. This was essential, since the BBC requires a physically different modulation (baseband signaling) than frequency modulated voice, amplitude modulation, or direct sequence spread spectrum. Although the signaling it uses is less efficient than other forms, it is only required for the initial key transfer. Once this is complete, the modulation should be changed and encrypted using the transferred key, similar to the process of using computationally complex public keys algorithms to encrypt symmetric keys for efficient secure communication.

FUTURE WORK

The next phase of this project is to implement direct sequence spread spectrum (DSSS). This form of modulation spreads out a narrowband signal with the use of pseudo random noise (PN). The PN is usually modulated in the form of phase shift keying (PSK), and the PN sequence is the spreading waveform, or code. The receiver uses the same spreading sequence or code to multiply against the incoming signal which recovers the original message.[17] The PN in this case would be derived from the symmetric key that was transferred, which is a secure shared secret. Therefore, the transmission would be spread among a band using a sequence that is unknown to a jamming station. The result is an *efficient* method of jam resistant communication that was enabled by the key transfer displayed in this project.

Another possibility is a chat and/or voice over IP (VoIP) implementation that could be used to further demonstrate the secure communications once they are established. This would provide a demo that could easily be understood and possibly serve as a requirement for an implementation into the JTRS. The DSSS modulation would be used for the initial key transfer with the PN being picked by DSSS. After the key transfer, DSSS would still be used; however the PN sequence would be determined by the transferred symmetric key. In addition to point to point communication, protocols for multiple radios to communicate in a radio net could be created and evaluated as well. This would require negotiation of a master certificate that can be requested when a user joins the network, and sharing of a common symmetric key among a group of radios. The master certificate would be shared using the BBC algorithm.

An update to the current project in the future could include a tighter integration with the BBC algorithm. For experimental purposes, the BBC encoding and modulator is loosely coupled with the CAC card GUI application. This is due to the current implementation being replaced by a streaming implementation where BBC calls and transmissions would be directly called from the application. The current project separates itself from the BBC code by saving key information to files, and letting the BBC codec read the data and send it with a GNU radio transmission program. Although this allows great flexibility in a lab testing environment, an application that is fielded will have to have these parts tightly integrated so that they consume fewer resources and operate more efficiently.

REFERENCES

1 Project Manager Tactical Radio Communications Systems. "Single Channel Ground and Airborne Radio System (SINCGARS)" p. 1. http://peoc3t.monmouth.army.mil/trcs/pdfs/SINCGARS.pdf

2 A. Brawerman, D. Blough, and B. Bing. Securing the Download of Radio Configuration Files for Software Defined Radio Devices. MobiWac 2004, Philadelphia, Pennsylvania. p. 101.

3 Sass, Paul. Communications Networks for the Force XXI Digitized Battlefield. Mobile Networks and Applications, Volume 4, Issue 3 (October 1999). p 139-155.

4 JPEO JTRS Website Mission Statement http://jpeojtrs.mil.

5 L. Baird, W. Bahn, and M. Collins. Jam-Resistant Communication Without Shared Secrets Through The Use of Concurrent Codes. US Air Force Academy Technical Report USAFA-TR-2007-01. p. 13.

6 Mozilla developer Center. NSS http://developer.mozilla.org/en/docs/NSS

7 Mozilla FIPS Validation http://wiki.mozilla.org/FIPS_Validation

8 FIPS 140-1 http://csrc.nist.gov/publications/fips/fips1401.htm

9 FIPS 140-2 http://csrc.nist.gov/publications/fips/fips140-2/fips1402.pdf

10 FIPS Publications http://csrc.nist.gov/publications/PubsFIPS.html

11 Fedora Directory Server. Coolkey http://directory.fedoraproject.org/wiki/CoolKey

12 RSA Security Incorporated. PKCS #11: Cryptographic Token Interface Standard. http://www.rsa.com/rsalabs/node.asp?id=2133

13 PCSC Workgroup Specifications Overview. http://www.pcscworkgroup.com/specifications/overview.php

14 Corcoran, David. MUSCLE PC/SC Lite API Introduction/Overview. http://pcsclite.alioth.debian.org/pcsc-lite/node2.html

15 ITU-T Recommendation X.690. www.itu.int/ITU-T/studygroups/com17/languages/X.690-0207.pdf

16 Netscape Portable Runtime http://www.mozilla.org/projects/nspr/

17 D.R. Bull, C.N. Canagarajah, and A.R. Nix. Insights into Mobile Multimedia Communications Academic Press, 1999. p. 464