Architecture for Issuing DoD Mobile Derived Credentials

David A. Sowers

Thesis submitted to the faculty of the Virginia Polytechnic Institute and State University
in partial fulfillment of the requirements for the degree of

Master of Science
In
Computer Engineering

T. Charles Clancy, Chair
Sandeep Shukla
Luiz A. DaSilva

May 2, 2014
Falls Church, Virginia

Keywords: Derived Credentials, Public Key Infrastructure, Common Access Card,
Department of Defense, x509, Mobile Phone

Architecture for Issuing DoD Mobile Derived Credentials

David A. Sowers

ABSTRACT

With an increase in performance, dependency and ubiquitousness, the necessity for secure mobile device functionality is rapidly increasing. Authentication of an individual's identity is the fundamental component of physical and logical access to secure facilities and information systems. Identity management within the Department of Defense relies on Public Key Infrastructure implemented through the use of X.509 certificates and private keys issued on smartcards called Common Access Cards (CAC). However, use of CAC credentials on smartphones is difficult due to the lack of effective smartcard reader integration with mobile devices. The creation of a mobile phone derived credential, a new X.509 certificate and key pair based off the credentials of the CAC certificates, would eliminate the need for CAC integration with mobile devices This thesis describes four architectures for securely and efficiently generating and delivering a derived credential to a mobile device for secure communications with mobile applications. Two architectures generate credentials through a software cryptographic module providing a LOA-3 credential. The other two architectures provide a LOA-4 credential by utilizing a hardware cryptographic module for the generation of the key pair. In two of the architectures, the Certificate Authority's (CA) for the new derived credentials is the digital signature certificate from the CAC. The other two architectures utilize a newly created CA, which would reside on the DoD network and be used to approve and sign the derived credentials. Additionally, this thesis demonstrates the prototype implementations of the two software generated derived credential architectures using CAC authentication and outlines the implementation of the hardware cryptographic derived credential.

Acknowledgements

I would like to first thank the US Coast Guard for the opportunity to attend Virginia Tech while serving as an active duty officer. The graduate school program through the Coast Guard is one of the most beneficial and rewarding experience of my military career.

I would like to thank Dr. Charles Clancy for all your support, guidance, and leadership throughout the thesis and research process.

A very special thanks to Kiran Karra for his help in compiling the python code used for the prototype implementation portion of the thesis. Your help was greatly appreciated.

Contents

List of Figures

List of Tables

1 Introduction

On August 27, 2004, President G.W. Bush signed into effect the Homeland Security Presidential Directive 12 (HSPD-12) Policy for a Common Identification Standard for Federal Employees and Contractors. The primary purpose of HSPD-12 was the mandatory implementation of a government wide standard for secure and reliable form of identification [1]. In response to the directive, the National Institute of Standards and Technology (NIST) provided implantation instructions through the delivery of the Federal Information Processing Standard 201 (FIPS 201). The Common Access Card (CAC), already in use by the Department of Defense (DoD) at the time, was the government's vehicle to comply the requirements set forth by HSPD 12 and FIPS 201.

The CAC allows the DoD to successfully implement a Public Key Infrastructure (PKI) architecture, securely storing all user x509 certificates and private keys which are used for authentication for building access, computer and file access and most importantly email access and digital signatures. Currently, the DoD requires that all persons accessing network files and email must use a CAC for authentication. The use of a CAC with a mobile device requires a USB or Bluetooth interface using a software middleware for interaction between the device and the CAC. With the recent popularity and reliance of mobile devices drastically increasing, the ability to securely authentication via a mobile device is burdensome or not practical. The current solution to DoD mobile security without the use of a CAC relies on derived credentials.

A derived credential is an X.509 certificate that is generated based off the credentials of the Personal Identity Verification (PIV) card certificates. These new certificates and private keys

would then allow the user authenticate to computer networks, servers and files from their mobile device. While derived credentials have been a hot topic in recent years, standards and policy are not in place for the generation and delivery of these credentials. Scalability, security and efficiency will all play a large role in the architecture used for the generation and delivery of derived credentials to DoD members. With over 1.4 million active duty men and women, and 718,000 civilian personnel [2], the process for generating derived credentials must be scalable and efficient; importing it to the mobile device and using the derived credential must also be extremely secure and resistant to attack. This thesis comprehensively details four separate architectures for generation and delivery of derived credential for DoD mobile devices. Additionally, this thesis demonstrations a prototype implementation of two of the architectures for the creation of DoD mobile derived credentials that will allow Android and iPhone devices to securely access DoD computer networks and files.

2 Review of Standards and Prior Work

This section evaluates relevant standards and previous work conducted in the field.

2.1 HSPD-12

President G.W. Bush's Homeland Security Presidential Directive 12, a "Policy for a Common Identification Standard for Federal Employees and Contractors" required the development and implementation of a mandatory standard for secure and reliable forms of identification for all federal employees and contractors [1]. The main purpose of this directive was to develop and use a secure and reliable method of identification that: "(a) is issued based on sound criteria for verifying an individual employee's identity; (b) is strongly resistant to identity fraud, tampering, counterfeiting, and terrorist exploitation; (c) can be rapidly authenticated electronically; and (d) is issued only by providers whose reliability has been established by an official accreditation process" [1].

As a response to HSPD-12, NIST and their Computer Security Division started a new program which would develop all the documentation, standards, and policy for improving the identification and authentication of Federal employees and contractors for access to computer networks and information system [3]. The resulting Federal Information Protection Standard (FIPS) 201, PIV of Federal Employees and Contractors was developed, approved and issued on February 25, 2005.

FIPS 201 incorporated three technical Special Publications (SP); NIST SP 800-73 "Interfaces for Personal Identity Verification", NIST SP 800-76 "Biometric Data Specification for

Personal Identity Verifications", and NIST SP 800-78 "Cryptographic Algorithms and Key Sizes for Personal Identity Verification" [3]. In addition to these publications, several guidelines, reference implementations and conformance tests were identified by NIST for the implantation of their PIV system. These guidelines include, but are not limited to, how to "implement and use the PIV system; protect the personal privacy of all subscribers of the system; authenticate identity source documents to obtain correct legal name of the person applying for the PIV card, electronically obtain and store biometric data from PIV system subscriber; create a PIV card that is personalized with data needed by the PIV system to later grant access to the subscriber to Federal facilities and information systems; assure appropriate levels of security for all applicable Federal applications; and provide interoperability among Federal organizations using the standards." [3].

All publications, guidelines and standards can be found on NIST website (http://csrc.nist.gov) under their publications tab. Below, this thesis will highlight the relevant details contained in the NIST publications which are pertinent to this thesis.

2.2 FIPS 201

The main purpose of FIPS 201 is to specify the architecture and technical requirements for common identification standards for the Federal employees and select contractors. FIPS 201 mandates the use of a PIV Card that can accurately determine a Federal employee's identity for access to buildings, computer systems, and data. In August 2013, the updated FIPS 201-2 was released. One of the major changes to FIPS 201-2 was the inclusion of Derived PIV Credentials. The standards allows for PIV cards to be used as the basis for issuing Derived PIV Credentials in accordance with NIST SP 800-157, Guidelines for Derived Personal Identity Verification

Credentials [4]. NIST SP 800-157 is still in draft form at the time of this writing, therefore, no requirements have been finalized for the issuance of derived credentials.

Sections 1-3 of FIPS 201-2 defines who is authorized a PIV credential and the proofing process on how to issue the cards. It is essential to accurately determine a person's identity and authority for access prior to issuing a credential. The PIV Card is the primary component to the PIV system and is "a credit card-size form factor, with one or more embedded integrated circuit chips (ICC) that provide memory capacity and computational capability." [4]. The PIV Card will be initiated with card writers, and will authenticate through card readers, Personal Identification Number (PIN) input devices and biometric readers at access points [4].

Sections 4-6 of FIPS 201-2 provides detailed technical specifications of components and processing required for interoperability of PIV Cards with personal authentication, access control, and PIV card management systems across Federal government. Section 4 defines the PIV Front-End Subsystem, including the PIV Card logical characteristics. At a minimum, the PIV Card must store a PIN, a Cardholder Unique Identifier (CHUID), PIV authentication data (one asymmetric private key and a corresponding public key certificate), two fingerprint templates, electronic facial image, and card authentication data (one asymmetric private key and corresponding certificate) [4].

For government issued email accounts, the PIV Card must contain an asymmetric private key and corresponding X.509 certificate for both digital signature and key management. Cryptographic operations with this key are performed only through the contact interface. The PIV Card shall implement the following cryptographic operations and support functions as defined in

SP 800-78 and SP 800-73: “RSA or elliptic curve key pair generation, RSA or elliptic curve private key cryptographic operations, and importation and storage of X.509 certificates.” [4]. While all cryptographic operations must be performed on the card, message hashing may be performed off-card.

There are numerous slots for keys on the PIV card, with some being mandatory and some options. The PIV authentication key shall be asymmetric private key supporting card authentication for an interoperable environment and it is mandatory. An asymmetric Card authentication key is a mandatory private key that supports card authentication for an interoperable environment. A symmetric (secret) card authentication key is optional. A digital signature key is an asymmetric private key supporting document signing and is mandatory for all users with government-issued email. A key management key is an asymmetric private key supporting key establishment and transport and is optional. Lastly, a card management key is a symmetric key used for personalization and post issuance activities and is optional. All PIV cryptographic keys shall be generated within a FIPS 140 validated cryptographic module with overall validation at Level 2 or above [4] [9].

Section 5 is a critical section of FIPS 201-2 for the purposes of this project, as it defines PIV key management requirements. The PIV Card authentication system will use a hierarchical PKI managed by the Federal PKI. All cryptographic algorithms and key sizes are specified in the NIST SP 800-78, which is detailed below. All certificates are issued in accordance with the X.509 Certificate Policy for the U.S. Federal PKI Common Policy Framework (COMMON) and the

contents of the X.509 certificates associated with the PIV private key are based on the X.509 Certificate and CRL Profile for the Common Policy (PROF) [4].

Lastly, Section 6 defines the PIV Cardholder Authentication using PIV asymmetric cryptography. For authentication using the PIV Authentication certificate (PKI-AUTH), the PIV Authentication certificate is read from the PIV Card. The certificate is then checked for validity. The cardholder is then prompted to submit a PIN, and the submitted PIN is used to activate the card. The card reader then issues a challenge string to the card and requests an asymmetric operation in response. The challenge is signed using the PIV authentication private key and is returned. The signature is then verified by the relying system. The unique identifier from the PIV authentication certificate is extracted and passed as input to the access control decision. For authentication with the Card Authentication certificate (PKI-CAK), the same steps are followed as with the PIV Authentication certificate, except the Card Authentication certificate is used for validity, cryptographic responses, and reference and there is no input of a PIN prior to cryptographic operations being performed. PKI-AUTH offers a level of assurance (LOA) comparable LOA-4 which is very high confidence in the asserted identity's validity. Comparably, PKI-CAK only offers LOA-2 which is some confidence in the asserted identity validity [4]. Table 1 below, taken from FIPS 201 [4], details the PIV assurance levels attained through PIV authentication mechanisms.

Table 2.1: Authentication of Logical Access; NIST FIPS 201. Available at http://csrc.nist.gov/. Used under fair use, 2014 [4]

PIV Assurance Level Required by Application/Resource	Applicable PIV Authentication Mechanism	
	Local Workstation Environment	Remote/Network System Environment
LITTLE or NO confidence	CHUID	
SOME confidence	PKI-CAK	PKI-CAK
HIGH confidence	BIO	
VERY HIGH confidence	BIO-A, OCC-AUTH, PKI-AUTH	PKI-AUTH

The levels of assurance from Table 2.1 are very similar to the levels of assurance that are contained in M-04-04 [12], which are detailed in section 2.3.4 below. Table 2.2 below, also from FIPS 201 [4], shows the relationship between PIV and E-Authentication assurance levels.

Table 2.2: Relationship Between PIV and E-Authentication Assurance Levels; NIST FIPS 201. Available at http://csrc.nist.gov/. Used under fair use, 2014 [4]

PIV Assurance Levels	Comparable OMB E-Authentication Levels	
	Level Number	Description
LITTLE or NO confidence	Level 1	Little or no confidence in the asserted identity's validity
SOME confidence	Level 2	Some confidence in the asserted identity's validity
HIGH confidence	Level 3	High confidence in the asserted identity's validity
VERY HIGH confidence	Level 4	Very high confidence in the asserted identity's validity

2.2.1 NIST SP 800-73-3

The NIST SP 800-73 contains all of the technical specifications to interface with the PIV Card for retrieval and use of the identity credentials it contains. This is critical information for the prototype implementation demonstration performed for this thesis. SP 800-73 identifies the PIV

data elements' identifiers, structure and format. Lastly, it gives us the client application programming interface and card command interface for use with the PIV Card [5].

Each PIV card has five mandatory data objects. The Card Capability Container (CCC) facilitates compatibility of Government Smart Card Interoperability Specification applications with End-Point PIV Cards. The Card Holder Unique ID (CHUID) is a number assigned to each PIV Card in accordance with the Technical Implementation Guidance: Smart Card Enabled Physical Access Control System (TIG SCEPACS). The X.509 Certificate for PIV Authentication and its associated private key is used specifically to authenticate the PIV card and the cardholder. Cardholder Fingerprints hold the primary and secondary fingerprints in accordance with FIPS 201. Lastly, the security object is used to map ContainerIDs in the PIV data model to the 16 data groups specified in the Machine Readable Travel Documents (MRTD) [5]. Along with these five mandatory data objects, there are twenty-eight optional data elements of FIPS-201. The optional data elements, essential for the purposes of this thesis, are the X.509 Certificates for digital signatures and key management. The certificate and private key for digital signature is used to sign a document or email and encryption as well. The digital signature private key is PIN protected with "PIN always" meaning that the PIN is always required in order to use the private key. Also essential to PKI, the PIV Card has non-mandatory retired x509 certificates. Up to twenty retired certificates can be stored on the PIV Card.

Table 2.3, taken from SP 800-73-3, shows PIV Card Application key reference values reserved for use [5]. This table identifies the key reference value, the key type, and the security condition for the use for the authentication key and the digital signature key. For the PIV

authentication, PIV Digital Signature, and PIV key management keys, the specific values are found in SP 800-78.

Table 2.3: PIV Card Application Authentication and Key Reference; NIST SP 800-73-3. Available at http://csrc.nist.gov/. Used under fair use, 2014 [5]

Key Reference Value	PIV Key Type	Authenticatable Entity / Administrator	Security Condition for Use	Retry Reset Value	Number of Unblocks
'00'	Global PIN	Cardholder	Always	Platform Specific	Platform Specific
'80'	PIV Card Application PIN	Cardholder	Always	Issuer Specific	Issuer Specific
'81'	PIN Unblocking Key	PIV Card Application Administrator	Always	Issuer Specific	Issuer Specific
See Table 6-1 in SP 800-78	*PIV Authentication Key*	PIV Card Application Administrator	PIN	N/A	N/A
See Table 6-1 in SP 800-78	*Card Management Key*[9]	PIV Card Application Administrator	Always	N/A	N/A
See Table 6-1 in SP 800-78	*Digital Signature Key*	PIV Card Application Administrator	PIN Always	N/A	N/A
See Table 6-1 in SP 800-78	*Key Management Key*	PIV Card Application Administrator	PIN	N/A	N/A
See Table 6-1 in SP 800-78	*Card Authentication Key*	PIV Card Application Administrator	Always	N/A	N/A

2.2.2 NIST Special Publication 800-78-3

In accordance with FIPS 201, NIST SP 800-78 specifies the cryptographic algorithms and key sizes for the PIV system. FIPS 201 identified the PIV Authentication Key, Digital Signature Key, and the Key Management key as cryptographic keys for use with the PIV card [4]. Table 2.4: , from section 3.1 of SP 800-78, identifies the algorithm and key size requirements for each certificate identified in FIPS 201 [6]. After 12/31/2013, and in effect currently, all key sizes must be RSA 2048-bits for all keys. This is critical for the prototpe implementaiton portion of the thesis and the generation of public/private keys for derived credentials.

Table 2.4: Algorithm and Key Size Requirements; NIST SP 800-78-3. Available at http://csrc.nist.gov/. Used under fair use, 2014 [6]

PIV Key Type	Time Period for Use	Algorithms and Key Sizes
PIV Authentication Key	Through 12/31/2013	RSA (1024 or 2048 bits) ECDSA (Curve P-256)
	After 12/31/2013	RSA (2048 bits) ECDSA (Curve P-256)
Card Authentication Key	Through 12/31/2010	2TDEA 3TDEA AES-128, AES-192, or AES-256 RSA (1024 or 2048 bits) ECDSA (Curve P-256)
	1/1/2011 through 12/31/2013	3TDEA AES-128, AES-192, or AES-256 RSA (1024 or 2048 bits) ECDSA (Curve P-256)
	After 12/31/2013	3TDEA AES-128, AES-192, or AES-256 RSA (2048 bits) ECDSA (Curve P-256)
Digital Signature Key	After 12/31/2008	RSA (2048 bits) ECDSA (Curves P-256 or P-384)
Key Management Key	After 12/31/2008	RSA key transport (2048 bits); ECDH (Curves P-256 or P-384)

The application programming interface, the End-Point PIV Client Application Programming Interface, and a set of mandatory card commands are defined in section 5 of NIST SP 800-78. When interfacing with the PIV Card, one-byte identifiers are used that specifies a cryptographic key according to its PIV Key Type [6]. Table 2.5 shows the key reference values for each specific PIV key. The key reference values are critical when interfacing with the PIV card or CAC card during the prototype implementation. In the implementation section, the digital signature key, reference value “9C” will be used in initializing the Gemalto smart cards and also in the demonstration.

Table 2.5: Key Reference Values; NIST SP 800-78-3. Available at http://csrc.nist.gov/. Used under fair use, 2014 [6]

PIV Key Type	Key Reference Value
PIV Authentication Key	'9A'
Card Management Key	'9B'
Digital Signature Key	'9C'
Key Management Key	'9D'
Card Authentication Key	'9E'

The PIV Card algorithm identifier is another one-byte identifier that specifies the algorithm and the size of the key. The algorithm identifiers are detailed in Table 2.6, which is from section 6.2 of NIST SP 800-78 [6]. When initializing the Gemalto Smart Cards for the implementation portion of the thesis, the algorithm identifier "07" for RSA 2048-bit keys is required.

Table 2.6: Algorithm Identifiers; NIST SP 800-78-3. Available at http://csrc.nist.gov/. Used under fair use, 2014 [6]

Algorithm Identifier	Algorithm – Mode
'00'	3 Key Triple DES – ECB
'01'	2 Key Triple DES – ECB
'03'	3 Key Triple DES – ECB
'06'	RSA 1024 bit modulus, $65,537 \leq$ exponent $\leq 2^{256} - 1$
'07'	RSA 2048 bit modulus, $65,537 \leq$ exponent $\leq 2^{256} - 1$
'08'	AES-128 – ECB
'0A'	AES-192 – ECB
'0C'	AES-256 – ECB
'11'	ECC: Curve P-256
'14'	ECC: Curve P-384

Lastly, Table 2.7, from NIST 800-78 summarizes the set of algorithms supported for each key reference value based on time periods.

Table 2.7: PIV Card Keys: Key Reference and Algorithms; NIST SP 800-78-3. Available at http://csrc.nist.gov/. Used under fair use, 2014 [6]

PIV Key Type	Key Reference Value	Time Period for Use	Permitted Algorithm Identifiers
PIV Authentication Key	'9A'	Through 12/31/2013	'06', '07', '11'
		After 12/31/2013	'07', '11'
Card Management Key	'9B'	Through 12/31/2010	'00', '01', '03', '08', '0A', '0C'
		After 12/31/2010	'00', '03', '08', '0A', '0C'
Digital Signature Key	'9C'	After 12/31/2008	'07', '11', '14'
Key Management Key	'9D'	After 12/31/2008	'07', '11', '14'
Card Authentication Key	'9E'	Through 12/31/2010	'00', '01', '03', '06', '07', '08', '0A', '0C', '11'
		1/1/2011 Through 12/31/2013	'00', '03', '06', '07', '08', '0A', '0C', '11'
		After 12/31/2014	'00', '03', '07', '08', '0A', '0C', '11'

2.2.3 X.509 Certificate Policy for the U.S. Federal PKI Common Policy Framework

The certificate policies which are used for all PKI components of all Federal Enterprise Architectures is defined in the X.509 Certificate Policy for the U.S. Federal PKI Common Policy Framework (COMMON). The COMMON requires the use of FIPS 140 validated cryptographic modules. It also states that all keys must be either 2048-bit RSA keys or 256-bit elliptic curve keys along with SHA-256 and SHA-384 hash algorithms [7]. If an entity is associated with a Secure Hash Algorithm version 1 (SHA-1) Federal Root Certificate Authority (CA), then it can issue a SHA-1 certificate. The PKI policies within the COMMON provides security management services of key generation/storage, certificate generation, modification, re-key, and distribution, certificate

revocation list generation and distribution, directory management of certificate related items, certificate token initialization/programming/management, system management functions [7].

Section 3 of the COMMON defines the requirements for the distinguished name characteristics of the X.509 certificates. These vary depending on the status of the person (military, federal employee, contractor, etc.). With the implementation of derived credentials, there has been a change proposal that would add additional distinguished name characteristics for derived credential certificates [10]. For almost all X.509 certificates the Organization will be U.S. Government, the first Organizational Unit will be the department, the second Organizational Unit will be the agency, and the third Organizational Unit will be the structural container (PKI). The Common Name of the certificates vary. Section 3.2 of the COMMON requires that when a party named in a certificate generates its own keys, that party shall be required to prove possession of the private key, which corresponds to the public key in the certificate request. For signature keys, this may be done by the entity using its private key to sign a value supplied by the CA. The CA can then validate the value by using the party's public key [7].

2.2.4 NIST SP 800-63-2

NIST SP 800-63 supplements OMB guidance, E-Authentication Guidance for Federal Agencies (OMB 0404) [12] that defines four levels of authentication (LOA) Levels 1 to 4, in terms of the consequences of the authentication errors and misuse of credentials [8]. Level 1 is the lowest assurance and Level 4 is the highest. For the purposes of this thesis, we are most concerned with levels 3 and 4. Level 3 provides high confidence that the asserted identity is valid, while level 4 provides very high confidence that the asserted identity is valid [8].

LOA-3 provides multi-factor remote network authentication. Level 3 authentication is based on the proof of possession of a key or one-time password through a cryptographic protocol. The key must be, at a minimum, two-factor authentication protected. LOA-3 may be in 3 different token forms; “soft” cryptographic tokens, “hard” cryptographic tokens, and “one-time password” device tokens. For authentication with an LOA-3, the user must prove control of the token (first factor), and must unlock the token with either a password (PIN) or biometric (second factor) [8].

LOA-4 is intended to provide the highest practical remote network authentication assurance. LOA-4 is very similar to LOA-3, except LOA-4 requires the use of a hardware cryptographic token in accordance with FIPS 140 Level 2 (discussed in Section 2.2.5 below) or higher overall with at least FIPS 140 Level 3 physical security. A hardware cryptographic token helps insure two-factor authentication [8].

2.2.5 FIPS 140-2

FIPS 140-2 specifies the security requirements for cryptographic modules utilized within a security system. FIPS 140-2 defines four security levels; Level 1, Level 2, Level 3, and Level 4. For the purposes of this thesis, Level 1, Level 2 and Level 3 are applicable [9].

Security Level 1 is the lowest of the security level. No specific physical security mechanisms are required beyond basic requirement for production-grade components. “An example of a Security Level 1 cryptographic module is a personal computer (PC) encryption board” [9]. Security Level 1 software components on of a cryptographic module to be executed on a general purpose computing system using an unevaluated operating system.

Security Level 2 enhances Security Level 1 by adding the requirement of tamper-evidence. Tamper-evidence can include coatings or seals which reveal if a module has been tampered with or removed. Security Level 2 also requires, at a minimum, role-based authentication. The operator must authenticate to the cryptographic module in order determine permissions of the operator and perform a corresponding set of services. The software component of a Level 2 cryptographic module may only be executed on a general purpose computing system with an operating systems that meets the requirements of FIPS 140 [9].

Level 3 extends on the tamper-evident physical security mechanisms of Security Level 2. Security Level 3 attempts to prevent the intrusion into the cryptographic module. The physical security of Level 3 is intended to have a high probability of detecting and responding to attempts at physical access, use or modifications of the cryptographic module. These physical security mechanism can be achieved through the use of strong enclosures or tamper detection/response circuitry [9].

2.2.6 NIST SP 800-157

The newest release of FIPS 201-2 specifies a new derived credential for use with mobile devices where a PIV card is not practical. The purpose of the derived credential is designed to serve as a Federal government-wide standard for a secure and reliable identity credential that can be used across all agencies. The original FIPS 201 specified that all credentials, certificates and keys, be stored on the PIV card. However, while using a PIV card with a desktop or laptop works well, the use of a PIV card with mobile devices is not practical. FIPS 201-2 authorizes the use of a derived credential for use with mobile device. NIST SP 800-157 is being written to provide the

technical guidelines for implementation of Derived PIV Credentials [10]. Currently, SP 800-157 is in draft form and should be fully published soon. Figure 2.2.1: , from SP 800-157, shows the intended use of derived credentials.

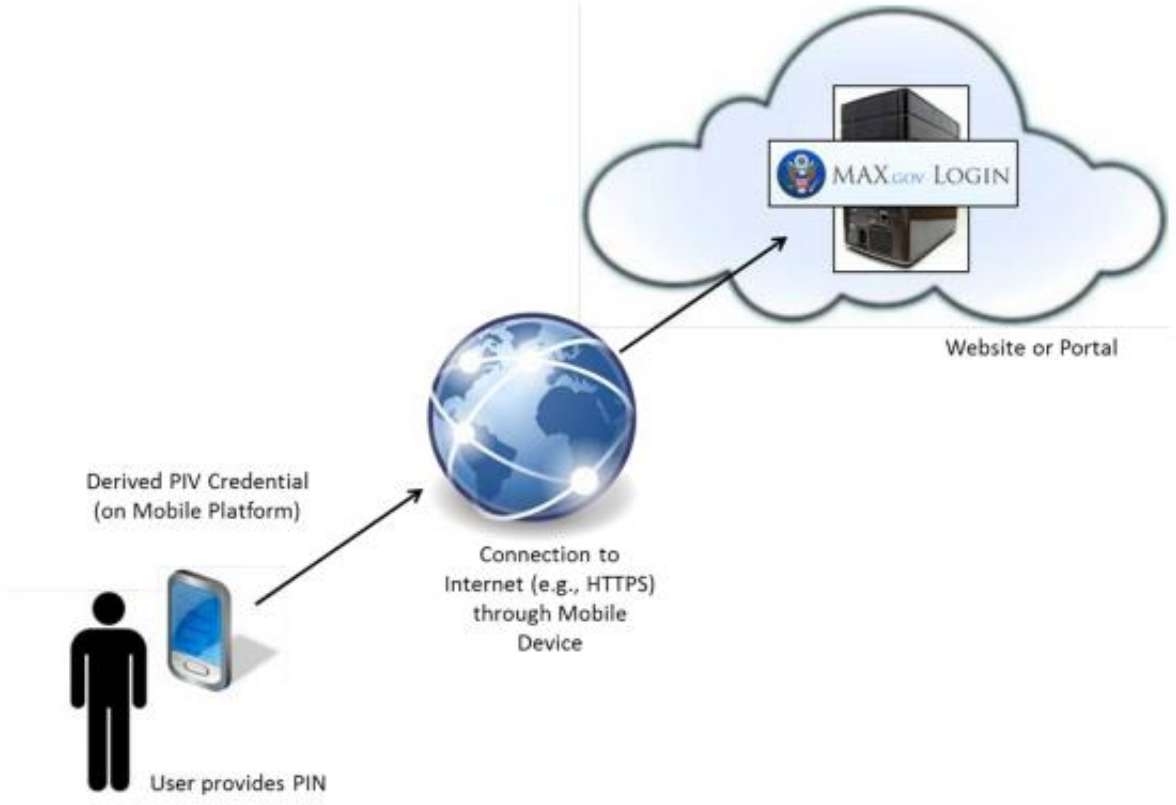

Figure 2.2.1: Use of Derived Credentials; NIST SP 800-157. Available at http://csrc.nist.gov/. Used under fair use, 2014 [10]

NIST 800-63 defines a derived credential "as a credential issued based on proof of possession and control of a token associated with a previously issued credential, so as not to duplicate the identity proofing process" [8]. SP 800-63 requires that a derived credential shall only be issued after the verification of the applicants identity has been completed using the PIV Authentication key on the applicants existing PIV card. A derived credential with LOA-3 can be issued remotely or in person in accordance with SP 800-63. If a remote electronic session is used

for delivery, then encryption shall be used. A derived credential with LOA-4 can be issued in person only and must use a biometric authentication of the applicant. SP 800-157 also does not preclude the issuance of multiple Derived PIV Credentials to the same applicant on the basis of the same PIV card [10].

Derived credentials must be issued as either `id-fpki-common-pivAuth-derived-hardware` (LOA-4) or the `id-fpki-common-pivAuth-derived` (LOA-3) policy of the COMMON [7]. The `id-fpki-common-pivAuth-derived-hardware` and `id-fpki-common-pivAuth-derived` are not currently listed in the COMMON and are both included in a change proposal that would include them in the revision. The cryptographic algorithms and key size requirements are the same as those required for the PIV card authentication certificate from SP 800-78 [6] [10]. For an LOA-4 derived credential, the key pair shall be generated within a hardware cryptographic module that is FIPS 140 level 2 validated or higher. The hardware cryptographic module must also provide level 3 physical security to protect the private key while in storage and that does not permit exportation of the private key. For an LOA-3 derived credential, the key pair shall be generated within a cryptographic module that has been validate to FIPS 140 level 1 or higher [10].

SP 800-157 also specifies the types of approved cryptographic tokens that may be used to use on mobile devices. The token may be hardware or software. There are three types of removable (non-embedded) hardware cryptographic tokens approved for derived credentials storage of the private keys. A secure SD card with cryptographic module of any size (original, mini, or micro) may be used. For most mobile devices, the micro SD card with cryptographic module is most

likely. A Universal Integrated Circuit Card (UICC) is the second approved non-embedded option. The UICC represents the new generation Subscriber Identity Module (SIM) card. The UICC is currently an unlikely solution since it would require the mobile device carrier to allow DoD access to the SIM card in the phone. The last option is a USB token with cryptographic module. This option would not be likely for use with mobile devices as it would require the user to carry around a token that connects to the mobile device through some USB sized cable. Embedded cryptographic tokens embedded within the mobile device may be used with private keys and their associated certificate. These modules may be either hardware cryptographic modules or software cryptographic modules that run on the device. All private keys stored on a hardware cryptographic module must be PIN protected, in the same manner as the keys on a PIV card. For a LOA-3, software cryptographic module key, a password based mechanism shall protect the private key. In addition to an authentication key and certificate, SP 800-157 authorizes the storage of other keys, such as the digital signature and key management private key and corresponding certificates [10].

2.3 Public Key Infrastructure

PKI is defined as a system of public key encryption using digital certificates from Certificate Authorities and other registration authorities that verify and authenticate the validity of each party involved in an electronic transaction [13]. In PKI, a digital certificate and associated public and private key pair are associated to a specific user through a Certificate Authority (CA). A CA is a trusted entity that issues that certificates to users only after verifying their identity. Having a digital certificate from a trusted CA allows any entity to confidently accept the identity of the person associated with the digital certificate [14]. CAs can be arranged in multiple different

topologies with each CA able to use different standards or procedures to verify the identity of the person whom they issue certificates [35]. The topology in which the CAs are arranged depends on the needs of the particular entity using the certificates. A CA hierarchy has a highest level of Root CA that is deployed. Then CAs are deployed that are subordinate to the Root CA. Then the subordinate CAs can issue the certificates to the users themselves [35]. A drawback to this topology is a single point of value at the Root CA level. However, by disturbing the certificate issuing to multiple subordinate CAs, an organization is able to handle a large scale PKI architecture. The figure below represents a CA hierarchy.

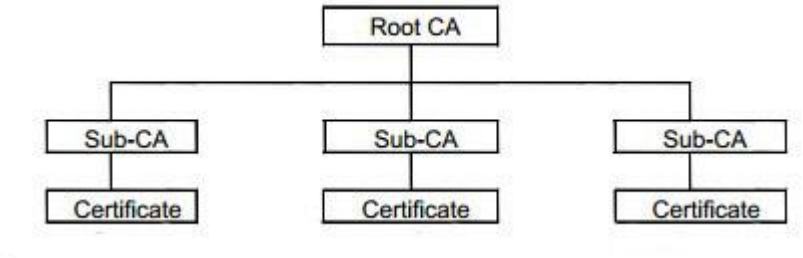

Figure 2.3.1: CA Hierarchical Model

Public key encryption is a cryptographic technique used to securely communicate on insecure networks and verify the identity of a user with a digital signature. In public key encryption, an asymmetric key pair is generated for each user. The keys are different in that knowledge of one key cannot lead to the derivation of the other key. One key is a public key and is made publicly available through some sort of database. The other key is a private key and should only be known/possessed by the user. The concept of public key encryption through asymmetric keys relies on the principal that the user is the only person in the world with knowledge of the private key. While the keys are different from each other, they are related in the fact that one key

can be used for encryption while the other can be used for decryption. Public key cryptography through the use of RSA keys is secure based on the fact that factorization of the large integers used to create the keys, while possible, has so far resisted all attempts to algorithmically feasible computation [14].

After the generation of a public/private key pair, a Certificate Signing Request (CSR) is created. The CSR contains identifying information that will be used to populate the user's digital certificate that will be signed by the CA. The CSR is an encrypted file that contains information such as common name, organization name, organizational unit name, country, email, and the public key. Once the CSR is ready, it is sent to the CA for approval. If approved, the CA will generate a X.509 certificate signed by the CA private key that can now be used by the user [14].

Public key cryptography enables a number of secure digital services that were unavailable or not possible with symmetric ciphers. Encryption through the use of public key cryptography is rather simple; data can be encrypted with the public key of a user and then decrypted with the private key. However, because encryption and decryption using 1024-bit or 2048-bit asymmetric keys can be computationally expensive, often a one-time symmetric key is generated and used to encrypt the data, encrypted with the public key of the intended recipient, and exchanged [14]. When extracted, the symmetric key is then used to encrypt and decrypt the data for the remainder of the sessions. Encryption in public key cryptography provides data confidentiality and data integrity. Data confidentiality means that the data encrypted can only be decrypted by the intended recipient. Data integrity means that the data being sent has not been altered during the transmission. To encrypt data with public key cryptography, a one-time symmetric key is generated. The

message is then encrypted with the symmetric key. The session key itself is then encrypted with the recipient's public key. The encrypted session key and the encrypted message are then sent as one message to the recipient. Once received, the encrypted message and encrypted session key are retrieved by the recipient. The symmetric key is then decrypted with the recipient's private key. Using the decrypted symmetric key, the recipient can then decrypt the message. As mentioned above, only the recipient has knowledge of the private key, and thus no one else can decrypt the message [14] [15]. Figure 2.3.2 below shows the encryption process while Figure 2.3.3 shows decryption with public key cryptography [15].

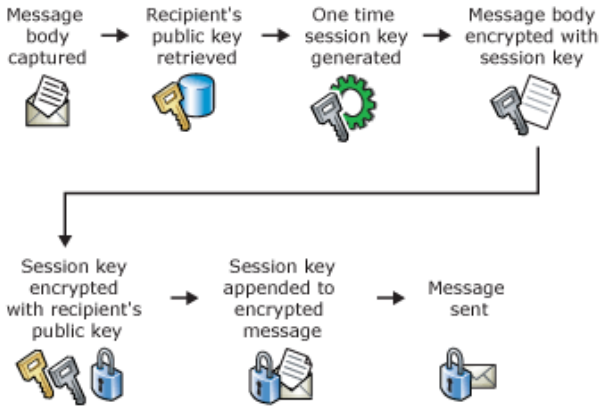

Figure 2.3.2: Public Key Data Encryption; Understanding Public Key Cryptography. Available at http://technet.microsoft.com/en-us/library/aa998077(v=exchg.65).aspx. Used under fair use, 2014 [15]

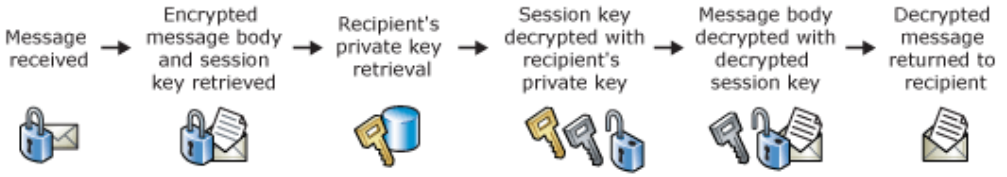

Figure 2.3.3: Public Key Data Decryption; Microsoft Exchange. Understanding Public Key Cryptography. Available at http://technet.microsoft.com/en-us/library/aa998077(v=exchg.65).aspx. Used under fair use, 2014 [15]

Another service of public key cryptography is digital signatures. Digital signatures are a cryptographic technique for authenticating the owner or creator of a document, or to signify ones

agreement with a documents content. Digital signatures provide authentication, data integrity, and non-repudiation. Authentication is the confidence that the message came from the person who it says it came from. Non-repudiation is the inability of the sender to later deny that the message was sent by them.

To digitally sign a document or message, a hash value of the message is calculated. The hash value is then encrypted with the sender's private key. The encrypted hash value is appended to the message as a digital signature and the message is sent. When the recipient receives the message, the digital signature is retrieved and so is the message. The recipient calculates a hash of the message. Then, the encrypted hash value is decrypted using the sender's public key. The recipient compares the decrypted hash with the hash value calculated by the recipient and if the values match, the digital signature is valid. Since only the public key which correlates to the private key of the digital signature can decrypt the message and since the sender is the only person with access to his private key, it can be assumed that the sender indeed sent the message. Figure 2.3.4 shows the process of a user digitally signing a message and Figure 2.3.5 shows a recipient verifying the digital signature [15].

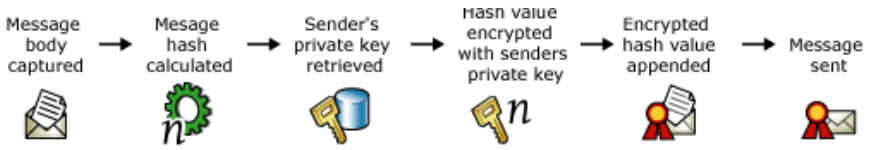

Figure 2.3.4: Digitally Signing a Message; Microsoft Exchange. Understanding Public Key Cryptography. Available at http://technet.microsoft.com/en-us/library/aa998077(v=exchg.65).aspx. Used under fair use, 2014 [15]

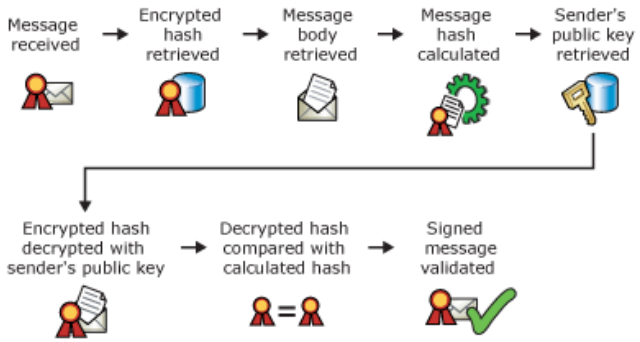

Figure 2.3.5: Verifying a Digital Signature; Microsoft Exchange. Understanding Public Key Cryptography. Available at http://technet.microsoft.com/en-us/library/aa998077(v=exchg.65).aspx. Used under fair use, 2014 [15]

Both message encryption and digital signatures can also be used in conjunction with each other to provide the most secure exchange of communication possible. A hash value of the message is calculated and encrypted with the sender's private key. The signed (encrypted) hash value is then appended to the message as a digital signature. A one-time symmetric session key is generated and used to encrypt the entire message. Next, the session key is encrypted with the recipient's public key and is appended to the encrypted message. The entire message is then sent to the recipient. When the message is received, the encrypted message and the encrypted session key are retrieved. The session key is decrypted with the recipient's private key. The message is then decrypted with the session key. A hash is taken of the newly decrypted message. The recipient then decrypts the encrypted hash value using the sender's public key. The decrypted hash value is then compared to the calculated hash value of the recipient. If the values match than the signature is valid and the message is returned to the recipient. Figure 2.3.6 show the process of combining encryption and digital signatures with public key cryptography while Figure 2.3.7 shows the decryption and verification process [15].

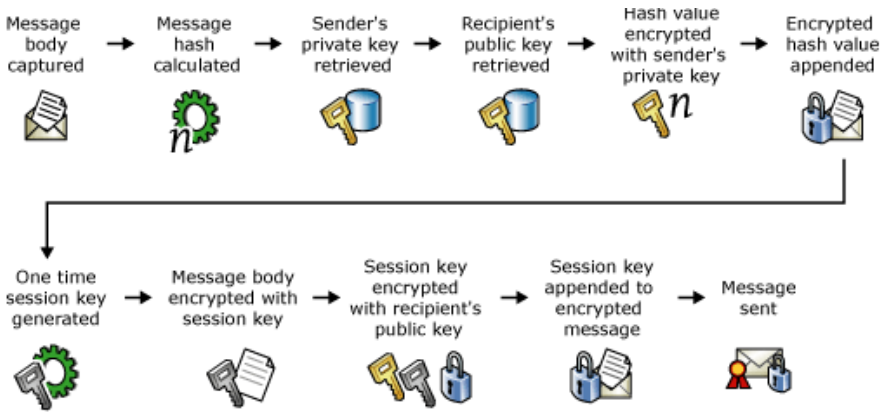

Figure 2.3.6: Digitally Signing and Encrypting a Message; Microsoft Exchange. Understanding Public Key Cryptography. Available at http://technet.microsoft.com/en-us/library/aa998077(v=exchg.65).aspx. Used under fair use, 2014 [15]

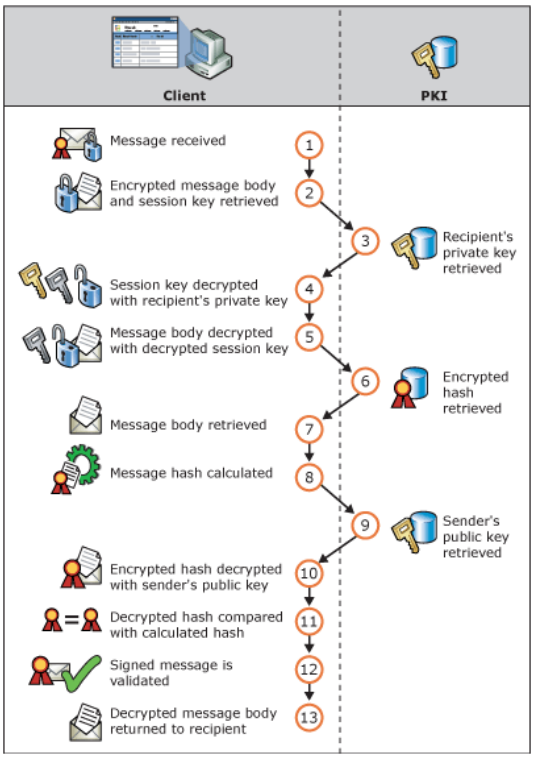

Figure 2.3.7: Decryption and Verification Process with Public Key Cryptography; Microsoft Exchange. Understanding Public Key Cryptography. Available at http://technet.microsoft.com/en-us/library/aa998077(v=exchg.65).aspx. Used under fair use, 2014 [15]

2.4 X.509 Certificates

X.509 specifies the standard format for public key certificates, certificate revocation lists, attribute certificates, and a certification path validation algorithm. The X.509 certificate profile is outlined in RFC 5280 [16]. As discussed above, the CA signs and issues a X.509 certificate that binds a public key to the distinguished name characteristics in the X.509 certificate. The X.509 certificates are then published to a database where they can be reached by all users. There are several required fields for a X.509 certificate. The version field identifies the version of the certificate. If extensions are used, the version must be 3. Since DoD X.509 certificates all have extensions, versions 3 is always used.

The serial number is a positive integer assigned by the CA. Each certificate has a unique serial number. The signature field contains the algorithm identifier for the algorithm used by the CA to sign the certificate. The issuer field identifies the entity that signed and issued the certificate. The issuer field contains the contents of the distinguished name characteristics of the issuer. The distinguished name characteristics can consist of country (C), organization (O), organizational unit (OU), distinguished name qualifier, state or province, common name (CN), and serial number. The validity field defines the period of time in which the certificate is valid. The validity field will have a valid begin and a valid end date. The subject field contains the identity information about the entity associated with the public key stored in the certificate. In addition to the distinguished name characteristics as defined above, the subject field can also contain and email address for the subject. The subject public key info field carries the public key and identifies which algorithm was used to generate the key. In a version 3 certificate, the unique identifiers field is utilized to handle the

possibility of reuse of a subject and/or issuer names over time. Lastly, the extensions fields of a version 3 X.509 certificate is a sequence of one or more certificate extensions [16]. Figure 2.4.1 below shows the contents of an X.509 certificate [29].

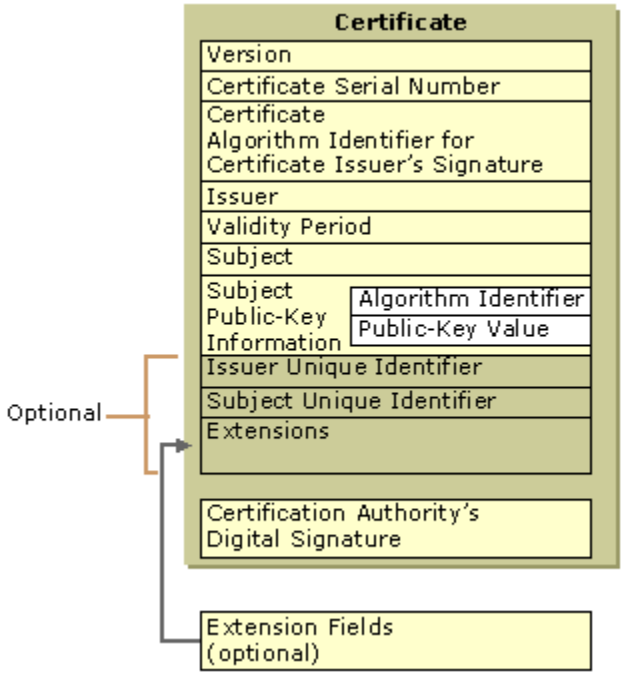

Figure 2.4.1: X.509 Certificate; Microsoft Tech Net. Digital Certificates. http://technet.microsoft.com/en-us/library/cc962029.aspx. Used under fair use, 2014 [29]

The extensions added to X.509 certificates allows for the certificate to provide additional attributes with users or public keys. Extensions on a certificate can be critical or non-critical. A critical extensions cannot be ignored and if a system does not recognize the extensions then it must reject the certificate. Non-critical extensions may be ignored if not recognized but must be processed if recognized. The extensions can contain an object identifier (OID) or ASN.1 structure.

The authority key identifier extensions provides a means of identifying the public key corresponding to the private key used to sign a certificate. The key usage extensions defines the

purpose of the key contained in the certificate. The key usage can be set to multiple values, including non-repudiation, digital signature, certificate signing and more. The certificate policies extensions contains OID's that highlight the terms of the policy under which the certificate is used. Applications with specific policy requirements will compare the OID's in the certificate policies with those that are accepted. If the extension is critical, the application must match the OID's, if it is non-critical then the application may ignore any unrecognized OID's. The subject alternative name allows additional identities to be bound to the subject of the certificate. The basic constraints extension identifies whether or not the subject is a CA. The maximum depth of the basic constraints extension provides the depth of the valid certification paths that include the CA certificate. The extended key usage extensions indicates the purposes for which the certificate public key may be used in addition to the key usage extension values. This extension is normally only seen in end entity certificates. The CRL distribution point's extension identifies how the CRL information is obtained. The last extension seen in DoD CAC certificates is the authority information access extension. This extension defines the use of the authority information access extensions in a CRL [16]. There are many more extensions that can be added to X.509 certificates, but for the purposes of DoD CAC X.509 certificates, these are the only extension fields that will be seen [16].

The certification path validation verifies a binding between the distinguished name characteristics (and possibly the subject alternate name) with the subject's public key. The basic constraints and certificate policy extensions allow the validation process to be automated. At the top of the validation path is the trust anchor. In a hierarchical PKI structure, the root CA is the trust anchor. Since a hierarchical PKI structure has intermediate CA's under the root CA, applications may rely different trust paths that begin with any set of trust anchor. The X.509 certificate will

contains the certification path [16]. A certificate will be trusted by a computer if the CA's in the certification path are installed on the computer. Most computers come pre-loaded with several well-known root CA's. Other CA's, such as the DoD CA's, must be installed on a computer before the certificates are trusted.

2.5 Prior Work

Since the term derived credential is such a new and unproven topic, there is not much prior work that has been done in the field of derived credentials themselves. Currently, NIST, DISA, and several contracted organizations are undergoing pilot projects that utilize the derived credential. These projects focus more on the implementation of the derived credential and how it will be used once it is loaded/stored on a mobile device. To be best of my knowledge, there has not been any published documentation which specifically defines an architecture for creation and delivery of a derived credential.

Currently, work-arounds to derived credentials can be achieved through use of a PIV card reader integrated with a mobile device. While effective, this method would require that the mobile device be able to attach to a contact PIV card reader via USB or Bluetooth. It would also require that the user carry around a Bluetooth or wired card reader with them at all times. Modern PIV cards are also RFID capable. Therefore, the RFID capable CAC could contactlessly communicate with the Near Field Communication (NFC) on the mobile device. However, only select mobile devices are NFC capable and NIST PIV specification prohibit cryptographic communication over NFC.

In September of 2012, Francisco Corella and Karen Lewison published a whitepaper titled "Techniques for Implementing Derived Credentials" [11]. Their focus was techniques that can facilitate the implementation, deployment, and use of derived credentials. The foundation of their method is the ability to eliminate the need for X.509 certificate storage on the mobile device itself. The device is given a key pair which is stored either in the web browser, front-end, or elsewhere on the mobile device. The paper does not specify how the key pair would be generated or how it would be delivered to the device. In a second method, the authors describe a two- and three-factor authentication method. In this method, the authors show how instead of encrypting the private key on the mobile device with a key-encryption key derived from the PIN, the authors use a method where the complete key pair is regenerated from the PIN [11]. This method, they argue, has the effect that all PIN values yield a well-formed key, pair, making it impossible to test PINs offline. Not having an X.509 certificate on a mobile device leaves the mobile device without an actual derived credential. In general, their method of regenerating a key pair from PIN and using that key pair for authentication is no different than a username and password authentication method. While the authors methods shown in this paper are very creative and innovative, the still leave the mobile device without an actual X.509 certificate capable of digital signatures and authentication and thus defeat the premise of having a derived credential on the mobile device.

2.5.1 Hardware vs. Software Cryptographic Module

While cryptographic modules can be implemented using both hardware and software, there are security implication of both methods. While both modules can prove effective, it has been shown that software-based cryptographic modules are less secure than hardware. In PKI, the

private key must be kept secure and private, therefore the storage of the key is critical to security. A hardware cryptographic module that can carry out the cryptographic operation on a microprocessor and store the private key are highly secure [32]. A hardware token may be vulnerable to malicious software residing on the device the token is connected to. When the user PIN is entered to unlock the token, malicious software on the device could have access to the token. This access, however, is limited to the amount of time the token is unlocked or connected to the device [32]. If a user's hardware token is physically stolen, an adversary would have control of the private keys stored on it, however, they would need the password or PIN to unlock the token itself. A side channel attack can be carried out against a hardware token, if the adversary has access to the physical token and is able to learn or crack the password to unlock it [32].

A smart card, which contains a hardware token, provides protection against the attacks listed above. While there is still somewhat a threat from malicious software, the appropriate anti-virus software can significantly reduce the threat of this attack. The modern smart card also allows for the protection of side-channel attacks by limiting the number of guessed password attempts an adversary can enter before the token locks up [32].

A software token, in relation to a hardware token, is significantly less secure. Since the key is stored in software and retrieved for cryptographic operations, the keys must be stored in software on all devices in which the user will work on. This increases the security vulnerabilities of software keys. An obvious attack against a software token is physical access. Any device containing a key in a software token can be physically accessed by an adversary. This physical access can occur through theft of the actual machine, or by an adversary simply accessing a machine that is left on

[32]. Another threat to software tokens is malicious software. Viruses, worms, trojan horses and more types of malicious software are very present in modern computing [32]. The countermeasure against these threats is to use password encryption for the stored keys. However, if the encrypted token is stolen somehow, an offline dictionary attack can be carried out by the adversary. For protection against an offline dictionary attack, a long, randomly generated password would be required. Since most users are not capable of remembering such passwords, most of the time common words are used for passwords which make them less secure [32].

3 DoD PKI

This section describes the DoD PKI infrastructure implemented by the CAC and X.509 certificates.

3.1 Common Access Card

In response to HSPD-12 and the subsequent publications from NIST, the Department of Defense instituted a department wide PKI architecture through the use of a CAC. Authentication of an individual is fundamental component of physical and logical access control process to enable physical access to buildings and controlled spaces, and it provides access to defense computer networks and systems. The CAC is a credit card-sized smart card, and is the standard identification for active duty uniformed service personnel, selected reserve, DoD civilian employees, and eligible contractor personnel. The other primary purpose of the CAC is to control physical access to buildings and controlled spaces, and to provide access to DoD computer networks and systems [17]. For physical access, the CAC CHUID can be read via RFID/NFC in addition to visual verification. The CAC provides two-factor authentication in that the user must for possess the card, and then the user must know the PIN which is only known by the user. The front of CAC consists of visual identification of a user through a facial picture, name, and rank, affiliation, service/agency and expiration date along with a seal of the department. It has a bar code on both the front and back of the card and a magnetic strip on the back which can be used to pull general identification information off the card. Lastly, the CAC has an Integrated Circuit Chip (ICC). The following figure shows a sample CAC [17]:

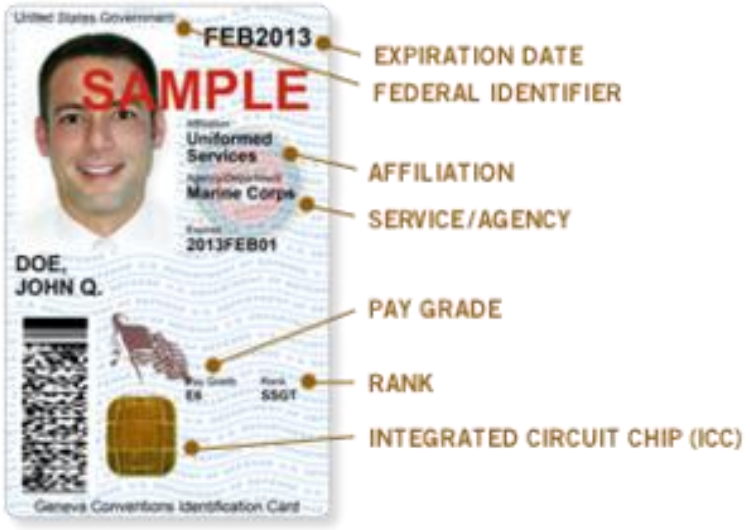

Figure 3.1.1: Sample Common Access Card; DoD Common Access Card. Available at http://cac.mil/. Used under fair use, 2014 [17]

The ICC is where the PKI X.509 certificates and keys are contained, as well as, the user PIN and other PKI relevant information. The ICC can also perform cryptographic operations on the chip itself. In accordance with FIPS 201, every CAC contains a user authentication certificate and corresponding private key. Also, every CAC user who has network and email access has a digital signature certificate and private key stored on the ICC. All DoD keys are 2048-bit RSA in accordance with NIST Special Publication 800-78-3 [6]. All private keys are protected with a user 6-digit PIN. The CAC interacts with a electronic device via a CAC reader and software middleware. Upon insertion of the CAC card, the user is prompted for the PIN. If the correct PIN is entered, then the middleware interact with the ICC. For authentication, the digital certificates can be accessed by the middleware. For digital signatures or encryption, since the private keys on the ICC can never be exported, all cryptographic oporations utilizing the private keys must be done on the CAC itself. The private keys on a CAC are also protected such that after 3 unseccessful PIN attempts, the CAC will become locked and unusable.

3.2 DoD PKI Architecture

The DoD PKI architecture is organized as a hierarchical PKI. As discussed in Section 2.3, a hierarchical PKI architecture means that there is a Root-CA with subordinate intermediate CAs. The subsequent intermediate CAs then have authority to issue credentials to users. In the DoD hierarchy, there is a DoD Root CA 2. Under the DoD Root CA are several intermediate CAs currently numbered 11-30. There are intermediate CAs that issue Email (Digital Signature) certifictes and just a DoD Intermediate CA that issues with authentication certificates. Under the intermediate CAs are the subscribers certificates, stored on the CAC. Figure 3.2.1 shows a broad overview of the DoD PKI hierachy of the Root CA 2 [18].

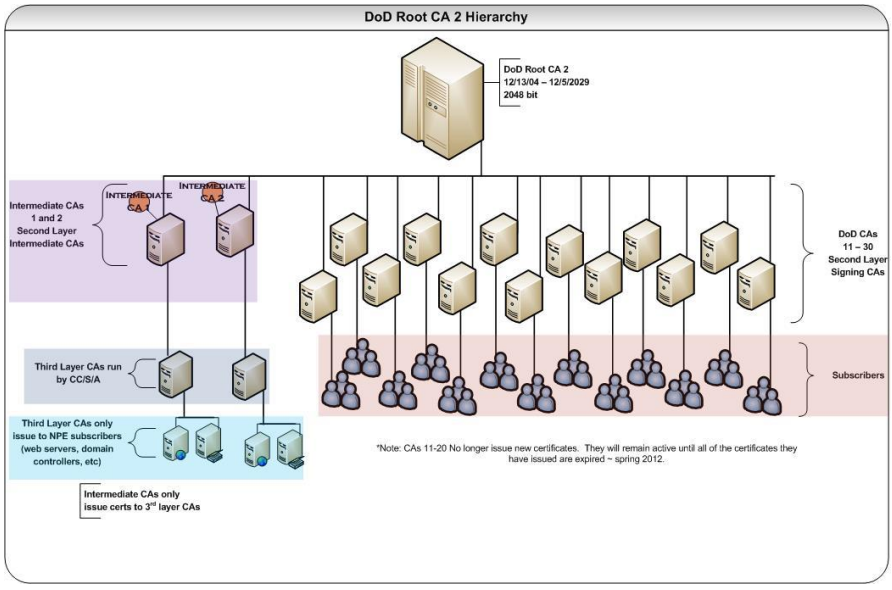

Figure 3.2.1: DoD Root CA 2 Hierarchy; Military CAC. Available at http://militarycac.com. Used under fair use, 2014 [18]

The DoD PKI participants are the DoD Policy Management Authority (PMA), the CAs, the registration authorities, and the subscribers [19]. The DoD registration authorities are the verifying officer/local registration authority (VO/LRA). The VO has several critical responsibilites

in the DoD PKI portion of the CAC distirbution process. The VO verifies the identity of a subscriber and uses a RAPIDS workstation to issue the CAC to the card reciptient. The VO assists in the management of the recipients keys and certificates as well as and updating that is required to any information or keys on the ICC. The VO is responsible for executing revocation request received from authorized sources [20].

3.3 DoD X.509 Certificates

The CAC stores three primary certificates for PKI funcationality for the subscriber; authentication certificate, digital signature certificate and the key management certificate. Appendix A contains the actual information of a CAC authentication, digital signature, and key mangemante certficates. The certificates contains the following verison 3 fields; version, serial number, signature algorithm, signature hash algorithm, issuer, valid from, valid to, subject, and public key. For the issuer and subject fields, the following distinguished names acrynymes are used: common name (CN), organizational unit (OU), organization (O), and country (C). The issuer CN is either specific to the CA that issues the certificate.

The information referenced from the DoD X.509 certificate is from an actual user certificate. Therefore, some of the information is unique to the specific user. For this users authentication certificate, the CN is DoD CA-30. However, for the digital signature and key management key, the CN is DoD Email CA-30. Since DoD has multiple intermediate CAs, the intermedia CAs may have different numbers. The CN for the subject is the unique name of the subscriber in accordance with COMMON. The organization is the U.S. Government. The country is US. There are multiple organizational units for each certificate. The OU will contain the

department (DoD), the branch of service (Coast Guard for this user), and PKI. The certification path of the authentication certificate is from the DoD Root CA 2, to a DoD intermediate CA (DoD CA-30), to the subscriber as seen below. For the digital signature and key management keys, the path still has DoD Root CA 2, but the intermediate CA is DoD Email CA-30. Figure 3.3.1 below are snapshots taken from actual DoD Authentication certificates.

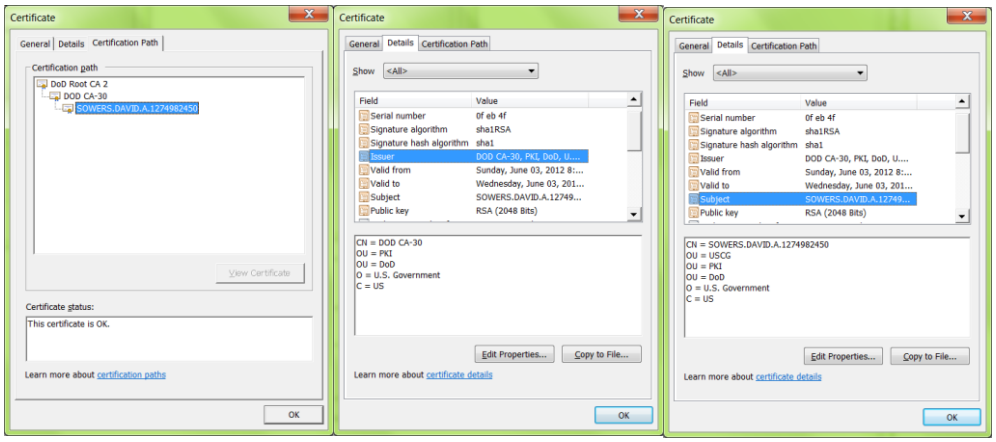

Figure 3.3.1: DoD CAC Authentication Certificate

There are several key extension added to the certificates including: authroity key identifier, crl distribution points, certificate policies, subject key identifier, authroity information access, subject alternative name, key usage, and extended key usage. As discribed in Section 3.2, the authrity key identifier provides a means for relating the public key to the private key used to sign the certificate. As seen in Appendix A, the authentication authority key identifier for this user is keyid:08:4E:D5:A4:3C:2A:04:9B:93:1B:B7:04:08:8E:74:B9:06:7C:0D:A3, while the digital signature and key management certificate authority key identifier is keyid:35:61:66:28:09:BC:56:25:5B:8B:CC:BF:81:5E:61:2C:30:39:D3:21. These values are unique to this specific user.

The CRL distirbution points provides the location of the crl file relating to this specific certificate. For the authentication certificate, the CRL distirbution point for DoD EMAIL CA-30 is URI:http://crl.disa.mil/crl/DODCA_30.crl while the digital signature and key management key CRL distribution points is URI:http://crl.disa.mil/crl/DODEMAILCA_30.crl. The authrotiy information access extension has the CRL access information for each certificate. The certificate polices are the same for all three certificates and are given in the form of OID's. The first OID, 2.16.840.1.101.2.1.11.9, coorelates to DoD External Certificate Authority (ECA) Medium Hardware Assurance (id-US-dod-mediumhardware) and validates that the certificates are generated and stored on a FIPS 140 Level 2 hardware token (smartcard) only [19]. The second certificate policy, 2.16.840.1.101.2.1.11.19, is Medium Hardware assurance certificates with 2048-bit RSA/DSA keys which are generated or stored in a FIPS 140 Level 2 hardware module [19].

The subject alternative name for the authentication certificate only has the principal name listed. The digital signature and key management keys also have the email of the user listed. In these other certificates, it would look like this "RFC822 Name=First.M.Last@agency.mil". The key usage of all three certificates is marked as a critical extensions. For the authentication certificate, the key usage is digital signautre. Section 3.2 shows how the digital signature can be used to provide authenticaiton. The digital sigature certificate key usage propterties are digital signature and non repudiation. Lastly, the only key management certificate key usage is key encipherment. The extended key usage extension clearly lables the key usages for this certificate and has a cooresponding OID. The the authentication certificate, the extneded key usages are smart card longon (OID=1.3.6.1.4.1.311.20.2.2) and Client Authentication (OID=1.3.6.1.5.5.7.3.2). The

digital signature certificate has both extended key usages that the authentication certificate has, and it also includes an extended key usage of Secure Email (OID=1.3.6.1.5.5.7.3.4). There are no extended key usage properties for the key management certificate.

4 Derived Credentials

This section defines the 4 architectures for issuing derived credentials to mobile devices.

4.1 Overview

Derived Credentials for mobile devices have recently become hot topic. The derived credential is a new credential that is derived from the credential on the PIV card or CAC. In the most recent revision to FIPS 201, NIST included the approval for use of a Derived PIV Credential. We will simply refer to a Derived PIV Credential as a derived credential moving forward. The draft version of the special publication for derived credentials has been release, and should be finalized in the near future. Once SP 800-157 is finalized, the Department of Defense will move quickly to adapt the use of derived credentials across all agencies. In recent years, the significant gains in mobile device capabilities allow for the access to information and applications wherever and whenever needed. Mobile devices are one commonly carried device which an individual takes with them everywhere they go, at all times of the day. The ability to securely access DoD email, networks, and files from a mobile device would help increase performance and efficiency of the entire department. Currently, the CAC provides a reliable, secure way to distribute PKI certificates and keys to all eligible members of the Federal government and select contractors. Since the integration of the CAC with mobile device would require bulky add-on devices, the most logical answer to mobile device PKI is the derived credential. The derived credential, if utilized to its full potential, could provide the same reliable and secure PKI for a mobile device that the DoD now gets from the CAC. The premise behind the derived credential is that by providing proof of

possession of a valid PIV card, a user can receive a derived credential. While the PIV credential itself is the same for a PIV card and a derived credential, the form factor is different. Currently, there are technologies in place that could leverage derived credentials through software cryptographic modules and hardware cryptographic modules. Instead of a user making an appointment at a RAPIDS Identification Card Issuance Office, the user would be able to remotely provision a derived credential (at a lower assurance level) by successfully authenticating through their PIV card [30]. With the number of derived credentials that would need to be issued, the deployment of derived credentials must provide a secure, yet efficient method of generation and delivery. Requiring DoD employees to make appointments to visit the RAPIDS Identification Card Issuance Office and have the registration authority officer issue a new credential for mobile devices would essentially just be a new credential and would not be a derived credential as defined by SP 800-157. Through the process of evaluating the requirements and standards imposed by NIST, this thesis develops 4 distinct architectures for remotely generating and delivering derived credentials to DoD mobile devices that meet all NIST requirements.

4.2 Software Generated LOA-3 Derived Credential

The first two solutions proposed here for the generation and delivery of a derived credential uses a software cryptographic module that generates the public/private key pair for the user using FIPS 140 validated Level 1 software. Unlike the CAC, which utilizes a hardware token to generate and store the PIV keys, a software token stores the keys in flash memory on the mobile device. The keys are then protected (encrypted) with a PIN or password which adheres to the entropy requirements of NIST SP 800-63-1 Table 6 Level 2 [8] for memorized secret tokens. The Level 2

token requirements state that the memorized secret may be a randomly generated PIN consisting of 6 or more digits, a user generated string consisting of 8 or more characteristics chosen from an alphabet of 90 or more characters, or a secret with equivalent entropy. According to SP 800-63-1 Table A.1, an 8 character password provides 18 bits of entropy [8]. SP 800-63-1 also states that "the verifier shall implement a throttling mechanism that effectively limits the number of failed authentication attempts an Attacker can make on the Subscriber's account to 100 or fewer in any 30-day period" [8]. When trusted applications are developed for DoD, they will attempt to access the software token stored credential by prompting an input of the password or PIN by the user. Once the key is unlocked, the application would have access to the key. A security measure of a derived credential stored in software is that the token must lock after a certain number of improper attempts at the PIN. This would stop an offline dictionary attack by the adversary by limiting the number of attempted guessed PIN or passwords.

As was discussed in Section 2.5.2, a software generated and stored derived credential would be less secure than a hardware token. The threats against physical access and offline attacks make the software token a security liability [32][33]. Another problem with a software token is that the software token itself does not know what application is accessing it. As long as the key is entered, the application would have access to the key. Therefore, multiple pop-up authentication methods may be required to ensure on proper key usage [34]. According to the SP 800-157 draft, a software token derived credential would limit keys and certificates generated to LOA-3 [10]. However, despite only enabling LOA-3 credentials, the architecture provide a secure, efficient, and remote method of generating a derived credential for a user. Since the software tokens are less secure, the validity dates can also be adjusted for a shorter period of time. This would require the

user to more frequently authenticate for a new derived credential. If a new credential is issued, the retired keys would need to be stored to access any files that were encrypted or signed by the previous keys. However, since the keys are stored in memory, there is more space for retired tokens.

4.2.1 CAC Signed Derived Credential

From Section 3.3, we know that the DoD PKI infrastructure is organized in a hierarchical architecture. The highest level being the DoD Root CA with the next level is a DoD intermediate CA. The lowest level, currently, is the actual user certificates. The premise behind this proposed architecture is that the user themselves, would be able to generate and sign their derived credential using their CAC as a CA to sign the CSR. Thus, this architecture would extend the DoD hierarchy to include another level, the derived credential, below the user certificate. A DoD CAC holder, who has already been issued Digital Signature and Key Management keys, has already been vetted and approved for permission granted by the X.509 certificates on the CAC. The method would allow the user to use a federally-approved application to generate a public/private key pair and CSR, then use the CAC to sign the CSR and return a X.509 certificate which in turn has the CAC user certificate in the CA chain. One of the biggest advantages to this method is the remote generation and delivery of the derived credential to the user. The user themselves are now authorizing the derived credential, based off the permissions approved on their CAC. If the CAC certificates become invalid (revoked or expire) than the derived credential certificate itself would expire.

One of the main disadvantages of this architecture is that current digital signature certificates on the CAC do not have "certificate signing authority" privileges. Therefore, DoD employees who are authorized the use of derived credentials would have to be reissued their digital signature certificate on their CAC. By doing this, however, the DoD would be able to control who is authorized the use of derived credentials. Any user trying to sign a CSR with a digital certificate that does not have certificate signing authority, would be denied this privilege. The DoD intermediate CA's also have a path length constraint of 0, which means that they are the last CA in the certification path. This number would have to be increased by one since the users CAC certificate would not be acting as a CA under the intermediate CA's. The application would be set up such that after the CSR is approved, the extensions added to the new derived credential certificate would be added.

Currently, there is not a certificate policy OID for derived credentials. The closest applicable OID, 2.16.840.1.101.2.1.11.5, certificates issued to users in software format [31]. Since the keys are being generated in a software cryptographic module, they would be delivered to the mobile device through a PKCS#12 file. The software token generation of keys limits the derived credential to only provide LOA-3. Once the derived credential certificate is signed, it would be packaged in a PKCS#12 file with the corresponding private key. A PKCS#12 file is an encrypted, password-protected file. A QR-Code will be generated with a link to the PKCS#12 file. The user is able to scan the QR-Code with a mobile device application and download the PKCS#12 file to the mobile device. As discussed above, the derived credential would then be stored on the mobile device in a software cryptographic module and protected by encrypting it under a key derived from a randomly generated 6-digit PIN or from a password with equivalent entropy, in accordance with

SP 800-157. Once stored, the derived credential is then ready for use by approved DoD mobile applications.

The steps of signing a CSR with a CAC digital signature certificate is as follows:

1 The user would log into a DoD workstation. Using the CAC authentication certificate (two factor authentication), the user would authenitcate with a certificate and PIN to an approved computer software application that is FIPS 140 Level 1 approved for generation of public/private key pairs.

2 If authentication is successful, the application will generate a 2048-bit RSA public/private key pair for the user.

3 After the key pair is generated, a CSR is generated using critical information pulled from the CAC digital signature certificate (name, email address, permissions) and the public key.

4 The CSR is then sent to the CAC for an aproval signature. Since the private key never leaves the CAC, the cryptographic operation of signing the CSR actually happens on the ICC of the CAC.

5 After approved by the CAC, the derived credential extensions would be added to the X.509 certificate.

6 The signed X.509 is published to the DoD PKI certificate repository.

7 The X.509 certificate and associated private key would be packaged into a password-protected PKCS#12 file.

8 A QR-Code of the PKCS#12 file would be generated and displayed. The user would scan the QR-Code with their mobile device to transfer the PKCS#12 file from the computer to their mobile device.

9 The PKCS#12 file would then be extracted into a software cryptographic module on the mobile device in accordance with SP 800-157.

4.2.2 Server-Driven Derived Credential

One of the easiest software driven architectures to implement the generation and delivery of derived credentials is a server driven method. A web portal for derived credentials would need to be built by the DoD for derived credential services. The DoD would need to create a new intermediate CA. For the purposes of this architecture, we will call this new intermediate CA the DoD DC CA. The DoD DC CA would then be the certificate authority on all CSR's from the web portal. The only requirement from DoD would be setting up the web portal with the proper software cryptographic modules to generate the RSA 2048-bit keys and the certificates. Once mobile devices are ready to receive derived credentials, the following steps could be taken for generation and delivery. A CAC authentication would be required to access the web portal. As we discussed earlier, the applicant must demonstrate possession and control of their PIV card via CAC authentication. The web portal would then be responsible for generating the RSA 2048-bit public/private key pair and CSR. The web portal would then use the DoD DC CA to sign the CSR, and return the certificate and key to the user via a PKCS#12 file through QR-Code. The transmission of the encrypted, password-protected, PKCS#12 file would be through a secure connection with the mobile device. Similar to the CAC signed certificate, the server driven certificate would then be stored into protected software token on the mobile device.

One advantage to having a web portal within the DoD intranet would be access control. A CAC authentication certificate would be required for access and since it resides on the DoD

intranet, it would be provide easy oversight for the DoD. Since the CA resides on the DoD intranet, it would be easy to institute and maintain the derived credential PKI. The web portal would also provide a remote method for generating and delivering the derived credential. This will provide a secure, efficient and instantaneous delivery of the credential and take the bourdon off of the RAPIDS Identification Card Issuance Offices. A DoD user would not need to visit an ID center, but could log into the web portal from any computer with a CAC reader and middleware.

The steps for a server driven derived credential are as follows:

1 The user would log into a DoD web portal designated for derived credentials using the CAC authentication certificate and PIN (two factor authentication).
2 Using a software cryptographic module, the web portal would generate a 2048-bit RSA public/private key pair.
3 Using critical information pulled from the CAC certificates (name, email, permissions), the web portal generates a CSR for the user derived credential.
4 The CSR is then approved and signed by the DoD DC CA. The result is he users derived credential X.509 digital certificate with the approved derived credential extensions.
5 The signed X.509 is published to the DoD PKI certificate repository.
6 The X.509 certificate and the private key are packaged into an encrypted, password-protected, PKCS#12 file. A link to the PKCS#12 file is on the web portal is turned into a QR-Code for the user to scan.
7 The user scans the QR-Code and the PKCS#12 file is transferred, using a secure connection with the server.
8 The PKCS#12 file would then be extracted into a software cryptographic module on the mobile device in accordance with SP 800-157.

4.3 Hardware Generated Derived Credential

The other two architectures use similar certificate signing methods as the software generated derived credentials, only the public/private keys are generated on a hardware cryptographic module that is FIPS 140 validated Level 2. By generating the keys on a removable (micro-SD) hardware token for the mobile device, the derived credential provides a LOA-4. The private key on the hardware token is protected by a 6 digit PIN. The hardware token would provide a means to protect the derived credential after a certain number of consecutive failed authentication attempts. A hardware cryptographic module provides multiple advantages and disadvantages over the software cryptographic module. From Section 2.5.2, we know that a hardware token is intrinsically more secure than a software credential. One major advantage of the hardware generated credential is that the private key is generated on the hardware module itself. Therefore, only the person in possession of the cryptographic token and with knowledge of the PIN are able to use the private key for any PKI function. Since the private key never leaves the hardware token, all cryptographic operations using the private key are performed on the token and not in software. In essence, a removable hardware cryptographic token would provide the same security features as a CAC, just sized to integrate into a mobile device.

The main disadvantage of the hardware cryptographic token is that it would require the purchase of a token for each mobile device approved for derived credentials. There would also be an associated cost with reissuing defective, lost, or stolen mobile devices that contain a cryptographic micro-SD card. Secondly, not all commercial mobile devices are equipped with micro-SD slots such as the iPhone and certain Android based mobile devices.

4.3.1 Crypto Micro-SD

NIST approved cryptographic micro-SD cards are FIPS 140-Level 3 validated and can provide a LOA-4 credential. For the purposes of derived credential implementation, a cryptographic micro-SD is most well suited if it can generate the public/private key pair on the cryptographic module. A cryptographic micro-SD card acts in a similar manner to the ICC on the CAC. The processors on a cryptographic micro-SD card allow for the generation of a public/private key pair with a restriction on the exportation of the private key. The private key can be PIN protected in the same way that the CAC user PIN protects its private key. Since the CAC is too large to be plugged into a mobile device, a FIPS 140 approved cryptographic micro-SD card is an extremely viable option to implement secure derived credentials on mobile devices. As derived credentials become more defined and more widely implemented by the DoD, commercial companies will continue to develop more specialized and more customized NIST approved cryptographic micro-SD cards and mobile devices. Below are two cryptographic micro-SD cards that are currently possible solutions.

The Motorola CRYPTR MICRO is an encryption unit that would fit the necessary requirements for DoD derived credential enabled mobile devices. The CRYPTR MICRO is FIPS 140 Level 3 complaint and provides high assurance key management and encryption capabilities. Only approved software is able to execute on the CRYPTR micro unit, which validates the integrity of the software upon each power up. The CRYPTR MICRO can support up to 512 keys and 32 certificates, well above the amount required for DoD PKI. The CRYPTR MICRO supports a full NSA Suite B, including AES and Elliptic Curve Cipher Suites [21].

Figure 4.3.1: Motorola CRYPTR Micro SD Card; 21. Motorola CRYPTR MICRO. MicorSD Encryption Unit for Mobile Devices. Product Spec Sheet. Available at http://www.motorolasolutions.com/web/Business/Products/Two-Way%20Radios%20-%20Public%20Safety/Encryption/AME1000/CRYPTR_micro_Encryption_Unit/CRYPTR_micro_spec_sheet.pdf. Used under fair use, 2014. [21]

Another cryptographic micro-SD card that is FIPS Level 3 approved is the ScanDisk TrustedFlash v1.0 [22]. The TrustedFlash v1.0 is outfitted with an onboard processor and a high performance cryptographic engine. The TrustedFlash supports the generation of RSA keys and implements AES, Triple-DES, PKCS, and SHA cryptographic algorithms [22].

4.3.2 CAC-Signed Hardware Derived Credential

By utilizing a removable cryptographic micro-SD card with the mobile devices, the DoD can implement LOA-4 derived credentials. This architecture would on be available with mobile devices that contain a micro-SD expansion slot. Tablet devices could also utilizes a cryptographic SD card of a different size that fits their expansion slots. This architecture combines the same principles from the CAC signed derived credential with a software cryptographic module, except

utilizes the hardware token for key generation. Again, this method would require that new CAC digital signature certificates be issued with certificate signing authority. An approved software application that integrates with the hardware token would be required for this method. The hardware token would also need to be connected to the computer with the software application interface, thus allowing the software to interact with the card. The software application would first need to authenticate the CAC holder prior moving forward in generating the derived credentials. The software application would then interface with the cryptographic micro-SD card to generate the public/private key pair. As stated above, the hardware token would provide two factor authentication in that the user would have to have possession of the token, and the user would need to know the PIN to use the private key. Once the key pair is generated on the token, the same steps are followed to sign the CSR with the CAC as before. Since the private key is already stored on the hardware token, there is no need to transfer any kind of PKCS#12 file back to the mobile device. The signed X.509 digital certificate are meant to be public, thus it can be published to the DoD certificate directory and then downloaded to the cryptographic hardware token on the mobile device from the computer or DoD PKI repository.

1 The user would log into a DoD workstation. Using the CAC authentication certificate (two factor authentication), the user would authenticate with a certificate and PIN to an approved computer software application that interfaces with the cryptographic micro-SD card.
2 If authentication is successful, the application will generate a 2048-bit RSA public/private key pair on the cryptographic micro-SD card.
3 After the key pair is generated, a CSR is generated using critical information pulled from the CAC digital signature certificate (name, email address, permissions) and the public key.

4 The CSR is then sent to the CAC for approval and signature. Since the private key never leaves the CAC, the cryptographic operation of signing the CSR actually happens on the ICC of the CAC.

5 After approved by the CAC, the derived credential extensions would be added to the X.509 certificate.

6 The signed X.509 is published to the DoD PKI certificate repository.

7 The X.509 certificate is than imported back to the mobile device cryptographic micro-SD card for storage.

4.3.3 User-Driven Hardware Derived Credential

The user driven hardware derived credential is generated in the same manner as the CAC signed hardware derived credential, except this credential is signed by a DoD DC CA and not signed by the CAC digital signature certificate. This method would alleviate the burden of reissuing CAC digital signature certificates to include the certificate signing authority. It would again keep the certificate signing and approval on the DoD servers just as with the server driven software derived credential. The user would be required to authenticate to a web portal via the CAC authentication certificate. Once the user has access to the portal, the user would upload the CSR via the software application on the computer. The web portal would then approve and sign CSR submitted by users granted access. The DoD DC CA would then sign the CSR and publish it in the DoD certificate repository. As in the previous architecture, the signed X.509 digital certificate is meant to be public, and therefore can easily be downloaded back to the mobile device for storage on the cryptographic micro-SD card from the DoD PKI repository.

1. Using the CAC authentication certificate (two factor authentication), the user would authenticate with a certificate and PIN to an approved computer software application that interfaces with the cryptographic micro-SD card.
2. If authentication is successful, the application will generate a 2048-bit RSA public/private key pair on the cryptographic micro-SD card.
3. After the key pair is generated, a CSR is generated using critical information pulled from the CAC digital signature certificate (name, email address, permissions) and the public key.
4. User must authenticate to DoD web portal using CAC authentication certificate. Once authenticated, the user then would have the permission to upload CSR for approval.
5. The CSR is then sent to the DoD DC CA for approval and signature through the DoD web portal. After approved by the CA, the derived credential extensions would be added to the X.509 certificate.
6. The signed X.509 is published to the DoD PKI certificate repository.
7. The X.509 certificate is than imported back to the mobile device cryptographic micro-SD card for storage.

5 Prototype Implementation

Chapter 4 detailed four distinct architectures for generating and deploying derived credentials to mobile devices. This chapter details the prototype implementation of the architectures. The testing setups and tools used are detailed below. Finally, the implementation was completed and the results are described in detail and shown in full as appendices.

5.1 Test Environment

A virtual box installation of Ubuntu 13.10 was used as the primary computer responsible for the implementation. A generic contact USB Smart Card reader was used to interact with both a CAC and purchased Gemalto PIV II smart cards. OpenSSL, a software toolkit, was used to generate 2048-bit RSA keys, CSR's, X.509 Certificates. OpenSSL is detailed below. To interact with the CAC and Gemalto smart cards, the OpenSC toolkit and subsequent libraries (piv-tool, pkcs11-tool, pkcs15-tool, and pkcs15-crypt) were used. After a mock DoD CA hierarchy was created using OpenSSL, OpenSSL and OpenSC were used to set up two mock CAC's using the Gemalto smart cards. Using OpenSC, the key pairs were generated on the Gemalto smart card. Next, OpenSSL was used to generating the CSR. Depending on the architecture, OpenSSL or OpenSC was used to implement the certificate signing with the CA. The architecture segments were converted to a python script which, when executed, fully implements the architecture. Figure 5.1.1 shows the setup used for implementation testing. The CAC and the 2 Gemalto smart cards can be seen on the left portion of the photo along with the USB smart card reader.

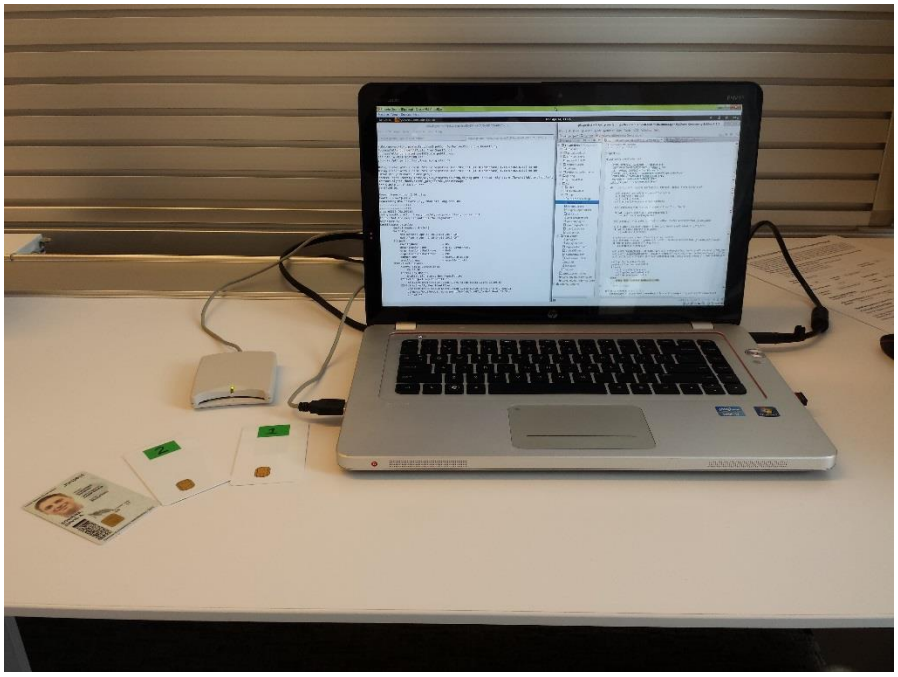

Figure 5.1.1: Test Setup

5.1.1 OpenSSL

OpenSSL is an open-source, commercial-grade, toolkit that is used to implement Secure Socket Layer (SSL) and Transport Layer Security (TLS) protocols [24]. OpenSSL is managed by a worldwide group of volunteers. OpenSSL has been validated as a Level 1, FIPS 140 software cryptographic module. This is important for the implementation of the software architectures, as NIST 800-157 requires the Level 1 validation of the software cryptographic module to generate LOA-3 credentials. Information, documents, and command definitions for OpenSSL can be found at www.openssl.org [24].

5.1.2 OpenSC

OpenSC is another open source, commercial-grade set of libraries and tools that work directly with smart cards. Just like OpenSSL, OpenSC is written by a host of volunteers online. OpenSC implements the PKCS#11 API and attempts to be compatible with every smart card that utilizes the PKCS#15 standard [25]. Essential to being able to sign a CSR with a CAC was the engine_pkcs11 library, as well as the libp11 library. OpenSC, along with the other required libraries, are available on the toolkits GitHub web site at www.github.com/OpenSC/OpenSC/wiki [25].

5.2 Mock DoD Certificate Authority Setup

This section details the process of setting up the simulated DoD CA used for the implementation.

5.2.1 OpenSSL Configuration File

In order to accurately prototype the derived credential architectures, it was first required to have a CA hierarchy that was mimicked from the actual DoD PKI hierarchy. The first step in this process was to set up the OpenSSL configuration file. The full configuration file can be found as Appendix A. The configuration file directs OpenSSL on the creation of CSR's and certificates. Since our hierarchy is multi-level, there are multiple CA's identified in the configuration file, each with identifying names. Each CA labeled has the specific path to the directories and files that are required for that CA. It provides the distinguished names for the CSR's. The CA section also

defines which set of extensions are to be assigned when issuing the digital certificate. The other key element of the CA section is the length of time (in days) that the certificate is valid for. The policy of each CA points to a different section of the configuration file that identifies the policy matches required for approving a CSR. Lastly, the unique subject line is set to yes if multiple certificates with the same common name are permissible by the CA.

The policy sections of the configuration file identify which distinguished name characteristics are required for the CA to approve a CSR. If a distinguished name is set to match, that specific distinguished name must match the CA. If it is set to supplied, then the CA must use the distinguished name from the CSR. If it is set to optional, the CSR does not need to match the CA.

The `req` section of the configuration file sets the default settings for a CSR. In the configuration file used for the mock DoD setup, the distinguished names are coded into the configuration file [24].

Lastly, each extensions section of the configuration file sets the extensions that are to be added to a certificate signed by the CA section that specifies that section. The extensions of a X.509 certificate are identified in 3.2. The extension sections were created with as many similar extensions as possible to the actual DoD CA and user certificates.

5.2.2 DoD Mock Certificates

Once the configuration file setup was complete, the mock DoD CA certificate could be created. All of the commands and outputs from the entire mock set up are included as Appendix

B. The first step in this process was to create all the directories within the SSL folder on the Ubuntu machine. Within each directory, an index file and serial file were created. The index maintains a database of the signed CSR's for each CA and serial file provides the serial for the next generated certificate.

The mock DoD Root CA was the first CA that needed to be created. Since this is a mock setup, the Root CA is a self-signed certificate. The first step is to generate the key pair and generate a CSR using OpenSSL. This `req` command creates a new CSR [24]. A RSA key size of 2048 bits is specified as well as SHA-1 encryption standard. When creating the CSR, the command prompts the user for input of the distinguished names. Since the country, organization, and organizational unit are the same for all the CAs, they were hard coded into the configuration file. The common name is different and is input in the command prompt. To self-sign the DoD Root CA CSR, the `x509` command is used [24]. OpenSSL will prompt the user for the private key password, and after passing verification, the certificate is signed.

For creating the intermediate CAs, the same step is followed to create the CSR. However, since in the hierarchy scheme used by DoD the intermediate CAs fall below the Root CA, the Root CA is used to sign and approve the intermediate CA CSRs. Therefore, once the CSRs are created, the OpenSSL command `ca` is used to specify an action being performed by a CA [24]. After the `ca` command is issued, the name of the `ca` section from the configuration file is identified. Once executed, OpenSSL prompts the user for the CA private key password. When approved, the CSR is signed by the CA, and the extensions specified in the configuration file are added to the

certificate. In Appendix B the creation of the *DoD CA I Sim* certificate and the *DoD DC CA Sim* certificate can be seen in detail.

5.2.3 Gemalto PIV II Smart Cards Initialization

Once the DoD CA and intermediate CA certificates are completed, the next step was to create the mock CAC cards. Two Gemalto PIV II smart cards were purchased, which closely mirror the specifications and capabilities of the CAC [23]. Gemalto is one company that provides the DoD with CAC smart cards [23].

To interact with a smart card, the OpenSC toolkit is utilized [25], specifically the piv-tool [26]. The piv-tool is used for card administration operations. A file containing the Gemalto card authentication password is saved and the path is exported. To generate the key pair on the card, the following command was given [26]:

```
$ piv-tool -A M:9B:03 -G 9C:07 -o \
  /home/al/pki_project/gemalto_card/pubkey.02.user1
```

In accordance with NIST SP 800-78, an RSA key of 2048 bits is generated using `-G` and the algorithm identifier from Table 7 above; 07 [6]. Table 6 above shows the key reference values for the specific key on the card; 9A for the authentication certificate and 9C for the digital signature certificate [9]. The `-A` identifies the administrative password location on the card. The `-o` outputs the slot key for the specific reference value on the card [26].

The ability for OpenSSL to interact with a hardware driver comes from the implementation of the engine_pkcs11 interface [28]. To set the engine, the path to the engine .so file is given. The

several engine pre commands at given. Then the path to the module is set. After running the following command, the engine is initialized and ready for use [28].

```
OpenSSL>
engine dynamic \
-pre SO_PATH:/usr/lib/ssl/engines/engine_pkcs11.so \
-pre ID:pkcs11 -pre NO_VCHECK:1 -pre LIST_ADD:1 -pre LOAD \
-pre MODULE_PATH:/usr/lib/x86_64-linux-gnu/opensc-pkcs11.so
```

Once the engine is initialed, then OpenSSL can be used to generate a CSR using the public key from the smart card. The same type command is given for the generation of the CSR as in the previous sections, except this time the engine is set, and the key is inserted from the `slot_1` (the slot of the Smart Card reader) and `id_01` (for the authentication key, ID of 02 would identify the digital signature key). The `-out` command give the filename of the CSR and location. The card authentication PIN is required to access the key and generate a CSR [24].

```
req -new -config /home/al/pki_project/ssl/openssltest.cnf \
-engine pkcs11 -sha1 -key slot_1-id_01 -keyform engine \
-out /home/al/pki_project/ssl/usercert.csr -text
```

Once the CSR is created, it is signed by the CA using the OpenSSL `ca` command [24]. The user certificates fall under the intermediate CA in the hierarchy, and therefore are signed by our mock DoD CA I certificate. Once the CSR is signed by the CA, it is stored onto the smart card. The piv-tool is used to accomplish this. The `-A` sets the authentication of the administrator. `The -C` writes a certificate to the card. Finally, the `-i` specifies the location of the certificate to be input [26].

```
$ piv-tool -A M:9B:03 -C 9A -i \
/home/al/pki_project/ssl/DoDRootCA/DoDCAI/UserCerts/ \
```

```
certs/user2.crt
```

Appendix C.1 then shows the steps used to verify that the certificate was indeed installed to the correct slot on the card and that the certificate is valid. Additionally, the same processes was used to setup and install a digital signature certificate for user 1 and an authentication and digital signature certificate for the user 2 smart card. The only difference in the process is that the id for the digital signature certificate is 9C, not 9A [9].

5.3 Implementation

This section provides the results from the prototype implementation of the architectures for issuing derived credentials.

5.3.1 CAC Signed Derived Credential

The python script, Appendix D.1, demonstrates a prototype implementation of the CAC signed derived credential. The first step in the architecture is to authentication the users PIV card. For the purposes of this implementation, the Gemalto PIV card with the mock DoD certificates was used. Upon running the script, the first thing that is done is the user authentication process. The certificate from the PIV card is pulled and checked against the CA certificates for authenticity. This is performed using the OpenSSL `-verify` command, and providing a concatenated file with all the allowable CA's. If the certificate validation passes, the second step is the actual authentication process. A hashed file is signed by the user private key and sent to the application for verification. The application then takes the signed hash, decrypts it using the public key provided by the PIV card and compares the decrypted hash with the hash of the original message.

This process is performed using the OpenSSL -dgst command with -verify selected. If either of these two authentications fail, then the program will no longer continue to run.

The second portion of the program parses through the PIV digital signature certificate and find the essential data to provide for the derived credential. Our program specifically pulls the common name and email address from the certificate. However, it can be adjusted to take any other subject characteristics or extensions that are required by NIST for the derived credential certificate. After we have collected the information from the PIV card certificate, we create a temporary OpenSSL configuration file. The temporary configuration file allows us to input the information taken from the PIV card certificate, and add that information to the CSR and signed certificate once created.

Once the temporary configuration file is created, the next step is to create the actual derived credential. The 2048-bit key pair is generated using OpenSSL genrsa command. Following the key pair generation, the CSR is generated using the information in the temporary configuration file and the public key just generated.

The next step is to approve and sign the CSR with the digital signature certificate that is on the PIV card. The key to being able to sign a CSR with the PIV card, is the engine_pkcs11 interface. The engine has to first be initialized, using the following OpenSSL command:

```
engine dynamic -pre \
SO_PATH:/usr/lib/engines/engine_pkcs11.so -pre ID:pkcs11 \
-pre NO_VCHECK:1 -pre LIST_ADD:1 -pre LOAD \
-pre MODULE_PATH:/usr/lib/x86_64-linux-gnu/opensc-pkcs11.so
```

Once the engine is initialized, the OpenSSL `ca` command is executed to perform the certificate signature. The OpenSSL command utilizes the engine interface to interact with the PIV card and use the private key stored on the card for signature. The engine interface instructs OpenSSL that the `keyform` for signature is the engine and it is in `slot_1` and `id_02` (slot 1 is the card reader and id 02 is the digital signature key). While the program performs this OpenSSL command, the user will be prompted to input the PIN protecting the private key of the digital signature. In our implementation, the PIN is "1234".

Once the certificate is created, it is ready to be transferred to the mobile device. The program then packages the signed certificate with the associated private key in an encrypted, password-protected, PKCS#12 file. When this action is completed, the user is prompted to input the password that protects the PKCS#12 file. The PKCS#12 file is then published on an Apache2 server that is running on the terminal. The program generates a QR-Code for the PCKS#12 file and when the user scans the QR-Code, it will begin the transfer of the derived credential PKCS#12 file to the mobile device for extraction and use. Figure 5.3.1 shows the QR-Code generated white contains the path to the PKCS#12 file containing the smart card signed derived credential and private key on the Apache server.

Figure 5.3.1: QR-Code for a PKCS#12 File on Apache Server

When we inspect the derived credential certificate that we have created, seen below as Figure 5.3.2, indeed the certificate is issued to *User.2.Dig.Sig* and was issued by *User 2 Dig Sig*; User 2 Dig Sig is he certificate stored on the Gemalto smart card. The certificate path of the certificate shows that its starts at the *DoD Root CA Sim*, then the *DoD CA I Sim*, then the *User 2 Dig Sig* (the certificate stored on the PIV card), and then finally the derived credential certificate itself. In our prototype, we limit the validity of the certificate signed by a PIV card to one week (7 days). As discussed in Section 4.2, this is a built in security factor for a derived credential stored in a software token. While one week was the length that we selected, the validity period is hard coded in the program and can be adjusted for whatever requirements are approved by NIST. If the phone is lost or stolen, the 7 day valid certificate will expire and be inherently useless.

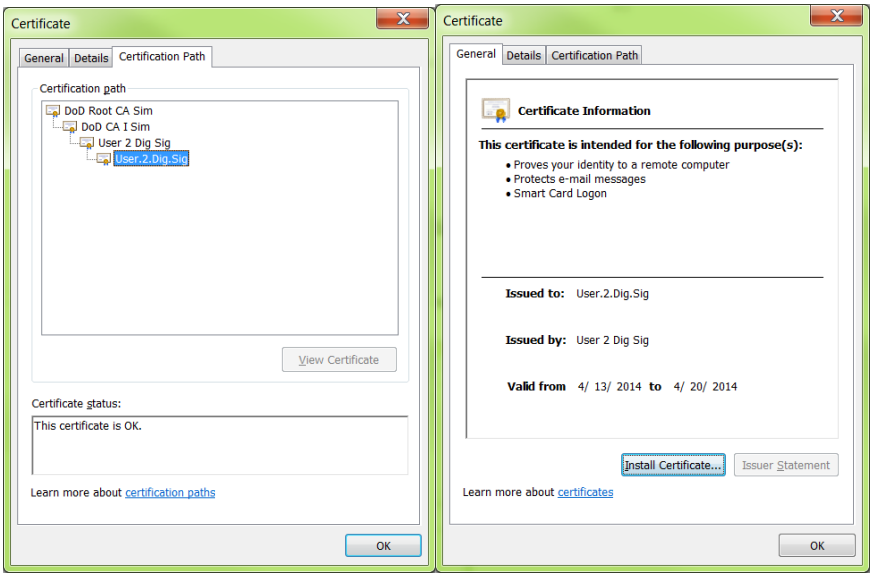

Figure 5.3.2: Certificate Signed by Smart Card

The full details of the derived credential certificate can be found as Appendix D.3. Further inspection reveals that the certificate has the same distinguished name characteristics as the original PIV card certificate, including the common name and email addresses. Since the derived credential certificate is only for use on mobile device for authentication and digital signatures, the only key usage permissions that are added are digital signature and non-repudiation. The only other extensions that are added are basic constraints, subject key identifier, and authority key identifier. When DoD policy is created for derived credentials, a certificate policy OID may be appropriately added to the certificate.

5.3.2 Server-Driven Derived Credential

The server driven derived credential architecture is very similar to the CAC signed derived credential. The primary difference between the two architectures is the CA. For the CAC signed

derived credential, the CA was the users digital signature certificate on the CAC. However, with the server driven architecture, the CA is a new intermediate CA created by DoD; *DoD DC CA Sim* in our prototype.

The complete python code that implements the server driven derived credential can be found as Appendix E.1. The full output results of the python code can be found as Appendix E.2. The user now authenticates to a web portal instead of a software application residing on the terminal itself. Our prototype models the authentication process by first, checking for validity of the users PIV certificate. If the certificate presented by the user is valid, the next step in the authentication process is to verify the certificate. The PIV card private key is used to encrypt a hashed message. The public key retrieved from the PIV certificate is used to decrypt the encrypted hash. The original hash and the decrypted hash value are compared, and the certificate is valid if the two hash values match.

In the server driven architecture, once the user is authenticated to the web portal, the rest of the process is driven on the server side. The server generates the new 2048-bit key pair for the derived credential. . In the prototype, the program only extracts the common name and the email. As NIST finalizes the requirements for the derived PIV credential, select extensions can also be chosen from the original PIV certificate for the derived credential. One the information is extracted from the PIV certificate, a temporary OpenSSL configuration file is created with the proper distinguished names inserted. Then the server creates the CSR for the user with the new public key and the temporary configuration file. The resulting CSR is then approved and signed by the CA, *DoD DC CA Sim*. Once signed, the certificate is then packaged with the private key in a password-

protected PCKS#12 file. The user is prompted for the PCKS#12 password for input. Once the PKCS#12 file is complete, it is placed in the proper file on the Apache server running on the terminal. A corresponding QR-Code is generated with the URL to the file. The user can then scan the QR-code and the PKCS#12 file is transferred to the mobile device.

Inspection of the server driven derived credential, seen below as Figure 5.3.3, reveals the certification path. The root CA is still the *DoD Root CA*. The root CA is followed by the intermediate CA, *DoD DC CA I*.

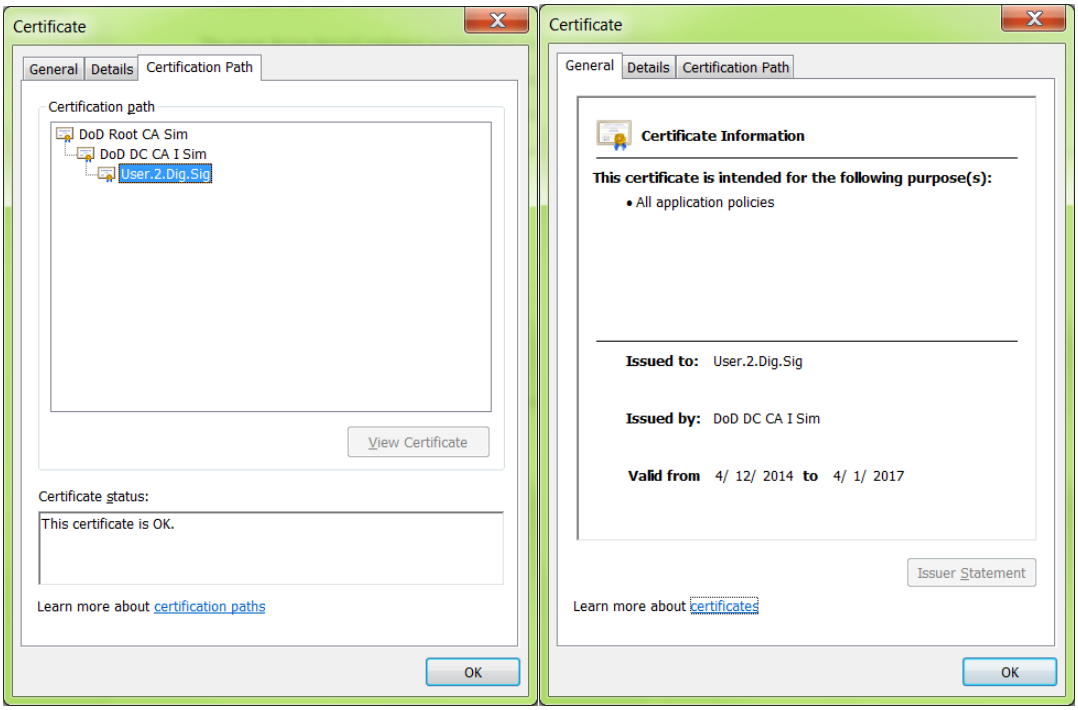

Figure 5.3.3: Derived Credential Signed By DoD DC CA

The full details of the server driven derived credential certificate can be found as Appendix E.3. Further inspection again shows that the certificate has the same distinguished name

characteristics as the original PIV card certificate, including the common name and email addresses. Also, since the derived credential certificate is only for use on mobile device for authentication and digital signatures, the only key usage permissions that are added are digital signature and non-repudiation. The only other extensions that are added are basic constraints, subject key identifier, and authority key identifier.

5.3.3 Hardware Token Credentials

Although we did not implement the hardware token credentials, both server driven and CAC signed, the same steps that are outlined in sections 4.3.2 and 4.3.3 were taken when completing the setup of the Gemalto Smart Cards. The main difference being that the derived credentials would be generated on a micro-SD card rather than a smart card. NIST 800-157 requires that the derived credential is only issued once the requester has been authenticated via PIV card. Therefore, the PIV must either interact directly with the mobile device via USB contact reader or Bluetooth reader, or the mobile device and the PIV must be connected to the same terminal. Connecting the mobile device to the terminal via a micro-USB cable is the most practical solution. Each terminal would have an application that would interact with the cryptographic micro-SD card on the mobile device as well as the PIV card. The first process of the application would be to authenticate the user via the PIV card. Once the authentication process is complete, then process of creating the derived credential would begin.

The first step of creating the derived credential is generating the keys on the cryptographic micro-SD card. To generate keys on a USB connected hardware token, the OpenSC `piv-tool` is utilized [26].

```
$ piv-tool -A <{M|A}> -G <ref>:<alg> -o <file>
```

The A identifies the admin authentication and is specific to the cryptographic token. For the Gemalto cards, M:9B:03 was used. The G function generates the key pair. Ref refers to the key reference values identified in NIST SP 800-78 and seen in table 6 above. Alg is the algorithm identifier that is defined in NIST SP 800-78 and seen in table 7 above. For a RSA 2048-bit key, the algorithm identifier is 07 [9].

Once the keys are created, the CSR must be completed. First, the engine_pkcs11 must be initialized to specify that the keys to be used for the CSR are on a hardware token [28].

```
openssl>
engine dynamic -pre
SO_PATH:/usr/lib/engines/engine_pkcs11.so -pre ID:pkcs11 \
-pre NO_VCHECK:1 -pre LIST_ADD:1 -pre LOAD \
-pre MODULE_PATH:/usr/lib/x86_64-linux-gnu/opensc-pkcs11.so
```

After the engine is initialized, the OpenSSL req command is used to generate a CSR.

```
req -new –config ./openssl.cnf -engine pkcs11 -sha1 \
-key slot_1-id_02 -keyform engine -out file
```

Config identifies the path to the OpenSSL configuration file to be used. Engine tells OpenSSL to use the pkcs11 engine. Sha1 identifies the encryption algorithm. The key to be used is identified by the slot and id listed [24]. The slot should be whatever slot the mobile device cryptographic micro-SD can be found. The id is the specific certificate that is to be used. The authentication certificate is 01, the digital signature certificate is 02, and the key management certificate is 03

[26]. If it is unknown, the slot the hardware token is stored in can be found using the `pkcs11-tool` to list the slots. To list the slots, the following command is used [26]:

```
$ pkcs11-tool --list-slots
```

Once the CSR is generated, it must be signed by the CA. The CAC Signed Derived credential architecture would send the CSR to the CAC for signature. The process for signing a CSR with a CAC requires that the engine be initialized again since the CAC digital signature certificate is on a hardware token. Once the engine is initialized, the follow OpenSSL `ca` command will sign a CSR with a digital certificate on the PIV card:

```
$openssl ca -engine pkcs11 -config ./openssl.cnf \
-cert file -keyform engine -keyfile slot_1-id_02  \
-name CA_usercerts -in CSRfile -out dcfile
```

Again, engine specifies the pkcs11 engine. The `config` identifies the path to the configuration file. `Keyform` tells OpenSSL that the CA private key can be found in the engine. The `keyfile` again is identified by the `slot` and the `id` of the CA private key to be used. `Name` identifies the configuration file section to use. The `in` file is the path to the CSR and the `out` is the path of the newly signed derived credential certificate [24].

For the server driven derived credential, the process to sign the CSR would include the OpenSSL `ca` command, however, it is not necessary to initialize the engine since the CA private key is kept on the same web portal that the user has accessed. Once the user authenticates to the web portal, and the CSR is generated by the web portal, the CSR is signed by the intermediate

derived credential CA: *DoD DC CA*. The OpenSSL command to sign the server driven derived credential would be [24]:

```
$ openssl ca -config ./openssl.cnf -cert file \
-name CA_usercerts -in CSRfile -out dcfile
```

It is not necessary to identify the location of the CA private key in this instance since its location is static and consistent. The location would be hard coded in the configuration file [24].

Once the certificate is signed, it can be downloaded to the mobile device in a few different ways. Since it is a public certificate, it can be published by DoD to their certificate repositories and then downloaded to the mobile device via a mobile application. Alternatively, the certificate can be returned to the terminal the user is accessing to generate the derived credential and imported directly to the mobile device. The command to store a certificate to a cryptographic hardware token is:

```
piv-tool -A <{M|A}> -C <ref> -i DCfile
```

The `admin` authentication is the same as above. `C` is used to specify the action of writing a certificate to the card. The `ref` is the same key reference where the RSA keys were generated since the derived credential correlates to that key pair. `I` specifies the file name for input operation [26].

6 Conclusion

This section provides a conclusion for the thesis and touches on the future work in the area of derived credentials for DoD mobile devices.

6.1 Summary

The deployment of derived credentials to DoD mobile devices can greatly affect the efficiency and security of the entire department. Modern smartphones are nearly as powerful and functional as desktop PC's yet portable enough they are integrated into almost every moment of our everyday lives. Enabling a mobile device the same PKI access as the DoD CAC would allow secure mobile authentication to DoD web portals and the use of secure email at all times of the day. While the concept of derived credentials is being molded, the generation and delivery architectures are still unknown. The goal of this thesis was to explore possible solutions for a DoD derived credential architecture for efficiently and securely generating approved derived credentials and delivering them to the users mobile device.

NIST standards allow for both a software and hardware cryptographic module generation of public/private key pairs. Two of the architectures detailed utilize a software cryptographic module that generates the public/private keys through the use of OpenSSL. One architecture allows the user to act as the CA and sign the CSR with the CAC card while the implements a server driven CA and the CSR is signed by a DoD derived credential CA residing on a DoD server. Since a software cryptographic module is less secure than a hardware cryptographic module, these derived credentials would only provide the user with a LOA-3 credential. However, by limiting the validity

period of the credentials, and storing them in an encrypted software token can greatly increase the security of the credentials and protect them against vulnerabilities.

The last two architectures utilized a non-embedded hardware cryptographic module (micro-SD card) to generate the public/private key pair. In the same way, one architecture allows the user to act as the CA and sign the CSR with the CAC card while the implements a server driven CA and the CSR is signed by a DoD derived credential CA. Due to the highly secure nature, a hardware generated derived credential provides the user with a LOA-4 credential. By protecting the credentials with a PIN and storing them on a hardware cryptographic module, the DoD can securely deploy derived credentials to all mobile devices.

The prototype implementation showed the possibility of allowing the user to sign a CSR with a CAC. While NIST SP 800-157 authorizes both generation of the RSA key pair through software and hardware cryptographic modules, I believe that the hardware cryptographic module will be the preferred method of implementation by DoD. The cryptographic micro-SD card, in theory, acts exactly as the CAC in that the key generation and all subsequent PKI operations are performed on the cryptographic module itself. The key never leaves the hardware token and is tamper proof. The loss of a mobile device containing a hardware token does not threaten the security of the DoD PKI structure. Whichever method DoD policy decides to authorize for the generation and delivery of derived credentials, one of, or a combination of, the architectures in this thesis will satisfy their requirements.

6.2 Recommendations for Future Work

Since term derived credential is so new and the standards and requirements are still undefined, this thesis has only begun to explore the possible solutions for an architecture for deployment. In the same regards, the architecture for deployment of the derived credential is only one aspect of the full implementation requirements for derived credentials. The first thing that will need to happen is NIST to publish the approved copy of SP 800-157. There are also updates to the COMMON and NIST SP 800-78 and 800-73 that will need to include the derived credential specifications. Certificate policy OID's will need to be created specifically for derived credentials. Once NIST finalizes all the documentation for derived credentials, the DoD is going to need to establish their own policy for derived credentials. Once DoD finalizes their policy on derived credentials, it will provide better insight into which architecture provided would work best for DoD. The architectures can be adjusted or combine to meet the policies set forth by the DoD.

While not specifically detailed in this paper, embedded hardware security modules are becoming more and more common in mobile devices. After NIST approves derived credentials for use on Federal mobile devices, industry may begin developing more phones with embedded hardware tokens. An advantage to embedded hardware tokens is that it would not be necessary to purchase additional hardware tokens after the initial purchase of the phone. The embedded hardware token would come standard in the phone and could provide unique security features not supported by other approaches [30].

The next step, once the derived credentials are created and deployed to mobile devices, is the generation of DoD approved mobile applications that interact with the derived credential. The

first mobile application on the phone would need to be written to correctly parse and store the PKCS#12 once the QR-Code is available to be scanned. The other mobile applications that must be developed would use the derived credential for secure email or DoD server access. The DoD could also implement secure voice or secure messaging since the public/private keys and certificates on the mobile device can be used for encryption. Once the derived credentials are approved and securely stored on the mobile device for use, the DoD will be able to approve mobile applications with a wide array of functionalities. It will be up to them to decide how the derived credentials can and cannot be used.

References

1. George W. Bush. Homeland Security Presidential Directive 12: Policy for a Common Identification Standard for Federal Employees and Contractors, August 2004. Available at http://www.dhs.gov/xabout/laws/gc_1217616624097.shtm.

2. U.S. Department of Defense. Web. April 14, 2014. Available at http://www.defense.gov/about/.

3. NIST Computer Security Division: Computer Security Resource Center. About Personal Identity Verification (PIV) of Federal Employees and Contractors. Available at http://csrc.nist.gov/groups/SNS/piv/

4. FIPS Publication 201-2, Personal Identity Verification (PIV) of Federal Employees and Contractors, NIST, August 2013, or as amended. Available at http://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.201-2.pdf.

5. NIST Special Publication 800-73-3, Interfaces for Personal Identity Verification, NIST, Feb 2010, or as amended. Available at http://csrc.nist.gov.

6. NIST Special Publication 800-78-3, Cryptographic Algorithms and Key Sizes for 897 Personal Identity Verification, NIST, Dec 2010, or as amended. Available at http://csrc.nist.gov.

7. X.509 Certificate Policy for the U.S. Federal PKI Common Policy Framework, Version 1.21, December 2012. Available at http://www.idmanagement.gov/documents/common-policy-framework-certificate-policy.

8. NIST Special Publication 800-63-2, Electronic Authentication Guideline, NIST, August 2013, or as amended. Available at http://csrc.nist.gov.

9. FIPS Publication 140-2, Security Requirements for Cryptographic Modules, NIST, May 25, 2001, or as amended. Available at http://csrc.nist.gov/publications/fips/fips140-2/fips1402.pdf.

10. Special Publication 800-157, DRAFT Guidelines for Derived Personal Identity Verification (PIV) Credentials, NIST, May 2014. Available at http://csrc.nist.gov.

11. Corella, Franscisco PhD and Karen Lewison, PhD. Techniques for Implementing Derived Credentials. Revised September 13, 2012. Available at http://pomcor.com/whitepapers/DerivedCredentials.pdf.

12. OMB Memorandum M-04-04, E-Authentication Guidance for Federal Agencies, OMB, December 2003. Available at http://www.whitehouse.gov/sites/default/files/omb/memoranda/fy04/m04-04.pdf.

13. Public key infrastructure. (n.d.). The Free On-line Dictionary of Computing. Retrieved March 30, 2014, from Dictionary.com website: http://dictionary.reference.com/browse/public key infrastructure.

14. Adams, Carlisle & Lloyd, Steve (2003). Understanding PKI: concepts, standards, and deployment considerations. Addison-Wesley Professional. pp. 11–15. ISBN 978-0-672-32391-1.

15. Microsoft Exchange. Understanding Public Key Cryptography. Available at http://technet.microsoft.com/en-us/library/aa998077(v=exchg.65).aspx.

16. Cooper, et al. Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile, RFC 5280, May 2008.

17. DoD Common Access Card. Available at http://cac.mil/.

18. Military CAC. Available at http://militarycac.com.

19. Department of Defense. United States Department of Defense X.509 Certificate Policy. May 4, 2011. Available at http://iase.disa.mil/pki-pke/downloads/unclass-dod_cp_v10-2final_4may11.pdf

20. CAC/PKI Training Guide. Common Access Card/Public Key Infrastructure. Available at http://www.idmanagement.gov/sites/default/files/documents/CACpkiTrainingGuide.pdf.

21. Motorola CRYPTR MICRO. MicorSD Encryption Unit for Mobile Devices. Product Spec Sheet. Available at http://www.motorolasolutions.com/web/Business/Products/Two-Way%20Radios%20-%20Public%20Safety/Encryption/AME1000/CRYPTR_micro_Encryption_Unit/CRYPTR_micro_spec_sheet.pdf.

22. SanDisk Corporation. TrustedFlash v1.0 – microSD. Non-Proprietary Security Policy, version 2.2 October 22, 2009. Available at http://csrc.nist.gov/groups/STM/cmvp/documents/140-1/140sp/140sp1191.pdf.

23. Gemalto IDPrime PIV Card v1.55. Technical Brochure. Available at http://www.gemalto.com/products/piv_card/product_brief.html.

24. OpenSSL and SSL/TSL toolkit. Available at https://www.openssl.org/.

25. OpenSC – tools and libraries for smart cards. Available at https://github.com/OpenSC/OpenSC/wiki.

26. Piv-tool. Available at https://www.opensc-project.org/opensc/wiki/PivTool.

27. Pkcs15-tool. Available at http://linux.die.net/man/1/pkcs15-tool.

28. Engine_pkcs11. Available at https://github.com/OpenSC/OpenSC/wiki/Engine-pkcs11-quickstart.

29. Microsoft. TechNet. Digital Certificates. Available at http://technet.microsoft.com/en-us/library/cc962029.aspx.

30. Draft NIST Interagency Report 7981, Mobile, PIV, and Authentication, March 2014. Available at http://csrc.nist.gov.

31. Turner, Sean, and Russ Housley. Implementing Email Security and Tokens: Current Standards, Tools, and Practices. Indianapolis: Wiley Publishing, 2008.

32. Lindell, Yehuda. The Security Advantages of Hardware Tokens over Software Toekns for PKI Applications. White Paper. Available at ftp://ftp.ealaddin.com/pub/marketing/eToken/White_Papers/WP_eToken_HardSoftTokens.pdf

33. Bar-El, Hagai. Security Implications of Hardware vs. Software Cryptographic Modules. Discretix Technologies Ltd. Available at http://leetupload.com/database/Misc/Papers/Asta%20la%20Vista/security_implications_of_hw_vs_sw_cryptographic_modules.pdf

34. Corella, Franscisco PhD and Karen Lewison, PhD. An Example of a Derived Credentials Architecutre. Revised April 27, 2014. Available at http://pomcor.com/techreports/DerivedCredentialsExample.pdf

35. Linn, John. Trust Models and Management in Public-Key Infrastructures. RSA Laboratories. November 6, 2000. Available at ftp://ftp.rsa.com/pub/pdfs/PKIPaper1.pdf

Appendix A: Actual DoD CAC Certificates

A.1: CAC Authentication Certificate

```
al@al-gnome:~/pki_project/cac_certs$ openssl x509 -in cac_auth.crt -text
Certificate:
    Data:
        Version: 3 (0x2)
        Serial Number: 1043279 (0xfeb4f)
    Signature Algorithm: sha1WithRSAEncryption
        Issuer: C=US, O=U.S. Government, OU=DoD, OU=PKI, CN=DOD CA-30
        Validity
            Not Before: Jun  4 00:00:00 2012 GMT
            Not After : Jun  3 23:59:59 2015 GMT
        Subject: C=US, O=U.S. Government, OU=DoD, OU=PKI, OU=USCG,
                 CN=SOWERS.DAVID.A.1274982450
        Subject Public Key Info:
            Public Key Algorithm: rsaEncryption
                Public-Key: (2048 bit)
                Modulus:
                    00:ba:83:95:cd:56:6e:72:a0:5d:d5:2a:e9:44:b3:
                    82:0e:a0:8e:4a:e0:95:73:d5:c0:c8:df:e6:fd:34:
                    c3:2e:03:8c:07:cf:37:e6:57:28:2f:1a:8e:2f:25:
                    5e:9c:d2:38:1c:76:13:9b:52:d1:c2:b7:0e:15:6c:
                    23:19:0f:6c:ec:b9:94:66:43:0f:17:ca:98:df:27:
                    fa:a0:d0:00:06:57:c2:65:cc:94:c5:4c:a3:bd:b8:
                    2f:aa:d9:a1:86:97:ea:1a:36:72:a0:38:eb:82:f4:
                    da:d7:64:50:0f:69:d4:92:70:81:db:d5:9a:83:47:
                    e5:53:d6:e1:7c:d8:1c:79:f0:03:ee:3d:b1:02:ca:
                    ba:7e:80:14:59:0e:3b:53:62:5a:d4:25:71:3d:b5:
                    e1:7e:70:ea:81:14:ff:ff:1b:14:eb:82:d9:31:dc:
                    76:8d:35:f3:00:7d:17:cf:6e:f1:75:11:25:b0:6d:
                    cf:7e:d3:34:8a:d2:cd:6e:16:5e:f3:10:aa:dc:7f:
                    61:62:1e:08:ee:6f:9c:3d:f3:6f:1c:b1:6e:43:18:
                    14:21:8b:a5:3d:85:81:6a:22:ca:71:aa:fc:ad:f2:
                    e1:cf:9b:3d:72:92:09:46:b4:2d:43:fd:72:c4:d7:
                    52:38:66:bb:a2:6f:ee:b1:91:a0:30:96:d4:7f:04:
                    59:ed
                Exponent: 65537 (0x10001)
        X509v3 extensions:
            X509v3 Authority Key Identifier:
```

keyid:08:4E:D5:A4:3C:2A:04:9B:93:1B:B7:04:08:8E:74:B9:06:7C:0D:A3

X509v3 CRL Distribution Points:

Full Name:
URI:http://crl.disa.mil/crl/DODCA_30.crl

X509v3 Key Usage: critical
Digital Signature
X509v3 Certificate Policies:
Policy: 2.16.840.1.101.2.1.11.9
Policy: 2.16.840.1.101.2.1.11.19

X509v3 Subject Key Identifier:
56:74:52:18:55:AA:17:A3:D1:5E:19:00:3B:A8:65:FA:78:E0:38:AD
Authority Information Access:
CA Issuers - URI:http://crl.disa.mil/sign/DODCA_30.cer
OCSP - URI:http://ocsp.disa.mil

X509v3 Subject Alternative Name:
othername:<unsupported>, othername:<unsupported>
X509v3 Subject Directory Attributes:
0.0...+.....1...US
X509v3 Extended Key Usage:
Microsoft Smartcardlogin, TLS Web Client Authentication
Signature Algorithm: sha1WithRSAEncryption
2e:6a:78:78:17:59:01:10:2c:84:b9:dd:74:32:1c:37:d1:8d:
b0:76:ce:2e:7e:e5:6a:f3:fd:43:4a:f8:c7:47:cb:7c:59:36:
d3:7e:be:80:db:83:87:10:eb:21:d7:7b:48:d5:8a:63:d0:e0:
cb:39:b9:92:d7:75:01:8a:1a:49:e5:5d:16:c3:f0:a6:0f:2f:
65:68:91:0d:e2:53:96:98:f1:9e:5b:54:c2:e9:d9:2e:31:b1:
7b:b2:16:2f:11:69:f4:fd:50:77:6b:0a:a4:51:51:4e:1c:9b:
96:33:82:1e:22:41:57:98:4c:a4:76:2b:4f:71:55:ca:34:0a:
4e:2c:b1:4a:72:6f:4e:7f:93:cd:3c:27:e2:29:57:12:83:db:
7d:5f:5f:7f:e5:38:4d:aa:c4:e7:71:05:7c:11:19:67:57:5b:
ef:aa:da:86:aa:1a:4f:63:73:fc:55:45:22:fe:f0:ac:3e:e9:
98:0b:f1:cf:d5:03:2d:87:af:bd:70:33:f4:f0:e8:35:e5:6b:
7f:c8:70:d2:6f:85:1f:be:78:0d:d2:4e:f8:5b:78:f2:87:95:
04:f8:6d:9e:f5:42:83:c1:84:d4:ec:b6:57:33:c0:fc:d1:5b:
13:87:27:8e:a3:bb:b2:04:71:1f:da:a6:17:10:e3:4e:85:63:
86:dd:26:41

A.2: CAC Digital Signature Certificate

```
al@al-gnome:~/pki_project/cac_certs$ openssl x509 -in cac_digsig.crt -text
Certificate:
    Data:
        Version: 3 (0x2)
        Serial Number: 1057905 (0x102471)
    Signature Algorithm: sha1WithRSAEncryption
        Issuer: C=US, O=U.S. Government, OU=DoD, OU=PKI, CN=DOD EMAIL CA-30
        Validity
            Not Before: Jun  4 00:00:00 2012 GMT
            Not After : Jun  3 23:59:59 2015 GMT
        Subject: C=US, O=U.S. Government, OU=DoD, OU=PKI, OU=USCG,
                 CN=SOWERS.DAVID.A.1274982450
        Subject Public Key Info:
            Public Key Algorithm: rsaEncryption
                Public-Key: (2048 bit)
                Modulus:
                    00:ca:93:9b:98:7b:de:d6:f7:0f:c9:3d:7d:61:6b:
                    c3:0e:b6:ce:15:8f:8e:62:b6:87:78:10:2c:4b:f5:
                    3d:72:09:06:57:0f:c7:60:60:b9:93:2f:43:3d:1d:
                    ef:43:28:9a:ea:5c:07:26:53:7e:f7:3a:fd:97:15:
                    7b:3b:29:1f:91:2d:6e:12:63:14:94:a9:17:8a:7e:
                    aa:6f:81:5c:02:c6:98:03:87:a2:2b:04:f9:4a:77:
                    6f:ed:c3:d8:d0:68:74:ab:7b:03:0f:e5:69:39:ad:
                    f9:2c:04:1e:8e:2f:1a:cc:31:10:10:76:f1:50:63:
                    af:3b:54:c5:2e:1e:05:87:8e:5c:70:30:0d:e2:16:
                    6b:c2:a3:ad:cc:bc:cd:43:2e:ad:69:9f:67:4d:c2:
                    d0:60:c8:9d:ec:88:a3:93:4d:e3:eb:a9:e0:80:d5:
                    5e:cf:f6:b0:4b:5e:83:e4:c9:b9:f7:11:e9:8a:bc:
                    62:bc:12:07:2e:8c:52:53:6e:a5:04:9f:f7:f1:6e:
                    40:af:79:89:d9:de:11:f2:16:d2:4a:b7:07:64:48:
                    0e:af:43:90:25:3c:88:47:dc:cb:c5:be:1f:b3:69:
                    cd:6e:a7:92:ba:e1:a8:bd:6c:90:9e:7e:00:ca:77:
                    88:d5:20:2b:be:4f:ee:29:aa:99:6c:2e:e4:79:a9:
                    11:27
                Exponent: 65537 (0x10001)
        X509v3 extensions:
            X509v3 Authority Key Identifier:
                keyid:35:61:66:28:09:BC:56:25:5B:8B:CC:BF:81:5E:61:2C:30:39:D3:21

            X509v3 CRL Distribution Points:
```

Full Name:
URI:http://crl.disa.mil/crl/DODEMAILCA_30.crl

X509v3 Key Usage: critical
Digital Signature, Non Repudiation
X509v3 Certificate Policies:
Policy: 2.16.840.1.101.2.1.11.9
Policy: 2.16.840.1.101.2.1.11.19

X509v3 Subject Key Identifier:
0C:D4:9C:77:4A:28:14:B2:78:06:86:4E:27:A9:07:35:D9:C4:AB:2E
Authority Information Access:
CA Issuers - URI:http://crl.disa.mil/sign/DODEMAILCA_30.cer
OCSP - URI:http://ocsp.disa.mil

X509v3 Subject Alternative Name:
email:David.A.Sowers@uscg.mil, othername:<unsupported>
X509v3 Subject Directory Attributes:
0.0...+.....1...US
X509v3 Extended Key Usage:
Microsoft Smartcardlogin, TLS Web Client Authentication, E-mail Protection
Signature Algorithm: sha1WithRSAEncryption
7b:e9:c0:ec:08:83:b4:ee:53:fc:6a:41:0a:55:cd:27:60:d9:
bd:44:28:84:b9:b9:1b:26:ca:33:91:b1:31:91:27:3a:19:d2:
f0:e4:99:9d:6a:4a:3c:ec:7a:b2:70:42:82:62:ca:f5:e5:1c:
4d:88:3e:a7:18:e5:96:4c:6a:1f:90:62:0c:22:3a:37:fa:0b:
05:d1:e2:ee:22:83:a8:b8:a4:ea:b2:cc:b3:66:92:24:ec:6c:
91:7e:0e:84:b0:ba:bc:41:91:be:d3:54:c4:c8:ef:4c:11:86:
fc:9f:81:4a:33:6c:b0:58:dd:cf:e5:f5:45:cc:59:21:57:96:
08:4d:a2:8a:10:35:e5:7d:a9:26:92:29:6d:a7:6b:2d:b1:84:
89:43:9e:c9:ee:43:69:20:c8:b3:60:e1:64:a0:35:23:1e:a2:
1f:30:84:d7:b9:d6:9d:3e:44:dc:33:50:3e:a7:18:e6:b1:ab:
ff:0c:ac:30:8f:da:0c:5a:c8:6e:f0:88:69:aa:9a:c7:6b:e4:
33:29:40:41:5b:3a:3f:b8:b1:a2:62:cb:5b:fb:7e:7d:f6:b7:
34:6e:cf:46:ac:18:03:bb:30:43:9c:52:c6:2f:34:00:a4:f3:
f0:c8:63:e4:0b:b3:bb:60:79:5e:90:ad:b1:3f:a2:d4:c7:6f:
4b:e7:1a:15

A.3: CAC Key Management Key

```
al@al-gnome:~/pki_project/cac_certs$ openssl x509 -in cac_keyman.crt -text
Certificate:
    Data:
        Version: 3 (0x2)
        Serial Number: 1057908 (0x102474)
    Signature Algorithm: sha1WithRSAEncryption
        Issuer: C=US, O=U.S. Government, OU=DoD, OU=PKI, CN=DOD EMAIL CA-30
        Validity
            Not Before: Jun  4 00:00:00 2012 GMT
            Not After : Jun  3 23:59:59 2015 GMT
        Subject: C=US, O=U.S. Government, OU=DoD, OU=PKI, OU=USCG,
                 CN=SOWERS.DAVID.A.1274982450
        Subject Public Key Info:
            Public Key Algorithm: rsaEncryption
                Public-Key: (2048 bit)
                Modulus:
                    00:96:4e:07:73:91:c9:d2:88:21:b7:15:41:34:82:
                    65:78:29:3e:fa:d2:44:0f:a7:2e:8a:7e:43:cb:d7:
                    f9:ca:06:fd:86:45:86:2f:ae:f1:7d:a6:85:48:32:
                    0f:d5:34:8e:dd:49:fa:a4:7d:b5:f7:c8:54:88:26:
                    9c:08:e2:c6:b9:0e:d8:5a:63:21:ae:99:64:24:38:
                    3f:f6:89:46:5c:3c:39:9e:65:a3:09:99:21:44:21:
                    e1:c3:a2:2e:d7:f4:42:60:bd:c9:ba:8b:09:fd:d1:
                    60:e0:0c:fe:f9:fd:fe:8e:09:ea:ae:2f:28:5e:07:
                    45:8a:b5:31:d1:bd:9a:40:74:71:27:23:07:e6:1b:
                    7f:25:ac:e7:63:74:0d:43:aa:27:15:fe:0c:0a:e1:
                    96:8a:f2:02:9c:b4:5c:a5:02:d8:fa:0b:91:e4:a2:
                    0e:da:cd:5c:ee:d4:95:5d:b0:0f:84:de:bb:42:96:
                    5f:52:11:a3:f8:64:d9:c8:29:ab:6f:00:f3:85:ec:
                    1e:44:83:92:85:74:e5:57:48:e6:63:f3:62:07:1f:
                    8f:ce:8b:ab:78:77:df:f3:03:7e:c1:c8:85:cd:98:
                    86:28:c9:17:5d:3a:db:89:6b:fa:1d:d7:86:00:10:
                    e9:17:b8:a2:1e:83:54:71:0a:24:38:bd:87:c1:7e:
                    d2:4d
                Exponent: 65537 (0x10001)
        X509v3 extensions:
            X509v3 Authority Key Identifier:
```

keyid:35:61:66:28:09:BC:56:25:5B:8B:CC:BF:81:5E:61:2C:30:39:D3:21

X509v3 CRL Distribution Points:

Full Name:
URI:http://crl.disa.mil/crl/DODEMAILCA_30.crl

X509v3 Key Usage: critical
Key Encipherment
X509v3 Certificate Policies:
Policy: 2.16.840.1.101.2.1.11.9
Policy: 2.16.840.1.101.2.1.11.19

X509v3 Subject Key Identifier:
A1:4D:99:AE:1E:F6:EA:2C:68:0B:EF:38:2C:E4:71:59:0B:24:6E:5E
Authority Information Access:
CA Issuers - URI:http://crl.disa.mil/sign/DODEMAILCA_30.cer
OCSP - URI:http://ocsp.disa.mil

X509v3 Subject Alternative Name:
email:David.A.Sowers@uscg.mil
X509v3 Subject Directory Attributes:
0.0...+.....1...US
Signature Algorithm: sha1WithRSAEncryption
84:ba:f7:dc:70:1e:19:86:a8:5a:14:73:80:9b:6b:ce:df:e8:
a2:70:f8:d1:a3:9d:9a:3b:a1:fe:44:b1:30:48:51:da:c2:b3:
a3:40:5d:95:b8:51:f1:57:0a:54:b1:a9:88:4d:9b:eb:c1:7a:
a2:32:38:6d:64:a8:4a:a6:da:68:e0:9c:34:73:e7:db:14:a6:
74:57:cf:31:e4:5c:6b:6d:0d:4d:ad:d7:3b:37:c9:a9:e1:67:
3e:68:e0:a7:68:28:9e:ea:25:be:7b:70:d3:dc:57:ca:56:79:
5c:3d:f3:be:9d:9d:52:1b:ff:42:27:05:38:e7:82:f7:ff:2d:
67:6d:9b:c3:c7:4b:38:ae:dc:0d:d3:fc:fb:aa:d0:d0:3a:e6:
20:7b:c2:6c:0b:43:a5:be:54:20:a4:12:83:b9:b6:60:31:22:
2e:d1:41:f4:7d:09:cb:5f:39:76:30:ba:bf:ab:bb:23:c0:2d:
b4:d3:cb:ee:7b:24:c4:a3:1f:9d:49:ef:c2:27:4d:01:e0:62:
30:02:8a:8a:1c:68:cb:34:42:06:15:48:d1:d1:7b:ff:2d:06:
b4:a1:da:25:ff:a7:9d:9c:4c:30:cc:b3:91:b6:78:7f:ae:cd:
ef:b9:4e:c9:f3:5e:47:e0:3a:0c:e3:20:34:c8:84:06:1c:0e:
f5:0b:3e:47

Appendix B: OpenSSL Configuration File

```
#
# OpenSSL Configuration file for simulated DoD Multi-level CA
#

# This definition stops the following lines choking if HOME isn't
# defined.
HOME            = .
RANDFILE        = $ENV::HOME/.rnd

####################################################################
[ ca ]
default_ca  = CA_default        # The default ca section

####################################################################
[ CA_dodrootca ]

dir         = /home/al/pki_project/ssl/DoDRootCA     # Where everything is kept
certs       = $dir/DoDCAI                           # Where the issued certs are kept
database    = $dir/index/index.txt                  # database index file.
new_certs_dir   = $dir/DoDCAI                       # default place for new certs.

certificate = $dir/dodrootca.crt          # The CA certificate
serial      = $dir/serial/serial    # The current serial number
#crl        = $dir/crl.pem          # The current CRL
private_key = $dir/dodrootca.key      # The private key
RANDFILE    = $dir/.rand                # private random number file

x509_extensions = dodcai      # The extentions to add to the cert
name_opt    = ca_default        # Subject Name options
cert_opt    = ca_default        # Certificate field options

default_days    = 3650          # how long to certify for
default_crl_days= 7               # how long before next CRL
default_md  = sha1                # which md to use.
preserve    = no                  # keep passed DN ordering

policy      = policy_match

# A new one - quells a complaint from openssl ca
#unique_subject = yes
```

```
#####################################################################
[ CA_dodcai]

dir         = /home/al/pki_project/ssl/DoDRootCA/DoDCAI   # Where everything is kept
certs       = $dir/UserCerts/certs                        # Where the issued certs are kept
#crl_dir    = $dir/crl                                    # Where the issued crl are kept
database    = $dir/index/index.txt                        # database index file.
new_certs_dir   = $dir/UserCerts/certs                    # default place for new certs.

certificate = $dir/dodcai.crt               # The CA certificate
serial      = $dir/serial/serial    # The current serial number
private_key = $dir/dodcai.key           # The private key
RANDFILE    = $dir/.rand                    # private random number file

x509_extensions = usr_cert      # The extentions to add to the cert

# Comment out the following two lines for the "traditional"
# (and highly broken) format.
name_opt    = ca_default        # Subject Name options
cert_opt    = ca_default        # Certificate field options

default_days    = 1085          # how long to certify for
default_crl_days= 7           # how long before next CRL
default_md  = sha1          # which md to use.
preserve    = no            # keep passed DN ordering

policy      = policy_user

#####################################################################

[ CA_dccai]

dir         = /home/al/pki_project/ssl/DoDRootCA/DoDDCCAI         # Where everything is kept
certs       = $dir/DCCerts                        # Where the issued certs are kept
#crl_dir    = $dir/crl                    # Where the issued crl are kept
database    = $dir/index/index.txt                # database index file.
new_certs_dir   = $dir/DCCerts                    # default place for new certs.

certificate = $dir/doddci.crt               # The CA certificate
serial      = $dir/serial/serial    # The current serial number
#crl        = $dir/crl.pem          # The current CRL
private_key = $dir/doddci.key                 # The private key
RANDFILE    = $dir/.rand                    # private random number file
```

```
x509_extensions = dc      # The extensions to add to the cert

# Comment out the following two lines for the "traditional"
# (and highly broken) format.
name_opt    = ca_default        # Subject Name options
cert_opt    = ca_default        # Certificate field options

default_days    = 1085          # how long to certify for
default_crl_days= 7           # how long before next CRL
default_md  = sha1          # which md to use.
preserve    = no          # keep passed DN ordering

policy      = policy_user
#####################################################################

[ CA_usercerts ]

dir             = /home/al/pki_project/ssl/DoDRootCA/DoDCAI/UserCerts         # Where everything
is kept
certs           = $dir/DCcerts                          # Where the issued certs are kept
#crl_dir        = $dir/crl                          # Where the issued crl are kept
database        = $dir/index/index.txt                  # database index file.
new_certs_dir   = $dir/DCcerts                          # default place for new certs.

certificate     = $dir/usercerts.crt        # The CA certificate
serial          = $dir/serial/serial    # The current serial number
#crl            = $dir/crl.pem          # The current CRL
private_key     = $dir/usercerts.key    # The private key
RANDFILE        = $dir/.rand            # private random number file

x509_extensions = dc              # The extentions to add to the cert

# Comment out the following two lines for the "traditional"
# (and highly broken) format.
name_opt        = ca_default            # Subject Name options
cert_opt        = ca_default           # Certificate field options

default_days    = 7                 # how long to certify for
default_crl_days= 7                   # how long before next CRL
default_md      = sha1                  # which md to use.
preserve        = no                # keep passed DN ordering

policy          = policy_dc
```

```
unique_subject = yes

# For the CA policy
[ policy_match ]
countryName             = match
#localityName          = match
organizationName        = match
organizationalUnitName  = optional
commonName              = supplied
emailAddress            = optional

[policy_user]
countryName             = match
#localityName          = match
organizationName        = match
organizationalUnitName  = optional
commonName              = supplied
emailAddress            = supplied

[policy_dc]
countryName             = match
#localityName          = match
organizationName        = match
organizationalUnitName  = optional
commonName              = supplied
emailAddress            = match

####################################################################
[ req ]
default_bits        = 1024
default_keyfile     = privkey.pem
distinguished_name  = req_distinguished_name
x509_extensions     = v3_req # The extentions to add to the self signed cert

string_mask = nombstr

req_extensions = v3_req # The extensions to add to a certificate request

[ req_distinguished_name ]
C = US
O = U.S. Government
0.OU = DoD
1.OU = PKI
```

```
[dodrootca]

#These extensions are added to the self signed Root Cert
basicConstraints=CA:TRUE
keyUsage = digitalSignature, keyCertSign, cRLSign
subjectKeyIdentifier=hash

[dodcai]

#Extensions added to certs signded by the DoD Root for DoD CA I
basicConstraints=CA:TRUE
keyUsage = digitalSignature, keyCertSign, cRLSign
subjectKeyIdentifier=hash
authorityKeyIdentifier=keyid
certificatePolicies=2.16.840.1.101.2.1.11.5, 2.16.840.1.101.2.1.11.9, 2.16.840.1.101.2.1.11.10

[ usr_cert ]

# These extensions are added when 'ca' signs a user request.
basicConstraints=CA:TRUE
keyUsage = digitalSignature, keyCertSign, nonRepudiation
extendedKeyUsage= clientAuth, msSmartcardLogin, emailProtection
subjectKeyIdentifier=hash
authorityKeyIdentifier=keyid,issuer:always
certificatePolicies= 2.16.840.1.101.2.1.11.9, 2.16.840.1.101.2.1.11.19
subjectAltName=email:copy


[dc]
basicConstraints=CA:FALSE
keyUsage = digitalSignature, nonRepudiation
subjectKeyIdentifier=hash
authorityKeyIdentifier=keyid,issuer:always

[ v3_req ]

# Extensions to add to a certificate request

basicConstraints = CA:FALSE
keyUsage = nonRepudiation, digitalSignature, keyEncipherment


[ crl_ext ]
```

```
# CRL extensions.
# Only issuerAltName and authorityKeyIdentifier make any sense in a CRL.

# issuerAltName=issuer:copy
authorityKeyIdentifier=keyid:always,issuer:always
```

Appendix C: Mock DoD CA Setup

Setting up the simulated DoD Certificate Authority
########## Configuration File ##############
########## End Config ##############
############## Set up Directories ##############
##Set up all directories for CA's
sudo mkdir /DoDRootCA
sudo mkdir /DoDRootCA/DoDCAI
sudo mkdir /DoDRootCA/DoDCAI/UserCerts
sudo mkdir /DoDRootCA/DoDDCCAI
sudo mkdir /DoDRootCA/DoDDCCAI/DCCerts

Set up directory for User and DC certs

sudo mkdir /DoDRootCA/DoDCAI/UserCerts/certs
sudo mkdir /DoDRootCA/DoDCAI/UserCerts/DCCerts
sudo mkdir /DoDRootCA/DoDCAI/UserCerts/private_keys

sudo mkdir /DoDRootCA/DoDDCCAI/certs
sudo mkdir /DoDRootCA/DoDDCCAI/P12
sudo mkdir /DoDRootCA/DoDDCCAI/private_keys

In each folder for CA's and User/DC folders
sudo mkdir index
sudo mkdir serial
cd ./index
touch index.txt
cd ./serial
echo "01" > serial

############# End Directories ##################################
############ Self Signed CA ############

Generate Root CA Private Key and CSR
pass phrase "root"

al@al-gnome:~/thesis/ssl/DoDRootCA$ openssl req -newkey rsa:2048 -sha1 -keyout dodrootca.key -out dodrootca.csr -config /home/al/thesis/ssl/openssl.cnf
Generating a 2048 bit RSA private key
.....+++
.....+++

writing new private key to 'dodrootca.key'
Enter PEM pass phrase:
Verifying - Enter PEM pass phrase:

You are about to be asked to enter information that will be incorporated
into your certificate request.
What you are about to enter is what is called a Distinguished Name or a DN.
There are quite a few fields but you can leave some blank
For some fields there will be a default value,
If you enter '.', the field will be left blank.

US [US]:
Organization [U.S. Government]:
Organizatinal Unit [DoD]:
Organizatinal Unit [PKI]:
Common Name (eg, YOUR name) []:DoD Root CA Sim
Email Address []:

Self Sign the Root CA Sim Cert

al@al-gnome:~/thesis/ssl/DoDRootCA$ openssl x509 -req -in dodrootca.csr -sha1 -signkey dodrootca.key -out dodrootca.crt -extensions dodrootca -days 3650
Signature ok
subject=/C=US/O=U.S. Government/OU=DoD/OU=PKI/CN=DoD Root CA Sim
Getting Private key
Enter pass phrase for dodrootca.key:

Install the certificate
sudo mkdir /usr/share/ca-certificates/extra
sudo cp dodrootca.pem /usr/share/ca-certificates/extra/dodrootca.crt
sudo dpkg-reconfigure ca-certificates
#select the /extra/dodrootca.crt to install
sudo update-ca-certificates

#Verify the root certificate
al@al-gnome:~/thesis/ssl/DoDRootCA$ openssl verify dodrootca.crt
dodrootca.crt: OK

############### End CA ######################
########### Set up Intermediate CAs ########
####### DoD CA I #####
al@al-gnome:~/thesis/ssl/DoDRootCA$ cd DoDCAI/
al@al-gnome:~/thesis/ssl/DoDRootCA/DoDCAI$ openssl genrsa -des3 -out dodcai.key 2048
Generating RSA private key, 2048 bit long modulus

.....+++
.....+++
e is 65537 (0x10001)
Enter pass phrase for dodcai.key:
Verifying - Enter pass phrase for dodcai.key:

##pass phrase "dodcai"

##Generate CSR
al@al-gnome:~/thesis/ssl/DoDRootCA/DoDCAI$ openssl req -new -key dodcai.key -config /home/al/thesis/ssl/openssl.cnf -out dodcai.csr
Enter pass phrase for dodcai.key:
You are about to be asked to enter information that will be incorporated
into your certificate request.
What you are about to enter is what is called a Distinguished Name or a DN.
There are quite a few fields but you can leave some blank
For some fields there will be a default value,
If you enter '.', the field will be left blank.

US [US]:
Organization [U.S. Government]:
Organizatinal Unit [DoD]:
Organizatinal Unit [PKI]:
Common Name (eg, YOUR name) []:DoD CA I Sim
Email Address []:

#Sign CSR with the DoD Root CA
al@al-gnome:~/thesis/ssl/DoDRootCA/DoDCAI$ openssl ca -config /home/al/thesis/ssl/openssl.cnf -name CA_dodrootca -in dodcai.csr -out dodcai.crt
Using configuration from /home/al/thesis/ssl/openssl.cnf
Enter pass phrase for /home/al/thesis/ssl/DoDRootCA/dodrootca.key:
Check that the request matches the signature
Signature ok
Certificate Details:
    Serial Number: 1 (0x1)
    Validity
      Not Before: Mar 1 19:22:44 2014 GMT
      Not After : Feb 27 19:22:44 2024 GMT
    Subject:
      countryName   = US
      organizationName   = U.S. Government
      organizationalUnitName = DoD
      organizationalUnitName = PKI
      commonName   = DoD CA I Sim

X509v3 extensions:
X509v3 Basic Constraints:
CA:TRUE
X509v3 Key Usage:
Digital Signature, Certificate Sign, CRL Sign
X509v3 Subject Key Identifier:
1F:EE:0E:91:B3:0B:4A:D8:BE:7B:90:B8:14:F5:84:1B:C0:DD:C7:4B
X509v3 Authority Key Identifier:
0.
X509v3 Certificate Policies:
Policy: 2.16.840.1.101.2.1.11.5
Policy: 2.16.840.1.101.2.1.11.9
Policy: 2.16.840.1.101.2.1.11.10

Certificate is to be certified until Feb 27 19:22:44 2024 GMT (3650 days)
Sign the certificate? [y/n]:y

1 out of 1 certificate requests certified, commit? [y/n]y
Write out database with 1 new entries
Data Base Updated

Verify
al@al-gnome:~$ openssl verify /home/al/thesis/ssl/DoDRootCA/DoDCAI/dodcai.crt
/home/al/thesis/ssl/DoDRootCA/DoDCAI/dodcai.crt: OK

########## DoD DC Intermediate ######################

Generate RSA Key
al@al-gnome:~/thesis/ssl/DoDRootCA/DoDDCCAI$ openssl genrsa -des3 -out ./private_keys/doddci.key 2048
Generating RSA private key, 2048 bit long modulus
.....+++
.....+++
e is 65537 (0x10001)
Enter pass phrase for ./private_keys/doddci.key:
Verifying - Enter pass phrase for ./private_keys/doddci.key:

##Pass phrase "doddci"

Generate CSR

al@al-gnome:~/thesis/ssl/DoDRootCA/DoDDCCAI$ openssl req -new -key ./private_keys/doddci.key -config /home/al/thesis/ssl/openssl.cnf -out doddci.csr

Enter pass phrase for ./private_keys/doddci.key:
You are about to be asked to enter information that will be incorporated
into your certificate request.
What you are about to enter is what is called a Distinguished Name or a DN.
There are quite a few fields but you can leave some blank
For some fields there will be a default value,
If you enter '.', the field will be left blank.

US [US]:
Organization [U.S. Government]:
Organizatinal Unit [DoD]:
Organizatinal Unit [PKI]:
Common Name (eg, YOUR name) []:DoD DC CA I Sim
Email Address []:

Sign the CSR with the DoD Root CA

al@al-gnome:~/thesis/ssl/DoDRootCA/DoDDCCAI$ openssl ca -config /home/al/thesis/ssl/openssl.cnf -name CA_dodrootca -in doddci.csr -out doddci.crt
Using configuration from /home/al/thesis/ssl/openssl.cnf
Enter pass phrase for /home/al/thesis/ssl/DoDRootCA/dodrootca.key:
Check that the request matches the signature
Signature ok
Certificate Details:
    Serial Number: 2 (0x2)
    Validity
      Not Before: Mar 1 19:51:51 2014 GMT
      Not After : Feb 27 19:51:51 2024 GMT
    Subject:
      countryName   = US
      organizationName   = U.S. Government
      organizationalUnitName = DoD
      organizationalUnitName = PKI
      commonName   = DoD DC CA I Sim
    X509v3 extensions:
      X509v3 Basic Constraints:
        CA:TRUE
      X509v3 Key Usage:
        Digital Signature, Certificate Sign, CRL Sign
      X509v3 Subject Key Identifier:
        40:C1:8C:FE:D4:64:B9:76:8D:B2:7A:07:59:B1:FC:9A:F4:7F:AC:21
      X509v3 Authority Key Identifier:
        0.
      X509v3 Certificate Policies:

Policy: 2.16.840.1.101.2.1.11.5
Policy: 2.16.840.1.101.2.1.11.9
Policy: 2.16.840.1.101.2.1.11.10

Certificate is to be certified until Feb 27 19:51:51 2024 GMT (3650 days)
Sign the certificate? [y/n]:y

1 out of 1 certificate requests certified, commit? [y/n]y
Write out database with 1 new entries
Data Base Updated

Appendix D: User Certificates on Gemalto PIV II Smart Card

D.1 User1 Authentication Certificate

Generate Keys on the card

Set Path to the PIV_EXT_AUTH_KEY
al@al-gnome:~$ export PIV_EXT_AUTH_KEY=/home/al/pki_project/gemalto_card/piv_auth_key
Generate a RSA:2048 key
al@al-gnome:~$ piv-tool -A M:9B:03 -G 9A:07 -o /home/al/pki_project/gemalto_card/pubkey.01.user2

Generate CSR
Set path for slot key (use output from step 1)
al@al-gnome:~$ export PIV_9A_KEY=/home/al/pki_project/gemalto_card/pubkey.1.user2

al@al-gnome:~$ openssl
Set Engine first
OpenSSL> engine dynamic -vvvv -pre SO_PATH:/usr/lib/ssl/engines/engine_pkcs11.so -pre ID:pkcs11 -pre NO_VCHECK:1 -pre LIST_ADD:1 -pre LOAD -pre MODULE_PATH:/usr/lib/x86_64-linux-gnu/opensc-pkcs11.so
(dynamic) Dynamic engine loading support
[Success]: SO_PATH:/usr/lib/ssl/engines/engine_pkcs11.so
[Success]: ID:pkcs11
[Success]: NO_VCHECK:1
[Success]: LIST_ADD:1
[Success]: LOAD
[Success]: MODULE_PATH:/usr/lib/x86_64-linux-gnu/opensc-pkcs11.so
Loaded: (pkcs11) pkcs11 engine
    SO_PATH: Specifies the path to the 'pkcs11-engine' shared library
        (input flags): STRING
    MODULE_PATH: Specifies the path to the pkcs11 module shared library
        (input flags): STRING
    PIN: Specifies the pin code
        (input flags): STRING
    VERBOSE: Print additional details

(input flags): NO_INPUT
QUIET: Remove additional details
(input flags): NO_INPUT
LOAD_CERT_CTRL: Get the certificate from card
(input flags): [Internal]
INIT_ARGS: Specifies additional initialization arguments to the pkcs11 module
(input flags): STRING

Generate CSR

```
OpenSSL> req -new -config /home/al/pki_project/ssl/openssltest.cnf -engine pkcs11 -sha1 -key slot_1-id_01 -keyform engine -out /home/al/pki_project/ssl/usercert.csr -text
engine "pkcs11" set.
PKCS#11 token PIN: ################## USE 1234 Auth PIN ###############
You are about to be asked to enter information that will be incorporated
into your certificate request.
What you are about to enter is what is called a Distinguished Name or a DN.
There are quite a few fields but you can leave some blank
For some fields there will be a default value,
If you enter '.', the field will be left blank.
-----
Country Name (2 letter code) [US]:
Organization Name (eg, company) [U.S. Government]:
Second Organization Name (eg, company) [PKI]:
Organizational Unit Name (eg, section) [DoD]:
Common Name (eg, YOUR name) []:User 2
Email Address []:user2@mil.mil
OpenSSL> quit
al@al-gnome:~$ openssl ca -config /home/al/pki_project/ssl/openssl.cnf -name CA_dodcai -in /home/al/pki_project/ssl/usercert.csr -out /home/al/pki_project/ssl/DoDRootCA/DoDCAI/UserCerts/certs/user2.crt
Using configuration from /home/al/pki_project/ssl/openssl.cnf
Enter pass phrase for /home/al/pki_project/ssl/DoDRootCA/DoDCAI/dodcai.key:
Check that the request matches the signature
Signature ok
Certificate Details:
        Serial Number: 2 (0x2)
        Validity
            Not Before: Mar 11 01:41:11 2014 GMT
            Not After : Feb 28 01:41:11 2017 GMT
        Subject:
            countryName               = US
            organizationName          = U.S. Government
            organizationalUnitName    = PKI
            organizationalUnitName    = DoD
```

commonName = User 2
emailAddress = user2@mil.mil
X509v3 extensions:
X509v3 Basic Constraints:
CA:TRUE
X509v3 Key Usage:
Digital Signature, Non Repudiation, Certificate Sign
X509v3 Extended Key Usage:
TLS Web Client Authentication, Microsoft Smartcardlogin, E-mail Protection
X509v3 Subject Key Identifier:
6B:4A:D4:B2:CB:C9:B0:22:12:12:7C:9A:F5:11:19:9F:3A:11:DA:5D
X509v3 Authority Key Identifier:
keyid:1F:EE:0E:91:B3:0B:4A:D8:BE:7B:90:B8:14:F5:84:1B:C0:DD:C7:4B
DirName:/C=US/O=U.S. Government/OU=DoD/OU=PKI/CN=DoD Root CA Sim
serial:01

X509v3 Certificate Policies:
Policy: 2.16.840.1.101.2.1.11.9
Policy: 2.16.840.1.101.2.1.11.19

Certificate is to be certified until Feb 28 01:41:11 2017 GMT (1085 days)
Sign the certificate? [y/n]:y

1 out of 1 certificate requests certified, commit? [y/n]y
Write out database with 1 new entries
Data Base Updated

Store the Certificate on the Card

Set path for Auth Key again
al@al-gnome:~$ export PIV_EXT_AUTH_KEY=/home/al/pki_project/gemalto_card/piv_auth_key

Store Cert on card
al@al-gnome:~$ piv-tool -A M:9B:03 -C 9A -i /home/al/pki_project/ssl/DoDRootCA/DoDCAI/UserCerts/certs/user2.crt
Using reader with a card: SCM Microsystems Inc. SCR 331 [CCID Interface] (21120710216109) 00 00

Verify private key on card (Can also check certificate with --list-certificate
al@al-gnome:~$ pkcs15-tool -k
Using reader with a card: SCM Microsystems Inc. SCR 331 [CCID Interface] (21120710216109) 00 00

```
Private RSA Key [PIV AUTH key]
        Object Flags   : [0x1], private
        Usage          : [0x2E], decrypt, sign, signRecover, unwrap
        Access Flags   : [0x1D], sensitive, alwaysSensitive, neverExtract, local
        ModLength      : 2048
        Key ref        : 154 (0x9A)
        Native         : yes
        Auth ID        : 01
        ID             : 01

#############################################################################


####    Verify Signature on Smart Card
al@al-gnome:~/pki_project/ssl/DoDRootCA/DoDCAI/UserCerts/DCcerts/test$ pkcs15-tool --read-certificate 02 --output user2digsig.crt
Using reader with a card: SCM Microsystems Inc. SCR 331 [CCID Interface] (21120710216109) 00 00
al@al-gnome:~/pki_project/ssl/DoDRootCA/DoDCAI/UserCerts/DCcerts/test$ ls
hashed  message  signed  user2digsig.crt
al@al-gnome:~/pki_project/ssl/DoDRootCA/DoDCAI/UserCerts/DCcerts/test$ openssl x509 -in ./user2digsig.crt -pubkey -noout > user2digsigpub.pem

al@al-gnome:~/pki_project/ssl/DoDRootCA/DoDCAI/UserCerts/DCcerts/test$ openssl dgst -verify user2digsigpub.pem -sha256 -signature signed < message
Verified OK
```

D.2 User 1 Digital Signature Certificate

```
########        USER 1 DigSig
al@al-gnome:~$                                                          export PIV_EXT_AUTH_KEY=/home/al/pki_project/gemalto_card/piv_auth_key
al@al-gnome:~$          piv-tool        -A      M:9B:03         -G      9C:07           -o /home/al/pki_project/gemalto_card/pubkey.02.user1
Using reader with a card: SCM Microsystems Inc. SCR 331 [CCID Interface] (21120710216109) 00 00
al@al-gnome:~$ export PIV_9C_KEY=/home/al/pki_project/gemalto_card/pubkey.02.user1

al@al-gnome:~$ openssl
OpenSSL> engine dynamic -vvvv -pre SO_PATH:/usr/lib/ssl/engines/engine_pkcs11.so -pre ID:pkcs11 -pre NO_VCHECK:1 -pre LIST_ADD:1 -pre LOAD -pre MODULE_PATH:/usr/lib/x86_64-linux-gnu/opensc-pkcs11.so
```

(dynamic) Dynamic engine loading support
[Success]: SO_PATH:/usr/lib/ssl/engines/engine_pkcs11.so
[Success]: ID:pkcs11
[Success]: NO_VCHECK:1
[Success]: LIST_ADD:1
[Success]: LOAD
[Success]: MODULE_PATH:/usr/lib/x86_64-linux-gnu/opensc-pkcs11.so
Loaded: (pkcs11) pkcs11 engine
    SO_PATH: Specifies the path to the 'pkcs11-engine' shared library
        (input flags): STRING
    MODULE_PATH: Specifies the path to the pkcs11 module shared library
        (input flags): STRING
    PIN: Specifies the pin code
        (input flags): STRING
    VERBOSE: Print additional details
        (input flags): NO_INPUT
    QUIET: Remove additional details
        (input flags): NO_INPUT
    LOAD_CERT_CTRL: Get the certificate from card
        (input flags): [Internal]
    INIT_ARGS: Specifies additional initialization arguments to the pkcs11 module
        (input flags): STRING
OpenSSL> req -new -config /home/al/pki_project/ssl/openssltest.cnf -engine pkcs11 -sha1 -key slot_1-id_02 -keyform engine -out /home/al/pki_project/ssl/user1digsig02.csr -text
engine "pkcs11" set.
PKCS#11 token PIN:
You are about to be asked to enter information that will be incorporated
into your certificate request.
What you are about to enter is what is called a Distinguished Name or a DN.
There are quite a few fields but you can leave some blank
For some fields there will be a default value,
If you enter '.', the field will be left blank.

Country Name (2 letter code) [US]:
Organization Name (eg, company) [U.S. Government]:
Second Organization Name (eg, company) [PKI]:
Organizational Unit Name (eg, section) [DoD]:
Common Name (eg, YOUR name) []:User 1 DigSig
Email Address []:user1@mil.mil
OpenSSL> quit
al@al-gnome:~$ openssl ca -config /home/al/pki_project/ssl/openssl.cnf -name CA_dodcai -in /home/al/pki_project/ssl/user1digsig.csr -out /home/al/pki_project/ssl/DoDRootCA/DoDCAI/UserCerts/certs/user1digsig.crt
Using configuration from /home/al/pki_project/ssl/openssl.cnf

```
Enter pass phrase for /home/al/pki_project/ssl/DoDRootCA/DoDCAI/dodcai.key:
Check that the request matches the signature
Signature ok
Certificate Details:
        Serial Number: 4 (0x4)
        Validity
            Not Before: Mar 11 04:19:36 2014 GMT
            Not After : Feb 28 04:19:36 2017 GMT
        Subject:
            countryName               = US
            organizationName          = U.S. Government
            organizationalUnitName    = PKI
            organizationalUnitName    = DoD
            commonName                = User 1 DigSig
            emailAddress              = user1@mil.mil
        X509v3 extensions:
            X509v3 Basic Constraints:
                CA:TRUE
            X509v3 Key Usage:
                Digital Signature, Non Repudiation, Certificate Sign
            X509v3 Extended Key Usage:
                TLS Web Client Authentication, Microsoft Smartcardlogin, E-mail Protection
            X509v3 Subject Key Identifier:
                B2:96:96:32:AC:F1:81:58:A9:A3:AE:A1:C0:F7:94:A8:4C:57:FD:23
            X509v3 Authority Key Identifier:
                keyid:1F:EE:0E:91:B3:0B:4A:D8:BE:7B:90:B8:14:F5:84:1B:C0:DD:C7:4B
                DirName:/C=US/O=U.S. Government/OU=DoD/OU=PKI/CN=DoD Root CA Sim
                serial:01

            X509v3 Certificate Policies:
                Policy: 2.16.840.1.101.2.1.11.9
                Policy: 2.16.840.1.101.2.1.11.19

Certificate is to be certified until Feb 28 04:19:36 2017 GMT (1085 days)
Sign the certificate? [y/n]:y


1 out of 1 certificate requests certified, commit? [y/n]y
Write out database with 1 new entries
Data Base Updated
```

Appendix E: CAC Signed Derived Credential

E.1 Python Implementation Code

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-

import os
import subprocess
import shlex
import ConfigParser
import time
import shutil
import qrcode
import shutil

class GenDerivedCredential:

    _CERT_FROM_SMARTCARD_NAME = 'digsig.ctr'
    _CERT_FROM_SMARTCARD_PUBKEY = 'digsig.pem'
    _CERT_PARSED_OUTFILE = 'certParse.txt'
    _PRIVATE_KEY_FOLDER = 'DoDRootCA/DoDCAI/UserCerts/DCcerts/private_keys'
    _CERTS_DIR = 'DoDRootCA/DoDCAI/UserCerts/DCcerts'
    _P12_DIR = 'DoDRootCA/DoDCAI/UserCerts/DCcerts/P12'
    _DEST_P12_DIR = '/var/www/p12files/'

    def __init__(self, opensslConfigFile, rootDir, dodCA, serverIpAddress):

        self.opensslConfigFile = opensslConfigFile
        self.rootDir = rootDir
        self.dodCA = dodCA
        self.serverIpAddress = serverIpAddress

        self.workingDirTmp = os.path.join(self.rootDir, 'tmp')

        if(not os.path.isdir(self.workingDirTmp)):
            os.mkdir(self.workingDirTmp)

        self.certsDir = os.path.join(self.rootDir, GenDerivedCredential._CERTS_DIR)

        self.p12Dir = os.path.join(self.rootDir, GenDerivedCredential._P12_DIR)
```

```
        if(not os.path.isdir(self.p12Dir)):
            os.mkdir(self.p12Dir)

        # make the private keys directory if we need to
        self.pkeysDir = os.path.join(self.rootDir,
GenDerivedCredential._PRIVATE_KEY_FOLDER)
        if not os.path.isdir(self.pkeysDir):
            os.mkdir(self.pkeysDir)

        self.certFromSmartcard = os.path.join(self.workingDirTmp,
GenDerivedCredential._CERT_FROM_SMARTCARD_NAME)
        self.certFromSmartcardPubkey = os.path.join(self.workingDirTmp,
GenDerivedCredential._CERT_FROM_SMARTCARD_PUBKEY)
        self.certParsed = os.path.join(self.workingDirTmp,
GenDerivedCredential._CERT_PARSED_OUTFILE)

        self.getCertFromSmartCard()
        good = self.authenticateCAC()
        if good:
            self.parseCertificate()
            self.genDerivedCredential()
            self.deliverCredential()
        else:
            print 'CAC Card Not Authenticated!'

#        self.cleanup()

    def getCertFromSmartCard(self):
        commandLine = 'pkcs15-tool --read-certificate 02 --output ' + self.certFromSmartcard
        args = shlex.split(commandLine)
        p = subprocess.Popen(args)

        # wait for the file to show up
        fileExists = os.path.exists(self.certFromSmartcard)
        while not fileExists:
            time.sleep(1)
            fileExists = os.path.exists(self.certFromSmartcard)
        print 'Successfully got certificate from SmartCard!'

        # generate the public key from the certificate
        commandLine = 'openssl x509 -in ' + self.certFromSmartcard + ' -pubkey -noout'
        args = shlex.split(commandLine)
        p = subprocess.Popen(args, stdout=subprocess.PIPE)
        stdout, stderr = p.communicate()
```

```
    f = open(self.certFromSmartcardPubkey, 'w')
    f.write(stdout)
    f.close()

    # wait for the file to show up
    fileExists = os.path.exists(self.certFromSmartcardPubkey)
    while not fileExists:
        time.sleep(1)
        fileExists = os.path.exists(self.certFromSmartcardPubkey)
    print 'Successfully generated certificate public key!'


def authenticateCAC(self):
    commandline = 'openssl verify -CAfile ' + self.dodCA + ' ' + self.certFromSmartcard
    args = shlex.split(commandline)
    p = subprocess.Popen(args, stdout=subprocess.PIPE)
    out,err = p.communicate()
    print '*** CAFile VERIFICATION ***'
    print out
    # parse 'out' to make sure it looks like this: /home/al/pki_project/ssl/tmp/digsig.ctr: OK
    if('OK' not in out):
        return False

    time.sleep(1)

    # create a dummy message to sign w/ cac card's private key
    msgFile = os.path.join(self.workingDirTmp, 'message')
    f = open(msgFile,'w')
    f.write('sometext')
    f.close()

    time.sleep(1)

    # create a hash of the message
    hashedFile = os.path.join(self.workingDirTmp, 'hashed')
    commandLine = 'openssl dgst -sha256 -binary -out ' + hashedFile + ' ' + msgFile
    args = shlex.split(commandLine)
    p = subprocess.Popen(args)

    time.sleep(1)

    # sign the hashed message w/ the cac card's private key
    signedFile = os.path.join(self.workingDirTmp, 'out.signed')
    commandLine= 'pkcs15-crypt -s --sha-256 --pkcs1 -k 02 -i ' + hashedFile + ' -o ' +
```

```
signedFile
        args = shlex.split(commandLine)
        p = subprocess.Popen(args)
        p.communicate()

        # verify
        commandLine = 'openssl dgst -verify ' + self.certFromSmartcardPubkey + ' -sha256 -
signature ' + signedFile + ' ' + msgFile
        print commandLine
        args = shlex.split(commandLine)
        p = subprocess.Popen(args, stdout=subprocess.PIPE)
        out,err = p.communicate()
        print '*** DigSig VERIFICATION ***'
        print out
        if ('OK' not in out):
            return False

        return True

    def parseCertificate(self):
        f = open(self.certParsed, 'w')
        # run openssl on this cert to get the info
        commandLine = 'openssl x509 -in ' + self.certFromSmartcard + ' -text -noout'
        args = shlex.split(commandLine)
        subprocess.Popen(args, stdout=f)
        f.flush()
        os.fsync(f.fileno())
        f.close()

        # wait 1 second before proceeding, sometimes the file isn't showing up :(
        time.sleep(1)

        # read it line by line, and look for the 'Subject:', if so parse it
        with open(self.certParsed) as f:
            for line in f:
                # look for the commmon name
                if 'Subject:' in line:
                    # parse this line for the CN
                    lineWithoutWhitespace = line.strip()
                    lineSplit = lineWithoutWhitespace.split(',')
                    # loop through and find the CN= tag
                    for keyValPair in lineSplit:
                        kvPair = keyValPair.strip()
                        kvPairSplit = kvPair.split('=')
```

```
                k = kvPairSplit[0]
                v = kvPairSplit[1]
                if(k=='CN'):
                    nameAndEmail = v
                    nameAndEmailSplit = nameAndEmail.split('/')
                    name = nameAndEmailSplit[0]
                    self.cn = '.'.join(name.split())
                    self.email = kvPairSplit[2]

    print 'Common Name = ' + self.cn
    print 'Email = ' + self.email

    self.genTempConfigFileFromOrig(self.cn, self.email)

def genTempConfigFileFromOrig(self, commonName, email):
    f = open(self.opensslConfigFile, 'r')
    origConfigFileLines = f.readlines()
    newConfigFile = list(origConfigFileLines)
    origIdx = 0
    newIdx = 0
    for line in origConfigFileLines:
        if '[ req ]' in line:
            newIdx = newIdx + 1
            lineToAdd = 'prompt = no\n'
            newConfigFile.insert(newIdx, lineToAdd)

        if '[ req_distinguished_name ]' in line:
            newIdx = newIdx + 1
            # add the commonName_default
            lineToAdd = 'CN=' + self.cn+'\n'
            # add the line
            newConfigFile.insert(newIdx, lineToAdd)
            newIdx = newIdx + 1
            lineToAdd = 'emailAddress=' + self.email+'\n'
            newConfigFile.insert(newIdx, lineToAdd)

        origIdx = origIdx + 1
        newIdx = newIdx + 1

    self.tmpSSLConfigFile = self.opensslConfigFile+'.tmp'
    f = open(self.tmpSSLConfigFile, 'w')
    for item in newConfigFile:
        f.write("%s" % item)
    f.close()
```

```
    def genDerivedCredential(self):
        # generating the key
        self.keyName = os.path.join(self.pkeysDir,self.cn+'key.key')
        commandLine = 'openssl genrsa -out ' + self.keyName + ' 2048'
        args = shlex.split(commandLine)
        subprocess.Popen(args)

        time.sleep(1)

        # generate the csr
        self.csrFile = os.path.join(self.workingDirTmp, self.cn + 'dc.csr')
        commandLine = 'openssl req -new -key ' + self.keyName + ' -config ' +
self.tmpSSLConfigFile + ' -out ' + self.csrFile
        args = shlex.split(commandLine)
        p = subprocess.Popen(args)

        time.sleep(1)
        os.unlink(self.tmpSSLConfigFile)

        # sign the csr w/ the CA
        self.certFile = os.path.join(self.certsDir, self.cn+'dc.crt')

#        ############ AL ADDED BATCH TO NOT PROMPT FOR CA SIGNING
#        commandLine = 'openssl ca -batch -config ' + self.opensslConfigFile + ' -key doddci -
name CA_dccai -in ' + self.csrFile + ' -out ' + self.certFile
#        args = shlex.split(commandLine)
#        p = subprocess.Popen(args)
#        p.communicate()
        p = subprocess.Popen('openssl', stdin=subprocess.PIPE, stdout=subprocess.PIPE)
        p.communicate(input='engine dynamic -pre SO_PATH:/usr/lib/engines/engine_pkcs11.so -
pre ID:pkcs11 -pre NO_VCHECK:1 -pre LIST_ADD:1 -pre LOAD ' + \
                    '-pre MODULE_PATH:/usr/lib/x86_64-linux-gnu/opensc-pkcs11.so\nca -engine
pkcs11 -batch ' + \
                    '-config ' + self.opensslConfigFile + ' -cert ' + self.certFromSmartcard +\
                    ' -keyform engine -keyfile slot_1-id_02 -name CA_usercerts -in ' + self.csrFile +\
                    ' -out ' + self.certFile + '\n')

        # make the P12 file
        self.p12Filename = self.cn+'dc.p12'
        self.p12File = os.path.join(self.p12Dir, self.p12Filename)
        commandLine = 'openssl pkcs12 -export -inkey ' + self.keyName + ' -in ' + self.certFile + ' -
out ' + self.p12File
        args = shlex.split(commandLine)
```

```
        p = subprocess.Popen(args)
        p.communicate()

        # copy the p12 file to the apache directory where it will be delivered
        shutil.copy(self.p12File, GenDerivedCredential._DEST_P12_DIR)
        self.p12FileInWebDir = os.path.join(GenDerivedCredential._DEST_P12_DIR,
self.p12Filename)

    def deliverCredential(self):
        hyperlink = self.serverIpAddress + '/p12files/' + self.p12Filename

        qr = qrcode.QRCode(
            version=1,
            error_correction=qrcode.constants.ERROR_CORRECT_L,
            box_size=10,
            border=4,
        )
        qr.add_data(hyperlink)

        qr.make(fit=True)
        img = qr.make_image()

        qrCodeFile = '/home/al/pki_project/qr_png/p12_qr.png'
        f = open(qrCodeFile,'w')
        img.save(f)
        f.close()

        #subprocess.Popen([qrCodeFile], shell=True)

    def cleanup(self):
        time.sleep(5)
        # delete everything from the tmp directory
        flist = os.listdir(self.workingDirTmp)

        for f in flist:
            try:
                os.unlink(os.path.join(self.workingDirTmp,f))
            except Exception as e:
                print e

if __name__=='__main__':
    rootDir = '/home/al/pki_project/ssl'
    ####### AL CHANGED THE PEM FILE TO POINT TO USER CA INSTEAD OF DOD
    dodCA = '/home/al/pki_project/auth_ca_certs/simdodcacerts.pem'
```

```
opensslConfigFile = '/home/al/pki_project/ssl/openssl.cnf'
serverIpAddress = '192.168.1.12'
gdc = GenDerivedCredential(opensslConfigFile, rootDir, dodCA, serverIpAddress)
```

E.2 CAC Signed DC Results

```
al@al-gnome:~/pki_project/python$ python SignCSRWithSmartCard.py
Successfully got certificate from SmartCard!
Successfully generated certificate public key!
*** CAFile VERIFICATION ***
/home/al/pki_project/ssl/tmp/digsig.ctr: OK

Using reader with a card: SCM Microsystems Inc. SCR 331 [CCID Interface] (21120710216109)
00 00
Using reader with a card: SCM Microsystems Inc. SCR 331 [CCID Interface] (21120710216109)
00 00
Enter PIN [PIV Card Holder pin]:
openssl dgst -verify /home/al/pki_project/ssl/tmp/digsig.pem -sha256 -signature
/home/al/pki_project/ssl/tmp/out.signed /home/al/pki_project/ssl/tmp/message
*** DigSig VERIFICATION ***
Verified OK

Common Name = User.2.Dig.Sig
Email = user2@mil.mil
Generating RSA private key, 2048 bit long modulus
...+++
.....+++
e is 65537 (0x10001)
Using configuration from /home/al/pki_project/ssl/openssl.cnf
engine "pkcs11" set.
PKCS#11 token PIN:
Check that the request matches the signature
Signature ok
Certificate Details:
        Serial Number: 1 (0x1)
        Validity
            Not Before: Apr 13 14:56:52 2014 GMT
            Not After : Apr 20 14:56:52 2014 GMT
        Subject:
            countryName               = US
            organizationName          = U.S. Government
            organizationalUnitName    = DoD
```

```
            organizationalUnitName    = PKI
            commonName                = User.2.Dig.Sig
            emailAddress              = user2@mil.mil
        X509v3 extensions:
            X509v3 Basic Constraints:
                CA:FALSE
            X509v3 Key Usage:
                Digital Signature, Non Repudiation
            X509v3 Subject Key Identifier:
                53:6F:05:65:B1:B9:4B:12:00:A8:1C:7C:DA:96:C3:20:E5:17:AC:56
            X509v3 Authority Key Identifier:
                keyid:5E:1A:EB:F8:B8:64:08:FA:8B:81:53:F5:56:4C:2D:AC:8D:BD:4C:3D
                DirName:/C=US/O=U.S. Government/OU=DoD/OU=PKI/CN=DoD CA I Sim
                serial:03

Certificate is to be certified until Apr 20 14:56:52 2014 GMT (7 days)

Write out database with 1 new entries
Data Base Updated
Enter Export Password:
Verifying - Enter Export Password:
al@al-gnome:~/pki_project/python$
```

E.3: CAC Signed Derived Credential Certificate

```
al@al-gnome:~/pki_project/ssl/DoDRootCA/DoDCAI/UserCerts/DCcerts$ openssl x509 -in
User.2.Dig.Sigdc.crt -text
Certificate:
    Data:
        Version: 3 (0x2)
        Serial Number: 1 (0x1)
    Signature Algorithm: sha1WithRSAEncryption
        Issuer: C=US, O=U.S. Government, OU=PKI, OU=DoD, CN=User 2 Dig
Sig/emailAddress=user2@mil.mil
        Validity
            Not Before: Apr 13 14:56:52 2014 GMT
            Not After : Apr 20 14:56:52 2014 GMT
        Subject: C=US, O=U.S. Government, OU=DoD, OU=PKI,
CN=User.2.Dig.Sig/emailAddress=user2@mil.mil
        Subject Public Key Info:
```

```
        Public Key Algorithm: rsaEncryption
            Public-Key: (2048 bit)
            Modulus:
                00:c3:f2:94:4f:18:aa:28:71:c1:d6:39:05:31:ae:
                62:5f:f3:6c:53:a0:dd:06:31:bc:71:22:0a:21:3f:
                ca:b2:8e:a1:5c:ff:0b:0b:0d:bb:b6:4c:4c:a8:84:
                ef:74:83:d8:82:40:be:e8:c0:32:91:2e:46:07:67:
                a5:a0:66:e4:81:ca:86:61:fd:69:0d:2e:b4:fb:d3:
                85:db:2c:af:e5:e8:fc:32:bd:46:fb:75:6f:92:a8:
                06:60:d9:33:8f:2b:ea:c6:8e:5d:b9:98:bb:3b:59:
                75:9d:e0:e0:a7:a6:fa:09:67:40:cb:53:99:57:26:
                26:f1:2a:40:bc:3c:9d:d0:5c:f5:a6:0c:b0:be:24:
                f5:e9:a6:bf:e2:22:d4:12:60:c1:dc:96:58:49:4a:
                a8:ba:2e:63:8e:f9:37:a8:c0:67:48:54:26:b9:47:
                37:4b:78:0d:fe:f1:de:78:e9:c9:7c:ae:a2:48:83:
                03:3e:0d:c7:8b:7c:b3:06:90:90:b9:54:ec:5b:60:
                5d:4b:a4:48:ba:9f:e2:76:50:f7:be:ce:85:7b:75:
                bc:5d:49:f7:4b:56:20:a0:6e:c0:dd:99:68:51:0d:
                ed:7a:7d:b0:54:f1:8d:58:df:4f:9f:79:1f:ad:80:
                ca:ce:d5:8a:c3:27:5c:31:3d:7a:a5:43:8e:81:85:
                71:dd
            Exponent: 65537 (0x10001)
    X509v3 extensions:
        X509v3 Basic Constraints:
            CA:FALSE
        X509v3 Key Usage:
            Digital Signature, Non Repudiation
        X509v3 Subject Key Identifier:
            53:6F:05:65:B1:B9:4B:12:00:A8:1C:7C:DA:96:C3:20:E5:17:AC:56
        X509v3 Authority Key Identifier:
            keyid:5E:1A:EB:F8:B8:64:08:FA:8B:81:53:F5:56:4C:2D:AC:8D:BD:4C:3D
            DirName:/C=US/O=U.S. Government/OU=DoD/OU=PKI/CN=DoD CA I Sim
            serial:03

Signature Algorithm: sha1WithRSAEncryption
    1b:00:47:46:0d:17:3e:95:4d:d4:dc:29:26:d6:43:ef:0f:3e:
    ad:8b:ff:b6:18:ae:3f:07:2f:98:19:5b:e0:d1:d6:44:0d:f0:
    88:16:75:74:3f:b0:37:b5:38:32:92:10:42:91:8b:57:76:cb:
    7c:fc:37:94:56:29:f4:b6:87:30:fe:8f:42:2c:58:c6:c5:77:
    cd:67:e7:9f:43:f7:f5:fe:8d:cb:9c:d4:f7:d7:f9:df:b1:a8:
    af:07:09:dd:6b:63:11:30:33:e8:fb:b2:e8:51:b6:3f:89:37:
    48:df:13:cf:8e:a9:0b:92:24:81:cd:06:ea:68:e2:07:ea:2c:
    ad:2f:b8:d7:9d:e2:7c:7b:32:1b:6b:5b:47:22:3d:6e:65:e9:
    f2:40:59:df:00:37:1f:33:a4:03:0f:b5:8d:52:49:29:1a:4b:
```

c9:c8:8d:3b:11:20:a2:c1:ab:a3:0f:0d:1f:a7:8a:3d:66:bd:
ea:c1:85:77:ba:6a:39:4f:c4:02:0c:a8:df:4f:57:37:3a:9a:
30:42:24:99:9d:44:c8:03:d6:bb:91:1c:2f:fd:aa:80:b7:59:
76:b9:6b:8f:df:b2:8a:52:b3:21:e6:1e:2f:56:83:5b:11:03:
87:66:bb:7d:f8:7b:6e:ab:1b:64:2d:a9:8f:85:af:86:7b:79:
c0:d8:d1:e7

Appendix F: Server Driven Derived Credential

F.1: Python Implementation Code

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-

import os
import subprocess
import shlex
import ConfigParser
import time
import shutil
import qrcode
import shutil

class GenDerivedCredential:

    _CERT_FROM_SMARTCARD_NAME = 'digsig.ctr'
    _CERT_FROM_SMARTCARD_PUBKEY = 'digsig.pem'
    _CERT_PARSED_OUTFILE = 'certParse.txt'
    _PRIVATE_KEY_FOLDER = 'DoDRootCA/DoDDCCAI/private_keys'
    _CERTS_DIR = 'DoDRootCA/DoDDCCAI/DCCerts'
    _P12_DIR = 'DoDRootCA/DoDDCCAI/P12'
    _DEST_P12_DIR = '/var/www/p12files/'

    def __init__(self, opensslConfigFile, rootDir, dodCA, serverIpAddress):

        self.opensslConfigFile = opensslConfigFile
        self.rootDir = rootDir
        self.dodCA = dodCA
        self.serverIpAddress = serverIpAddress

        self.workingDirTmp = os.path.join(self.rootDir, 'tmp')

        if(not os.path.isdir(self.workingDirTmp)):
            os.mkdir(self.workingDirTmp)

        self.certsDir = os.path.join(self.rootDir, GenDerivedCredential._CERTS_DIR)

        self.p12Dir = os.path.join(self.rootDir, GenDerivedCredential._P12_DIR)
```

```
        if(not os.path.isdir(self.p12Dir)):
            os.mkdir(self.p12Dir)

        # make the private keys directory if we need to
        self.pkeysDir = os.path.join(self.rootDir,
GenDerivedCredential._PRIVATE_KEY_FOLDER)
        if not os.path.isdir(self.pkeysDir):
            os.mkdir(self.pkeysDir)

        self.certFromSmartcard = os.path.join(self.workingDirTmp,
GenDerivedCredential._CERT_FROM_SMARTCARD_NAME)
        self.certFromSmartcardPubkey = os.path.join(self.workingDirTmp,
GenDerivedCredential._CERT_FROM_SMARTCARD_PUBKEY)
        self.certParsed = os.path.join(self.workingDirTmp,
GenDerivedCredential._CERT_PARSED_OUTFILE)

        self.getCertFromSmartCard()
        good = self.authenticateCAC()
        if good:
            self.parseCertificate()
            self.genDerivedCredential()
            self.deliverCredential()
        else:
            print 'CAC Card Not Authenticated!'

#        self.cleanup()

    def getCertFromSmartCard(self):
        commandLine = 'pkcs15-tool --read-certificate 02 --output ' + self.certFromSmartcard
        args = shlex.split(commandLine)
        p = subprocess.Popen(args)

        # wait for the file to show up
        fileExists = os.path.exists(self.certFromSmartcard)
        while not fileExists:
            time.sleep(1)
            fileExists = os.path.exists(self.certFromSmartcard)
        print 'Successfully got certificate from SmartCard!'

        # generate the public key from the certificate
        commandLine = 'openssl x509 -in ' + self.certFromSmartcard + ' -pubkey -noout'
        args = shlex.split(commandLine)
        p = subprocess.Popen(args, stdout=subprocess.PIPE)
        stdout, stderr = p.communicate()
```

```
    f = open(self.certFromSmartcardPubkey, 'w')
    f.write(stdout)
    f.close()

    # wait for the file to show up
    fileExists = os.path.exists(self.certFromSmartcardPubkey)
    while not fileExists:
        time.sleep(1)
        fileExists = os.path.exists(self.certFromSmartcardPubkey)
    print 'Successfully generated certificate public key!'


def authenticateCAC(self):
    commandline = 'openssl verify -CAfile ' + self.dodCA + ' ' + self.certFromSmartcard
    args = shlex.split(commandline)
    p = subprocess.Popen(args, stdout=subprocess.PIPE)
    out,err = p.communicate()
    print '*** CAFile VERIFICATION ***'
    print out
    # parse 'out' to make sure it looks like this: /home/al/pki_project/ssl/tmp/digsig.ctr: OK
    if('OK' not in out):
        return False

    time.sleep(1)

    # create a dummy message to sign w/ cac card's private key
    msgFile = os.path.join(self.workingDirTmp, 'message')
    f = open(msgFile,'w')
    f.write('sometext')
    f.close()

    time.sleep(1)

    # create a hash of the message
    hashedFile = os.path.join(self.workingDirTmp, 'hashed')
    commandLine = 'openssl dgst -sha256 -binary -out ' + hashedFile + ' ' + msgFile
    args = shlex.split(commandLine)
    p = subprocess.Popen(args)

    time.sleep(1)

    # sign the hashed message w/ the cac card's private key
    signedFile = os.path.join(self.workingDirTmp, 'out.signed')
    commandLine= 'pkcs15-crypt -s --sha-256 --pkcs1 -k 02 -i ' + hashedFile + ' -o ' +
```

```
signedFile
        args = shlex.split(commandLine)
        p = subprocess.Popen(args)
        p.communicate()

        # verify
        commandLine = 'openssl dgst -verify ' + self.certFromSmartcardPubkey + ' -sha256 -
signature ' + signedFile + ' ' + msgFile
        print commandLine
        args = shlex.split(commandLine)
        p = subprocess.Popen(args, stdout=subprocess.PIPE)
        out,err = p.communicate()
        print '*** DigSig VERIFICATION ***'
        print out
        if ('OK' not in out):
            return False

        return True

    def parseCertificate(self):
        f = open(self.certParsed, 'w')
        # run openssl on this cert to get the info
        commandLine = 'openssl x509 -in ' + self.certFromSmartcard + ' -text -noout'
        args = shlex.split(commandLine)
        subprocess.Popen(args, stdout=f)
        f.flush()
        os.fsync(f.fileno())
        f.close()

        # wait 1 second before proceeding, sometimes the file isn't showing up :(
        time.sleep(1)

        # read it line by line, and look for the 'Subject:', if so parse it
        with open(self.certParsed) as f:
            for line in f:
                # look for the commmon name
                if 'Subject:' in line:
                    # parse this line for the CN
                    lineWithoutWhitespace = line.strip()
                    lineSplit = lineWithoutWhitespace.split(',')
                    # loop through and find the CN= tag
                    for keyValPair in lineSplit:
                        kvPair = keyValPair.strip()
                        kvPairSplit = kvPair.split('=')
```

```
                k = kvPairSplit[0]
                v = kvPairSplit[1]
                if(k=='CN'):
                    nameAndEmail = v
                    nameAndEmailSplit = nameAndEmail.split('/')
                    name = nameAndEmailSplit[0]
                    self.cn = '.'.join(name.split())
                    self.email = kvPairSplit[2]

    print 'Common Name = ' + self.cn
    print 'Email = ' + self.email

    self.genTempConfigFileFromOrig(self.cn, self.email)

def genTempConfigFileFromOrig(self, commonName, email):
    f = open(self.opensslConfigFile, 'r')
    origConfigFileLines = f.readlines()
    newConfigFile = list(origConfigFileLines)
    origIdx = 0
    newIdx = 0
    for line in origConfigFileLines:
        if '[ req ]' in line:
            newIdx = newIdx + 1
            lineToAdd = 'prompt = no\n'
            newConfigFile.insert(newIdx, lineToAdd)

        if '[ req_distinguished_name ]' in line:
            newIdx = newIdx + 1
            # add the commonName_default
            lineToAdd = 'CN=' + self.cn+'\n'
            # add the line
            newConfigFile.insert(newIdx, lineToAdd)
            newIdx = newIdx + 1
            lineToAdd = 'emailAddress=' + self.email+'\n'
            newConfigFile.insert(newIdx, lineToAdd)

        origIdx = origIdx + 1
        newIdx = newIdx + 1

    self.tmpSSLConfigFile = self.opensslConfigFile+'.tmp'
    f = open(self.tmpSSLConfigFile, 'w')
    for item in newConfigFile:
        f.write("%s" % item)
    f.close()
```

```
    def genDerivedCredential(self):
        # generating the key
        self.keyName = os.path.join(self.pkeysDir,self.cn+'key.key')
        commandLine = 'openssl genrsa -out ' + self.keyName + ' 2048'
        args = shlex.split(commandLine)
        subprocess.Popen(args)

        time.sleep(1)

        # generate the csr
        self.csrFile = os.path.join(self.workingDirTmp, self.cn + 'dc.csr')
        commandLine = 'openssl req -new -key ' + self.keyName + ' -config ' +
self.tmpSSLConfigFile + ' -out ' + self.csrFile
        args = shlex.split(commandLine)
        p = subprocess.Popen(args)

        time.sleep(1)
        os.unlink(self.tmpSSLConfigFile)

        # sign the csr w/ the CA
        self.certFile = os.path.join(self.certsDir, self.cn+'dc.crt')

        ############ AL ADDED BATCH TO NOT PROMPT FOR CA SIGNING
        commandLine = 'openssl ca -batch -config ' + self.opensslConfigFile + ' -key doddci -name
CA_dccai -in ' + self.csrFile + ' -out ' + self.certFile
        args = shlex.split(commandLine)
        p = subprocess.Popen(args)
        p.communicate()

        # make the P12 file
        self.p12Filename = self.cn+'dc.p12'
        self.p12File = os.path.join(self.p12Dir, self.p12Filename)
        commandLine = 'openssl pkcs12 -export -inkey ' + self.keyName + ' -in ' + self.certFile + ' -
out ' + self.p12File
        args = shlex.split(commandLine)
        p = subprocess.Popen(args)
        p.communicate()

        # copy the p12 file to the apache directory where it will be delivered
        shutil.copy(self.p12File, GenDerivedCredential._DEST_P12_DIR)
        self.p12FileInWebDir = os.path.join(GenDerivedCredential._DEST_P12_DIR,
self.p12Filename)
```

```
    def deliverCredential(self):
        hyperlink = self.serverIpAddress + '/p12files/' + self.p12Filename

        qr = qrcode.QRCode(
            version=1,
            error_correction=qrcode.constants.ERROR_CORRECT_L,
            box_size=10,
            border=4,
        )
        qr.add_data(hyperlink)

        qr.make(fit=True)
        img = qr.make_image()

        qrCodeFile = '/home/al/pki_project/qr_png/p12_qr.png'
        f = open(qrCodeFile,'w')
        img.save(f)
        f.close()

        #subprocess.Popen([qrCodeFile], shell=True)

    def cleanup(self):
        time.sleep(5)
        # delete everything from the tmp directory
        flist = os.listdir(self.workingDirTmp)

        for f in flist:
            try:
                os.unlink(os.path.join(self.workingDirTmp,f))
            except Exception as e:
                print e

if __name__=='__main__':
    rootDir = '/home/al/pki_project/ssl'
    ####### AL CHANGED THE PEM FILE TO POINT TO USER CA INSTEAD OF DOD
    dodCA = '/home/al/pki_project/auth_ca_certs/simdodcacerts.pem'
    opensslConfigFile = '/home/al/pki_project/ssl/openssl.cnf'
    serverIpAddress = '192.168.1.12'
    gdc = GenDerivedCredential(opensslConfigFile, rootDir, dodCA, serverIpAddress)
```

F.2 Server Driven Output

```
al@al-gnome:~/pki_project/python$ python GenDerivedCredentialGemalto.py
Successfully got certificate from SmartCard!
Successfully generated certificate public key!
*** CAFile VERIFICATION ***
/home/al/pki_project/ssl/tmp/digsig.ctr: OK

Using reader with a card: SCM Microsystems Inc. SCR 331 [CCID Interface] (21120710216109)
00 00
Using reader with a card: SCM Microsystems Inc. SCR 331 [CCID Interface] (21120710216109)
00 00
Enter PIN [PIV Card Holder pin]:
openssl dgst -verify /home/al/pki_project/ssl/tmp/digsig.pem -sha256 -signature
/home/al/pki_project/ssl/tmp/out.signed /home/al/pki_project/ssl/tmp/message
*** DigSig VERIFICATION ***
Verified OK

Common Name = User.2.Dig.Sig
Email = user2@mil.mil
Generating RSA private key, 2048 bit long modulus
.............+++
.............+++
e is 65537 (0x10001)
Using configuration from /home/al/pki_project/ssl/openssl.cnf
Check that the request matches the signature
Signature ok
Certificate Details:
        Serial Number: 4 (0x4)
        Validity
            Not Before: Apr 13 18:34:50 2014 GMT
            Not After : Apr  2 18:34:50 2017 GMT
        Subject:
            countryName               = US
            organizationName          = U.S. Government
            organizationalUnitName    = DoD
            organizationalUnitName    = PKI
            commonName                = User.2.Dig.Sig
            emailAddress              = user2@mil.mil
        X509v3 extensions:
            X509v3 Basic Constraints:
                CA:FALSE
```

X509v3 Key Usage:
Digital Signature, Non Repudiation
X509v3 Subject Key Identifier:
72:EC:69:01:17:98:E2:73:2D:3D:7C:A5:1D:86:32:11:AC:C7:B6:E8
X509v3 Authority Key Identifier:
keyid:40:C1:8C:FE:D4:64:B9:76:8D:B2:7A:07:59:B1:FC:9A:F4:7F:AC:21
DirName:/C=US/O=U.S. Government/OU=DoD/OU=PKI/CN=DoD Root CA Sim
serial:02

Certificate is to be certified until Apr 2 18:34:50 2017 GMT (1085 days)

Write out database with 1 new entries
Data Base Updated
Enter Export Password:
Verifying - Enter Export Password:

F.3 Server Driven Derived Credential Certificate

al@al-gnome:~/pki_project/ssl/DoDRootCA/DoDDCCAI/DCCerts$ openssl x509 -in User.2.Dig.Sigdc.crt -text
Certificate:
Data:
Version: 3 (0x2)
Serial Number: 4 (0x4)
Signature Algorithm: sha1WithRSAEncryption
Issuer: C=US, O=U.S. Government, OU=DoD, OU=PKI, CN=DoD DC CA I Sim
Validity
Not Before: Apr 13 18:34:50 2014 GMT
Not After : Apr 2 18:34:50 2017 GMT
Subject: C=US, O=U.S. Government, OU=DoD, OU=PKI, CN=User.2.Dig.Sig/emailAddress=user2@mil.mil
Subject Public Key Info:
Public Key Algorithm: rsaEncryption
Public-Key: (2048 bit)
Modulus:
00:b5:53:f8:95:76:a7:8c:15:27:7a:8a:b5:d5:58:
39:88:0a:a7:41:89:ee:d6:0a:28:87:e6:11:68:ce:
f7:23:2c:06:2b:c7:da:79:1a:ad:11:06:9f:ba:f3:
f7:cf:fe:16:f6:28:7a:8e:d3:3f:59:e9:fd:43:24:
bb:aa:0a:26:95:12:f2:5b:17:1b:ff:13:94:24:aa:
a8:cd:50:82:d8:39:64:70:4c:d1:73:5b:8e:1a:eb:
eb:92:5f:d1:6e:57:8d:9e:9b:3b:17:28:d7:e7:cf:

```
            b4:f7:2d:31:75:4d:dd:84:63:aa:ae:2e:c6:1d:07:
            dd:ea:94:88:cd:69:75:f0:6b:31:cb:40:af:25:19:
            fb:a3:16:ca:a3:4a:2d:5b:e0:e7:1f:53:dc:d7:b7:
            e1:7e:da:5b:f3:15:e8:50:65:3f:bb:36:8d:e1:25:
            0d:fe:2b:29:64:dc:35:48:2a:b7:d7:da:d8:97:a0:
            c1:f0:06:15:7f:cd:3e:a7:a0:5c:d7:95:ea:95:ca:
            62:8c:d8:a3:c6:f3:01:3a:3c:5a:ad:81:06:fc:bb:
            5d:03:f3:af:97:26:70:ba:6e:6e:f5:4b:81:77:f4:
            ec:92:14:51:6e:c5:97:01:45:c2:f8:d6:d8:c8:dd:
            55:e8:8b:64:8a:1d:81:f4:5b:9f:5d:4d:06:82:3e:
            97:2b
        Exponent: 65537 (0x10001)
    X509v3 extensions:
        X509v3 Basic Constraints:
            CA:FALSE
        X509v3 Key Usage:
            Digital Signature, Non Repudiation
        X509v3 Subject Key Identifier:
            72:EC:69:01:17:98:E2:73:2D:3D:7C:A5:1D:86:32:11:AC:C7:B6:E8
        X509v3 Authority Key Identifier:
            keyid:40:C1:8C:FE:D4:64:B9:76:8D:B2:7A:07:59:B1:FC:9A:F4:7F:AC:21
            DirName:/C=US/O=U.S. Government/OU=DoD/OU=PKI/CN=DoD Root CA Sim
            serial:02

Signature Algorithm: sha1WithRSAEncryption
    15:18:07:5c:da:ce:91:01:73:f5:b6:7e:c4:78:39:7d:c7:44:
    fd:63:d9:b0:cb:97:f4:4c:f1:b4:83:49:12:a0:72:90:8f:30:
    fd:fe:4d:a6:84:de:2e:70:30:5f:b1:f3:1e:38:09:54:0e:03:
    9d:db:e6:49:38:f8:36:6f:0a:23:62:9c:6e:9f:29:b0:b9:82:
    8b:52:50:d9:19:29:16:64:a8:cf:ec:d7:93:f1:fc:7d:d0:29:
    df:28:7a:29:5d:b5:2f:45:d2:7c:e0:49:d4:e3:7e:8e:0d:49:
    07:84:55:a5:fe:56:f6:a9:c3:fc:f1:b5:4c:9f:eb:7a:a9:2e:
    93:73:22:9c:12:86:03:23:3a:25:12:07:82:13:cd:92:78:1e:
    ca:0b:08:37:79:2e:6e:f5:1f:6f:6d:e4:9b:06:1b:4d:eb:0f:
    65:c6:11:50:b8:22:e2:79:98:8c:cd:f5:c7:59:d9:b1:33:1e:
    6c:73:77:02:b4:94:be:a8:0f:dc:a1:60:d3:5d:e4:c7:33:18:
    19:f8:96:34:1c:c0:56:e5:f5:1b:98:57:a5:6e:af:fe:00:ae:
    bd:48:02:c5:a4:96:e6:77:e7:06:d2:cb:1e:b5:e8:35:c6:bb:
    b2:51:ac:39:68:5e:2b:95:9a:c0:b0:6d:9d:00:e6:87:90:8a:
    8c:47:04:ce
```