# An Example of a Derived Credentials Architecture

Francisco Corella, PhD
fcorella@pomcor.com

Karen Lewison, MD
kplewison@pomcor.com

Original: March 31, 2014; updated:[*] April 27, 2014

**Abstract**

NIST has released drafts of two documents containing guidelines for issuing credentials to federal employees or contractors upon presentation of a PIV/CAC card and storing them on mobile devices, such credentials being known as derived credentials. The guidelines permit the storage of credentials in a software token, i.e. a software cryptographic module, protected by an activation passcode with 20 bits of entropy. While mere encryption under a key derived from such a passcode would not provide sufficient security, we propose a method of achieving strong security with such a passcode, by wrapping the credentials using a high-entropy key-wrapping key (KWK), storing the KWK in a secure back-end, and retrieving the KWK by authenticating to the back-end with a device credential regenerated from the passcode and a protocredential, in such a way that an adversary who captures the mobile device while the software token is inactive is not able to mount an offline guessing attack against the passcode. We suggest that the use of this method could serve as a compensating control to justify the storage of derived credentials in cryptographic modules that are not removable from the device and do not provide sufficient tamper resistance. We describe the use of the compensating control to increase the security provided by a software token, a TEE token (i.e. a cryptographic module embedded in a Trusted Execution Environment), and a traditional hardware token. We describe an example of a derived credentials architecture where the KWK is stored in a device record within a Mobile Device Management (MDM) database.

## Contents

[*]Updated on 4/24 as explained in footnote 2, and on 4/27 to correct Figure 4.

## List of Figures

# 1 Introduction

To comply with presidential directive HSPD-12 [1], US federal employees and contractors use Personal Identity Verification (PIV) cards, specified by NIST in FIPS 201-2 [2], Special Publication 800-73-4 (Draft) [3] and other special publications, for physical access to federal facilities and logical access to federal information systems. PIV cards used at the Department of Defense are called Common Access Cards (CAC). When used for logical access, a PIV/CAC card is typically plugged into a card reader connected to a desktop or laptop personal computer (PC). The PC authenticates the user to an information system of a federal agency by presenting a PIV Authentication certificate contained in the card to the information system together with a signature on a challenge executed within the card using a PIV Authentication private key associated with the certificate. Such authentication may take place, for example, during the establishment of a TLS connection from the PC to the information system, in which case the PIV Authentication certificate plays the role of a TLS client certificate.

Usually, a PIV/CAC card also contains credentials used for exchanging signed and encrypted S/MIME mail messages, including a "Digital Signature" private key and associated certificate, a current "Key Management" private key and associated certificate, and a collection of up to 20 retired key management private keys and certificates. Although it is not obvious from the name, the key management private key is used to decrypt mail messages. More specifically, it is used to unwrap the symmetric keys used to encrypt messages addressed to the user of the card, while the public key contained in the key management certificate is used by senders to wrap those keys. The retired key management private keys are used to decrypt older messages that the user has saved in their encrypted form.

In December 2011, NIST Special Publication 800-63-1 [4] introduced the concept of a *derived credential*, defined 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.*

with the purpose of allowing users to carry such credentials in mobile devices rather than PIV/CAC cards [5].

Storing derived credentials in mobile devices is complicated by the fact that many mobile devices do not provide tamper resistant storage, while HSPD-12 specifically requires tamper protection for federal credentials, and FIPS 201-2 [2] requires a PIV card to provide physical security level 3 as specified in FIPS 140-2 [6].

To address this difficulty, at the NIST Cryptographic Key Management workshop of September 2012 [7] we proposed to use as authentication credential an uncertified RSA key pair that, instead of being stored in the mobile device, would be regenerated before use from parameters including the prime factors of the RSA modulus together with user secrets consisting of a PIN and/or a biometric key, the biometric key being itself derived from a biometric sample and auxiliary data [8, 9, 10]. An adversary who captured the mobile device would not find the key pair in the device, and furthermore could not mount an offline guessing attack against the user secrets, because each guess could only be tested by attempting online authentication against a back-end that would limit the number of attempts.

We also proposed an architecture where a prover black-box (PBB) in the mobile device authenticates to a verifier black-box (VBB) in an agency back-end, obtaining a code[1] that front-ends of native and web-based applications use for authentication to their back-ends.

All this would have greatly simplified the development of new mobile applications, but would have meant a sharp departure from the certificate-based authentication method used by existing applications in federal agencies.

[1]The code plays the role of a "bearer token," and we referred to it as an "authentication token." In the present paper, however, we use the word "token" to refer to a cryptographic module, for consistency with the NIST documents, and we avoid other uses of the word as much as possible.

In subsequent work [11] we extended the credential regeneration method to other cryptosystems besides RSA, viz. DSA and ECDSA; we introduced the concept of a *protocredential* to refer to parameters stored in the device and combined with user-supplied secrets to regenerate a credential; and we devised a method for enterprise single sign-on (SSO) based on the sharing of a login session created by the VBB, among both native and web-based applications. We also proposed a method for storing a traditional credential consisting of a certificate and associated private key in a mobile device, the credential being encrypted under a data encryption key stored in a secure back-end and retrieved by authenticating to the back-end with a credential regenerated from a protocredential and user-supplied secrets [11, §5.5].

Recently NIST has published drafts of two documents providing guidelines for the implementation of derived credentials, NIST Interagency Report 7981 (NISTIR 7982) [12] and Special Publication 800-157 (SP 800-157) [13], and has asked for comments on the drafts. We have sent comments to NIST, and published them at [14]. This paper expands on some of the points made in the comments.

The NIST documents discuss the storage in a mobile device of an authentication private key and certificate, a digital signature private key and certificate, a current key management key and certificate and a collection of retired management keys and certificates. The authentication and digital signature keys and certificates are of the same type as those in a PIV card, but distinct from those in the card. On the other hand, the current and retired key management keys must be identical to those in the card, since they are to be used for decrypting the same mail messages.

Although SP 800-157 discusses mail-related credentials in informative appendix A and normative appendix B, it insists that only the authentication credential is to be considered a "Derived PIV Credential":

> *The Derived PIV Credential is a PIV Derived Authentication certificate, which is an X.509 public key certificate that has been issued in accordance with the requirements of this document and the X.509 Certificate Policy for the U.S. Federal PKI Common Policy Framework [COMMON]. While the PIV Card may be used as the basis for issuing other types of derived credentials, the issuance of these other credentials is outside the scope of this document. Only derived credentials issued in accordance with this document are considered to be Derived PIV credentials.*

As a practical matter, a term is needed to refer jointly to authentication and mail-related credentials stored in a mobile device. In this paper we shall use the term *derived credentials* for that purpose.

The NIST documents direct agencies to store derived credentials in hardware or software tokens, using the word "token" to refer to a cryptographic module. Although this is not completely clear, the documents seem to suggest that derived credentials stored in a software token are to be protected by encrypting them under a key derived from a randomly generated 6-digit PIN or from a password with equivalent entropy, i.e. with 20 bits of entropy. In our comments we point out that such low entropy provides no protection against an offline attack using a botnet, which the attacker could easily rent for a low fee; we propose instead protecting the derived credentials in a software token by encrypting their private keys under a high-entropy key stored in a secure back-end and retrieved by authenticating to the back-end with a credential regenerated from a protocredential and the PIN or password (i.e. using the technique of [11, §5.5] in the special case where the user-supplied secrets consist of a single PIN or password).

Henceforth we shall use the term *passcode* to refer a PIN, password or passphrase.

SP 800-157 observes that the OMB memorandum M-07-16 [15] requires storing derived credentials in a "device separate from the computer gaining access," but anticipates that OMB will issue alternative guidance allowing compensating controls to make up for the use of tokens integrated into the device, including software tokens. In our comments we point out that a compensating control is also needed for the lack of the protection against tampering required by HSPD-12, and we suggest framing the proposed technique for protecting derived credentials stored in a software token as a compensating control for both the lack of separation and the lack of tamper resistance of software tokens. Storing the proposed high-entropy key used to encrypt the derived credentials away from the mobile device seems a suitable compensating control for not storing the derived credentials themselves away from the device; and encrypting the derived credentials under a high-entropy key can be viewed as a form of virtual tamper resistance, compensating for the lack of physical tamper resistance.

The same compensating control is applicable to hardware tokens that provide no tamper resistance, such as a token implemented in the Trusted Execution Environment (TEE) provided by an ARM Cortex-A processor [16]; and it is useful even for hardware tokens that provide tamper resistance because, as discussed below, tamper resistance is never absolute. For all tokens, the compensating control provides the following security baseline:

> *An adversary who captures a mobile device containing a token (i.e. a cryptographic module) that is not active (i.e. that must be activated by a passcode before it can be used), and who does not know the passcode, has a negligible probability of extracting the private keys of the derived credentials protected by the compensating control, even if the passcode has as little as 20 bits of entropy.*

where the meaning of the term *negligible* is made precise below in Section 2.1. Security features of some kinds of tokens provide further security beyond the baseline, as discussed below.

As mentioned above, in previous papers [10, 11] we have discussed a technique for using a biometric key to regenerate a credential from a protocredential, instead of, or in addition to, a passcode. The technique could be used for token activation. However, while FIPS 201-2 [2] mentions the possibility of using a biometric sample to activate a PIV card, SP 800-157 only considers activation with a passcode. To keep this paper narrowly focused, we shall not further discuss the use of biometrics, and we shall only consider token activation with a passcode, and credential regeneration from a protocredential and a passcode.

The rest of the paper is organized as follows. Section 2 describes the compensating control in more detail, and how it can be used to increase the security provided by a *mobile cryptographic module*, which can be a software token, a hardware token implemented in a TEE, or a traditional hardware token. Section 3 provides a particular example of a system architecture that takes advantage of a mobile device management (MDM) infrastructure to implement the compensating control. Subsection 3.1 describes the overall system. Subsection 3.2 describes how the mobile cryptographic module is used via one or more APIs. Subsection 3.3 describes procedures for activating and deactivating the mobile cryptographic module. Subsection 3.4 describes procedures for registering the device and provisioning the derived credentials. Subsection 3.6 explains how the mobile cryptographic module can be used to implement effective data protection and how token activation can be integrated with effective device locking when the operating system itself implements the module. Subsection 3.5 suggests an optional enhancement of the architecture that can be used to increase the availability of derived credentials and allow offline decryption of saved mail messages. Subsection 3.7 envisions the specification of open API standards to facilitate the development of an ecosystem of interoperable components of the architecture. Finally, Section 4 recapitulates and Section 5 makes an intellectual property disclosure.

The paper also includes two appendices. Appendix A explains the process of regenerating a credential from a protocredential and a passcode, and Appendix B explains the process of initially generating a protocredential and the corresponding credential from the passcode.

## 2 A Compensating Control

Recall that the word token is used in the NIST documents to refer to a cryptographic module. In this section we describe the method of protecting derived credentials that can serve as a compensating control for software tokens and hardware tokens that do not provide tamper resistance, and can generally improve security for any kind of token. We first discuss the use of the compensating control to protect credentials in a software token, then in a token embedded in a TEE, and finally in a traditional hardware token.

### 2.1 Using the Control with a Software Token (i.e. with a Cryptographic Module Implemented Entirely in Software)

Figure 1 illustrates the use of the control in conjunction with a software token. The private keys of the derived credentials, including the derived PIV authentication key, as well as the derived digital signature (DS) and current and retired key management (KM) private keys if the user has an agency mail account, are stored encrypted under a high-entropy key-wrapping key (KWK, which could also be called a key-encryption key, KEK), which is stored in a device record within the agency back-end. Section 3 provides an example of a system architecture in which the device record is part of a database of device records maintained by a mobile device management (MDM) back-end; but an MDM back-end is not necessarily involved.

The KWK is retrieved from the back-end when the user activates the software token by entering a passcode. It is then used to unwrap the private keys of the derived credentials, and deleted immediately after use as illustrated by dark grey shading in the figure. The private keys remain present in the clear in the software token until the token is deactivated, as illustrated by light grey shading in the figure. Deactivation may occur upon explicit request by the user, when the user powers off or locks the device, after the token has been inactive for a configured period of time, or after a configured period of time has elapsed since activation.

To retrieve the KWK, the device authenticates to the agency back-end using a *device credential*, which consists of a device record handle that uniquely identifies the device record within the agency back-end, and a key pair pertaining to a digital signature cryptosystem such as a DSA, ECDSA or RSA. To that purpose the device establishes a TLS connection to the back-end. During the handshake, the back-end authenticates to the device using

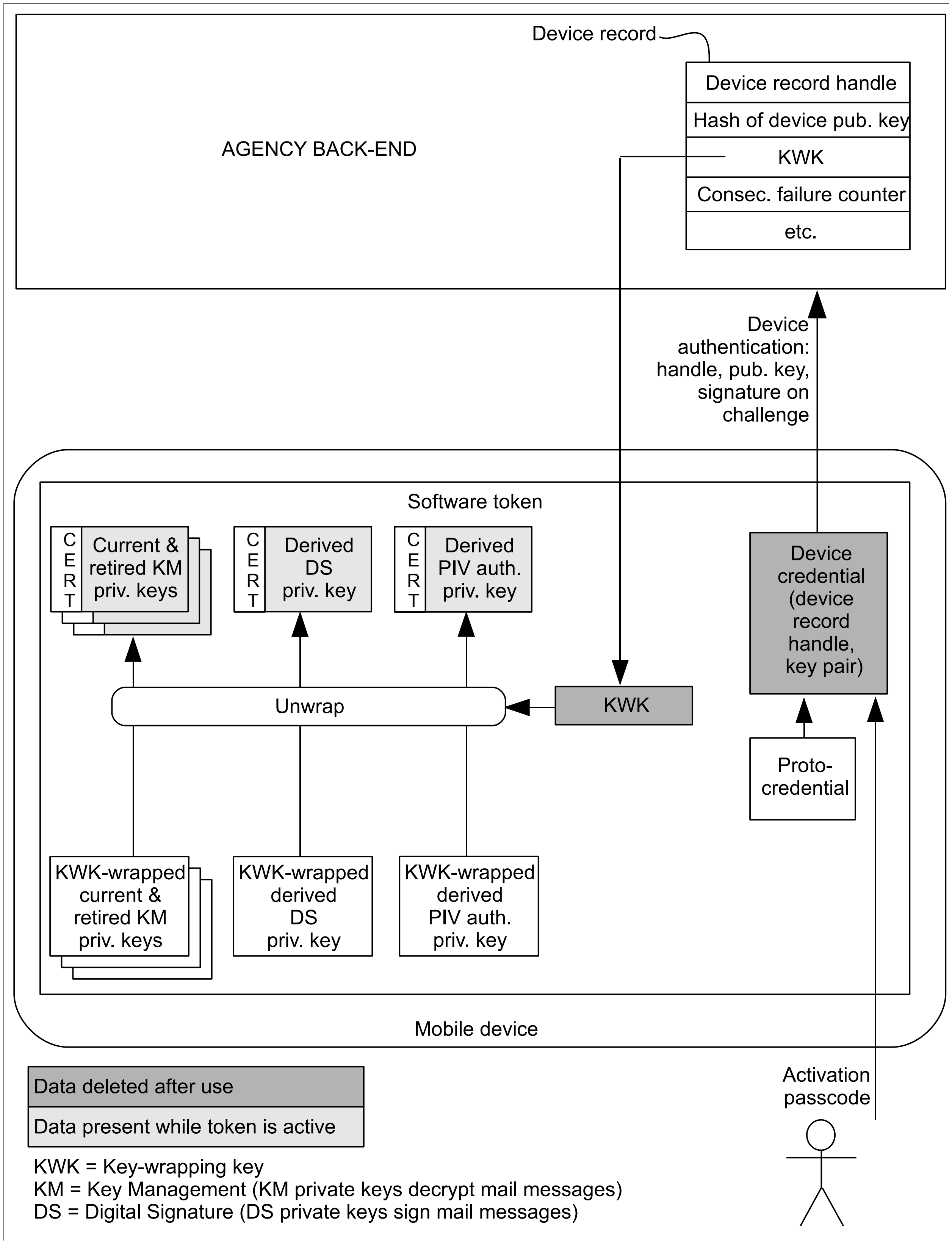

Device record
Device record handle
Hash of device pub. key
KWK
Consec. failure counter
etc.
AGENCY BACK-END
Device authentication: handle, pub. key, signature on challenge
Software token
CERT
Current & retired KM priv. keys
CERT
Derived DS priv. key
CERT
Derived PIV auth. priv. key
Device credential (device record handle, key pair)
Unwrap
KWK
Proto-credential
KWK-wrapped current & retired KM priv. keys
KWK-wrapped derived DS priv. key
KWK-wrapped derived PIV auth. priv. key
Mobile device
Data deleted after use
Data present while token is active
Activation passcode
KWK = Key-wrapping key
KM = Key Management (KM private keys decrypt mail messages)
DS = Digital Signature (DS private keys sign mail messages)

Figure 1: Software token with compensating control

a TLS server certificate. After the handshake, the device sends to the back-end the device record handle, the public key component of the device key pair, and a signature on a challenge derived by hashing together the TLS master secret and the back-end's TLS server certificate.[2] The back-end verifies the signature and compares a hash of the device public key to a hash stored in a field of the record during the device registration procedure. If authentication fails, the back-end increments a counter of consecutive authentication failures maintained in another field of the record, and disables the record if the counter reaches a limit set by agency policy. Any number in the range 3-10 would be a reasonable limit.

The device credential is regenerated from the activation passcode supplied by the user and a protocredential, as described in Appendix A. It is deleted after being used to authenticate the device to the back-end and retrieve the KWK, as illustrated by dark grey shading in the figure.

Recall the security baseline provided by the compensating control:

> *An adversary who captures a mobile device containing a token that is not active, and who does not know the passcode, has a negligible probability of extracting the private keys of the derived credentials protected by the compensating control, even if the passcode has as little as 20 bits of entropy.*

The following informal argument explains how the security baseline is achieved:

- The private keys of the derived credentials are not present in the clear in the device when the device is captured, because the device is assumed not to be active.
- The wrapped private keys are present in the device, and could be unwrapped with the KWK, but the KWK is not present in the clear in the device when the device is captured, because the plaintext KWK is always deleted after use.
- The KWK could be retrieved from the agency back-end by authenticating with the device credential, but the device credential is not present in the device, because it is always deleted after use.
- The protocredential is present in the device and could be used to regenerate the device credential, but that requires the activation passcode.
- The adversary may try to guess the passcode, but, as discussed below, has no information that would make it possible to test passcode guesses offline. The adversary can only test each passcode guess by regenerating the device credential and attempting to authenticate to the back-end, which can limit the number of attempts to not more than 10. The probability of guessing a passcode with 20 bits of entropy in no more than 10 tries may be deemed negligible based on the fact that Table 6 of SP800-63-2 [18] allows 100 tries every 30 days for that amount of entropy.

[2] In the original version of the paper, the challenge was derived from the master secret only. However, using an Unknown Key-Share (UKS) vulnerability of TLS recently reported in [17], an attacker might be able to trick the device into establishing a TLS connection to the attacker, while simultaneously establishing a TLS connection from the attacker to the back-end having the same master secret. The attacker might then be able to obtain a signature by the device on a challenge derived from the common master secret, and use it to authenticate to the back-end. We discussed another, less compelling reason for including the back-end certificate in the challenge material in [11, §2.1].

An adversary who captures a device that is not active does not have information for testing a passcode guess offline for the following reasons:

- No information related to the device credential is stored in the device other than the protocredential, and every passcode yields a well-formed credential when combined with the protocredential.[3]
- After regenerating a credential from the passcode guess, the adversary cannot match the public key of the credential against a public key contained in a certificate because no certificate is issued for the public key in the device credential.
- When the device authenticates to the back-end while in possession of the legitimate user, the public key and the signature on the challenge sent to the back-end cannot be sniffed by the adversary for future use in an offline test after device capture, because they are sent encrypted after the TLS connection has been established.
- The hash of the public key stored in the device record is treated by the back-end as a secret.

The requirement to treat as secrets the hashes of public keys stored in a database of device records is reminiscent of the requirement to treat as secrets the salted hashes of passwords stored in a traditional database of user records. However the consequences of a database security breach are very different in the two cases. An adversary who breaches the security of the database of salted password hashes can mount an offline dictionary attack against each password, and many passwords will fall to the attack. On the other hand, an adversary who breaches the security of the database of device records also needs to breach the security of an individual device in order to acquire credentials pertaining to the device.

## 2.2 Using the Control with a TEE Token (i.e. with a Cryptographic Module Embedded in a Trusted Execution Environment)

In our comments [14] on the NIST documents we argue that a particular kind of embedded hardware token, which may be called a *TEE token*, deserves special mention. A TEE token is a cryptographic module within a Trusted Execution Environment (TEE). A TEE, a.k.a. a TrustZone, is a computing environment provided by a secure OS running on the same processor as a normal OS. One or more trusted applications (TAs) run under the secure OS. A hardware bus architecture ensures that a portion of the flash storage can only be accessed by the secure OS. Both OSes can access the touchscreen, but a security indicator lets the user know when the screen is controlled by the secure OS and the user interface can be trusted. TEE specifications are being developed by GlobalPlatform [19]. TEEs are provided by ARM Cortex-A processors, where a TEE is also referred to as a TrustZone [16]. Development of trusted applications is supported at least by Trustonic [20] and Sierraware

[3]This is strictly true when the regenerated credential pertains to the DSA or ECDSA cryptosystems. Regeneration of an RSA credential may fail, but the probability of failure is very small, so regeneration success cannot be used to test passcode guesses in an offline attack.

[21]. The cryptographic module functionality of a TEE token is to be implemented by a trusted application, which we shall call the *token TA*, running under the secure OS.

A TEE token has important advantages as well as disadvantages for the storage of derived credentials. The GlobalPlatform TEE specifications include a Trusted User Interface API specification [22] that can be used to protect the activation PIN from being phished by malware. On the other hand, since the secure OS runs on the same processor as the normal OS and uses a portion of the same flash storage, a TEE token does not typically provide any tamper resistance.[4] The compensating control remedies this drawback of a TEE token.

Figure 2 illustrates how the compensating control is used in conjunction with a TEE token. While the TEE token is active, the plaintext private keys of the derived credentials are stored within the token and designated as not being extractable by software.[5] They thus have some protection against malware running on the device while the token is active: malware can use the private keys, by instructing the TEE token to perform cryptographic operations with them, but cannot extract them and send them to an adversary for use in a machine owned by the adversary.

The wrapped private keys, on the other hand, are stored outside the TEE token. This is because a typical cryptographic module API expects the wrapped argument to an unwrapping operation to be in working memory, which would require the wrapped private keys to be extracted before unwrapping if they were kept in the TEE token.

While the KWK used with the software token in Figure 1 is stored and retrieved in the clear, the KWK used with the TEE token is stored and retrieved wrapped under a KWK-wrapping key, which is permanently stored within the TEE. Otherwise malware running on the mobile device as the device is activated could capture the KWK and use it to unwrap the private keys stored outside the token.[6] A TEE token protects the activation passcode against being phished by malware running on the mobile device while the device is being used by the legitimate user. When the user activates the token, the passcode is prompted for by the token TA using the Trusted User Interface API specification [22], while the mobile device displays a security indicator[7] to let the user know that the touchscreen is controlled by the secure OS, that the passcode is being requested by a trusted application and that the passcode will be imported into the TEE token via a trusted path protected from malware.

Once it has been imported, the passcode never leaves the TEE token. The protocredential

[4]Tamper resistance is of course not *incompatible* with a TEE architecture. A GlobalPlatform specification [23] proposes to compensate for the lack of tamper resistance by using a secure element in addition to a TEE, the secure element providing tamper resistance while the TEE provides a trusted user interface. (However this negates the TEE cost savings achieved by using the same processor for the secure OS and the normal OS, since the secure element has its own processor; and it is a very complex solution, because the TEE and the secure element must communicate via the normal OS using an encrypted channel.) And the M-Shield technology of Texas Instruments [24] claims to be software-compatible with the ARM Trustzone technology while providing "tampering detection [that] triggers effective protection actions."

[5]The TEE token could be accessible through a variety of APIs, such as a future extension of the PKCS#11 API, or a future *TEE Functional API* envisioned in the GlobalPlatform TEE white paper [19]. If accessed through an extension of PKCS#11, the private keys could be made non-extractable by setting the value of their `CKA_EXTRACTABLE` attribute to `CK_FALSE`.

[6]The database of device records may contain records of devices with software tokens where the KWK is unwrapped and records of devices with TEE or other hardware tokens where the KWK is wrapped. Whether the KWK is wrapped or not is only of concern to the device, not to the back-end.

[7]The security indicator could be an LED controlled by the secure OS, or an image and/or phrase chosen by the user, stored in the TEE, and treated as a shared secret between the TEE and the user.

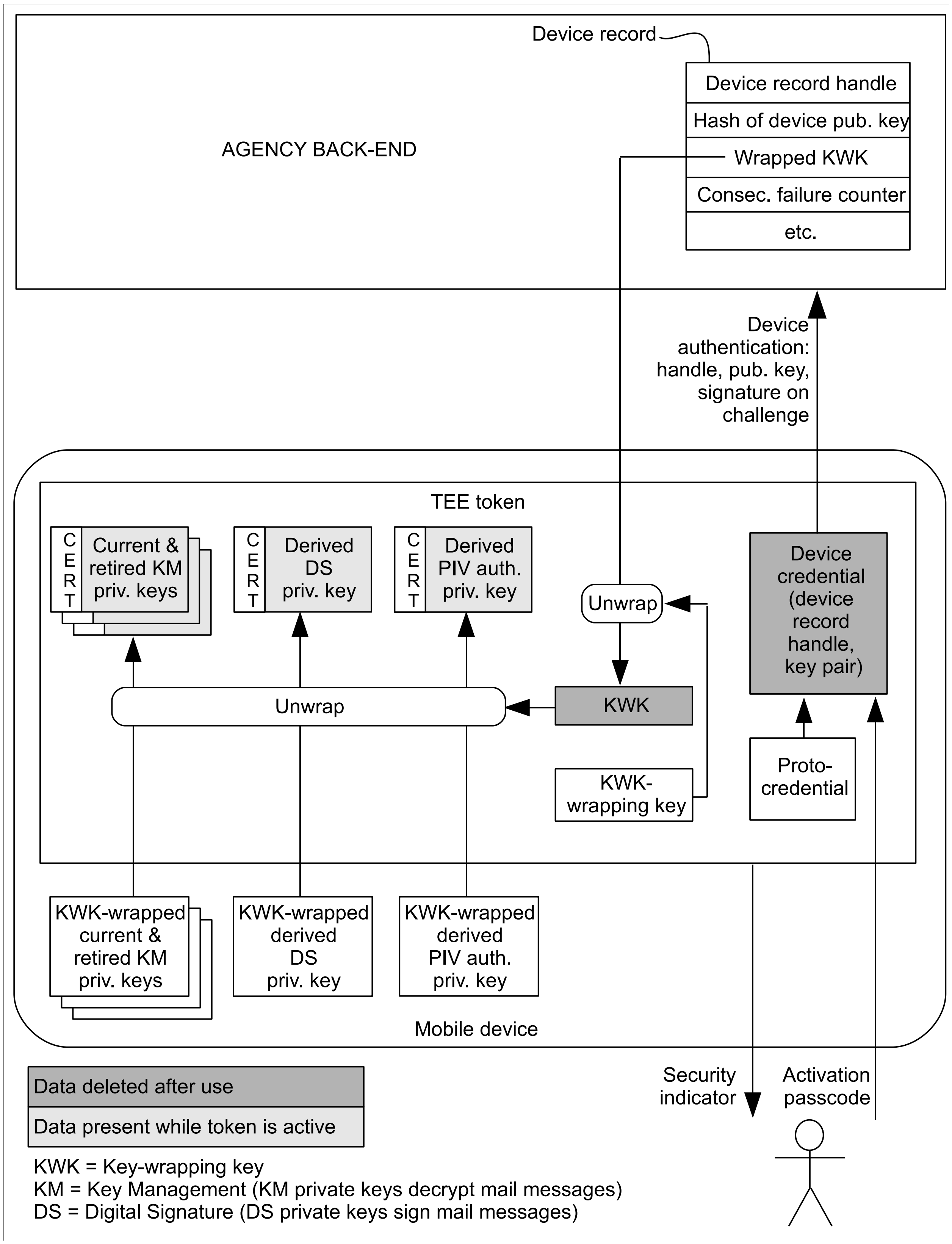

Device record
AGENCY BACK-END
Device record handle
Hash of device pub. key
Wrapped KWK
Consec. failure counter
etc.
Device authentication: handle, pub. key, signature on challenge
TEE token
CERT
Current & retired KM priv. keys
CERT
Derived DS priv. key
CERT
Derived PIV auth. priv. key
Unwrap
Device credential (device record handle, key pair)
Unwrap
KWK
KWK-wrapping key
Proto-credential
KWK-wrapped current & retired KM priv. keys
KWK-wrapped derived DS priv. key
KWK-wrapped derived PIV auth. priv. key
Mobile device
Security indicator
Activation passcode
Data deleted after use
Data present while token is active
KWK = Key-wrapping key
KM = Key Management (KM private keys decrypt mail messages)
DS = Digital Signature (DS private keys sign mail messages)

Figure 2: TEE token with compensating control

is stored within the token, and the device credential is regenerated within the token from the passcode and the protocredential. This requires, of course, that the token TA be programmed to support credential regeneration functionality and the token API provide access to the functionality.

As in the case of a software token, the compensating control used with a TEE token prevents an adversary who captures a mobile device while the token is not active from extracting the private keys, thus compensating for the lack of tamper resistance; but furthermore, the trusted interface feature of the TEE token protects the passcode from being phished by malware running on the mobile device while the legitimate user is using the device. This prevents the following two-phase attack. In phase I, the adversary uses malware while the legitimate user is using the device to phish the passcode. In phase II, the adversary captures the device and takes advantage of the lack of tamper resistance to extract the protocredential, the wrapped private keys and the KWK-wrapping key. Then the adversary computes the device credential from the protocredential and the passcode, uses the device credential to retrieve the wrapped KWK, uses the KWK-wrapping key to unwrap the KWK, and uses the KWK to unwrap the private keys of the derived credentials. (Another, minor advantage of a TEE token is that malware running on the device cannot activate the token, and therefore can only use the private keys if and when the user of the device activates the token.)

## 2.3 Using the Control with a Traditional Hardware Token (i.e. a Hardware Cryptographic Module Not Embedded in a TEE)

Traditional hardware tokens considered in the NIST documents include MicroSD cards, devices that plug into a USB port, UICC cards, and chips embedded into mobile devices.[8] Such hardware tokens do not provide a trusted interface feature that can be used to protect the passcode against phishing by malware. On the other hand, they may provide tamper resistance features. Traditional hardware tokens may use chips similar or identical to chips used for Digital Rights Management (DRM), which must be tamper proof against the intended user of the device and often incorporate sophisticated tamper resistance features.

A detailed account of the many tamper countermeasures included in a family of Infineon chips was presented at Black Hat DC 2010 [25]. The presentation shows that manufacturers of security hardware go to great lengths to prevent the hacking of tamper resistant chips. But it also illustrates the principle that tamper resistance is never absolute. Even though some of the chips in the family had passed Common Criteria certification [26], the countermeasures were defeated by a Class I attacker, i.e. a single attacker with no insider knowledge [27]. The attack required equipment that would be very expensive to buy, but is affordable if rented by the hour. It took the attacker months to discover the countermeasures and develop techniques to circumvent them, but it might take only hours to extract secrets from one of the chips in the family after developing the techniques.

Since tamper resistance is not absolute, the compensating control provides a useful complement to whatever tamper resistance is provided by a traditional hardware token. Figure 3 illustrates how the control can be used to improve the security of the token. The security baseline achieved by the control means that even an adversary who is able to circumvent the

[8]The NIST documents also mention "hardware security modules [. . . ] built into the SoC at the heart of the device." This may be a reference to what we call a TEE token, except that a TEE token is not a separate hardware module, since it uses the main processor of the device.

countermeasures against tampering will not be able to extract the private keys of the derived credentials from a captured device, as long as the device is not active. On the other hand, the tamper resistance provided by the device protects the private keys against adversaries who capture the device while active but are not able to circumvent the countermeasures.

As in the case of a TEE token, the plaintext private keys of the derived credentials are marked as not extractable, hence malware running on the device while the token is active can use but cannot extract the keys. However, there is no trusted interface, so malware may be able to phish the passcode, after which it will be able to activate the token at will in order to use the keys.

We assume that a traditional hardware token is not able to regenerate a credential from the passcode and a protocredential. Regeneration must therefore take place outside the token. The protocredential is kept outside the token to avoid having to extract it from the token before use.[9]

## 3 Example: An MDM-Based Derived Credentials Architecture

Figure 4 illustrates a derived credentials architecture that leverages an existing MDM infrastructure. An MDM back-end must implement a database of managed devices, and device records in the MDM database can be used to store the KWKs used by the compensating control. The KWK is stored unwrapped in a device record if the corresponding device uses a software token, wrapped if the device uses a TEE token or a traditional hardware token.

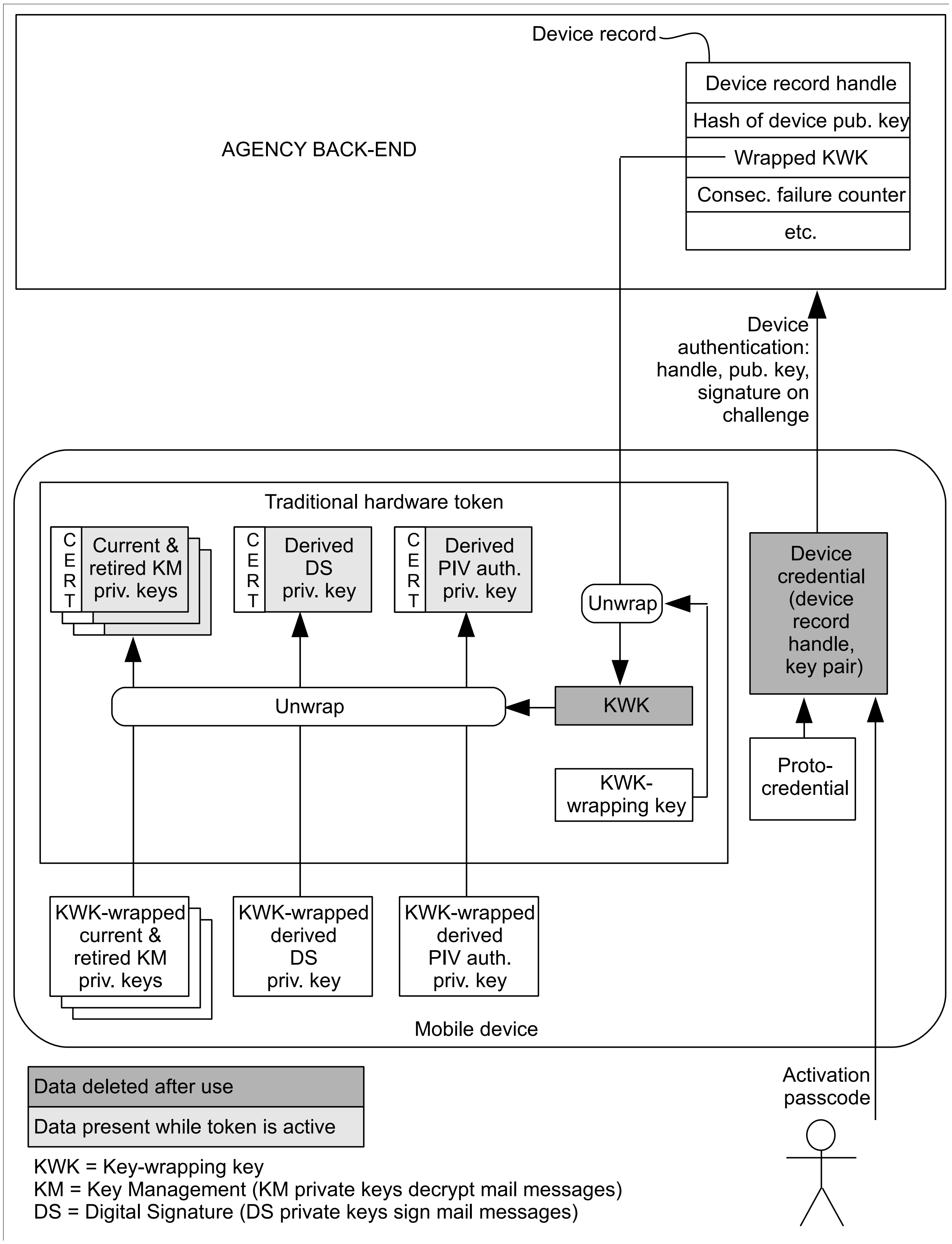
However this is only an example. The compensating control can be implemented in environments where there is no MDM infrastructure, or independently of an existing MDM infrastructure. All that the compensating control requires architecturally is a device record in a secure back-end where the KWK (wrapped or unwrapped) can be stored, and client code to retrieve and use the KWK.

### 3.1 System Architecture

Derived credentials are stored within a mobile device in a *mobile cryptographic module*, which may be implemented as a software token, a TEE token, or a hardware token; a software token is shown in the figure as an example. The derived credentials include a derived PIV authentication private key and its associated certificate, as well as a digital signature (DS) private key and current and retired key management (KM) private keys and their certificates if the user has an agency mail account. The authentication and digital signature credentials are provisioned by a CA while the key management credentials are provisioned by an escrow server as described below in Section 3.4.2. The CA and the escrow server may be the same entity, which interacts with the agency's identity management system (IDMS), as illustrated in the figure.

The user can process electronic mail using any mail client that supports one of the APIs exposed by the mobile cryptographic module. The client can be adapted to use the mobile

[9]An argument could be made in favor of storing the protocredential inside the token to benefit from tamper protection provided by the token. But the protocredential would have to be marked extractable, making it easy for an attacker to circumvent the tamper protection.



Device record
Device record handle
Hash of device pub. key
Wrapped KWK
Consec. failure counter
etc.
AGENCY BACK-END
Device authentication: handle, pub. key, signature on challenge
Traditional hardware token
CERT
Current & retired KM priv. keys
CERT
Derived DS priv. key
CERT
Derived PIV auth. priv. key
Unwrap
Unwrap
KWK
KWK-wrapping key
Device credential (device record handle, key pair)
Proto-credential
KWK-wrapped current & retired KM priv. keys
KWK-wrapped derived DS priv. key
KWK-wrapped derived PIV auth. priv. key
Mobile device
Activation passcode
Data deleted after use
Data present while token is active
KWK = Key-wrapping key
KM = Key Management (KM private keys decrypt mail messages)
DS = Digital Signature (DS private keys sign mail messages)

Figure 3: Traditional hardware token with compensating control

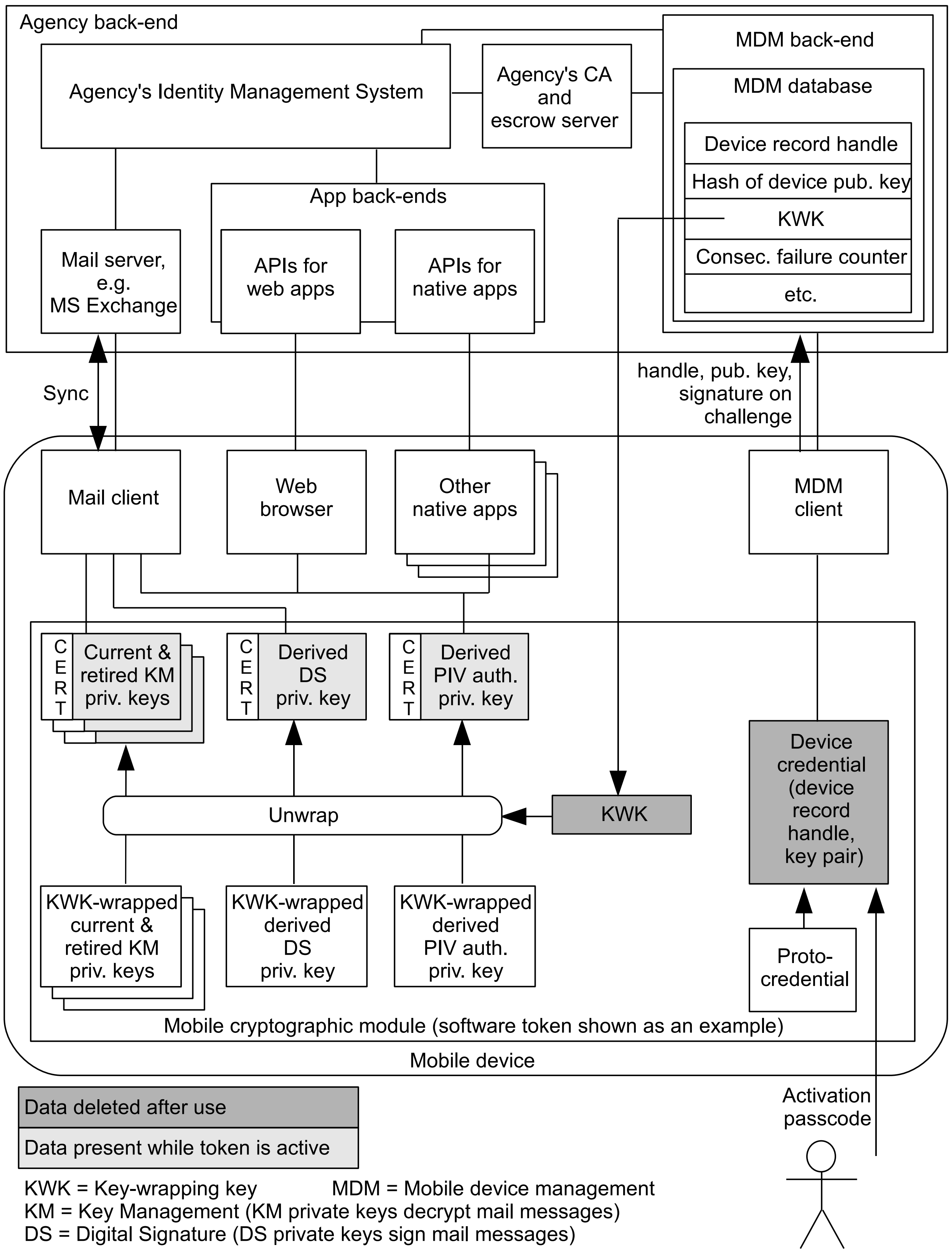

Agency back-end
Agency's Identity Management System
Agency's CA and escrow server
MDM back-end
MDM database
Device record handle
Hash of device pub. key
KWK
Consec. failure counter
etc.
App back-ends
Mail server, e.g. MS Exchange
APIs for web apps
APIs for native apps
Sync
handle, pub. key, signature on challenge
Mail client
Web browser
Other native apps
MDM client
CERT
Current & retired KM priv. keys
CERT
Derived DS priv. key
CERT
Derived PIV auth. priv. key
Unwrap
KWK
Device credential (device record handle, key pair)
KWK-wrapped current & retired KM priv. keys
KWK-wrapped derived DS priv. key
KWK-wrapped derived PIV auth. priv. key
Proto-credential
Mobile cryptographic module (software token shown as an example)
Mobile device
Activation passcode
Data deleted after use
Data present while token is active
KWK = Key-wrapping key
MDM = Mobile device management
KM = Key Management (KM private keys decrypt mail messages)
DS = Digital Signature (DS private keys sign mail messages)

Figure 4: MDM-Based Derived Credentials Architecture

cryptographic module by replacing the library that implements the API. For example, the user may use an Exchange ActiveSync (EAS) client that synchronizes mail messages (as well as calendar entries and contacts) with an existing Microsoft Exchange server. The client authenticates to the server using the derived PIV authentication private key and certificate, signs messages with the digital signature private key, and decrypts messages, new or old, with the key management private keys. The Exchange server requires no modification.

Native applications may authenticate to back-end APIs using the derived PIV authentication private key and certificate for TLS client authentication during the establishment of a TLS connection.

A web browser may also use the derived PIV authentication private key and certificate to establish TLS connections. Some manufacturers of PIV/CAC cards or readers supply their own web browsers. Some of those browsers could be interfaced to a mobile cryptographic module instead of a cryptographic module that accesses a card, by replacing the cryptographic libraries that they use. A browser interfacing to a mobile cryptographic module could also be built from scratch using WebKit [28].

The derived PIV authentication private key and certificate may be also used to set up a VPN tunnel from the mobile device to the agency back-end.

The mobile cryptographic module also contains the device credential used to authenticate the device to the back-end and retrieve the KWK, wrapped or unwrapped, as described above in Section 2. The device authentication and KWK retrieval protocol is carried out by a native MDM client application running on the mobile device and an MDM back-end included in the agency back-end.

## 3.2 Access to the Mobile Cryptographic Module through APIs

The derived credentials are used by applications such as a mail client, a web browser and other native applications. To accommodate existing applications, the module may expose one or more of the same APIs that are used today by applications to access the corresponding credentials in a PIV/CAC card, such as a subset of the PIV Client API [3, Part 3], a subset of PKCS#11 [29], or a subset of PC/SC [30, Part 8]. For the convenience of future applications, the module should also expose a new API to be designed and standardized with the specific purpose of using the derived credentials stored in a mobile cryptographic module.

Each mobile cryptographic module API is implemented by a library linked to each native application that uses the module. If the module is a TEE token or a traditional hardware token, the library in turn calls an API provided by the operating system for having cryptographic operations performed by the secure OS of the TEE or the separate processor or ad-hoc cryptographic hardware of the traditional hardware token. We shall refer to this second API as an *inner API*, and to the mobile cryptographic module API used by native applications to make use of the derived credentials as an *outer API*. An inner API is also used by the MDM client to perform tasks such as unwrapping the private keys of the derived credentials and regenerating the device credential.

Data used by the mobile cryptographic module but not stored within a TEE token or traditional hardware token, including all data used by a software token, the wrapped credentials kept outside a TEE token or traditional hardware token, and the protocredential and device credential kept outside a traditional hardware token, are stored in shared storage accessed by the code of the libraries that implement the outer API, which is linked to the native applications that use the derived credentials. The shared storage, and the inner API in the case of a TEE token or traditional hardware token, are accessed directly by the MDM client, but only through API library code by the native applications that use the derived credentials. Hence they need not be visible to developers of native applications.

In iOS, shared storage is provided by the keychain, with access to shared keychain items based on an "app ID prefix" shared by iOS applications deployed by the federal agency that issues the derived credentials. In Android, shared storage can be supplied by an SQLite database accessed via a ContentProvider implemented by the MDM client, with access to the ContentProvider restricted to applications signed with a code-signing credential used for agency-deployed Android applications.

Alternatively, the mobile cryptographic module and the MDM client may be implemented by the operating system, either in a standard OS version, or in a tailored version of an open-source OS such as Android. In that case the shared storage is internal to the operating system.

## 3.3 Module Activation and Deactivation

The module activation procedure comprises the following steps:

1. The MDM client asks the user to enter the passcode and regenerates the device credential from the passcode and the protocredential. If the mobile cryptographic module is a TEE token, the passcode is imported into the module via a trusted path from a trusted user interface, and the device credential is regenerated within the module.

2. The MDM client establishes a TLS connection with server-only authentication to the MDM back-end, then authenticates with the device credential over the TLS connection as described above in Section 2.1.

3. The MDM client retrieves the KWK, wrapped or unwrapped, over the TLS connection. The KWK is wrapped if the mobile cryptographic module is a TEE token or a traditional hardware token, unwrapped if the module is a software token.

4. The MDM client deletes the device credential from the mobile cryptographic module.

5. If the mobile cryptographic module is a TEE token or a traditional hardware token, the KWK is unwrapped using the KWK-wrapping key. The plaintext KWK is imported into the token and the wrapped KWK is deleted.

6. The KWK is used to unwrap the private keys of the derived credentials. If the mobile cryptographic module is a TEE token or a traditional hardware token, the plaintext private keys are imported into the token.

7. The MDM client deletes the plaintext KWK from the mobile cryptographic module.

The procedure for deactivating the mobile cryptographic module comprises just one step:

1. The MDM client deletes the plaintext private keys of the derived credentials from the module.

## 3.4 Device Registration and Provisioning

Preparing a mobile device so that it can provide logical access to the agency's information systems and handle signed and encrypted mail requires three steps: installing an KWK-aware MDM client in the device, registering the device with the MDM back-end, and provisioning derived credentials to the device. The code of a KWK-aware MDM client need not contain any confidential information, so it may be downloaded by the user from an agency application store without authentication. A remote registration process is described below in Section 3.4.1, and a remote provisioning process in Section 3.4.2. SP 800-157 requires in-person issuance of the derived PIV-credential at Level Of Assurance 4 (LOA-4); in-person registration and provisioning processes are described below in Section 3.4.3.

### 3.4.1 Remote Device Registration with the MDM Back-End

As illustrated in Figure 5, during the remote device registration process the user interacts both with a laptop or desktop personal computer (PC) and with the mobile device being registered. The process comprises the following steps, illustrated by numbered arrows in the figure:

1. The user plugs his or her PIV/CAC card to a card reader connected to the PC, activates the card by entering a PIN, and uses a web browser running on the PC to access the MDM back-end. The browser establishes a mutually authenticated TLS connection to the back-end using the PIV Authentication Key and certificate stored in the PIV/CAC card for TLS client authentication, and requests initiation of the registration process.

   The MDM back-end creates a device record in the MDM database, containing the device record handle, the certificate associated with the PIV Authentication Key, a cross-site request forgery (CSRF) prevention code, a registration code, a confirmation code, and a confirmation deadline, as illustrated in Figure 6(a). (The record also has fields left empty as shown in the figure, and may have fields besides those shown in the figure, used for other purposes by the MDM infrastructure.)

   The handle is a permanent identifier that uniquely identifies the record among all device records in the MDM database; it may be the value of a counter of device records maintained by the database management system. The certificate serves to associate the device record with the user and to include in the record user data that will be used later during the provisioning process.[10] The CSRF-prevention code, the registration code, and the confirmation code are generated by the MDM back-end: the CSRF-prevention code is a random high-entropy string; the registration code is a random medium-entropy code, such as an 8-digit number, unique among registration codes, which will be deleted after use; and the confirmation code is a random low-entropy code, such as a 4-digit number. (The reasons why medium entropy is suitable for the registration code and low entropy for the confirmation code are explained below.) The confirmation deadline is a time, a few minutes in the future, set according to policy; if confirmation has not been completed by the deadline, the device record will be deemed invalid and eligible for garbage-collection.

[10]This certificate is *not* a credential. It is a copy of the certificate in the PIV card. The private key is in the PIV card. Data extracted from the certificate could be stored in the record instead of the certificate itself.

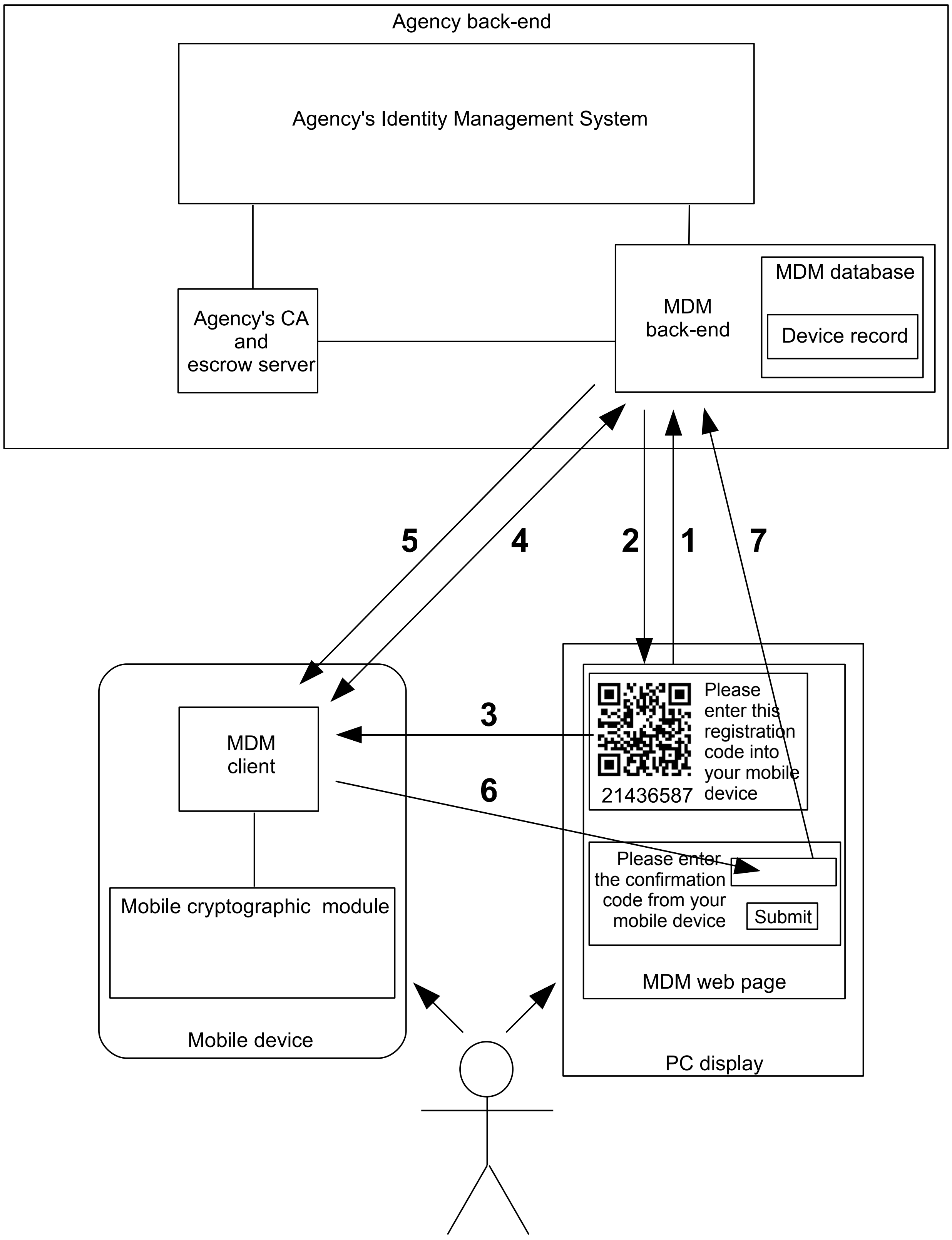

Agency back-end
Agency's Identity Management System
Agency's CA and escrow server
MDM back-end
MDM database
Device record
5
4
2
1
7
3
6
MDM client
Mobile cryptographic module
Mobile device
Please enter this registration code into your mobile device
21436587
Please enter the confirmation code from your mobile device
Submit
MDM web page
PC display

Figure 5: Remote device registration

2. In response to the registration initiation request, the MDM back-end returns a web page showing the registration record, both as a QR code and as a string of digits, and a form including: a field where the user will later enter the confirmation code; a hidden field containing the device record handle; and a hidden field containing the CSRF-prevention code. After downloading the web page the MDM back-end closes the TLS connection, in order to force the browser to authenticate again later when the user submits the form.

3. The user enters the registration code shown on the PC into the MDM client running in the mobile device, either by manually copying the string of digits, or, if the mobile device has a camera and the camera has not been disabled, by letting the MDM client use the camera to read the QR code.

4. The MDM client establishes a TLS connection to the MDM back-end where the back-end authenticates with a TLS server certificate, and sends the registration code.

   The MDM back-end finds the device record containing the registration code and returns the handle of the record over the TLS connection.

   The MDM client obtains a passcode to be used for activation of the mobile cryptographic module, either by prompting the user, or, if the module is a TEE token, by asking the TEE token to request the passcode using the trusted interface and import it into the token.

   The MDM client causes a protocredential and corresponding device credential to be generated from the device record handle and the passcode as described in Appendix B, either by generating them itself, or by instructing the mobile cryptographic module to generate them internally through the module API if the module is a TEE token.

   The MDM client also generates a random high-entropy KWK, as follows:

   - If the cryptographic module is a TEE token or a traditional hardware token, the MDM client asks the token to generate the KWK and a random high-entropy KWK-wrapping key (KWKWK), to wrap the KWK with the KWKWK, to export the wrapped KWK, and to delete the KWK from the token while retaining the KWKWK.
   - If the cryptographic module is a software token, the MDM client generates the KWK itself and does not wrap it.

   The wrapped or unwrapped KWK is not stored in the mobile cryptographic module. The MDM client keeps it in temporary storage, which could be volatile working memory or persistent private storage, until it sends it to the MDM back-end as described below.

   The MDM client obtains a push notification identifier, which can be used to send push notifications to the client. In iOS, the push notification identifier is called a "device token," and is issued by the Apple Push Notification service (APNs). In Android, the push notification ID is called a "Registration ID" and is issued by the Google Cloud Messaging for Android (GCM) service.

   The MDM client sends the following to the MDM back-end over the TLS connection: the wrapped or unwrapped KWK; the push notification identifier; the device record handle; the public key component of the device credential, and a signature computed with the associated private key on a challenge derived from the TLS master secret and the back-end's TLS server certificate..

The MDM client deletes the wrapped or unwrapped KWK from temporary storage, and causes the device credential to be deleted from the mobile cryptographic module, either by deleting it itself, or by instructing the module to perform the deletion through the module API if the module is a TEE token.

5. If the confirmation deadline has not been reached, the MDM back-end uses the public key to verify the signature on the challenge.

   Then it generates a provisioning session code and provisioning session expiration time, which will allow the user to request provisioning immediately after registration without having to reenter the passcode. The code is a high-entropy random string.

   Then it computes a hash of the public key, and stores the following in the device record: the hash of the public key; the wrapped or unwrapped KWK; the push notification identifier; the provisioning session code; and the provisioning session expiration time.

   Then it deletes the registration code from the device record, after which the state of the device record is as illustrated in Figure 6(b).

   Then it downloads the provisioning session code and the confirmation code to the MDM client over the TLS connection. The client stores the provisioning session code in its own private storage, displays the confirmation code and asks the user to enter the confirmation code into the web page shown on the PC.

6. The user manually copies the confirmation code from the mobile device to the form field in the web page shown on the PC.

7. The user clicks a button on the web page to submit the form containing the confirmation code. The browser establishes a new mutually authenticated TLS connection to the back-end using the PIV Authentication Key and certificate stored in the PIV/CAC card for TLS client authentication, which requires the user's PIV/CAC card to still be plugged into the card reader connected to the PC. Then the browser submits the form over the TLS connection, conveying the confirmation code, the device record handle, and the CSRF-prevention code to the MDM back-end.

   When it receives the form, the MDM back-end uses the device record handle to locate the device record, checks that the device record contains a CSRF-prevention code and a confirmation code that coincide with the ones in the form, and verifies that the confirmation deadline has not been reached. If so it deletes the CSRF-prevention code, the confirmation code and the confirmation deadline from the record, which is then as shown in Figure 6(c).

   In response to successful submission of the confirmation code, the MDM back-end uses the push notification identifier to send a notification to the mobile device informing the user that the registration process has been successfully completed and instructing the user to initiate the provisioning process on the mobile device, which the user can do by tapping on the notification.

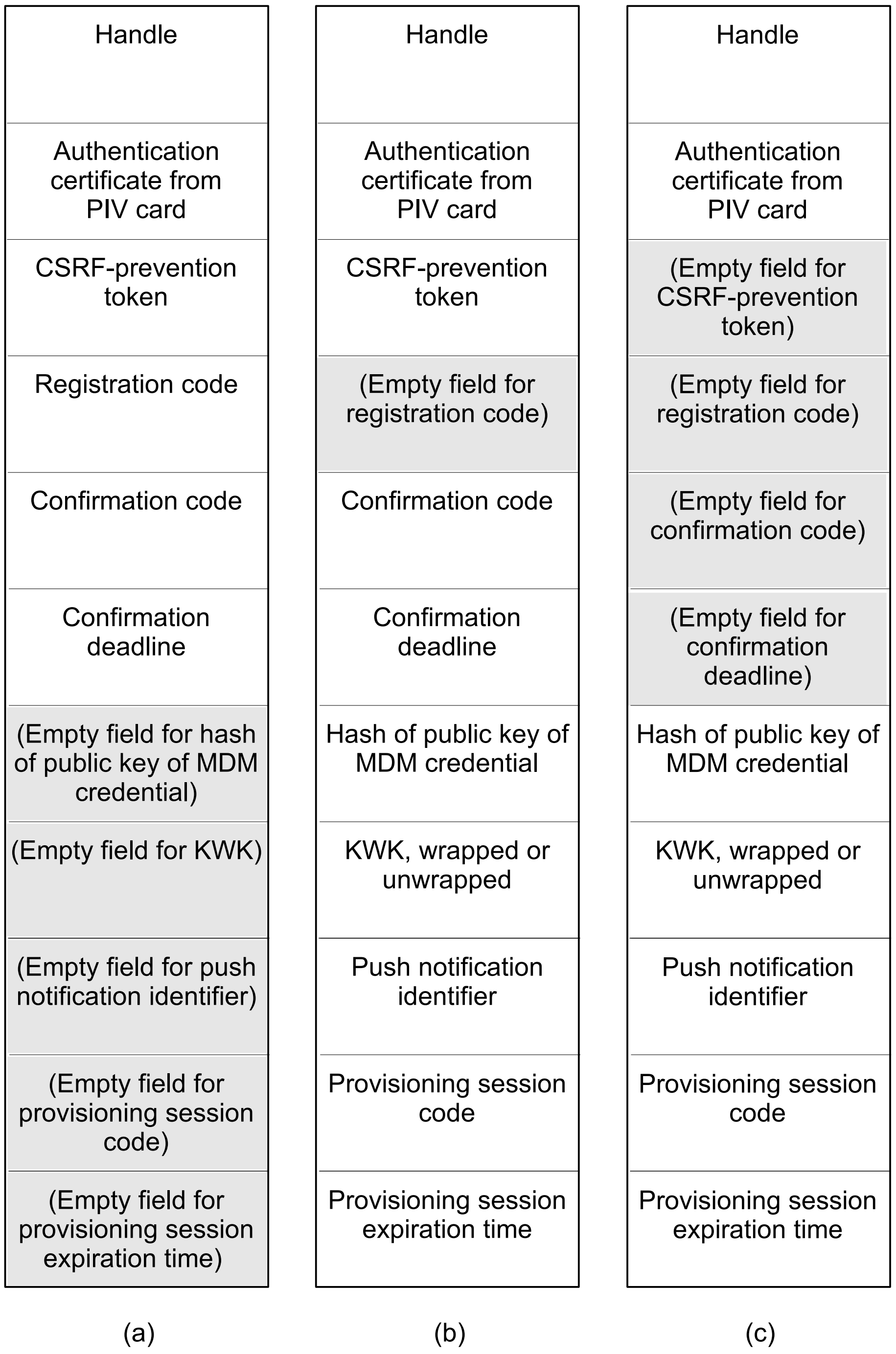

Handle
Authentication certificate from PIV card
CSRF-prevention token
Registration code
Confirmation code
Confirmation deadline
(Empty field for hash of public key of MDM credential)
(Empty field for KWK)
(Empty field for push notification identifier)
(Empty field for provisioning session code)
(Empty field for provisioning session expiration time)
(a)
Handle
Authentication certificate from PIV card
CSRF-prevention token
(Empty field for registration code)
Confirmation code
Confirmation deadline
Hash of public key of MDM credential
KWK, wrapped or unwrapped
Push notification identifier
Provisioning session code
Provisioning session expiration time
(b)
Handle
Authentication certificate from PIV card
(Empty field for CSRF-prevention token)
(Empty field for registration code)
(Empty field for confirmation code)
(Empty field for confirmation deadline)
Hash of public key of MDM credential
KWK, wrapped or unwrapped
Push notification identifier
Provisioning session code
Provisioning session expiration time
(c)

Figure 6: States of the device record during remote registration

An adversary may submit guesses of registration codes from one or more computers having connectivity to the MDM back-end. Submission of a correct guess will result in data sent by the adversary being stored in fields of the device record, instead of data sent by the user. The adversary will receive the confirmation code, but the CSRF-prevention code will prevent the adversary from submitting it via the user's browser in a cross-site request forgery. Medium entropy is sufficient for the registration code because a correct guess of the code by the adversary only results in the user having to restart the registration process. The purpose of using a medium rather than low entropy code is to minimize the chances that the code will be guessed and the user will be inconvenienced.

The confirmation code is not strictly necessary. It would be sufficient for the MDM client to show a message asking the user to click a confirmation button on the PC, which would result in the submission of a form containing just the device record handle and the CSRF-prevention code in hidden fields; but an impatient user might submit the form without waiting for the message to be displayed on the mobile device. Requiring entry of the confirmation code forces the user to wait for the message. A low entropy confirmation code is sufficient for that purpose.

### 3.4.2 Remote Provisioning of the Derived Credentials

Provisioning makes use of the MDM back-end in a role similar to that of a registration authority (RA). To simplify the description of the process we make the following assumptions:

- The user initiates the process immediately after registration. Otherwise the provisioning session could expired, in which case the user would have to reenter the passcode needed to regenerate the device credential, and the MDM client would use the device credential instead of the provisioning session code to authenticate to the MDM back-end.

- The user has a government-issued electronic mail account. Otherwise the digital signature credential and the current and retired key management credentials would not be provisioned.

- The roles of CA and escrow server are fulfilled by the same entity.

- The combined CA and escrow server provisions certificates and escrowed keys automatically without waiting for manual approval by a human operator. Otherwise the user would have to wait for approval, and push notifications would be sent to the mobile device when the certificates and/or keys become ready for download.

Under this assumptions the process comprises the following steps, illustrated by numbered arrows in Figure 7:

1. The MDM client establishes a TLS connection to the MDM back-end with server-only authentication and sends to the MDM back-end a provisioning request. The request includes the provisioning session code, which the back-end uses to locate the device record. (With high probability there is only one record containing the code, since the code is a high-entropy random string.) The MDM back-end verifies that the provisioning session expiration time has not been reached.

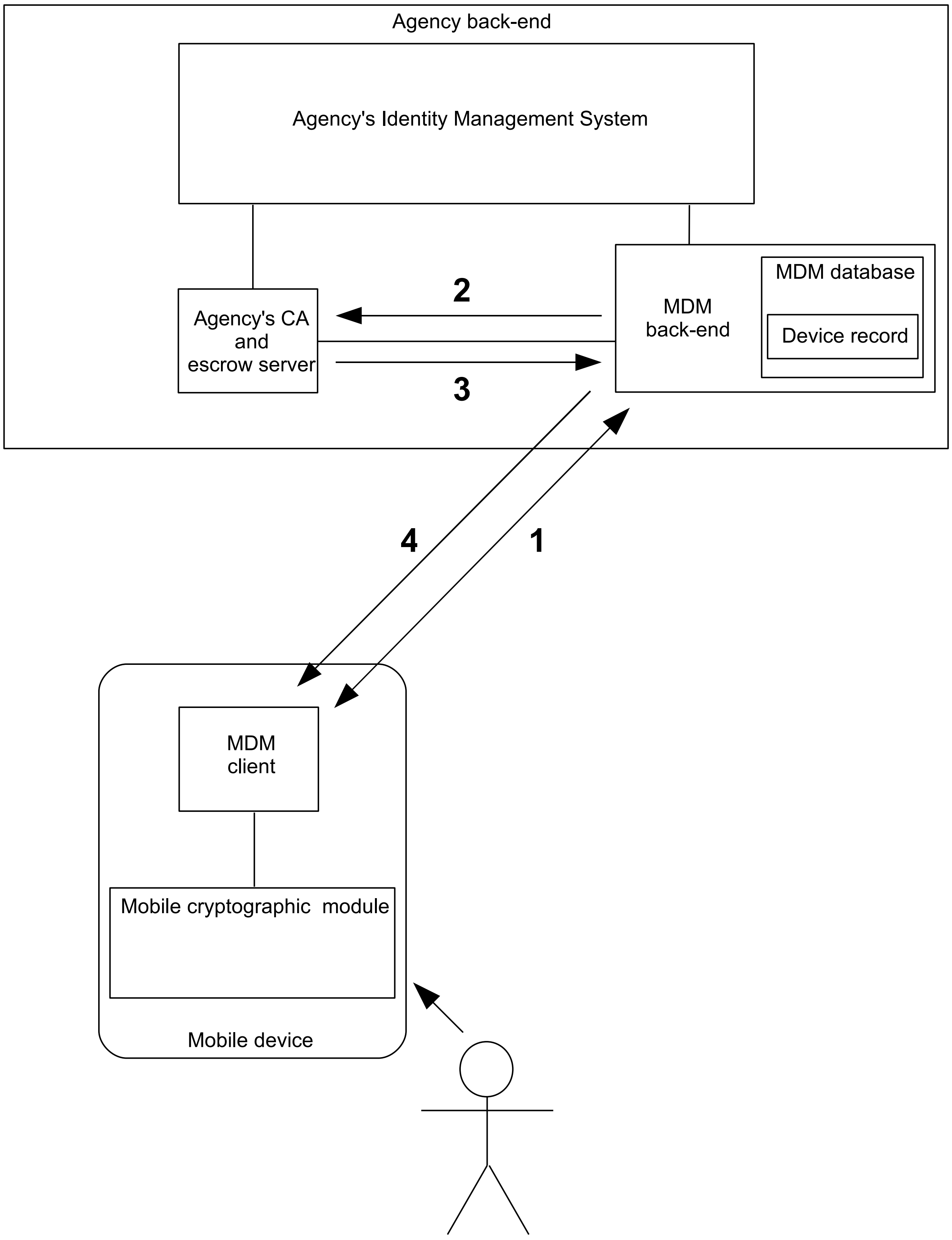

Agency back-end
Agency's Identity Management System
Agency's CA and escrow server
MDM back-end
MDM database
Device record
2
3
4
1
MDM client
Mobile cryptographic module
Mobile device

Figure 7: Remote provisioning

Then the back-end verifies that the PIV Authentication certificate found in the device record is still valid, and uses data in the certificate and/or in the agency's IDMS to assemble a set of user data to be included in the derived PIV authentication certificate to be provisioned, and a set of user data to be included in the digital signature certificate to be provisioned. (The two sets of user data may or may not be identical.)

The back-end sends the two sets of user data and the wrapped or unwrapped KWK found in the device record to the MDM client.

The MDM client generates key pairs and certificate signing requests (CSRs) for the derived PIV authentication and digital signature credentials, wraps the private keys with the KWK, stores the wrapped private keys, deletes the plaintext versions of the private keys, and deletes the KWK. (If the mobile cryptographic module is a TEE token or a traditional hardware token, the KWK must be unwrapped before use, and all the cryptographic operations are performed within the token.) With the purpose of protecting the escrowed key management private keys, the MDM client also generates an ephemeral Diffie-Hellman (DH) key pair using domain parameters that may have been chosen and published by the agency [31], and retains the DH private key.

Then the MDM client sends the CSRs and the DH public key to the MDM back-end.

2. The MDM back-end verifies that the sets of user data in the CSRs have not been altered, and forwards the CSRs, the DH public key, and the PIV Authentication certificate found in the device record to the entity that plays the role of CA and escrow server. (It should be noted that the back-end is not using the certificate for authentication, since it does not have the private key. Instead it authenticates itself to the entity as a kind of RA, and it will be trusted by the entity to forward the provisioned credentials to a mobile device pertaining to the subject of the certificate.)

3. The entity playing the role of CA and escrow server verifies the user data, generates its own ephemeral DH key pair using the same domain parameters as the MDM client, computes the DH shared secret, derives a symmetric key from the shared secret that it uses to wrap the escrowed current and retired key management private keys, issues the derived PIV authentication certificate and the derived digital signature certificate, and sends those certificates as well as its DH public key and the wrapped current and retired key management private keys and corresponding certificates to the MDM back-end.

4. The MDM back-end forwards everything received from the entity to the MDM client, adding the wrapped or unwrapped KWK found in the device record, in a response to the provisioning request of step 1 sent over the same TLS connection through which the request was received.

   The MDM client computes the DH shared secret, derives the symmetric key, unwraps the key management private keys, rewraps them with the KWK, stores the wrapped versions, and deletes the plaintext versions as well as the KWK. (If the mobile cryptographic module is a TEE token or a traditional hardware token, the KWK must be unwrapped before use, and all the cryptographic operations are performed within the token.)

The MDM client also stores all the certificates in the module, where they will be associated with the corresponding private keys whenever the mobile cryptographic module is active and the private keys are in the clear.

### 3.4.3 In-Person Registration and Provisioning

SP 800-157 requires in-person issuance of the derived PIV credential at LOA-4, with applicant identification using a biometric sample verified against the applicant's PIV Card. The above registration and provisioning processes can be modified as follows to meet this requirement.

For in-person registration, the PC used to request the registration code is replaced with a dedicated workstation located on agency premises. Access to the workstation is provided to the applicant after verification of a biometric sample against the applicant's PIV card.

For in-person provisioning, step 4 of the provisioning process of Section 3.4.2 (where the MDM back-end forwards credentials to the MDM client) does not take place in response to step 1. Instead it takes place in response to a new request, where the MDM client supplies an in-person provisioning code entered by the applicant, which the applicant obtains from an administrator after verification of a biometric sample against the applicant's PIV card. The in-person provisioning code is an additional field of the device record, whose value is set when the record is created.

## 3.5 Local Escrow

Recall that the KWK is needed to unwrap the private keys of the derived credentials, and is retrieved from the MDM back-end when the user activates the mobile cryptographic module. If the MDM back-end is down and the module is not active, the user cannot retrieve the KWK and unwrap the private keys, and thus is not able to authenticate to application back-ends, sign mail messages or decrypt mail messages. Also, the user may want to be able to read encrypted mail messages while offline; but if the mobile cryptographic is not active and the mobile device is offline, the key management keys needed to decrypt mail messages cannot be unwrapped.

Therefore it may be desirable to provide the user with an alternative method of obtaining the plaintext versions of the private keys of some or all of the derived credentials. One such alternative method is to also wrap those private keys separately under a random high-entropy key that would be printed as a long string of characters or as a QR code readable by the device camera. The printed QR code would ordinarily be stored in a secure cabinet, but the user could retrieve it when necessary and use it to unwrap the private keys of the desired credentials.

This method would amount to a local escrow of some or all of the private keys in the mobile device itself, protected by the high-entropy key. Which keys are locally escrowed would be a matter of policy. For example, the key management private keys used for mail decryption could be locally escrowed to allow the user to read mail offline, while the derived PIV authentication and digital signature private keys could by prohibited from being locally escrowed for the sake of security.

## 3.6 Data Protection and Device Locking

Besides protecting derived credentials, the mobile cryptographic module can be used to protect sensitive data such as decrypted mail attachments, documents downloaded from the agency back-end, and documents created or edited on the mobile device. To that purpose the mobile cryptographic module can store one or more symmetric data encryption keys wrapped by the KWK. When the user activates the module, the KWK is used to decrypt the data encryption keys, after which encryption/decryption services using the data encryption keys can be made available to authorized native applications installed on the device.

When the mobile cryptographic module is implemented by the operating system, it can be used to provide file system encryption in combination with device locking, as follows. A symmetric file system encryption key is stored in the mobile cryptographic module wrapped by the KWK. The user unlocks the device by entering a passcode. The activation procedure of Section 3.3 is then carried out using the unlocking passcode as activation passcode, and the file system key is unwrapped using the KWK retrieved from the MDM back-end. While the device remains unlocked the operating system uses the unwrapped file system key to decrypt files on the fly as they are accessed by applications.

It is also possible to implement two levels of file system encryption. In a two-level encryption scheme, ordinary data becomes available when the user unlocks the device by entering a passcode, but derived credentials and sensitive agency data only become available later, if and when the user enters a second passcode, which may or may not be the same as the first. A two-level scheme may be implemented, for example, by setting up two file systems in the mobile device, protected by different file system encryption keys.

## 3.7 Open Standards

The MDM-based derived credentials architecture involves the following components:

- A mobile operating system.
- Optional security hardware such as a TEE token or a traditional hardware token.
- An MDM client.
- An MDM back-end.
- A CA and an escrow server or a single entity playing the role of CA and escrow server.
- One or more mail clients.
- One or more mobile web browsers.
- Native applications other than the mail clients and web browsers.

Mail servers, web applications and back-ends of native applications are also part of the architecture, but without being aware of it.

Deployment of the architecture will be greatly facilitated by standard APIs that will allow components to interoperate even when developed and supplied by different parties.

The following interfaces could be standardized:

- A new API for using derived credentials stored in a mobile cryptographic module, which will play the role of outer API in mobile cryptographic modules, and may co-exist in that role with other APIs used by existing applications, as discussed above in Section 3.2.

- An API for directing a TEE token to perform cryptographic operations. Such an API, under the name *GlobalPlatform TEE Functional API*, is envisioned in the GlobalPlatform TEE white paper [19], but is not yet available on the GlobalPlatform web site. When available, it could be used as the inner API in TEE tokens. To that purpose, the cryptographic functionality exposed by the API should include regeneration of a credential from a protocredential as described in Appendix A and initial generation of a protocredential and its corresponding credential as described in Appendix B.

- The interface between the MDM client and the MDM back-end. Today's MDM infrastructures are monolithic: the MDM client and back-end are provided by the same party. However, in the context of derived credentials, it would be useful to enable different parties to provide the client and the back-end. A party providing MDM clients could then focus on supporting a variety of mobile platforms, traditional hardware tokens and TEE tokens, while a party providing MDM back-ends could focus on supporting a variety of enterprise IT architectures and providing good user interfaces to administrators.

- The interface between the MDM back-end and the entity or entities playing the role of CA and escrow server.

## 4 Conclusion

We have proposed a *compensating control* that can be used to increase the security of *derived credentials* stored in mobile devices, the derived credentials being issued by a federal agency to a user upon authentication of the user with a PIV card. The control compensates for the storage of derived credentials in tokens (cryptographic modules) that cannot be removed from the device and do not provide tamper resistance, such as software tokens (software cryptographic modules) and TEE tokens (cryptographic modules implemented within a Trusted Execution Environment). As compensation, the private keys of the derived credentials are wrapped under a random high-entropy key-wrapping key (KWK) that is stored in a secure back-end and retrieved by authenticating to the back-end with a device credential regenerated from a protocredential and a token activation passcode in such a way that an adversary who captures the mobile device is not able to mount an offline guessing attack against the passcode. The control is also useful in conjunction with traditional hardware modules that do provide tamper resistance, because tamper resistance is never absolute.

We have provided an example of a particular derived credentials architecture that takes advantage of the compensating control and leverages an existing MDM infrastructure by storing the KWK in a device record within an MDM database. We have described a procedure for registering the device credential with the MDM database upon user authentication with a PIV/CAC card, and a procedure for provisioning derived credentials to the device where the MDM back-end plays a role similar to that of a registration authority. Certificates for a derived PIV public key and a digital signature public key are issued by an agency certificate authority (CA), while current and retired key management private keys used for decryption of email messages, and their corresponding certificates, are downloaded securely from an escrow service that may be provided by the CA or by a separate entity within the agency. An optional local escrow mechanism may by used, in particular, to enable offline decryption of saved email messages. We have shown how the token storing derived credentials can also be used to store encryption keys for protecting sensitive data; if the token is implemented by the operating system, token activation can be integrated with device unlocking, and the compensating control can provide effective protection for all data stored in the device.

## 5 Disclosure

Pomcor has filed patent applications related to the compensating control and to the derived credentials architecture described in this paper.

## References

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

[2] NIST. FIPS 201-2 Personal Identity Verification (PIV) of Federal Employees and Contractors, August 2013. http://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.201-2.pdf.

[3] NIST. NIST Special Publication 800-73-4, DRAFT Interfaces for Personal Identity Verification (3 Parts)
Part 1- PIV Card Application Namespace, Data Model and Representation
Part 2- PIV Card Application Card Command Interface
Part 3- PIV Client Application Programming Interface
May 13, 2013. http://csrc.nist.gov/publications/PubsDrafts.html#SP-800-73--4.

[4] W. E. Burr, D. F. Dodson, and W. T. Polk. NIST SP 800-63-1 Electronic Authentication Guideline, December 2011. http://csrc.nist.gov/publications/nistpubs/800-63-1/SP-800-63-1.pdf.

[5] Hildegard Ferraiolo. FIPS 201-2 and Derived Credentials. Presentation at the Information Security and Privacy Advisory Board Meeting, February 1, 2012. http://csrc.nist.gov/groups/SMA/ispab/documents/minutes/2012-02/feb1_der_cred_ferraiolo_h_fips_201-2.pdf.

[6] NIST. FIPS 140-2 Security Requirements for Cryptographic Modules, May 2001. http://csrc.nist.gov/publications/fips/fips140-2/fips1402.pdf.

[7] NIST. Cryptographic Key Management Workshop. September 2012. http://www.nist.gov/itl/csd/ct/ckm_workshop_2012.cfm.

[8] F. Corella and K. Lewison. Techniques for implementing derived credentials. Presentation at the 2012 NIST Key Management Workshop. http://csrc.nist.gov/groups/ST/key_mgmt/documents/Sept2012_Presentations/CORELLA_DerivedCredentials.pdf.

[9] F. Corella and K. Lewison. Techniques for Implementing Derived Credentials on Mobile Devices. Blog post. August 22, 2012. http://pomcor.com/2012/08/22/techniques-for-implementing-derived-credentials-on-mobile-devices/.

[10] F. Corella and K. Lewison. Techniques for Implementing Derived Credentials. Revised September 13, 2012. http://pomcor.com/whitepapers/DerivedCredentials.pdf.

[11] F. Corella and K. Lewison. A Comprehensive Approach to Cryptographic and Biometric Authentication from a Mobile Perspective. Revised April 2013. http://pomcor.com/whitepapers/CryptographicAuthentication.pdf.

[12] Hildegard Ferraiolo, Andrew Regenscheid, David Cooper, Salvatore Francomacaro, and William Burr. Mobile, PIV, and Authentication. Draft NIST Interagency Report 7981. March 2014. http://csrc.nist.gov/publications/PubsDrafts.html#NIST-IR-7981.

[13] Hildegard Ferraiolo, David Cooper, Salvatore Francomacaro, Andrew Regenscheid, Jason Mohler, Sarbari Gupta, and William Burr. Guidelines for Derived Personal Identity Verification (PIV) Credentials. Draft NIST Special Publication 800-157. http://csrc.nist.gov/publications/PubsDrafts.html#SP-800-157.

[14] Francisco Corella and Karen Lewison. Comments on the NIST Guidelines on Derived Credentials. March 31, 2014. http://pomcor.com/documents/CommentsOnDerivedCredentials.txt.

[15] Office of Management and Budget (OMB). Safeguarding Against and Responding to the Breach of Personally Identifiable Information. M-07-16. May 22, 2007. http://www.whitehouse.gov/sites/default/files/omb/memoranda/fy2007/m07-16.pdf.

[16] ARM. TrustZone. http://www.arm.com/products/processors/technologies/trustzone/index.php.

[17] Karthikeyan Bhargavan, Antoine Delignat-Lavaud, Cédric Fournety, Alfredo Pironti, and Pierre-Yves Strub. Triple Handshakes and Cookie Cutters: Breaking and Fixing Authentication over TLS. March 2014. https://secure-resumption.com/tlsauth.pdf.

[18] William E. Burr, Donna F. Dodson, Elaine M. Newton, Ray A. Perlner, W. Timothy Polk, Sarbari Gupta, and Emad A. Nabbus. NIST SP 800-63-2 Electronic Authentication Guideline, August 2013. http://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-63-2.pdf.

[19] GlobalPlatform. The Trusted Execution Environment: Delivering Enhanced Security at a Lower Cost to the Mobile Market. February 2011. http://www.globalplatform.org/documents/GlobalPlatform_TEE_White_Paper_Feb2011.pdf.

[20] Trustonic. TrustZone and TEE. http://www.trustonic.com/technology/trustzone-and-tee.

[21] Sierraware. Open Virtualization for ARM TrustZone. http://www.openvirtualization.org/open-source-arm-trustzone.html.

[22] GlobalPlatform. Trusted User Interface API Version 1.0. June 2013. Available for licensed download at http://www.globalplatform.org/.

[23] GlobalPlatform. TEE Secure Element API. July 2013. Available for licensed download at http://www.globalplatform.org/.

[24] Jerome Azema and Gilles Fayad. M-Shield Mobile Security Technology: making wireless secure. http://focus.ti.com/pdfs/wtbu/ti_mshield_whitepaper.pdf.

[25] Christopher Tarnovsky. Hacking The Smartcard Chip (Blackhat 2010). Video broken into eight 10-minute segments. http://www.securitytube.net/video/945.

[26] Infineon. World's Most Stringent Security Tests Confirm Infineon's Security Competence in Smart Card ICs. http://www.smartcardalliance.org/articles/2006/01/04/worlds-most-stringent-security-tests-confirm-infineons-security-competence-in-smart-card-ics.

[27] Ross Anderson and Markus Kuhn. Tamper Resistance – a Cautionary Note. Second USENIX Workshop on Electronic Commerce, 1996. http://www.cl.cam.ac.uk/users/mgk25/tamper.pdf.

[28] The WebKit Open Source Project. http://www.webkit.org/.

[29] RSA Laboratories. PKCS #11: Cryptographic Token Interface Standard. http://www.emc.com/emc-plus/rsa-labs/standards-initiatives/pkcs-11-cryptographic-token-interface-standard.htm.

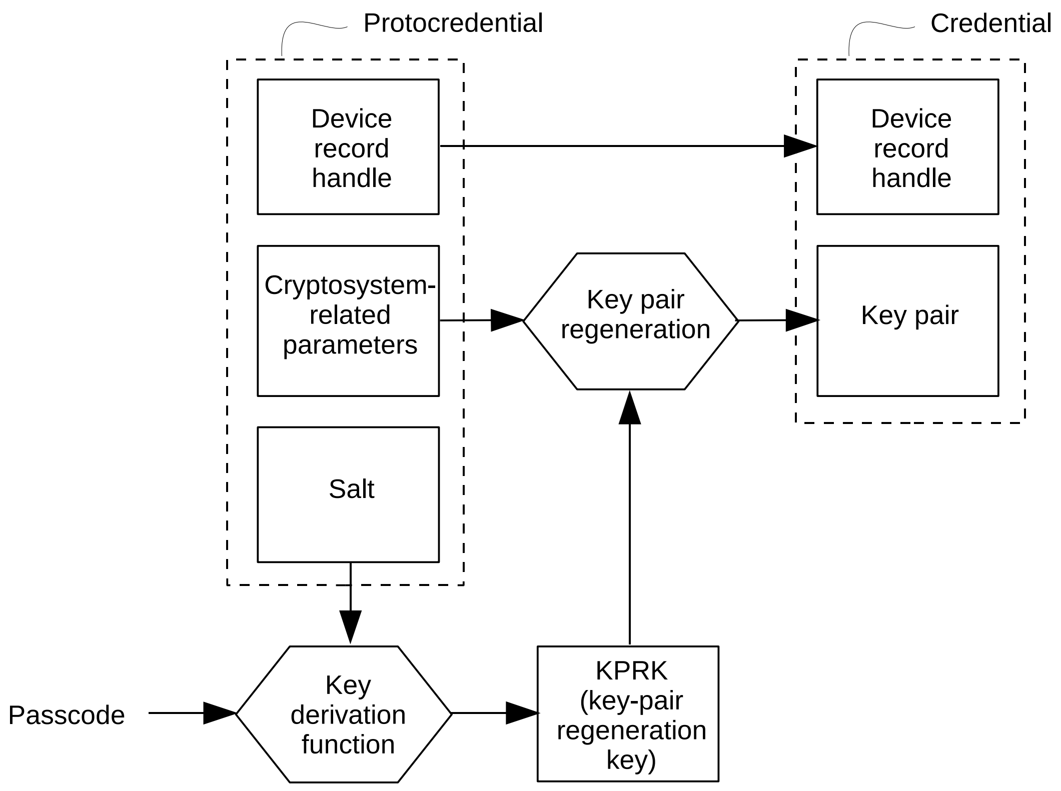


Figure 8: Regeneration of a credential from a protocredential and a passcode

[30] PC/SC Workgroup Specifications 2.01.14.
http://www.pcscworkgroup.com/specifications/specdownload.php.

[31] Elaine Barker, Lily Chen, Allen Roginsky, and Miles Smid. Recommendation for Pair-Wise Key Establishment Schemes Using Discrete Logarithm Cryptography. NIST Special Publication 800-56A Revision 2. http://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-56Ar2.pdf.

[32] H. Krawczyk and P. Eronen. HMAC-based Extract-and-Expand Key Derivation Function (HKDF), May 2010. http://tools.ietf.org/html/rfc5869.

[33] B. Kaliski. PKCS #5: Password-Based Cryptography Specification Version 2.0, September 2000. http://tools.ietf.org/html/rfc2898.

[34] NIST. Digital Signature Standard (DSS), July 2013. FIPS PUB 186-4,
http://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.186-4.pdf.

# A Credential Regeneration from a Protocredential

Figure 8 illustrates the process of regenerating a credential consisting of a device record handle and a key pair, from a protocredential and a passcode. The protocredential comprises the device record handle, parameters related to the cryptosystem that the key pair pertains to, and a random high-entropy secret salt.

The salt and the passcode are used to produce a key-pair regeneration key (KPRK) using

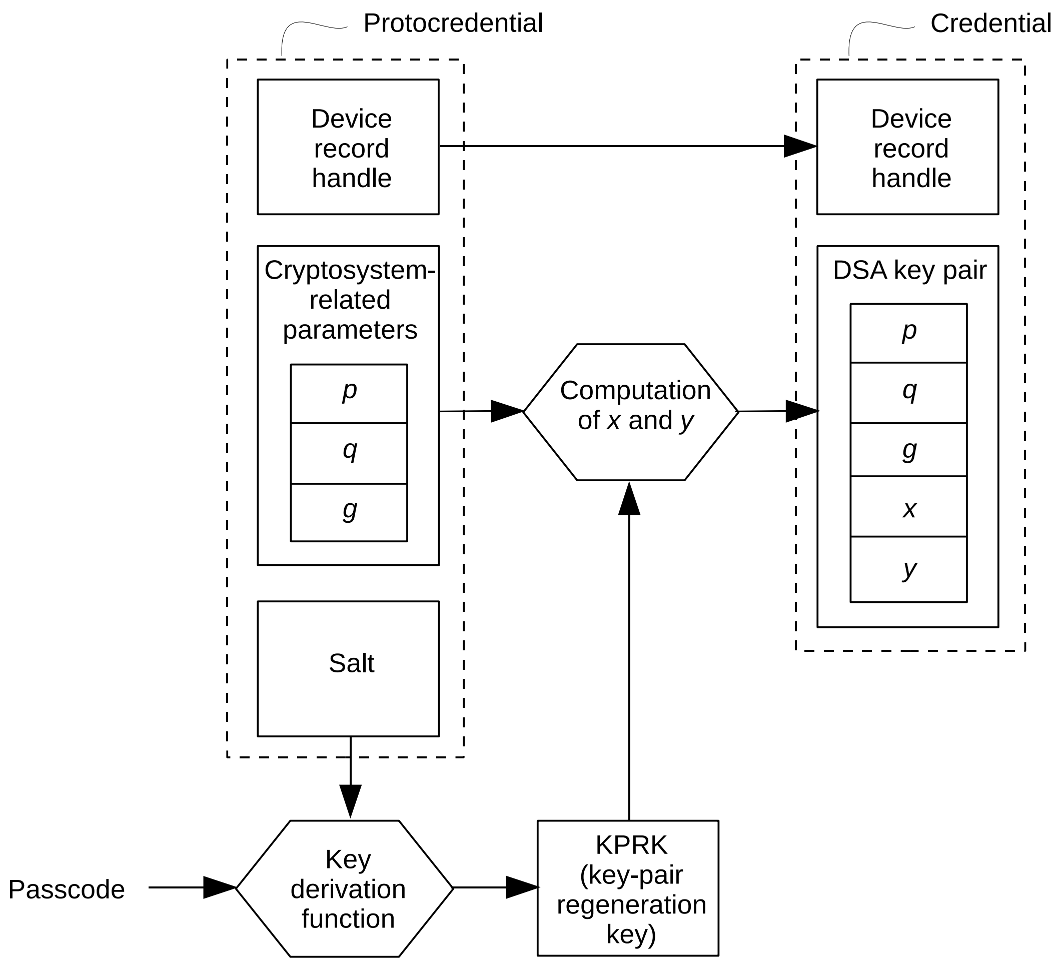


Figure 9: Regeneration of a DSA credential from a protocredential and a passcode

a key derivation function such as HKDF [32].[11] The KPRK is then used in conjunction with the cryptosystem-related parameters to regenerate the key pair. The key pair and the device record handle present in the protocredential comprise the regenerated credential.

Figure 9 illustrates more specifically the case where the cryptosystem is the Digital Signature Algorithm (DSA) specified in Section 4 of the Digital Signature Standard [34].

With the notations of the DSS, the cryptosystem-related parameters of the protocredential are $p$, $q$, and $g$. These are also the domain parameters, which may be common to different key pairs and may be publicly known. It is not necessarily the case, however, that the cryptosystem-related parameters are domain parameters; in other cryptosystems the cryptosystem-related parameters are specific to the key pair and are secret.

[11]PBKDF2 [33] with a large number of iterations could be used instead of HKDF, to provide additional security against an adversary who has somehow obtained information about the key pair, such as the hash of the public key or a signature on a challenge computed with the private key, in addition to obtaining the salt contained in the protocredential. But using PBKDF2 would incur a cost in terms of token activation latency and battery life.

The key pair comprises the parameters $p$, $q$, $g$, $x$ and $y$. The KPRK is a string of $N + 64$ bits, where $N$ is the bitlength of $q$, and $x$ and $y$ are computed as described in Appendix B.1.1 of the DSS, with the KPRK being substituted for the random string *returned_bits*. The private key is $x$ and the public key is $y$.

Regeneration of a credential comprising a key pair pertaining to the ECDSA cryptosystem specified in Section 6 of the DSS [34] is very similar. The cryptosystem parameters are again the domain parameters. The private and public keys are computed as described in Appendix B.4.1 of the DSS, with the KPRK being substituted for the random string *returned_bits*.

Several methods can be used to regenerate an RSA key pair. Either the decryption exponent or the encryption exponent can be derived from the KPRK, the other exponent being computed from the protocredential and the KPRK-derived exponent. The protocredential may comprise the prime factors $p$ and $q$ of the modulus as well as parameters that facilitate performance optimizations such as the use of the Chinese Remainder Theorem. The RSA regeneration process may fail for various reasons, but the probability of failure is very small, so regeneration success cannot be used to test guesses of passcodes in an offline attack.

## B Initial Generation of a Protocredential and the Corresponding Credential

Figure 10 illustrates the process of initial generation of a protocredential and the corresponding credential from a passcode, in the specific case where the credential pertains to the DSA cryptosystem.

The cryptosystem-related parameters $p$, $q$ and $g$ are generated using a random number generator (RNG) as described in Section 4.3.1 and Appendix A of the DSS [34]. However, since, as we saw above, they are also the domain parameters, they may be common to different key pairs and they may have been generated and published once and for all by a trusted party such as the federal agency issuing the derived credentials. Hence generation of the cryptosystem-related parameters is optional, as indicated by the dashed hexagon in the figure.

The random high-entropy secret salt is generated by the random number generator. The KPRK is derived from the salt and the passcode using the same key derivation function that will later be used for credential regeneration. The key pair $(p, q, g, x, y)$ is generated by computing $x$ and $y$ from the cryptosystem-related parameters $(p, q, g)$ and the KPRK as described in Appendix B.1.1 of the DSS, with the KPRK being substituted for the random string *returned_bits*. The device record handle is obtained by the MDM client from the MDM back-end during the registration process described above in Section 3.4.1. The protocredential consists of the handle, the crypto-system related parameters and the salt, while the credential consists of the handle and the key pair.

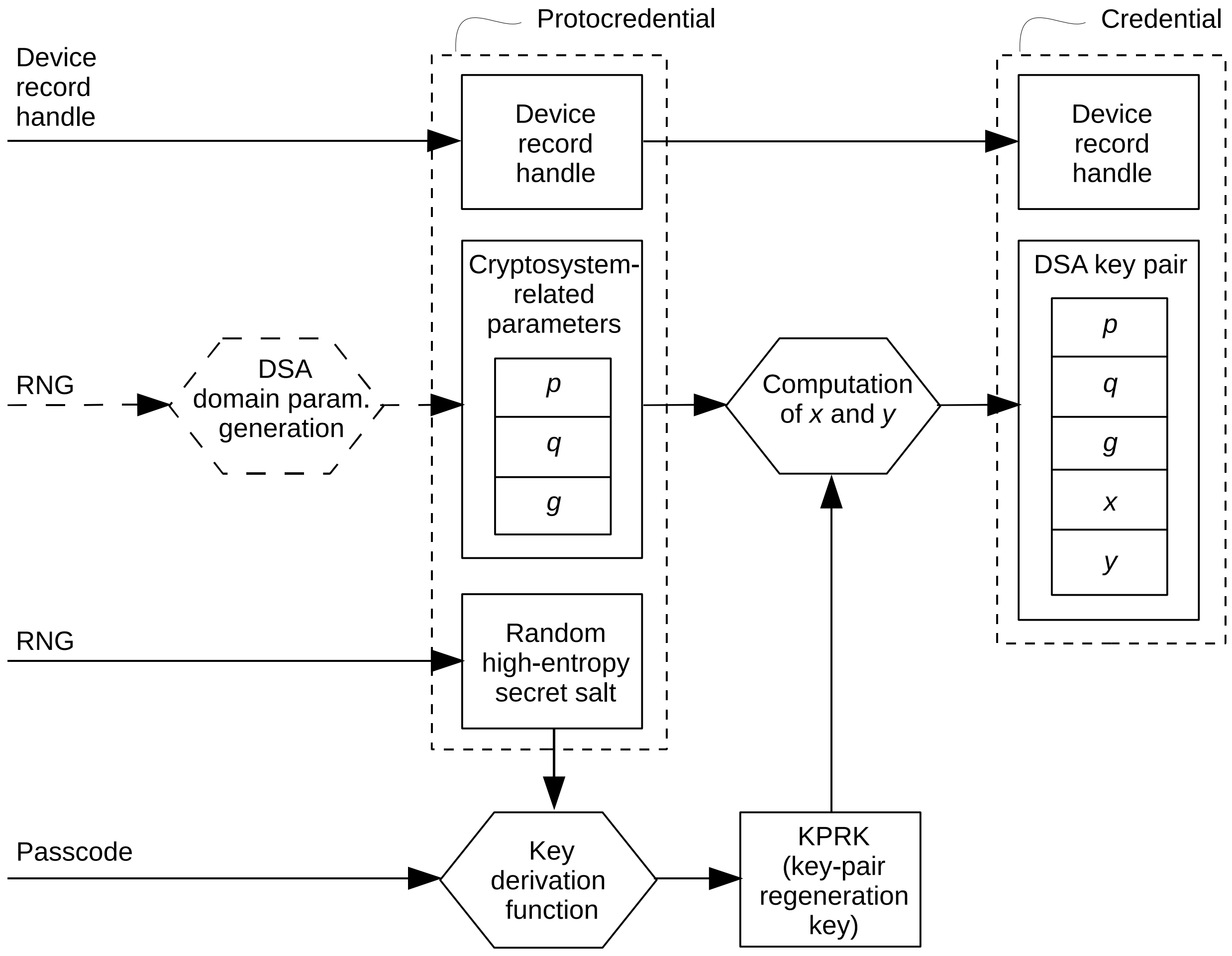


Figure 10: Generation of a DSA protocredential and its corresponding credential