Techniques for Implementing Derived Credentials

Francisco Corella, PhD
fcorella@pomcor.com

Karen Lewison, MD
kplewison@pomcor.com

Revised September 13, 2012

Abstract

The requirement of a Government-wide means of authenticating federal employees and contractors stated by HSPD-12 is traditionally met by PIV smartcards in civilian agencies and CAC smartcards in the Department of Defense. But there are substantial obstacles to using a smartcard for authentication when an information system is accessed via a mobile device. NIST is investigating alternative authentication methods that rely on *derived credentials* not contained in smartcards. In this paper we propose three techniques that can facilitate the implementation, deployment and use of derived credentials. The first technique eliminates the administrative cost that would be incurred by having to issue certificates to the mobile devices owned by users in addition to issuing certificates to the users themselves, by dispensing with client certificates. The second technique makes it possible to provide two- and three-factor authentication on a mobile device by using a PIN and/or a biometric sample to regenerate a key pair, obviating the need for tamper-resistant storage, which is not generally available on today's mobile devices. The third technique relieves developers from having to incorporate cryptographic and/or biometric functionality into mobile apps by outsourcing authentication to a prover black box and a verifier black box. These techniques would allow federal agencies to develop mobile apps quickly and inexpensively while complying with the authentication requirements of HSPD-12.

1 Introduction

In 2004, Homeland Security Presidential Directive No. 12 (HSPD-12) called for a "Government-wide standard for secure and reliable forms of identification issued by the Federal Government to its employees and contractors." In response, NIST developed a set of *Personal Identity Verification (PIV)* standards [1] specifying the PIV smart card, used both for physical access to buildings and for logical access (i.e. user authentication) to information systems.

When used for logical access, a PIV card is typically inserted into a card reader connected to a desktop or laptop computer. But the government has embraced mobile computing [2], and traditional card readers cannot be plugged into smart phones or tablets. As discussed in [3], there are two ways to authenticate a user of a mobile device in a manner compliant with HSPD-12.

One way is to use wireless communication for interaction between the mobile device and a PIV card. A traditional contact card can be placed in a battery-powered sled that communicates with the mobile device via WiFi or Bluetooth; but this requires the user to carry three different pieces of equipment. A contactless card can communicate directly with a mobile device via Near Field Communication (NFC); but mobile devices such as the iPhone are nor equipped with NFC today.

The other way is to use a *derived credential* installed in the mobile device itself. A *derived credential* is defined in the recently revised Electronic Authentication Guideline [4] 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." The derived credential in the mobile device would be issued upon presentation of a PIV card. Derived credentials are in the process of being standardized by NIST. The draft of Federal Information Processing Standard (FIPS) 201-2 [5], which updates the core PIV standard, refers to a future publication SP 800-157 that will provide guidelines on PIV derived credentials.

Derived credentials on mobile devices based on a traditional public key infrastructure present several challenges. One challenge arises from the administrative cost that would be incurred by having to issue certificates to the mobile devices owned by users in addition to issuing certificates to the users themselves; the number of certificates would double, and the complexity of the public key infrastructure (PKI) would be increased by having to keep track of the relationship between device certificates and user certificates. Another challenge is the lack of tamperproof storage for credentials and biometric data in mobile devices. Yet another challenge is the complexity of cryptographic and biometric processing, which stands in the way of agile development of mobile apps.

In this whitepaper we propose three techniques, described in the next three sections, that could simplify the implementation and facilitate the deployment and use of PIV derived credentials on mobile devices. The three techniques can be combined, as described in a companion whitepaper [6] concerned with mobile authentication in general terms (not specifically with authentication of federal employees and contractors); but each technique is of independent interest and can be used by itself, except that the technique of Section 3 is best used in conjunction with the technique of Section 2.

The proposed techniques are motivated by the desire to avoid having to use a PIV card for authentication on a mobile device. But they also make it possible to use derived credentials on laptops and desktops, using browser extensions. Use of the techniques on laptops and desktops is discussed in the companion whitepaper [6, §5].

2 Public Key Cryptography without Certificates

The first technique dispenses with client certificates by using a directory rather than a certificate to provide the user's identity. Figures 1 and 2 illustrate the technique in two different use cases. In Figure 1, the user interacts with an application that has a native front-end running on the mobile device and an online back-end. In Figure 2 the user interacts with a web-based application via a web browser; in this case the browser plays the role of application front-end, and we shall refer to the web-based application itself as

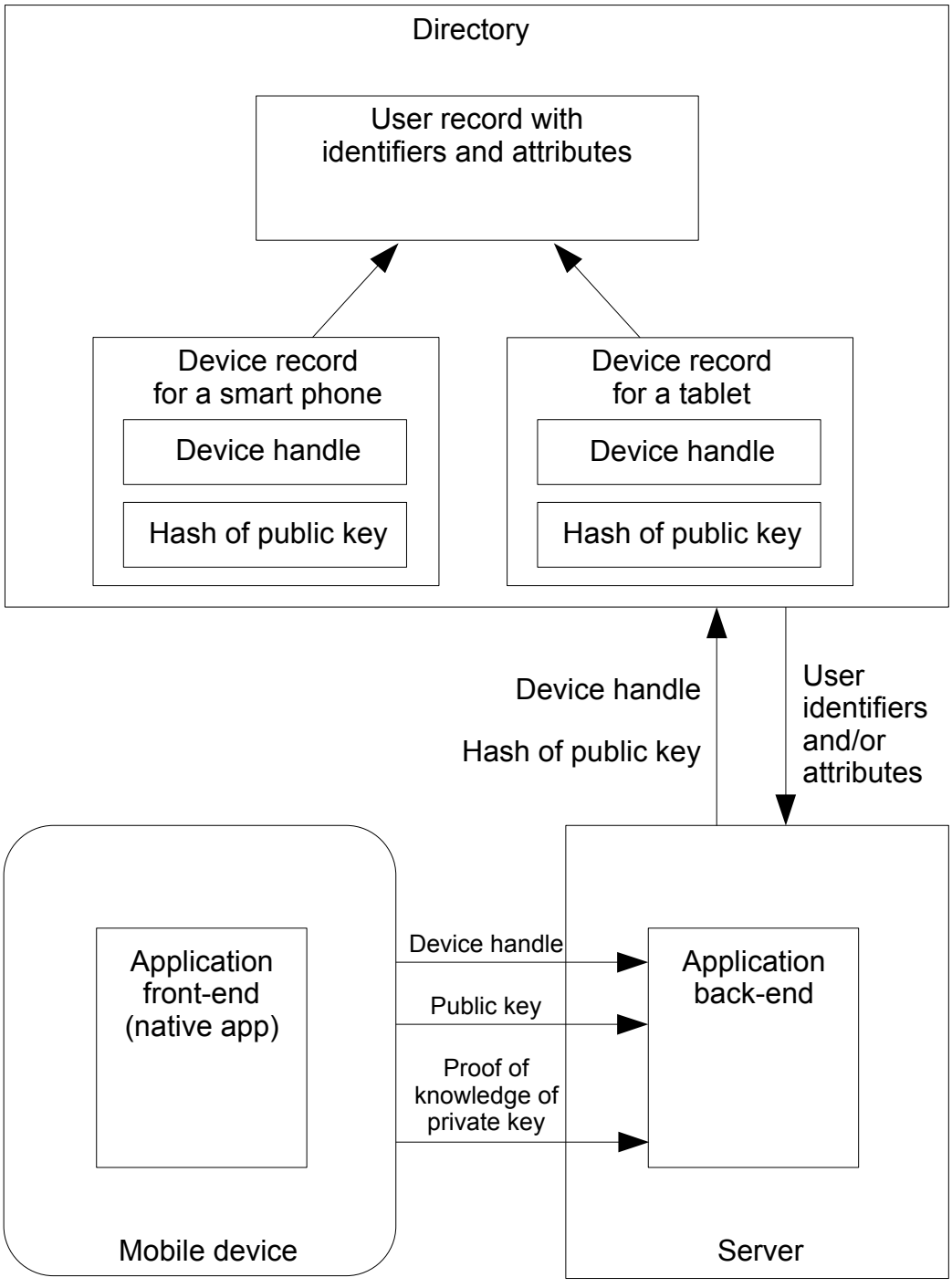

Figure 1. Authentication without certificates to an application with a native front-end

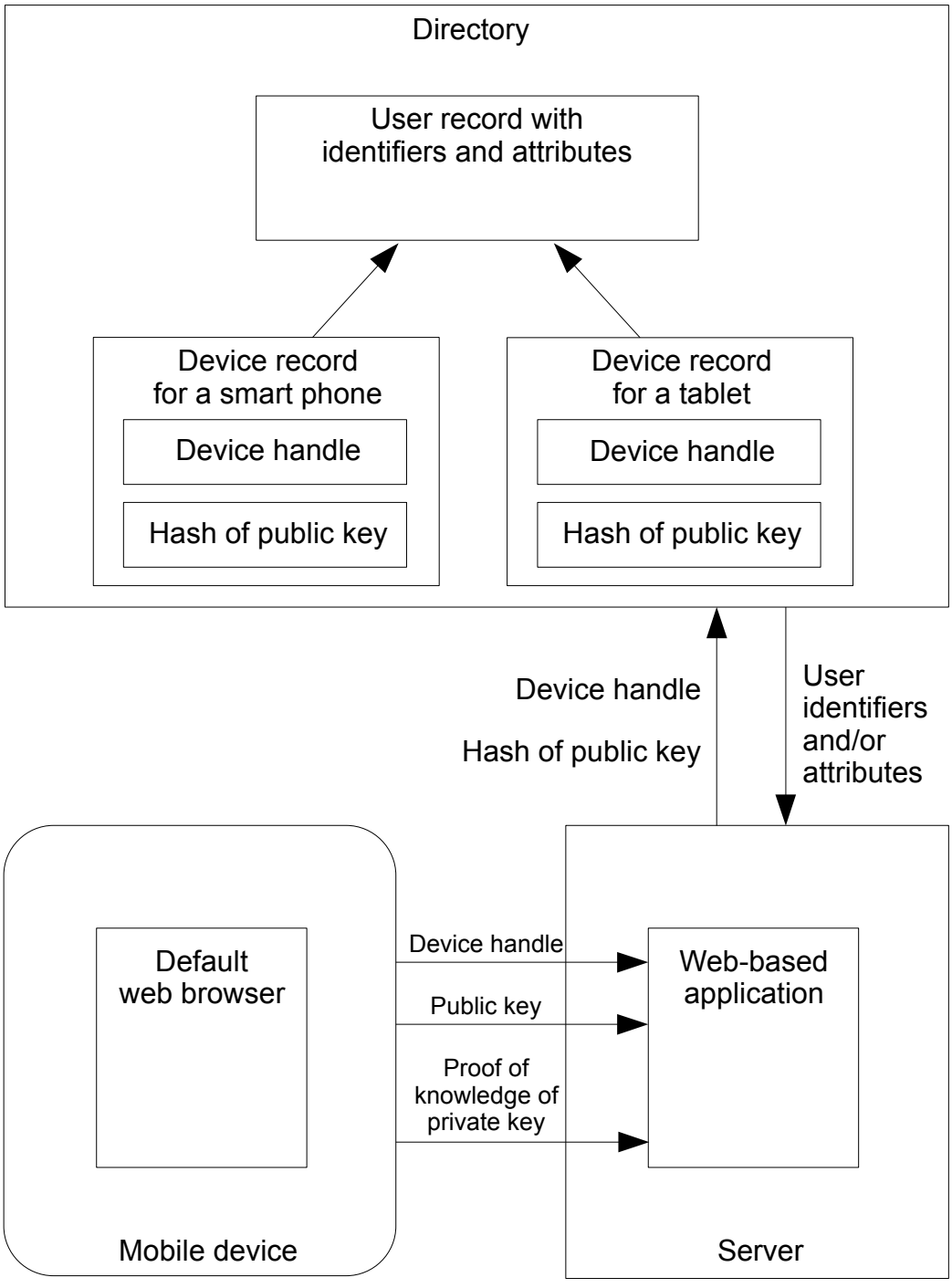

Figure 2. Authentication without certificates to a web-based application

its own back-end. In both cases the user authenticates to the application back-end. (There is no point in authenticating the user to the front-end, which resides in a non-tamperproof device and can therefore be modified by the user.)

The mobile device contains a key pair (not shown in the figures) pertaining to a public key cryptosystem, such as RSA, DSA, or ECDSA. The key pair may be kept in the browser of Figure 1, or in the native front-end of Figure 2, or elsewhere in the mobile device. In Section 4 below we propose placing the key pair in a specialized native app. It could also reside in the operating system. The key pair is used for authenticating the device, thus playing a role similar to that of a Card Authentication Key (CAK) in a PIV card, and indirectly authenticating the user as the owner of the device.[1]

The directory can be implemented in many different ways, e.g. as an LDAP directory, a relational database, or a NoSQL database. It contains records (or entries), corresponding to users and to devices owned by users. Each device record refers to the user record of the owner of the device. In a relational database the reference is expressed by a field of the device record, whose value equals the primary database key[2] of the user record. In an LDAP directory the reference is expressed by the fact that the device record is a child of the user record in the directory information tree. Multiple device records may refer to the same user record, since a user may have, e.g., a smart phone and a tablet, as illustrated in the figures.

Each device record contains a hash of the public key component of the key pair that the device uses for authentication, computed with a cryptographic hash function such as SHA-256, and a *device handle*, which uniquely identifies the device record.[3]

Each user record may contain user identifiers found in a PIV card such as the user's name and/or the Federal Agency Smart Credential Number (FASC-N) and/or the PIV Universally Unique IDentifier (UUID), as well as additional attributes not present in a PIV card, such as security clearance level and/or emergency responder capabilities. Derived credentials can thus provide both the authentication functionality of PIV cards and the attribute retrieval functionality of the Backend Attribute Exchange (BAE) protocol [7].

The user authentication process comprises two steps. In step 1, the mobile device sends the device handle and the public key to the application back-end, and demonstrates knowledge of the private key. In step 2, the application back-end locates the device record using the device handle, computes the hash of the public key received from the device and verifies that it coincides with the hash stored in the record, locates the user record

[1] We say that a device is owned by a user if it is used by the user, whether the user has purchased the device or a federal agency or contractor has purchased the device and assigned it to the user. The user may have actually purchased the device if the agency or contractor has a Bring-Your-Own-Device (BYOD) policy.

[2] Not to be confused with a cryptographic key.

[3] Notice that the hash of the public key also identifies the device record uniquely. But if the key pair is protected by a passcode for two-factor authentication and the back-end wants to limit the number of authentication attempts in order to thwart a passcode-guessing attack, the back-end must know what device record the authentication attempts pertain to. In that case the device record must be identified without knowing the hash of the public key, hence the need for a device handle. Notice also that the device handle is specific to the directory, and created at enrollment time. In cases where the mobile device is used to access applications pertaining to different organizations with different directories, there should be different device handles for different directories, for the sake of privacy.

referenced by the device record, and obtains user identifiers and/or attributes from the user record.

Step 1 may be carried out by a browser, a native front-end, a specialized application as described below in Section 4, or the operating system.

The details of step 2 depend on the type of directory being used. If the directory is implemented as a relational database, a single query performing a join of the table of device records and the table of user records can retrieve the desired attributes from the user record. If the diretory is an LDAP directory, LDAP protocol can be used to access the device record and then access the user record as the parent of the device record.

3 An Effective Cryptographic Alternative to Tamper Resistance

The second technique obviates the need to store derived credentials or biometric data in a tamper-resistant module when using multifactor authentication.

A PIV card provides one-factor authentication, two-factor authentication where the second factor is knowledge of a PIN, two-factor authentication where the second factor is a biometric sample, and three-factor authentication with both a PIN and a biometric sample. In two- and three-factor authentication the additional factors activate the card, enabling the use of a *PIV Authentication* private key stored in the card.

Implementing two- or three-factor authentication by using the additional factors to enable the use of the first factor has the important advantage that the relying party only has to verify the first factor.

3.1 An Alternative to Tamper Resistance for Authentication with PIN and Key Pair

Two-factor authentication with a PIV card and a PIN relies on the PIV card being tamper resistant. FIPS 201-2 [5] specifies that private keys must be stored in a cryptographic module that satisfies the tamper-resistance requirements of FIPS 140-2 level 3 [8]. Without tamper resistance, an attacker who possesses the card (the first factor) can extract the PIV Authentication private key and use it without knowing the PIN (the second factor), defeating the purpose of two-factor authentication.

But today's mobile devices are not tamper resistant, and they do not generally contain tamper resistant modules.[4] Therefore if two-factor authentication on a mobile device is to be accomplished by using a passcode to enable the use of a key pair, an alternative

[4]Smart phones equipped with Near-Field Communication (NFC) have a "secure element" that is sometimes described as being tamper resistant. But the level of tamper resistance is not specified, and, to our knowledge, none of the cryptographic modules certified by NIST as tamper resistant (FIPS 140-2 physical level 3 or 4) [9] is an NFC security element. Furthermore, the NFC architecture requires the secure element to be accessible to an online Trusted Service Manager (TSM). And iPhones are not equipped with NFC at this time. Google Wallet originally used an NFC secure element to store credit card numbers, but it now stores them online.

method must be found for protecting the private key component of the key pair. Two known methods come to mind, but neither is effective on mobile devices.

The first known method consists of encrypting the private key with a key-encryption key derived from a passcode (such as a PIN, password or passphrase). But without tamper resistance, an attacker can mount an offline guessing attack against the passcode. To withstand such an attack, the passcode must have high entropy. A high-entropy PIN is impractical because the user may not be able to remember it. A high entropy password or passphrase is impractical because it is difficult to enter it on the tiny touch-screen keyboard of a smart phone, which requires keyboard switching to enter different classes of characters; and it is not secure because the keyboard provides typing feedback by prominently displaying characters as they are being typed, which makes the password or passphrase easily observable over the user's shoulder, nullifying the security provided by a password input box that echoes characters as dots.

The second known method is to derive the key-encryption key from both a passcode and a hardware key. The hardware key is hard-wired into cryptographic hardware and cannot be extracted by a casual attacker who does not have the capability or does not make the effort of probing the hardware. However, even if it cannot be extracted, the hardware key can be used by the cryptographic hardware in which it is embedded. The iOS operating system uses a hardware key in combination with a passcode to derive encryption keys for data protection, and the data protection mechanism has been defeated [10, 11].

We propose a third method, applicable when no client certificate needs to be kept in the mobile device. The method is thus best used in conjunction with the technique of Section 2, which dispenses with client certificates altogether. We propose to use an RSA key pair [12, §8.2] as a derived credential. Use of the RSA key pair is enabled by a PIN. But instead of encrypting the private key component of the RSA key pair under a key-encryption key derived from the PIN, the complete key pair is *regenerated* from the PIN. This has the effect that (almost) every PIN yields a well-formed key pair, making it impossible to test PINs offline.

Generating an RSA key pair involves choosing the prime factors p and q of the RSA modulus $n = pq$, and encryption and decryption exponents e and d such that $ed \equiv 1 \pmod{\lambda}$, where λ is the least common multiple of $p-1$ and $q-1$. For simplicity, e and d may be chosen such that $ed \equiv 1 \pmod{\phi}$ where $\phi = (p-1)(q-1)$, avoiding the computation of λ. The private key is d and the public key is (n, e). (Therefore e is also called the public exponent and d the private, or secret, exponent.) Usually a small encryption exponent such as $e = 3$ or $e = 65537$ is chosen first, and the decryption exponent is computed as the unique integer d, $1 < d < \phi$, such that $ed \equiv 1 \pmod{\phi}$, which exists as long as e and ϕ are relatively prime. Instead, we derive the decryption exponent d from the PIN and compute the encryption exponent as the unique integer e, $1 < e < \phi$, such that $ed \equiv 1 \pmod{\phi}$, which exists as long as d and ϕ are relatively prime.

The decryption exponent d should not be too small, because several effective algorithms have been shown to be able to compute d' such that $1 < d' < \lambda$ and $ed' \equiv 1 \pmod{\lambda}$ from the public key (n, e), if d' is small; d' will be small if d is small, because $d' = d \bmod \lambda$. Specifically, those algorithms [13, 14, 15] are able to compute d' if the bit-length of d' is less than bounds ranging from 25% to 29% of the bit-length of the modulus. (The authors

of [15] have conjectured that their approach could eventually be improved to work when the bit-length of d' is up to 50% of the bit-length of the modulus.)

The decryption exponent d is derived from the PIN, from a random seed s of sufficient length (e.g. 256 bits), from ϕ, and from the set S of *small prime factors* of ϕ, a prime factor being deemed to be small if it is less than 100. The process of deriving d, using a process variable D, is as follows:

1. A randomized extended hash of the PIN with random seed s, of same byte length as the modulus, is computed and assigned to D. (A randomized extended hash of any byte length can be computed using the `P_hash` mechanism of the TLS protocol [16, §5], which uses a hash function such as SHA-256 [17], the HMAC mechanism [18], a secret, which in this case is the PIN, and a random seed, in this case s, to produce a cryptographic hash of the secret and the seed of the desired length.)

2. If D if divisible by one or more elements of S, D is repeatedly divided by such elements, the result of each division being assigned to D, until no such elements remain.

The decryption exponent d is the value of D at the end of the process. If the resulting d is not relatively prime with ϕ, we start over, choosing different prime factors p, q for the modulus. The probability of d and ϕ not being relatively prime after any common prime factors less than 100 have been removed from d is similar to the probability of two random numbers having common prime factors greater than 100. We show in Appendix A that this probability is less than 0.2%. The bit-length of $d' = d \bmod \lambda$ may be less than the bit-length of the modulus, but, with high probability, by not more than a few bits.

The key pair is *regenerated* before each use. To make it possible to regenerate the key pair, the prime factors p and q and the random seed s are retained in the device when the key pair is generated for the first time.[5] The set S of small prime factors of ϕ is also retained, to facilitate the process. The decryption exponent is then calculated from the PIN, s, $\phi = (p-1)(q-1)$, and S by the same process used when the key pair is generated for the first time; the encryption exponent is calculated by the extended Euclidean algorithm [12, Algorithm 2.107] as the unique integer e, $1 < e < \phi$, such that $ed \equiv 1 \pmod{\phi}$; and the modulus as $n = pq$.

Any PIN produces a well-formed RSA key pair, so PINs cannot be tested offline using only data extracted from the mobile device. An attacker who tampers with the device and is able to read its persistent memory but does not know the public key (nor the hash of the public key stored in the directory according to Section 2 above) can only test a PIN by attempting an online authentication with the key pair produced by the PIN. Thus the attacker is not in a better position than an attacker who simply tries PINs without tampering with the mobile device. If the application back-end limits the number of failed authentications, e.g., to five, a 6-digit PIN should provide a reasonable amount of entropy.

The reader may have noticed the unusual requirement that the attacker must not know the RSA public key, nor a hash of the public key. Traditionally, in public key cryptography the public key is, naturally, *public*. There are good reasons for this. The public key of

[5] Notice that p and q can be computed from the key pair [12, §8.2.2(i)], so storing p and q is no less secure than storing the key pair, or the private key and a certificate containing the public key.

an encryption key pair must be public so that anybody can send an encrypted message to the owner of the key pair. The public key of a signature key pair must be public so that anybody can verify a signature produced with the private key. However, when a key pair is used for authentication there is no need for the public key to actually be public.

If the technique of Section 2 is used, an employee of a federal agency authenticates to the agency by demonstrating knowledge of the private key associated with a public key whose hash is stored in the agency's directory. The hash can be kept secret, just like the salted hash of the user's password would be kept secret if the user authenticated with a password. Even if an employee of an agency authenticates to a second agency, the hash does not need to be public. After the user sends the public key to the second agency and demonstrates knowledge of the private key, the second agency computes the hash of the public key and uses it to request the user's identity and/or attributes from the employee's own agency. The hash is then known to both agencies, but it need not be public.

Although the hash of the public key should be kept secret, a security breach that reveals the hash to an attacker is much less serious than one that gives access to a password hash. To exploit the breach, the attacker would also have to gain physical access to the mobile device, tamper with it and read its persistent memory, before being able to mount an offline attack against the PIN.[6]

3.2 Biometric Authentication with Derived Credentials

An alternative to tamper resistance is also required for using biometric authentication to enable the use of a derived credential.

Biometric authentication with a PIV card relies on the assumption that the biometric sample presented by the user is a live sample. This is a sound assumption if authentication is supervised by a human attendant, and may be a plausible assumption for unattended authentication using sophisticated sensors. The assumption, however, will not hold for a biometric sample such as an iris image collected by the one sensor found today on a mobile device, the device camera.

Since liveness of the sample cannot be verified, biometric authentication on mobile devices must rely instead on the assumption that the attacker does not have a biometric sample from the user. Like liveness, this assumption holds in some cases but not others. Clearly it does not hold for fingerprint biometrics, since fingerprints are certain to be found on the mobile device. It holds for iris biometrics if the device has been accidentally lost, then found by an attacker who does not know the user; and it may hold in the case of a casual attacker who steals the mobile device but does not take a suitable picture of the user's iris.[7]

Biometric authentication on a PIV card (either off-card or on-card) is achieved by comparing a biometric sample to a biometric template stored in the card. This method does

[6] Notice that d can be computed from p, q and e. Therefore it is important to store the hash of the public key in the directory rather than the public key itself, so that an attacker who breaks into the directory in addition to reading the persistent memory of the mobile device must mount an offline attack against the PIN to obtain d, rather than just computing d from p, q and e.

[7] It is assumed that the biometric enrollment process takes care of erasing the biometric sample used for enrollment.

not work for derived credentials for two reasons. First, due to the absence of tamper-resistant storage for storing the template, an attacker with physical access to the device could obtain the template, and may be able to derive a biometric sample from the template [19, 20], invalidating the biometric secrecy assumption. Second, due to the absence of tamper-resistant storage for the authentication software, program logic requiring a successful biometric match to enable use of the derived credential could be bypassed by the user.

In [6, §2.4] we have proposed an alternative biometric authentication methology, well suited for the purpose of enabling a derived credential. Several methods have been described for deriving a key, sometimes called a *biometric key*, from a genuine biometric sample and an auxiliary string [21, 22, 23, 24]. The auxiliary string is itself derived from an original biometric input and a random string. We propose to store such an auxiliary string, which does not require protection, instead of a biometric template, and to use the biometric key produced by a successful biometric authentication to regenerate the derived credential as explained above in Section 3.1, the biometric key playing the role that was played by the PIN. For further protection of the user's biometric and to achieve three-factor authentication, a randomized extended hash of a PIN is x-ored with the auxiliary string.

More details can be found in [6, §2.4].

4 Authentication Outsourcing Protocol

The third technique facilitates the implementation of derived credentials by removing the need for developers to incorporate cryptographic or biometric functionality into mobile apps.

This is accomplished using an *authentication outsourcing protocol*, which outsources the task of proving knowledge of a private key to a Prover Black Box (PBB), and the task of verifying the proof to a Verifier Black Box (VBB). The PBB contains the derived credential, including the device handle that identifies the device record in the directory. (We consider the device handle to be part of the credential just like a username is part of a username-and-password credential.) For two-factor authentication the PBB takes care of obtaining a passcode and using it to enable a derived credential; developers are thus relieved from the burden of protecting the passcode against offline guessing attacks. For three-factor authentication, the PBB takes care of obtaining a passcode and a biometric sample to enable the credential.

In this section we show how the outsourcing protocol can be used in conjunction with the technique of Section 2, in the case where the VBB is implemented as a generic server appliance that has no knowledge of the directory. In that case the role of the VBB is limited to verifying that the PBB has knowledge of the private key. Alternatively, the VBB could access the directory, check that the hash of the public key matches the hash in the device record, and obtain data from the user record, relieving the application from the need to perform these tasks. This alternative is discussed in [6, §2.2 and §3.3]. It is also possible for the PBB to authenticate to the VBB in a traditional way using a client certificate.

The PBB is most conveniently implemented as a separate native app using interapp communication facilities available in mobile operating systems such as iOS and Android.

We describe the protocol in that case in Section 4.3, after describing the interapp communication facilities in Section 4.1 and possible improvements to those facilities in Section 4.2. The PBB can also be embedded in the mobile app to which the user authenticates, or in a browser (as a browser extension), or in the operating system of the mobile device. Those cases are discussed in the companion whitepaper [6]. Implementing the PBB as a browser extension makes it possible to use the protocol on laptops and desktops, as described in [6, §5].

The PBB and VBB as described here provide a form of cryptographic agility: they can be replaced with a different PBB and a different VBB to support different types of derived credentials, without modifying the application code.

4.1 Interapp Communication

An interapp communication facility, provided at least by iOS and Android, allows native applications to send messages to each other. A native application can use a special kind of URL, which we shall call a *native URL*, to send a message via the operating system of the mobile device to another native application residing on the same device. Both iOS [25] and Android [26] implement native URLs using *custom schemes*, a custom scheme identifying the native application to which the message is to be sent. As in an HTTP GET request, the URL is both the address and the contents of the message. The URL identifies to the operating system the target application to which the message is being sent, and the operating system delivers the URL to the target application, which parses it to extract parameters, usually encoded in a so-called *query-string* portion of the URL. An important aspect of this communication method is how it is used by a browser, which is itself a native application. A URL with scheme `http` or `https` is delivered to the default browser and causes the default browser to send an HTTP GET request targetting the URL over a network. Furthermore, when a browser receives an HTTP redirection response to a request that it has sent over a network, if the URL specified by the redirection is a native URL, the browser sends that URL to the native application identified by the URL, via the operating system.

As currently implemented by iOS and Android, interapp communication might allow a malicious application to receive a message intended for a different application, because a custom scheme is registered at run time with the operating system of the device, and a native application is free to register any custom scheme it wants. System behavior is undefined if a malicious application registers a scheme previously registered by a legitimate application, or one that a legitimate application will register later. The protocol is therefore best suited for the case where mobile devices are deployed by an organization, such as a federal agency, that exerts control over what applications are installed on mobile devices, so that installed applications can be trusted to not impersonate each other. This is not incompatible with a Bring-Your-Own-Device (BYOD) program that allows employees to use their own devices, because some Mobile Device Managers (MDMs) provide app inventory control for BYOD devices.

4.2 Suggested Interapp Communication Improvements

While the interapp communication facilities of iOS and Android can be used as they are by the authentication outsourcing protocol, provided that installed applications are trusted, it would be good for cybersecurity if they were improved to eliminate the risk of application impersonation. The US government could bring about such an improvement by asking Apple, Google and other mobile operating system providers to strengthen the security of their operating systems.

Interapp communication could be improved as follows. Instead of using a different custom scheme for each application, a single scheme, e.g. `app`, could be registered with IANA[8] for the purpose of building native URLs. A native URL could then consist of that scheme, followed as in a web URL by "`://`", a DNS domain name, a path and a query string. An example of a native URL would then be:

```
app://example.com/path/to/endpoint?name1=value1&name2=value2
```

The DNS domain name would belong to the developer[9] of the application, and would be registered with the operating system of the mobile device at installation time (rather than at run time) for use in native URLs of application endpoints. The operating system would verify that the developer owns the domain name in one of two ways:

- By verifying that the application is downloaded from the named domain, or from a broader domain, over a TLS connection, using a server certificate chain rooted in a CA trusted by the operating system, or
- By verifying that the code is signed with a private key whose corresponding public key is bound to the named domain, or to a broader domain, by a certificate backed by a certificate chain rooted in a CA trusted by the operating system. (The certificate could be issued specifically for code signing, or it could be a TLS server certificate.)

Once these improvements have been made, it will be possible to use the authentication outsourcing protocol securely even in cases where not all installed applications can be trusted.

4.3 Outsourcing Protocol Flows

When describing the outsourcing protocol we shall refer to the mobile application to which the user authenticates as the *client application*.

We shall use the term *endpoint* to refer to a metaphorical "point" of a web server or native application that can receive HTTP POST requests, HTTP GET requets, or native

[8] URL schemes are supposed to be registered with IANA [27], but the custom schemes of native applications are not. (There is an informal database of iOS custom schemes [28] but it is not sanctioned by Apple or the IETF. We do not know of any database of custom schemes for Android.) Custom schemes may thus conflict with registered schemes, and two native applications may choose the same custom scheme. As mentioned above, a custom scheme conflict might allow a malicious application to receive a message intended for a legitimate application.

[9] By *developer* we mean the individual or organization who has control over the application.

URL messages. Each endpoint has a URL that does not have a query string. An HTTP POST request is addressed to the endpoint URL, with request parameters passed in the body of the request. An HTTP GET request or a native URL message is addressed to the endpoint URL augmented with a query string consisting of a question mark ("?") followed by one or more parameters separated by ampersands ("&"), each parameter being encoded as a parameter name followed by an equal sign ("=") and a parameter value.[10]

The PBB has an outsourcing endpoint where the native front-end sends an outsourcing request, and the client application has a callback endpoint where the PBB sends a reply to the outsourcing request. The VBB is implemented as a generic server appliance. It has a device-authentication endpoint where the PBB sends an authentication request, and a hash-retrieval endpoint from which the online back-end of the client application retrieves the hash of the public key.

The protocol has different flows when the client application has a native front-end and when it uses a general-purpose web browser as front-end. However the functionality of the PBB and the VBB is the same in both cases.

4.3.1 Native Front-End Flow

Figure 3 shows the protocol flow when the client application has a native front-end. The callback endpoint resides on the front-end. The protocol has eight steps indicated by circled numbers in the figure.

In step 1, the native front-end sends a message to the outsourcing endpoint of the PBB, using a native URL with the following parameters:

- The URL of the callback endpoint of the native front-end of the client application.
- The URL of the device-authentication endpoint of the VBB.
- An optional transaction ID that the client application may use internally to remember the context of the authentication transaction.

In step 2, the PBB uses the derived credential to authenticate to the VBB, after enabling use of the credential as needed with additional authentication factors provided by the user. A secure connection between the PBB and the VBB is used for the authentication process or established as a result of the authentication process. When the technique of Section 2 is used, the PBB sends the public key to the VBB and demonstrates knowledge of the private key, as illustrated in the figure. For multifactor authentication the public key needs confidentiality protection for reasons explained above in Section 3.1; several options for protecting the public key as the PBB authenticates to the VBB are described in [6, §2.1].

In step 3, the VBB generates a random high-entropy string to be used as a one-time authentication token, and creates a short-term authentication record containing the authentication token, which serves as a reference to the record, and the hash of the public key. The VBB sends the authentication token to the PBB over the secure connection established in step 2, after which it closes the connection.

[10] More accurately, an endpoint could be defined as the procedure, or event listener, that is invoked when the HTTP request or native URL message is received; the request parameters are passed to that procedure.

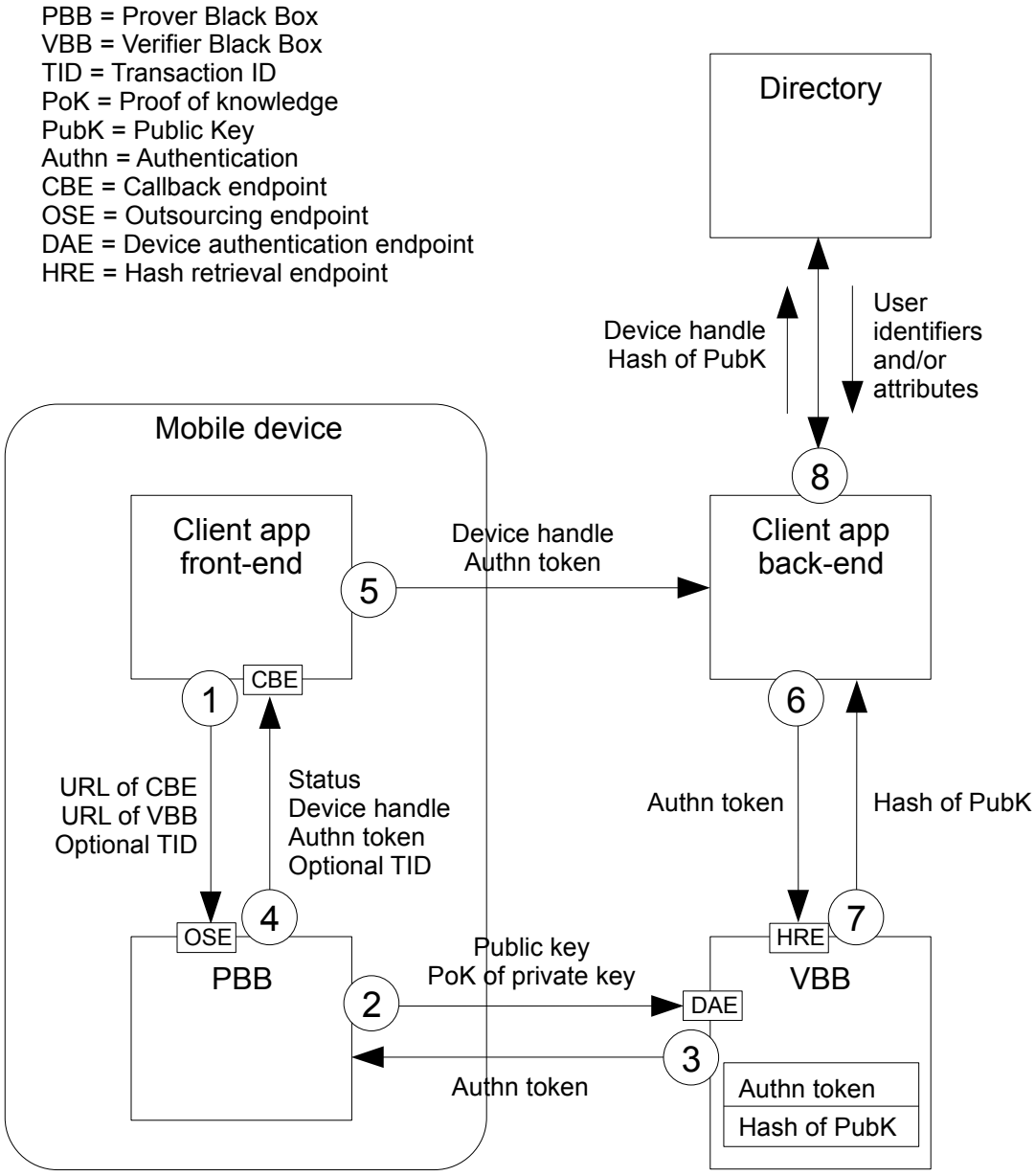

Figure 3. Authentication outsourcing protocol
when client application has a native front-end

An alternative to steps 2 and 3 that does not require the establishment of a connection can be found in [6, §2.1].

In step 4 the PBB sends the authentication token (obtained from the VBB in step 3) and the device handle (stored in the PBB as part of the credential) to the callback endpoint of the native front-end, along with a status code indicating success (assuming device authentication was successful) and the optional transaction ID, if one was received by the PBB from the native front-end in step 1.

In step 5 the native front-end authenticates to the online back-end by sending the authentication token and the device handle over a TLS connection with server authentication.

In step 6 the back-end sends the authentication token to the VBB in the body of an HTTP POST request. The request is sent over a secure connection, such as a TLS connection established from the back-end to the VBB with TLS server authentication, unless communication takes place within an intranet that is deemed secure.

In step 7 the VBB uses the authentication token to locate the authentication record, and sends the hash of the public key found in the record to the application back-end in the HTTP response to the HTTP POST request.

In step 8 the back-end uses the hash of the public key and the device handle to access the directory, verify that the hash matches the one in the device record, and obtain user identifiers and attributes from the user record, as explained above in Section 2.

4.3.2 Web-Based Flow

Figure 4 shows the protocol flow when the client application uses the default web browser of the mobile device as front-end for user interactions. The callback endpoint is now located on the online client application, and its URL is a web URL that uses the *https* scheme rather than a native URL that uses a custom scheme.

Whereas all native applications installed in the mobile device are trusted, web-based applications cannot all be trusted, since the browser provides access to any application on the world-wide web. The outsourcing protocol must therefore be hardened to cope with possibly malicious web-based client applications.

The user interacts with the client application via the browser, and the protocol is initiated as a result of an HTTP request sent by the browser to the online client, e.g. an HTTP request that requires authentication and is not accompanied by a login session cookie.

In step 1, the client issues an HTTP redirection (302) response to the HTTP request, redirecting the browser to a native URL that targets the outsourcing endpoint of the PBB, with the following parameters:

- The URL of the callback endpoint of the online client.
- The URL of the device-authentication endpoint of the VBB.
- A random high-entropy transaction ID, which identifies the authentication transaction.

The HTTP response also sets a cookie in the browser containing the same transaction ID.

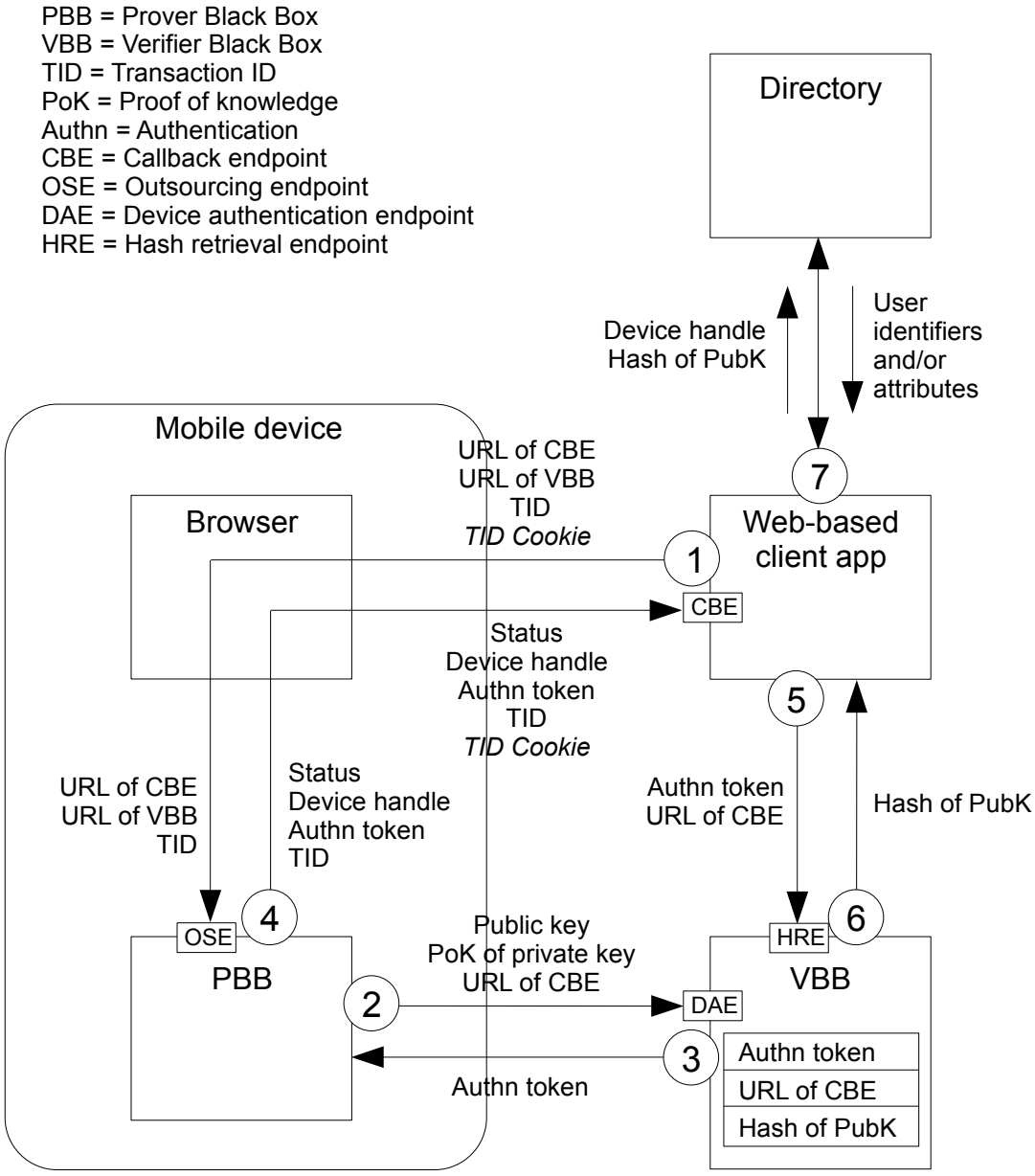

Figure 4. Authentication outsourcing protocol
when client application is web-based

Notice that the transaction ID, which was optional in the protocol flow of Section 4.3.1, is compulsory in this flow. This is because it now plays a security role in addition to helping the client remember the context of the authentication transaction. The transaction ID is sent to the PBB as a parameter and set in the browser as a cookie. In step 4, the client will receive a transaction ID from the PBB as a parameter and another transaction ID from the browser as a cookie. The client will then verify that the two transaction IDs are the same. This prevents a login-as-attacker attack described below in connection with step 4.

In step 2, the PBB authenticates the device to the VBB as explained above in Section 4.3.1 in connection with the native front-end flow, and establishes a secure connection to the VBB. Then the PBB sends the URL of the client callback endpoint to the VBB over the secure connection. The reason for sending this URL will become apparent in connection with step 5.

In step 3, the VBB generates a one-time authentication token and creates a short-term authentication record as in the the native front-end flow of Section 4.3.1. This time, in addition to storing the authentication token and the hash of the public key in the authentication record, the VBB also stores in the record the URL of the client callback endpoint. The reason for this will become apparent in connection with step 5. The VBB sends the authentication token to the PBB over the secure connection that was established in step 2, after which it closes the connection.

In step 4, the PBB constructs a client callback URL by augmenting the URL of the callback endpoint received from the client with a query-string comprising the following parameters:

- A status indicating that the outsourced device authentication succeeded (assuming that the PBB did obtain an authentication token from the VBB).
- The device handle, present in the PBB as part of the derived credential.
- The authentication token.
- The transaction ID received from the client in step 1.

The PBB asks the operating system of the mobile device to deliver the client callback URL to its destination. The client callback URL is a web URL, which the operating system delivers to the default browser. The browser sends an HTTP GET request to the URL received from the operating system, adding a cookie HTTP header to the request that contains the transaction ID set as a cookie in step 1.

In step 5, the client checks that the status parameter indicates success and the transaction ID parameter in the callback URL coincides with the transaction ID in the cookie header. If either check fails, the authentication fails.

The transaction ID check prevents an insider attack, by an attacker who is a legitimate user of the client application, in which the attacker would cause a victim user to submit an authentication token for the attacker's account to the client application, causing the victim user to authenticate as the attacker. The attacker could be an employee of a federal agency and the victim could be the head of the agency. The incorrect authentication could cause the head of the agency to log in to an account owned by the attacker without realizing it,

and to enter highly classified data into the attacker's account, making it available to the attacker. This attack is similar to the Login CSRF attack of [29]. Without the transaction ID check the attack could be easily mounted by using a client callback URL including the attacker's authentication token as the URL of a link in a web page controlled by the attacker, luring the victim into the page, and tricking the victim into clicking on the link. The transaction ID check prevents the attack because the attacker cannot set a cookie in the victim's browser that will be sent to the client application.

Still within step 5, if the status check and the transaction ID check are successful, the client sends the authentication token to the VBB in the body of an HTTP POST request (over a secure connection as in step 6 of the native front-end flow), together with the URL of its callback endpoint, which is used here as an identifier of the client application. By sending the URL of its callback endpoint, the client application self-asserts its own identity. The reason for this will become apparent momentarily.

In step 6 the VBB uses the authentication token to locate the authentication record, obtaining the hash of the public key and the callback endpoint URL stored in the authentication record.

The authentication fails if the callback endpoint URL in the record differs from the one received from the client application in step 5. This prevents another insider attack, where an attacker, who is a user, sets up a malicious web-based application and lures a victim user into authenticating to the malicious application with a mobile device; the malicious application obtains an authentication token pertaining to the victim's device in step 4 but does not perform step 5; the attacker authenticates to the legitimate application from the attacker's own mobile device (or computer simulating a mobile device) but, in step 4, substitutes the token pertaining to the victim's device obtained by the malicious application for the token pertaining to the attacker's own device, and thus authenticates as the victim. Checking that the URL of the callback endpoint sent by the client application in step 5 coincides with the one stored in the record prevents the attacker from substituting an authentication token intended for the malicious application when authenticating to the legitimate application.

Still within step 6, if the callback endpoint URLs agree, the VBB deletes the authentication record, and returns the hash of the public key to the client application in the HTTP response to the HTTP POST request received in step 5.

In step 7 the back-end uses the hash of the public key and the device handle to access the directory, verify that the hash matches the one in the device record, and obtain user identifiers and/or attributes from the user record, as explained above in Section 2.

5 Conclusion

We have proposed three techniques that can facilitate the implementation, deployment and use of derived credentials in mobile devices. The first technique eliminates the financial and performance costs associated with verifying client certificates and their certificate chains on mobile devices, by dispensing with client certificates altogether. The second technique makes it possible to provide two- and three-factor authentication on a mobile device by using

a PIN and/or a biometric sample to *regenerate* a key pair, obviating the need for tamper-resistant storage, which is not generally available on today's mobile devices. The third technique relieves developers from having to incorporate cryptographic and/or biometric functionality into mobile apps by outsourcing authentication to a prover black box and a verifier black box.

These techniques would allow federal agencies to develop mobile apps quickly and inexpensively while complying with the authentication requirements of HSPD-12.

Appendix

A Probability of two Random Numbers Having a Common Prime Factor Greater than 100

The probability of a random number chosen from an interval $[1, N]$ being divisible by a positive number p is less than or equal to $1/p$. Therefore the probability that two random numbers (chosen from the same such interval) have a common prime factor belonging to a set P of prime numbers is less than

$$\sum_{p \in P} \frac{1}{p^2}.$$

In particular, if p_i is the i-th prime number, the probability that two random numbers having a common prime factor greater than the j-th prime number is less than

$$\sum_{i>j} \frac{1}{p_i^2} = \sum_{i>0} \frac{1}{p_i^2} - \sum_{i<=j} \frac{1}{p_i^2}.$$

The value of

$$\sum_{i>0} \frac{1}{p_i^2}$$

is the value of the prime zeta function for the argument 2, which is 0.452247... [30]. There are 25 prime numbers less than 100, and

$$\sum_{i<=25} \frac{1}{p_i^2} = 0.450428....$$

Therefore the probability that two random numbers have a common prime factor greater than 100 is less than $0.452248 - 0.450428 = 0.0018 \approx 0.2\%$.

References

[1] NIST. PIV Standards & Supporting Documents. http://csrc.nist.gov/groups/SNS/piv/standards.html.

[2] J. Nicholas Hoover. Going Mobile. InformationWeek Government, February 2012.

[3] 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.

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

[5] NIST. Personal Identity Verification (PIV) of Federal Employees and Contractors REVISED DRAFT, July 2012. http://csrc.nist.gov/publications/drafts/fips201-2/draft_nist-fips-201-2_revised.pdf.

[6] F. Corella and K. Lewison. Strong and Convenient Multi-Factor Authentication on Mobile Devices. Pomcor whitepaper. http://pomcor.com/whitepapers/MobileAuthentication.pdf.

[7] ICAM. Backend Attribute Exchange (BAE) v2.0 Overview, January 2012. http://www.idmanagement.gov/documents/BAE_v2_Overview_Document_Final_v1.0.0.pdf.

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

[9] NIST. FIPS 140-1 and FIPS 140-2 Vendor List. http://csrc.nist.gov/groups/STM/cmvp/documents/140-1/1401vend.htm.

[10] Vladimir Katalov. ElcomSoft Breaks iPhone Encryption, Offers Forensic Access to File System Dumps, May 23, 2011. http://blog.crackpassword.com/2011/05/elcomsoft-breaks-iphone-encryption-offers-forensic-access-to-file-system-dumps/.

[11] Jean-Baptiste Bedrune and Jean Sigwald. iPhone data protection in depth, May 2011. Presentation at HITB 2011, http://conference.hackinthebox.org/hitbsecconf2011ams/materials/D2T2%20-%20Jean-Baptiste%20Be%cc%81drune%20&%20Jean%20Sigwald%20-%20iPhone%20Data%20Protection%20in%20Depth.pdf.

[12] Alfred J. Menezes and Paul C. Van Oorschot and Scott A. Vanstone and R. L. Rivest. Handbook of Applied Cryptography, 1997. http://cacr.uwaterloo.ca/hac/.

[13] Michael J. Wiener. Cryptanalysis of short rsa secret exponents. *IEEE Transactions on Information Theory*, 36:553–558, 1990.

[14] Eric R. Verheul and Henk C. A. van Tilborg. Cryptanalysis of 'less short' rsa secret exponents. *Appl. Algebra Eng. Commun. Comput.*, 8(5):425–435, 1997.

[15] Dan Boneh and Glenn Durfee. Cryptanalysis of rsa with private key d less than $n^{0.292}$. *IEEE Transactions on Information Theory*, 46(4):1339, 2000.

[16] T. Dierks and E. Rescorla. The transport layer security (tls) protocol version 1.2, August 2008. http://tools.ietf.org/html/rfc5246.

[17] NIST. FIPS PUB 180-4 Secure Hash Standard, March 2012. http://csrc.nist.gov/publications/fips/fips180-4/fips-180-4.pdf.

[18] H. Krawczyk, M. Bellare, and R. Canetti. HMAC: Keyed-Hashing for Message Authentication, February 1997. http://tools.ietf.org/html/rfc2104.

[19] Raffaele Cappelli, Dario Maio, Alessandra Lumini, and Davide Maltoni. Fingerprint image reconstruction from standard templates. *IEEE Trans. Pattern Anal. Mach. Intell.*, 29(9):1489–1503, 2007.

[20] Arun Ross, Jidnya Shah, and Anil K. Jain. From template to image: Reconstructing fingerprints from minutiae points. *IEEE Trans. Pattern Anal. Mach. Intell.*, 29(4):544–560, 2007.

[21] Y. Dodis, R. Ostrovsky, L. Reyzin, and A. Smith. Fuzzy Extractors: How to Generate Strong Keys from Biometrics and Other Noisy Data. *SIAM Journal on Computing*, 3(1):97–139, 2008.

[22] Xavier Boyen. Reusable Cryptographic Fuzzy Extractors. In *ACM CCS 2004, ACM*, pages 82–91. ACM Press, 2004.

[23] F. Hao, R. Anderson, and J. Daugman. Combining Cryptography with Biometrics Effectively. *IEEE Trans. Comput.*, 55(9):1081–1088, 2006.

[24] C. Rathgeb and A. Uhl. A Survey on Biometric Cryptosystems and Cancelable Biometrics. *EURASIP Journal on Information Security*, 3, 2011. http://jis.eurasipjournals.com/content/2011/1/3.

[25] Apple Inc. iOS Programming Guide: Communicating with other Apps; Implementing Custom URL Schemes. http://developer.apple.com/library/ios/#documentation/iPhone/Conceptual/iPhoneOSProgrammingGuide/AdvancedAppTricks/AdvancedAppTricks.html#//apple_ref/doc/uid/TP40007072-CH7-SW2.

[26] Google. IntentFilter. http://developer.android.com/reference/android/content/IntentFilter.html.

[27] IANA. URI Schemes. http://www.iana.org/assignments/uri-schemes.html.

[28] Zwapp. One Million App Schemes. http://onemillionappschemes.com/.

[29] A. Barth, C. Jackson, and J. C. Mitchell. Robust Defenses for Cross-Site Request Forgery. In *Proceedings of the 15th ACM Conference on Computer and Communications Security*, 2007. http://www.adambarth.com/papers/2008/barth-jackson-mitchell-b.pdf.

[30] WolframMathWorld. Prime Zeta Function.
http://mathworld.wolfram.com/PrimeZetaFunction.html.