Strong and Convenient Multi-Factor Authentication on Mobile Devices

Francisco Corella, PhD
fcorella@pomcor.com

Karen Lewison, MD
kplewison@pomcor.com

Revised September 6, 2012

Executive Summary

Authentication methods used today on mobile devices are both inconvenient and insecure.

Ordinary passwords are difficult to type on small touch-screen displays that require switching keyboards for entering digits or punctuation. They provide even less security on mobile devices than on desktops or laptops: a typing-feedback feature prominently displays each character after it is typed, eliminating the security provided by password input boxes that display dots in lieu of characters; and users are motivated to choose shorter and simpler passwords, which have less entropy.

One-time passwords are often used on mobile devices due to the lack of security of ordinary passwords. Authenticating with a one-time password requires: entering a PIN or a password to generate or request the one-time password; obtaining the one-time password from a hard token, a soft token, a text message, or an email message; and entering the one-time password. This is a very cumbersome procedure. One-time passwords afford only relative security because they can be observed or intercepted and they remain valid for several minutes.

We propose one-, two- and three-factor authentication methods for mobile devices that provide strong security and are more convenient to use than one-time or ordinary passwords. They are suitable both for enterprise and consumer use.

The proposed authentication methods are based on public key cryptography, but they are easy to implement and deploy. They are easy to implement because all cryptography is encapsulated in black boxes, so that developers do not have to program any cryptographic operations. They are easy to deploy because they avoid the use of certificates and do not require a public-key infrastructure.

In our one-factor authentication method the device uses a key-pair credential, and the user does not have to provide any input. The device authenticates by demonstrating knowledge of the private key. A hash of the associated public key is stored in a device record, which is linked to a user record in an enterprise directory or user database. If the device is lost or stolen, the key-pair credential can be revoked by removing the device record from the database.

In our two- and three-factor authentication methods, the device also uses a key-pair credential, but the key pair is not stored in the device; it is regenerated before use from the additional authentication factor(s).

In the two-factor authentication method, the user provides a passcode such as a PIN, which is used to regenerate the key pair. Because any passcode results in a well-formed key pair, the user's passcode is not exposed to an exhaustive offline guessing attack by an attacker who steals the mobile device, opens it, and reads its persistent memory.

In the three-factor authentication method, the user provides a passcode such as a PIN and a biometric sample such as an iris image taken by the camera of the mobile device. No biometric template is stored in the mobile device. Instead, the device contains an auxiliary string that is used in conjunction with the biometric sample to generate a biometric key. The biometric key is used in turn to regenerate the key pair. The auxiliary string is encrypted by a key derived from the passcode for additional security.

Although the proposed methods are motivated by the need to improve authentication on mobile devices, they can also be implemented on desktop or laptops using browser extensions.

1 Introduction

Two methods are used today for user authentication on mobile devices: ordinary passwords, and one-time passwords. Both of them have security and usability drawbacks. An alternative, public key certificates, provides strong security, but is difficult to deploy. Instead, we propose novel one-, two- and three-factor authentication methods based on public key cryptography without certificates. These new methods provide strong security and are easy to deploy and use.

1.1 Drawbacks of Current User Authentication Methods on Mobile Devices

1.1.1 Ordinary Passwords

Passwords have well-known drawbacks for user authentication. They are hard to remember and easy to guess [1]; they are often reused [2], allowing malicious Web sites to capture them and use them to impersonate the user on other sites; they are vulnerable to phishing attacks; and databases of hashed passwords, which sometimes are not even salted [3], are targets for hackers who can mount offline dictionary attacks after breaking in. In the enterprise, employees are often required to change their passwords regularly, which impacts productivity and causes resentment [4].

Passwords on mobile devices have additional drawbacks. It is difficult to enter a password accurately on a small touch-screen keyboard; repeated mistakes result in an account lockout that requires a password reset. Entering digits and punctuation symbols requires switching keyboards. Users are likely to choose shorter, simpler, and therefore less secure passwords because of the difficulty of typing and the inconvenience of switching keyboards. To aid in typing, characters are magnified when typed, so an attacker can easily observe the password by looking over the user's shoulder as it is being typed.

Ordinary passwords on mobile devices are thus cumbersome to use while providing poor security.

1.1.2 One-Time Passwords

Given the poor security provided by ordinary passwords on mobile devices, mobile applications with non-trivial security requirements use one-time passwords instead; but one-time passwords are even more inconvenient for the user than ordinary passwords.

Traditionally one-time passwords have been generated by hardware tokens, and manually copied by the user from the display of the token to a password input box. When used on mobile devices, they can be generated instead by a native application running on the device, often referred to as a "software token". A software token avoids the cost of a hardware token and the inconvenience of having to carry it. But using a software token is still highly inconvenient. The user has to launch the software token application, enter a PIN to generate the one-time password and copy the one-time password to the password input box. There are alternative ways of producing a one-time password: it can be sent to the user in an SMS message or an email message; or the user may obtain it by selecting images from a one-time grid; but it's not clear that these alternative methods are less inconvenient.

A one-time password is a two-factor authentication method, one factor being the PIN, and the other the possession of the hardware token or the mobile device where the one-time password is generated or received. One-time passwords provide more security than ordinary passwords, but less security than other strong authentication methods such as public key cryptography. They have limited entropy, they can be observed or intercepted, and they remain valid for several minutes.

1.2 Drawbacks of Public Key Certificates

In theory, public key certificates are an attractive alternative for user authentication, because they provide strong security while requiring no user input when used as a one-factor authentication method; however they have their own drawbacks:

- Relying parties must cope with the complexities of checking revocation using certificate revocation lists (CRLs) [5] or the Online Certificate Status Protocol (OCSP) [6].
- A certificate issuer must set up a CRL distribution point or an OCSP service or both, in addition to dealing with the complexities of issuing certificates.
- Developers dislike the ASN.1 encoding [7] of certificate data, which makes debugging cumbersome.
- Users are burdened by the need to apply for, install, and renew certificates.

This may explain, at least in part, the poor support provided by browsers for SSL client certificates, and some of the difficulties encountered by the US government when trying to

deploy PIV credentials for access to information systems, described in a recent GAO report [8].

1.3 Our Solution: Public Key Cryptography Without Certificates

Our solution takes advantage of the security and ease of use of public key cryptography while avoiding the drawbacks of user certificates.

Our solution aims at replacing passwords. A password is used to authenticate repeat visits or transactions, establishing *user continuity*. A user registers a password and later uses the password to demonstrate to an application that he or she is the same user who registered the password with the application. The password does not demonstrate that the user has a particular employee number, or a particular social security number, or any other attribute that is asserted by a third-party authority. (Such attributes may be established by other means in cases where they are needed. For example, a person may open a bank account by visiting a branch and presenting physical third-party documentation, such as a driver's license. A web-access account may then be created for the user by a bank employee, and the user may be given a temporary password.)

Similarly, our solution does not seek to prove attributes asserted by any third party. It simply demonstrates user continuity, with no third party involvement. The user authenticates to a mobile application that maintains user accounts. The mobile application may be web-based, or it may consist of a native application that serves as a front-end to an online back-end. A user may have multiple mobile devices, which he or she registers with the application. Upon registration, a record for the mobile device is created in a database or directory where the application maintains user accounts, and it is associated with the user's account. The device record stores a cryptographic hash of a public key. Upon subsequent authentication, the mobile device submits the public key and demonstrates knowledge of the associated private key. The application computes the hash of the public key, verifies that it coincides with the hash stored in the device record, and uses the associated user account to identify the user. No certificates are needed, which greatly facilitates the task of the application developer.

Furthermore, all cryptography is encapsulated in two black boxes: a prover black box (PBB), which runs on the mobile device, and an online verifier black box (VBB). The PBB may be implemented as a library linked into native application code, or as a separate native application, or as a browser extension for use by web-based applications. The VBB may be implemented as a physical or virtual server appliance, or hosted on an application server.

Our solution comes in three versions, which provide one-factor, two-factor and three-factor authentication.

In one-factor authentication, the one factor is possession of a mobile device that contains a *mobile credential* consisting of a handle that identifies the device record and a key pair, the hash stored in the device record being the hash of the public key component of that key pair. In multi-factor authentication, the first factor is again a mobile device containing a mobile credential, but the mobile credential comprises data that yields a key pair when combined with additional authentication factors, rather than the key pair itself.

In two-factor authentication the second factor is a passcode such as a PIN. The passcode could be used to decrypt the private key component of the key pair, or the entire key pair; but that would make the passcode vulnerable to an offline attack by an attacker who steals the device and reads its persistent memory. Instead, the passcode is used to regenerate the key pair from the mobile credential, as explained below in Section 2.3.

In three-factor authentication, an auxiliary string needed to derive a so-called *biometric key* from a biometric sample such as an iris image, is x-ored with a randomized extended hash of a passcode, of same length as the auxiliary string. The biometric key is used to regenerate the key pair from the mobile credential, as explained below in Section 2.4.

1.4 Outline of the Rest of the Paper

In Section 2 we describe the case where the mobile application has a native front-end, and the PBB is embedded in the native front-end. The VBB may be a generic appliance, or it may be application-specific.

In Section 3 we describe the case where mobile devices are deployed by an enterprise, the term meant broadly to include commercial enterprises, federal agencies, military units, and other organizations. The enterprise has control over what applications are installed on the devices, so that all installed applications are trusted. Instead of embedding the PBB into the application, the application plays the role of client in a protocol that outsources authentication to the PBB, which is implemented as a separate native application. The PBB can thus be shared by multiple native enterprise applications; more significantly, the PBB can then be used by web-based mobile applications accessed through a mobile browser. The same VBB can be used by all the mobile applications in the enterprise. It may be implemented as a generic server appliance, or it may be application-specific.

In Section 4 we recommend mobile operating system improvements that would obviate the need to trust all installed applications in the enterprise case; and we point out that the PBB could be embedded in a mobile operating system.

In Section 5 we explain how the proposed authentication methods can also be used on desktop or laptop computers using browser extensions.

In Section 7 we conclude by recapitulating the benefits of our solution.

2 Embedding the Prover in a Native Application

Figure 1 shows the architecture of a mobile application that uses a PBB embedded in a native front-end. For ease of exposition we shall refer to the native front-end and the PBB as separate entities, even though the PBB is part of the front-end.

The application comprises a native front-end, an online back-end, a VBB implemented as a generic server appliance, and a database where the back-end stores user records and mobile device records. To simplify the description we will assume throughout this paper that the database is a relational database with a table of user records and a table of device records, but other storage architectures can be used, such as a NoSQL database or an LDAP directory. Each user record includes a user handle that uniquely identifies the record. We use the term *handle* rather the term *key* used in the database literature to avoid confusion

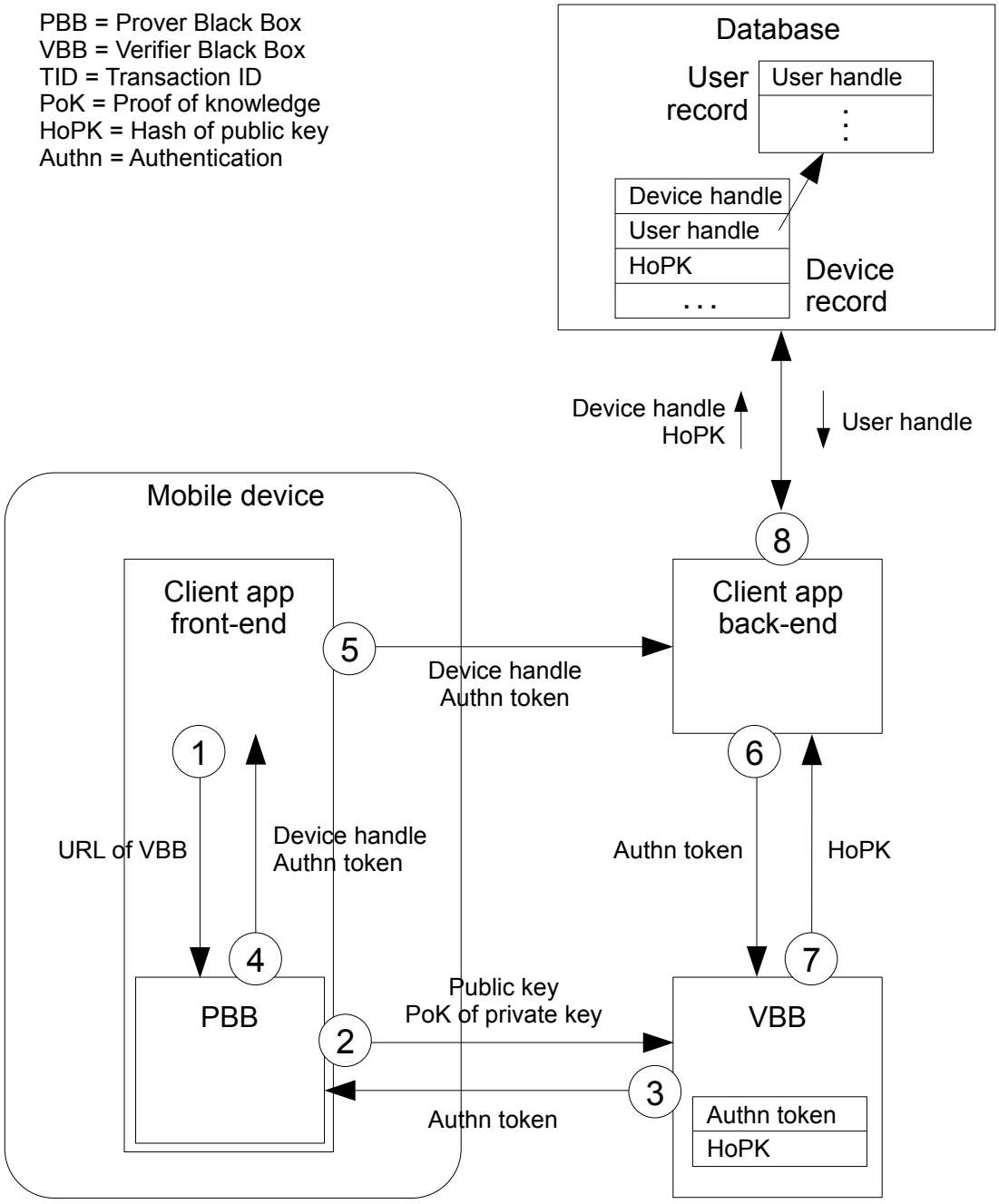

Figure 1. PBB embedded in native app, generic VBB appliance

with the term *key* as used in cryptography. Each device record comprises a device handle that uniquely identifies the record and user handle that refers to the record of the user who owns the device.[1] There may be multiple device records referencing the same user record, e.g. a record for a smart phone and a record for a tablet, both owned by the same user.

The PBB contains a mobile credential created when the user registers the device with the mobile application. The mobile credential consists of a device handle and a key pair pertaining to a public key cryptosystem, such as RSA, DSA, or ECDSA. The device handle plays the same role that a username plays in a username-and-password credential. The key pair may be explicit or implicit. An explicit key pair is physically stored in the PBB. An implicit key pair is not actually stored; it is generated at registration time, then regenerated at authentication time from secondary authentication factors, as explained below in sections 2.3 and 2.4. A hash of the public key component of the key pair is stored in the device record. The hash is computed using a cryptographic hash function such as SHA-256.

Connections between the native front-end and the online back-end are protected by TLS, with server or both client and server authentication as specified below. Connections between the back-end and the database, and between the back-end and the VBB, are protected by TLS, unless they take place over an intranet that is deemed secure, in which case the connections may be in the clear.

2.1 Authentication process

The authentication process may begin when the user launches the native front-end, or when the user clicks on a login button, or when the user initiates a transaction that requires authentication. It comprises the following steps, indicated by circled numbers in Figure 1.

In step 1, the native front-end passes the URL of the VBB to the PBB, and asks the PBB to authenticate to the VBB.

In step 2, the PBB retrieves the stored key pair, or regenerates it after requesting additional authentication factors from the user if multi-factor authentication is being used. Then it sends the public key component of the key pair to the VBB and demonstrates knowledge of the private key component; for one-factor authentication, this can be accomplished using a TLS extension where the TLS client authenticates to the server using a key pair without a certificate, as specified in [9].

But for two- and three-factor authentication the public key requires confidentiality protection, for reasons given below in sections 2.3 and 2.4, whereas in the TLS handshake the client's public key is sent in the clear.

The TLS protocol, or at least the extension of [9], could be improved by postponing client authentication until after the client's ChangeCipherSpec message.[2] Before such TLS improvements are available, the PBB can use an ordinary TLS connection with server authentication only, as follows: after the connection is established, the PBB sends its public key, the VBB sends a nonce, and the PBB sends another nonce and a signature on

[1] In an LDAP directory, the device record would be a child of the user record in the directory information tree.

[2] This change to the TLS handshake, which would increase user privacy when client authentication is used, has been discussed in the TLS working group.

a hash of both nonces computed with the private key.[3]

In step 3, the VBB generates a random high-entropy authentication token, computes the hash of the public key, and creates a short-term authentication record containing the token and the hash. (The record will be used later by the application back-end to retrieve the hash of the public key upon presentation of the token, and will be deleted when the back-end obtains the hash or after a short, configurable expiration time.) The VBB sends the authentication token to the PBB over the TLS connection that was established in step 2, after which it closes the connection.

If the key pair of the mobile credential pertains to a public-key encryption cryptosystem such as RSA, and the VBB also uses an encryption key pair for authentication, and the PBB knows the public key of the PBB, then steps 2 and 3 can be replaced with a simple two-step protocol that does not require the establishment of a TLS connection. The PBB sends the public key of the mobile credential to the VBB, encrypted under a symmetric key that is itself encrypted under the public key of the VBB. (An RSA public key is too large to be encrypted directly under another RSA public key, if the moduli are of same size.) Then the VBB sends the authentication token to the PBB encrypted under the public key of the mobile credential. There is no need for the PBB to demonstrate knowledge of the private key because, if it does not know the private key, it will not be able to decrypt and use the authentication token.

In step 4 the PBB passes the authentication token to the native front-end, together with the device handle.

In step 5 the native front-end authenticates to the online back-end, without cryptography, by sending the device handle and authentication token 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.

In step 7 the VBB uses the authentication token to locate the authentication record. It obtains the hash of the public key from the authentication record, deletes the authentication record, and sends the hash to the application back-end in the HTTP response to the HTTP POST request received in step 6.

In step 8 the back-end issues a query against the relational database that specifies a join of the table of device records and the table of user records, looking for a device record that contains the device handle and the hash of the public key, and a user record referenced by a user handle contained in the device record; the query fetches data such as the user handle. Authentication is successful if the query finds the specified records.

If authentication is successful, the back-end may log the user in by creating a session record containing a session ID and the device and user handles, and sending the session ID to the client front-end, which uses it for authentication while the session remains valid.

[3] To reduce the number of roundtrips, the PBB could sign a random string obtained from the TLS layer, jointly generated by the TLS client and server.

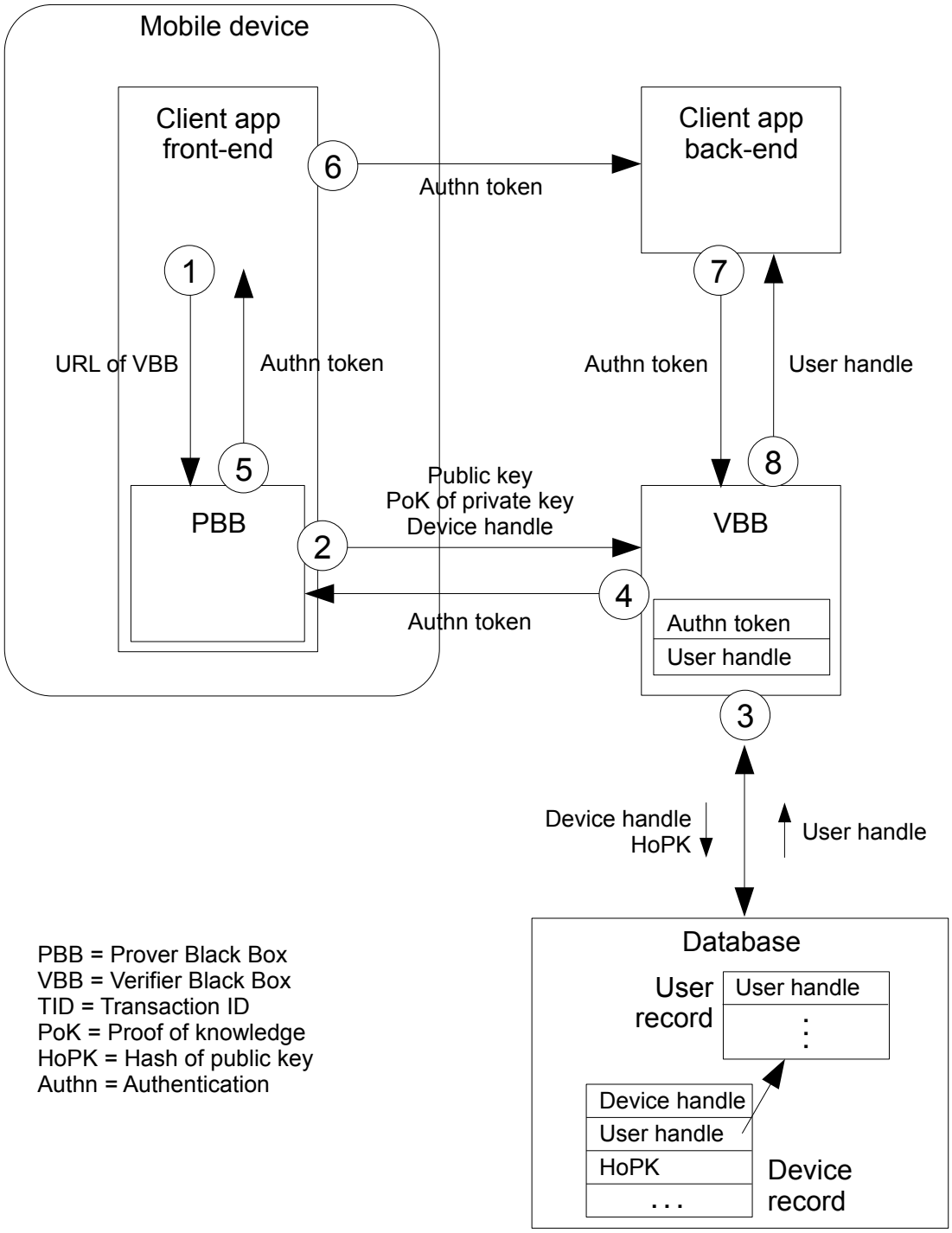

Figure 2. PBB embedded in native app, application-specific VBB

2.2 Application-specific VBB

Figure 2 shows a possible variant of the authentication process in the case where the VBB is specific to the application instead of being a generic server appliance. The VBB knows the structure of the database and accesses it on behalf of the client application, thus simplifying the client application. In step 2 the PBB sends the device handle to the VBB in addition to sending the public key and demonstrating knowledge of the private key. The VBB uses the device handle and the hash of the public key to access the database in step 3 and obtain data such as the user handle, which it stores in the authentication record. When the client application back-end receives the authentication token, it uses it to retrieve the data from the authentication record without having to access the database.

2.3 Two-Factor Authentication

A simple way of providing two-factor authentication would be to encrypt the private key with a key-encryption key derived from a passcode, such as a PIN. But in the absence of a tamper-resistant Trusted Platform Module (TPM),[4] a sophisticated attacker with physical access to the device might be able break into it and read the encrypted private key stored in persistent memory. It would then be trivial for the attacker to mount an offline attack, trying passcodes until one is found that produces a decrypted private key matching the public key. The match can be tested by checking that a signature produced by the private key can be verified with the public key. Encrypting the entire key pair rather than just the private key would not help, since passcodes could be tested by checking if they produce a well-formed key pair.

Instead, in our two-factor authentication method, the passcode is used to regenerate the key pair from the mobile credential. Any passcode produces a well-formed key pair, so passcodes cannot be tested offline using only data extracted from the mobile device. If the attacker does not have direct access to the database, a passcode can only be tested by attempting to authenticate with the key pair it produces, using the authentication process of Section 2.1. An attacker who tampers with the phone and reads its persistent memory has no advantage over an attacker who simply tries passcodes by typing them in as part of a normal authentication procedure. If the application back-end limits the number of guesses to five, a 6-digit PIN should provide a reasonable amount of entropy.

Of course, the attacker will be able to mount an offline attack against the passcode if he or she knows the public key, or a hash or other data derived from the public key. Therefore this method of preventing an offline attack requires confidentiality protection for the public key when it is transmitted from the PBB to the VBB in step 2, and for the hash of the public key stored in the device record.[5]

[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, no NFC secure element has yet been certified by NIST as tamper resistent. 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. Google Wallet originally used an NFC secure element to store credit card numbers, but it now stores them online.

[5]Traditionally, in public key cryptography the public key is, naturally, *public*. The public key of an

To regenerate the key pair, we propose to use RSA [10, §8.2] as the public key cryptosystem, and keep the prime factors p, q of the RSA modulus $n = pq$ in the PBB. When creating the key pair, instead of using a small encryption exponent e such as $e = 3$ or $e = 65537$ and computing the decryption exponent as $d = e^{-1} \pmod{\phi}$, $\phi = (p-1)(q-1)$, we propose to derive d from the passcode and compute $e = d^{-1} \pmod{\phi}$.

The decryption exponent d is derived from the passcode, 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 passcode with random seed s, of same byte length as the modulus[6], 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 [14, §5], which uses a hash function such as SHA-256 [15], the HMAC mechanism [16], a secret, which in this case is the passcode, 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, which is less thatn 0.2% [17, Appendix A].

Instead of storing the private key d and the public key (n, e) in the PBB, we store p, q and s. (The set S may be stored as well to facilitate to derivation of d.) The mobile credential is the tuple (h, p, q, s), where h is the device handle, which uniquely identifies the device record. The key pair is later regenerated from the mobile credential by using the above process to derive d, then computing $n = pq$ and $e = d^{-1} \pmod{\phi}$.

Notice that p and q can be computed from the key pair [10, §8.2.2(i)], so storing p and q is no less secure that storing the key pair, or the private key and a certificate containing the public key. Notice, however, that d can be computed from p, q and e. Therefore it is important to store the hash of the public key in the database rather than the public key itself, so that an attacker who breaks into the database in addition to reading the persistent

encryption key pair must be public so that anybody can send an encrypted message to the party that knows the private key. 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 without third party involvement, i.e. when it is used for user continuity verification, there is no reason why the public key, or a hash of the public key, cannot be a shared secret between the user and the party that authenticates the user. The hash of the public key should be protected just like the salted hash of a password. Notice, however, that a security breach that gives an attacker access to the hash of the public key is much less serious than one that gives access to a salted hash of a password. In the present case, 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, in order to be able to mount an offline attack against the passcode.

[6] We do not use a shorter hash in order to avoid attacks against short decryption exponents [11, 12, 13].

memory of the mobile device must make the additional effort of mounting an offline attack against the passcode to obtain d, rather than just computing d from p, q and e.

The two-factor authentication process is the same as the one-factor process of Section 2.1, except that the PBB prompts the user for the passcode in step 2 and uses it to regenerate the key pair.

2.4 Three-Factor Authentication

Several methods have been described for deriving a key from a biometric sample and an auxiliary string [18, 19, 20, 21]. The derived key is sometimes called a *biometric key*. The auxiliary string is derived from an original biometric sample and a random string. Biometric samples that are similar enough to the original biometric sample consistently produce the same biometric key. The original biometric sample cannot be recovered from the auxiliary string.

For our three-factor authentication method, we propose to use an RSA key pair as the first factor, a passcode such as a PIN as the second factor and a biometric sample, such as a an iris image obtained by a camera available on the mobile device, as the third factor. The passcode is used to protect an auxiliary string, and the biometric sample is used to produce a biometric key using the auxiliary string, by one of the methods described in the literature.

The auxiliary string is stored in the PBB, encrypted by x-oring it with a randomized extended hash of the passcode. The randomized extended hash, of same bit length as the auxiliary string, is computed using the TLS `P_hash` function and a random seed s, as explained above in Section 2.3. We shall call a the encrypted auxiliary string. Offline attacks are prevented as in the two-factor case, by storing in the PBB the prime factors p, q of the modulus $n = pq$ of the key pair rather than the key pair itself, and regenerating the key pair before use. The key pair is regenerated from the biometric key and a random seed s' (different from the random seed s used to compute the randomized extended hash of the of the passcode) as in the two-factor case, with the biometric key playing the role that was played by the passcode in the two-factor case. The mobile credential is the tuple (h, s, a, s', p, q), where h is the device handle.

The three-factor authentication process is the same as the one-factor process of Section 2.1, except that, in step 2, the PBB obtains the passcode and the biometric sample from the user, and uses them to regenerate the key pair. To recapitulate, the PBB performs step 2 as follows:

1. It prompts the user for the passcode.
2. It computes the randomized extended hash of the passcode using the random seed s and decrypts the auxiliary string by x-oring the randomized extended hash with the encrypted auxiliary string a.
3. It obtains the biometric sample from the user.
4. It uses the decrypted auxiliary string and the biometric sample to derive the biometric key.

5. It regenerates the key pair from the biometric key, the random seed s', and the prime factors p and q.

6. It sends the public key to the VBB and demonstrates knowledge of the private key.

2.5 Registration

The user may register the mobile device for use with a preexisting user account, or may register as a user and create a new user account at the same time as he or she registers the device for use with the account. In either case the device may be registered as part of the process of installing the native front-end in the mobile device, or after installation of the front-end.

In the case where the VBB is a generic server appliance the registration process is similar to the authentication process described above in Section 2.1 and illustrated in Figure 1. It comprises the following nine steps.

Step 1 depends on whether the user is creating a new account or registering a new device with a preexisting account.

To create a new user account and register the device with the new account, the user provides user data to the native front-end, e.g., by filling out a registration form.

To register the device for use with a preexisting account, the user uses a preexisting credential to demonstrate ownership of the account. That preexisting credential could be a username-and-password credential that the user uses on desktop or laptop computers, or it could be a temporary PIN or password provided to the user for the purpose of registering the new device. In either case the user provides his or her preexisting credential to the native front-end.

In step 2, the native front-end passes the URL of the VBB to the PBB, and asks the PBB to create a mobile credential and authenticate to the VBB.

In step 3 the PBB creates a mobile credential, sends the public key to the VBB and demonstrates knowledge of the private key.

Creation of the mobile credential depends on the number of authentication factors being used.

In the case of one-factor authentication, the mobile credential consists of the device handle that will uniquely identify the device record in the database, and a key pair. The PBB generates the device handle and the key pair, and stores them; the device handle is generated as a random high-entropy string, which will be different from other device handles in the database with high probability. The PBB sends the public key to the VBB, and demonstrates knowledge of the private key.

In the case of two-factor authentication, the PBB generates the prime factors p and q of an RSA modulus, asks the user for a passcode such as a PIN, and computes the RSA private and public keys from p, q, the passcode and a random seed s as described above in Section 2.3. (Random seeds must not be reused. A fresh random seed must be generated for each mobile credential.) The mobile credential is (h, p, q, s), where h is the device handle, generated as a random high-entropy string. The PBB sends the public key to the VBB, and demonstrates knowledge of the private key. The PBB discards the public and private keys after using them, keeping only the mobile credential (h, p, q, s).

In the case of three-factor authentication, the PBB obtains from the user a biometric sample and a passcode such as a PIN. It generates a random string and computes an auxiliary string and a biometric key from the biometric sample and the random string. (In a subsequent authentication process, the same biometric key will be derived from the auxiliary string and a presented biometric sample, if the presented biometric sample is close enough to the original biometric sample provided during registration.) It computes a randomized extended hash of the passcode from a fresh random seed s as described in Section 2.3, and x-ors it with the auxiliary string to obtain an encrypted auxiliary string a. It generates the prime factors p and q of an RSA modulus and computes the RSA private and public keys from p, q, the biometric key and a random seed s' as described above in Section 2.4. The mobile credential is the tuple (h, s, a, s', p, q), where h is the device handle, generated as a random high-entropy string. The PBB sends the public key to the VBB and demonstrates knowledge of the private key. The PBB discards the public and private keys after using them, keeping only the mobile credential (h, s, a, s', p, q).

Steps 4 and 5 are as steps 3 and 4 of the authentication protocol of Section 2.1.

In step 6 the native front-end sends the device handle and the authentication token to the back-end as in step 5 of the authentication protocol of Section 2.1. In addition to the handle and the token, the native front-end also sends to the back-end:

- The user data obtained from the user in step 1, if the user is creating a new account; or
- The preexisting credential obtained from the user in step 1, if the user is registering the device for use with a preexisting account.

Steps 7 and 8 are as steps 6 and 7 of the authentication protocol of Section 2.1.

In step 9, the back-end interacts with the database using the device handle received from the front-end in step 6 and the hash of the public key received from the VBB in step 8, as follows:

- If a new account is being created, the back-end creates a user record containing the user data obtained in step 1 and a user handle that uniquely identifies the record, as well as a device record containing the device handle received in step 6, the user handle that identifies the user record, and the hash of the public key received in step 8.
- If the device is being registered for use with a preexisting account, the back-end uses the preexisting credential obtained in step 1 to locate the user record, authenticate the user, and obtain the user handle that uniquely identifies the user record.[7] Then the back-end creates a device record containing the device handle received in step 6, the user handle, and the hash of the public key obtained in step 8.

[7] If the preexisting credential is a username-and-password credential, the username is used to locate the user record, and the user is authenticated by comparing a hash of the presented password and a salt stored in the user record against a hash stored in the user record. If the preexisting credential is a temporary credential consisting of a temporary PIN or password, the temporary credential is stored in the clear in the user record, and can thus be used to locate the user record, the user being authenticated if the record can be found.

In the case where the VBB is application-specific, the registration process is similar to the authentication process described above in Section 2.2 and illustrated in Figure 2.

3 Enterprise Use Case

Mobile operating systems including iOS and Android provide interapp communication facilities that allow native applications to send messages to each other. A native application can ask the operating system to deliver a URL to another native application. There are two kinds of URLs, which we shall call *web URLs* and *native URLs*. A web URL uses the scheme `http` or the scheme `https`. The operating system delivers it to the default browser and causes the default browser to send an HTTP GET request targetting the URL over a network. A native URL identifies a native application installed on the same device. The operating system delivers it to the native application, which does not forward it to the network.

Both iOS [22] and Android [23] 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, a native URL is both the address and the contents of the message. The native application 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 interacts with the HTTP redirection mechanism. 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.

We propose to use these interapp communication facilities to implement a protocol where the mobile application, which will be referred to as the *client application*, outsources authentication to the PBB, implemented as a separate native application. Unfortunately, as currently implemented by iOS and Android, interapp communication might allow a malicious application to receive a message intended for a different application.[8] For that reason, the security of the protocol depends on all installed applications being trusted. The protocol is therefore best suited for the case where mobile devices are deployed by an enterprise that exerts control over what applications are installed on the devices. (The term *enterprise* is meant broadly to include commercial enterprises, federal agencies, military units, and other organizations.) Notice that such control 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 application inventory control for BYOD devices. We refer to this case as the *enterprise use case*.

The outsourcing protocol allows the PBB to be shared by any number of native applications as described in the next section; furthermore, the PBB can be used by web-based

[8] 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.

mobile applications that interact with the user via a web browser. The outsourcing protocol also allows the same VBB to be used by all the mobile applications in the enterprise, whether web-based or having a native front-end.

3.1 Authentication to a Native Application

Figure 3 shows the process of authenticating to a client application that has a native front-end in the enterprise use case, using a VBB implemented as generic server appliance.

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 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.[9]

The PBB has an outsourcing endpoint where the native front-end sends an outsourcing request, and the native front-end has a callback endpoint where the PBB sends a reply to the outsourcing request. The VBB is implemented as an 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.[10]

As in Figure 1, a database contains a user record and a device record that references the user record. The device record contains a device handle that uniquely identifies it, a user handle that refers to the user record, and the hash of the public key component of a key pair. In one-factor authentication, the handle and the key pair are stored in the PBB, as a mobile credential. In two- and three-factor authentication the handle is stored in the PBB and the key pair is regenerated from a mobile credential before it is used, as described above in sections 2.3 and 2.4.

The authentication process has eight steps, similar to the eight steps of the authentication process of Section 2.1.

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.

[9] 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.

[10] The VBB endpoints were not shown in Figure 1 only because the concept had not been introduced yet. There is no difference between the VBB of Figure 1 and that of Figure 2.

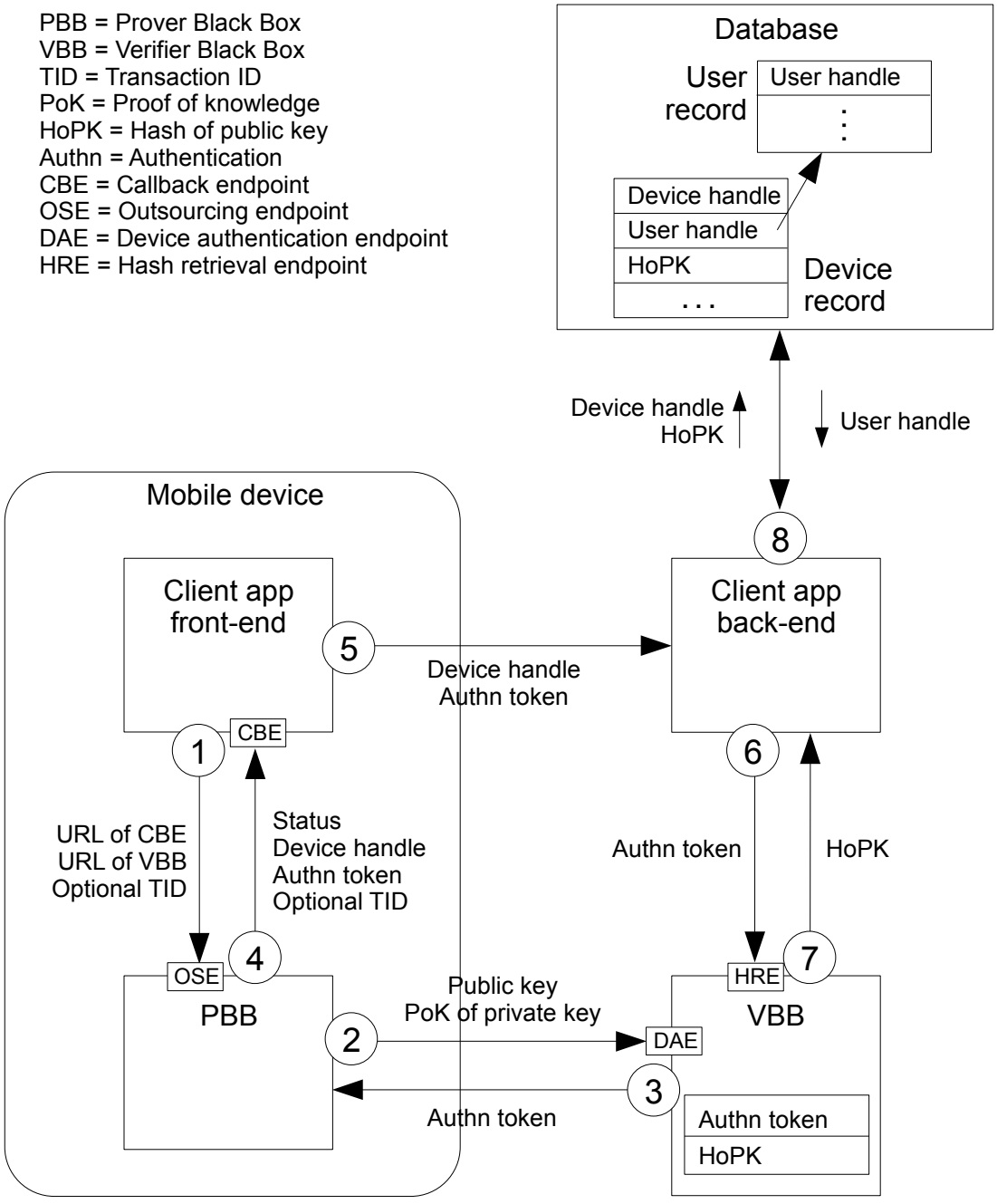

Figure 3. Native app, native PBB, generic VBB

Steps 2 and 3 are as in the case of an embedded PBB, described above in Section 2.1.

In step 4 the PBB sends the authentication token obtained from the VBB in step 3 to the callback endpoint of the native front-end, along with the device handle, 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.

Steps 5-8 are as in the embedded PBB case of Section 2.1.

3.2 Authentication to a Web-based Application

Figure 4 shows the process of authenticating to a web-based client application in the enterprise use case.

The native front-end of the client application is replaced with the default web browser of the mobile device. 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 authentication process must therefore be hardened to cope with possible malicious web-based client applications. Notice, however, that the same PBB and VBB can be used for both native and web-based clients.

The user interacts with the client application via the browser, and the process 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.

Notice that the transaction ID, which was optional in the authentication process of Section 3.1, is compulsory here. 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.

Steps 2 and 3 are as when authenticating to a native application using a VBB implemented as a generic server appliance and either an embedded PBB (Section 2.1, Figure 1) or a PBB implemented as a separate native PBB (Section 3.2, Figure 3), except for one

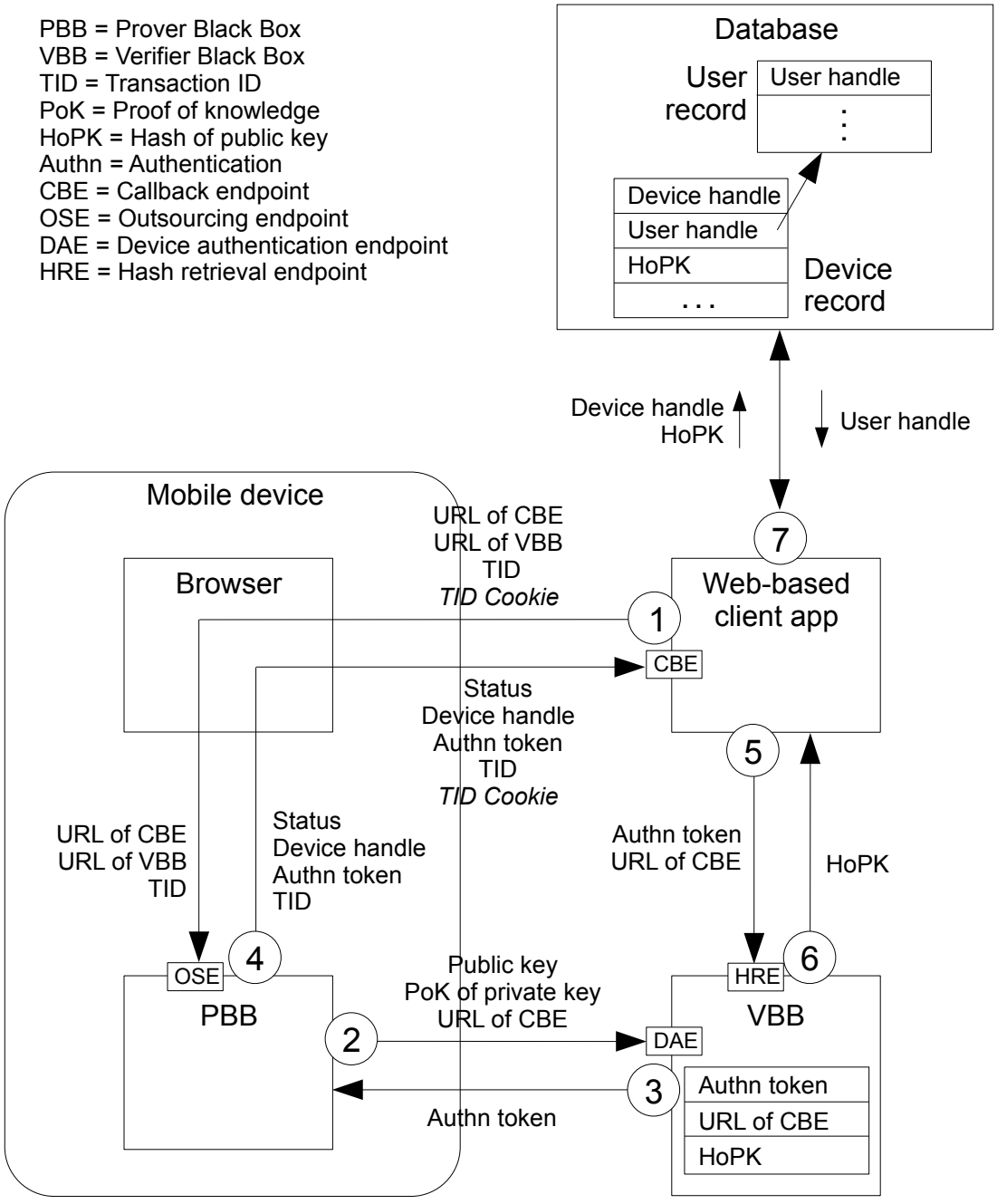

Figure 4. Web-based app, native PBB, generic VBB

important difference. The PBB sends to the VBB the URL of the client callback endpoint, and the VBB stores it as an additional field of the authentication record. The reason for this will become apparent below in connection with step 5.

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.
- 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, which 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 process fails.

The transaction ID check prevents an attack similar to the Login CSRF attack of [24] by an insider 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 company and the victim could be the CEO. The incorrect authentication could cause the CEO to log in to an account owned by the attacker without realizing it, and to enter sensitive data into the attacker's account, making it available to the attacker. 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, 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 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 sends 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, as in step 8 of the authentication process for a native application (sections 2.1 and 3.1, figures 1 and 3), the back-end issues a query against the relational database that specifies a join of the table of device records and the table of user records, looking for a device record that contains the device handle and the hash of the public key, and a user record referenced by a user handle contained in the device record; the query fetches data such as the user handle. Authentication is successful if the query finds the specified records.

If authentication is successful, the back-end may log the user in by creating a session record containing a session ID and the device and user handles, and setting the session ID in the browser as an authentication cookie.

3.3 Enterprise-specific VBB

Figures 5 and 6 illustrate the use of an enterprise-specific VBB to authenticate to enterprise mobile applications that use a native front-end or a browser, respectively. As in the case of an application-specific VBB (Section 2.2, Figure 2), the VBB knows the structure of the database and accesses it on behalf of the client application, thus simplifying the client application. The Device Authentication Endpoint (DAE) of figures 3 and 4 has been renamed User Authentication Endpoint (UAE) and the Hash Retrieval Endpoint (HRE) has been renamed User-data Retrieval Endpoint (URE).

The PBB sends the device handle to the VBB, and the VBB uses it together with the hash of the public key to access the database in step 3 and obtain data such as the user handle, which it stores in the authentication record. When the client application back-end receives the authentication token, it uses the token to retrieve the data from the authentication record without having to access the database.

3.4 Registration in the Enterprise Use Case

In the enterprise use case, the device is registered once to create a mobile credential in the PBB, which can then be used for authentication to all the mobile applications in the enterprise, both native and web-based. Registration is accomplished using a *registration application* that plays a role similar to that of the client application in the authentication

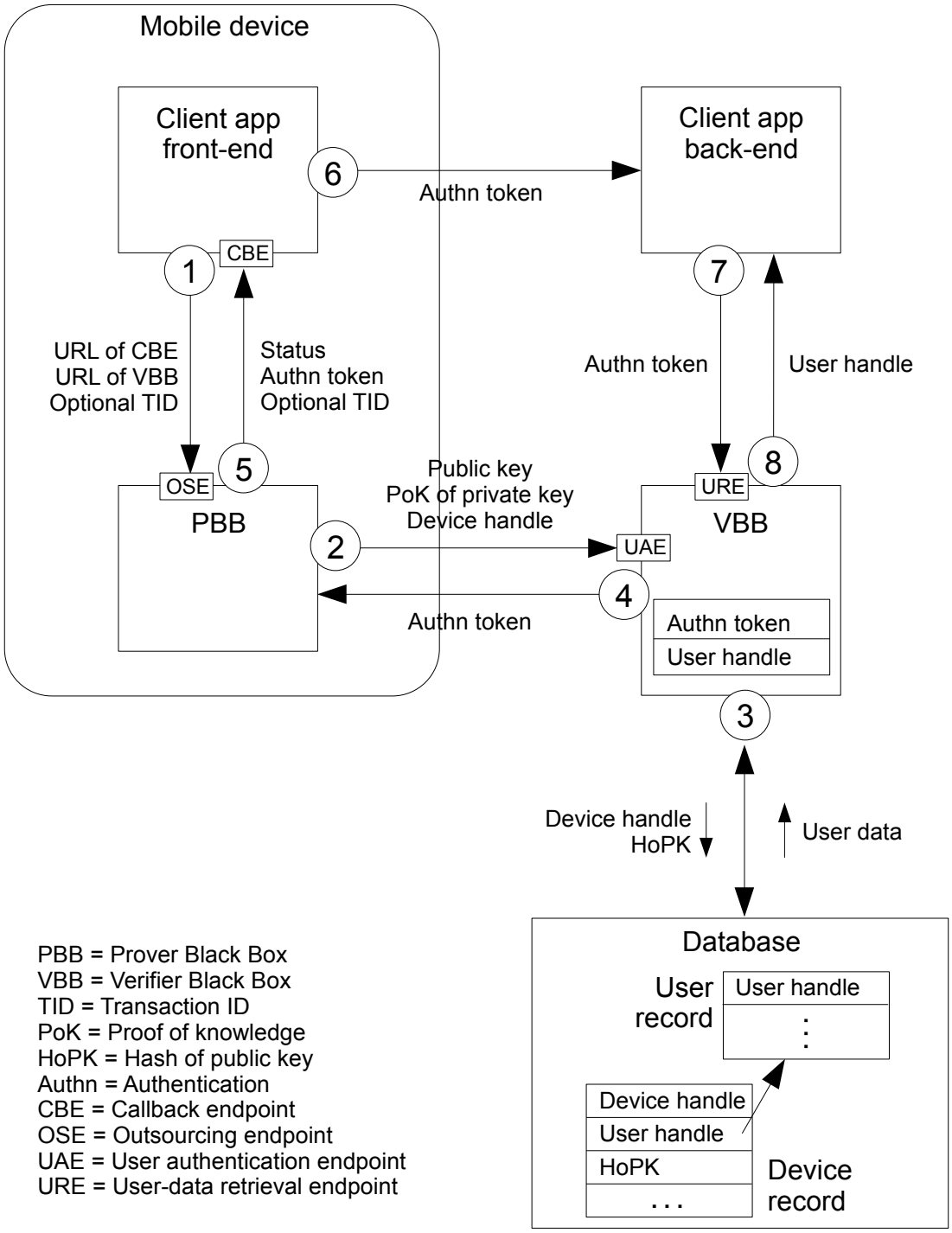

Figure 5. Native app, native PBB, enterprise-specific VBB

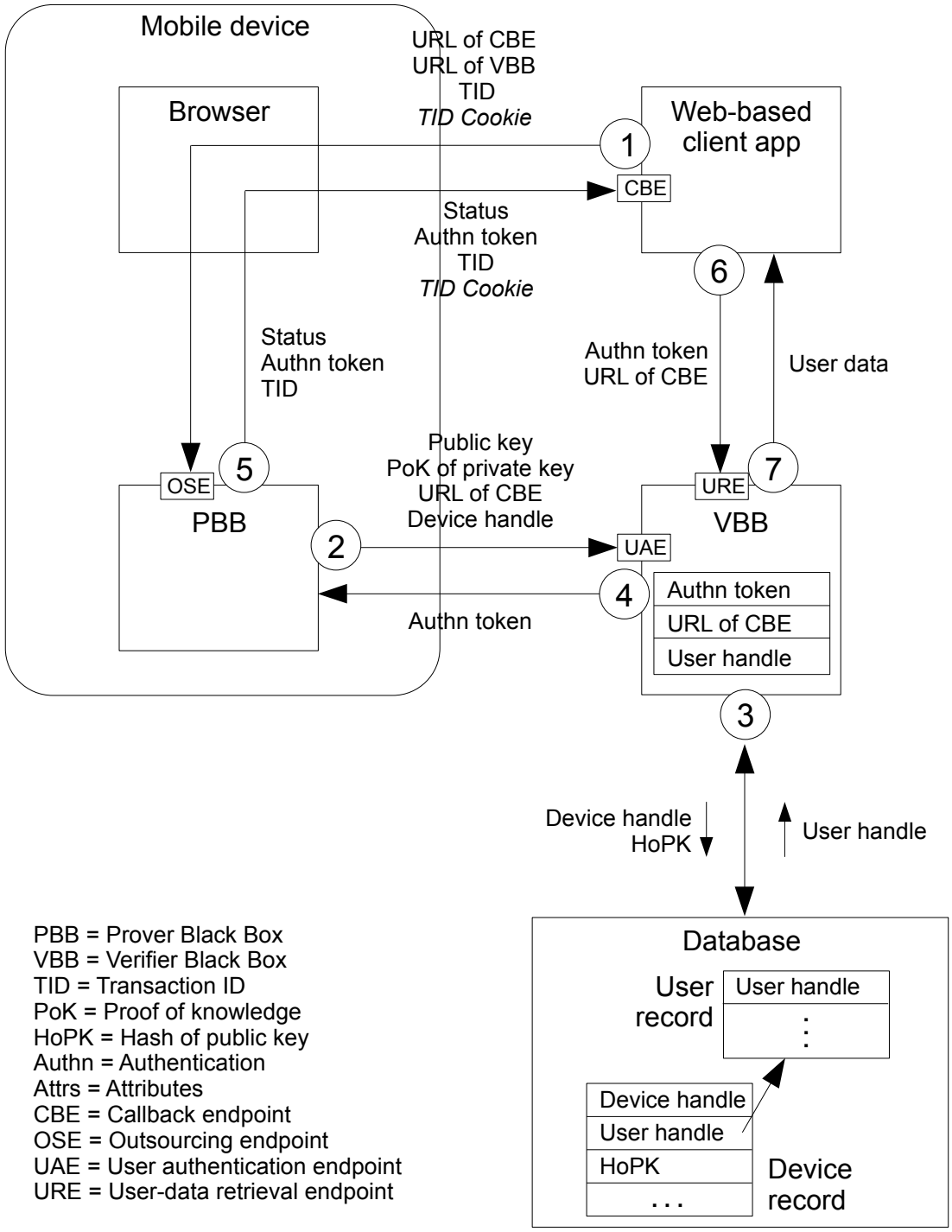

Figure 6. Mobile web-based app, native PBB, enterprise-specific VBB

process. All the cryptographic operations of the registration process are encapsulated in the PBB and the VBB. The registration application, like the client application, performs no cryptographic operations. The PBB has a *registration endpoint*, and the registration application has a callback endpoint.

We will only describe the registration application in the case where the VBB is a generic server appliance. The case of an enterprise-specific VBB is similar.

The registration application may be web-based or it may have a native front-end.

3.4.1 Registration Application with Native Front-End

If the registration application has a native front-end, the registration process is the same as the one described in Section 2.5 except for the fact that the native front-end and the PBB communicate through the interapp communication facility rather than through an API provided by the embedded PBB.

3.4.2 Web-Based Registration Application

If the registration application is web-based, the registration process comprises the following eight steps.

In step 1, the user enters user data to create a new user account, or a preexisting credential to register the device with a preexisting account. The user data or preexisting credential is conveyed by the browser to the web-based registration application in an HTTP request.

In step 2, the web-based registration application creates a transaction context referenced by a random high-entropy transaction ID, where it remembers the user data or preexisting credential received in step 1. Then it issues an HTTP redirection (302) response to the HTTP request, redirecting the browser to the registration endpoint of the PBB, with the following parameters:

- The URL of the callback endpoint of the registration application.
- The URL of the device-authentication endpoint of the VBB.
- The transaction ID.

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

In step 3 the PBB creates a mobile credential as in step 3 of the registration process of Section 2.5. Then it sends the public key and the URL of the callback endpoint to the VBB, and it demonstrates knowledge of the private key to the VBB.

Steps 4-7 are like steps 3-6 of the authentication process of Section 3.2 and Figure 4.

In step 8, the web-based registration application uses the transaction ID received in step 5 and uses it to recall the user data or preexisting credential obtained in step 1. Then it creates a device record in the database, or both a device record and a user record, as in step 9 of the registration process of Section 2.5.

4 Recommended Mobile Operating System Improvements

The interapp communication facilities provided today by iOS and Android 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[11] 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[12] 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 an authentication outsourcing protocol even in cases where not all installed applications can be trusted.

In addition to such improvements, an operating system could embed the functionality of the PBB within itself, so that applications can count on authentication outsourcing being available. Either a custom scheme registered with IANA, or a DNS domain name owned by the OS provider could be used to build the URLs of the device-authentication endpoint and the hash-retrieval endpoint.

5 Authentication on Desktops or Laptops

The authentication methods described above are motivated by the need to make authentication simpler and more secure on mobile devices. But authentication on desktops and

[11] URL schemes are supposed to be registered with IANA [25], but the custom schemes of native applications are not. (There is an informal database of iOS custom schemes [26] 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.

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

laptops is also in need of improvement, and it would simplify authentication for web applications if they could use the same methods on mobile devices as on desktops or laptops.

Two obstacles make it difficult to use the proposed mobile authentication methods on traditional desktop or laptop computers.

The first obstacle is the lack of the interapp communication facility described above in the preamble of Section 3. As the computing architecture of mobile devices becomes more on more popular, desktop operating systems may well implement the same interapp communication features found in mobile OSes, but this will take time. This obstacle rules out the use cases of figures 3-6.

The second obstacle is the fact that native applications are much less popular on desktops and laptops than on mobile devices, for a variety of reasons. This makes the use cases of figures 1-2 much less useful for desktop OSes than for mobile OSes.

We propose to overcome these obstacles by implementing the PBB as a browser extension capable of intercepting an HTTP redirect response addressed to a native URL, as illustrated in figures 7 and 8. (Most browsers have facilities that allow developers to create functionality extensions, variously called extensions, plug-ins, add-ons, helper objects, etc.)

Figure 7 shows the case where the VBB is a generic server appliance. It is identical to Figure 4, except for the fact that the PBB is embedded in the browser instead of being a separate native application.

In step 1, the PBB browser extension intercepts the HTTP redirect response that conveys the URL of the callback endpoint, the URL of the VBB, and the transaction ID to the outsourcing endpoint of the PBB.

In step 4, the PBB browser extension causes the browser to send an HTTP GET request that conveys the status, device handle, authentication token and transaction ID to the client callback endpoint in the query string of the callback URL; standard browser software adds the cookie that was set in step 1.

Similarly, Figure 8 shows the case where the VBB is application or enterprise-specific. It is to Figure 6 what Figure 7 is to Figure 4.

6 PBB as a Mobile Browser Extension

In the previous section we have proposed to implement the PBB as a browser extension to enable the implementation of the proposed authentication methods on desktops and laptops. But a PBB implemented as a browser extension may also be useful in mobile devices. In particular, it would be useful for authentication to web-based mobile applications on mobile devices in which some of the installed applications may be malicious.

7 Conclusion

We have proposed one-, two-, and three-factor authentication methods for mobile applications, both web-based applications and applications having a native front-end, that compare favorably with the authentication methods used today. The proposed methods are based

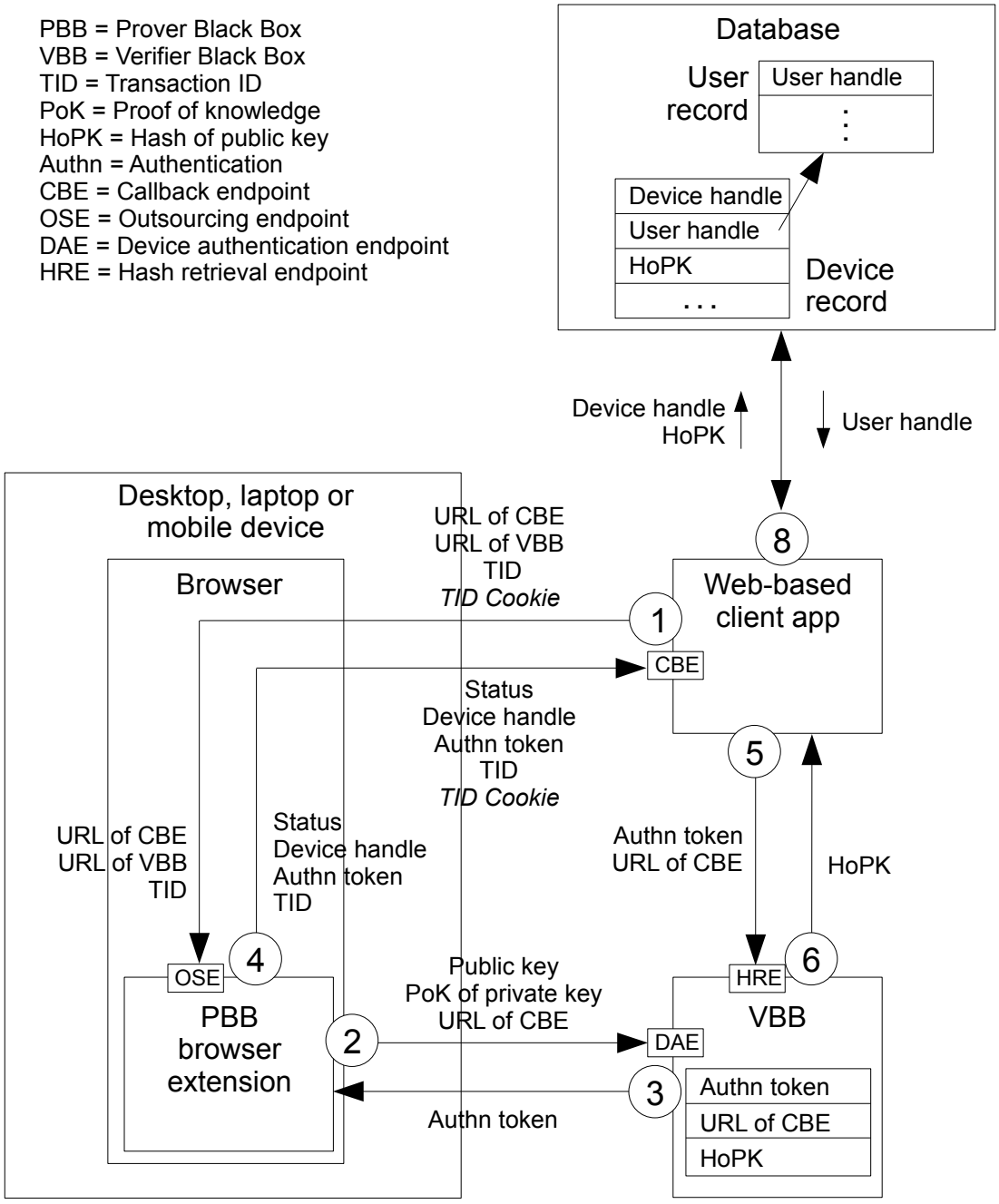

Figure 7. Web-based app, PBB browser plugin, generic VBB

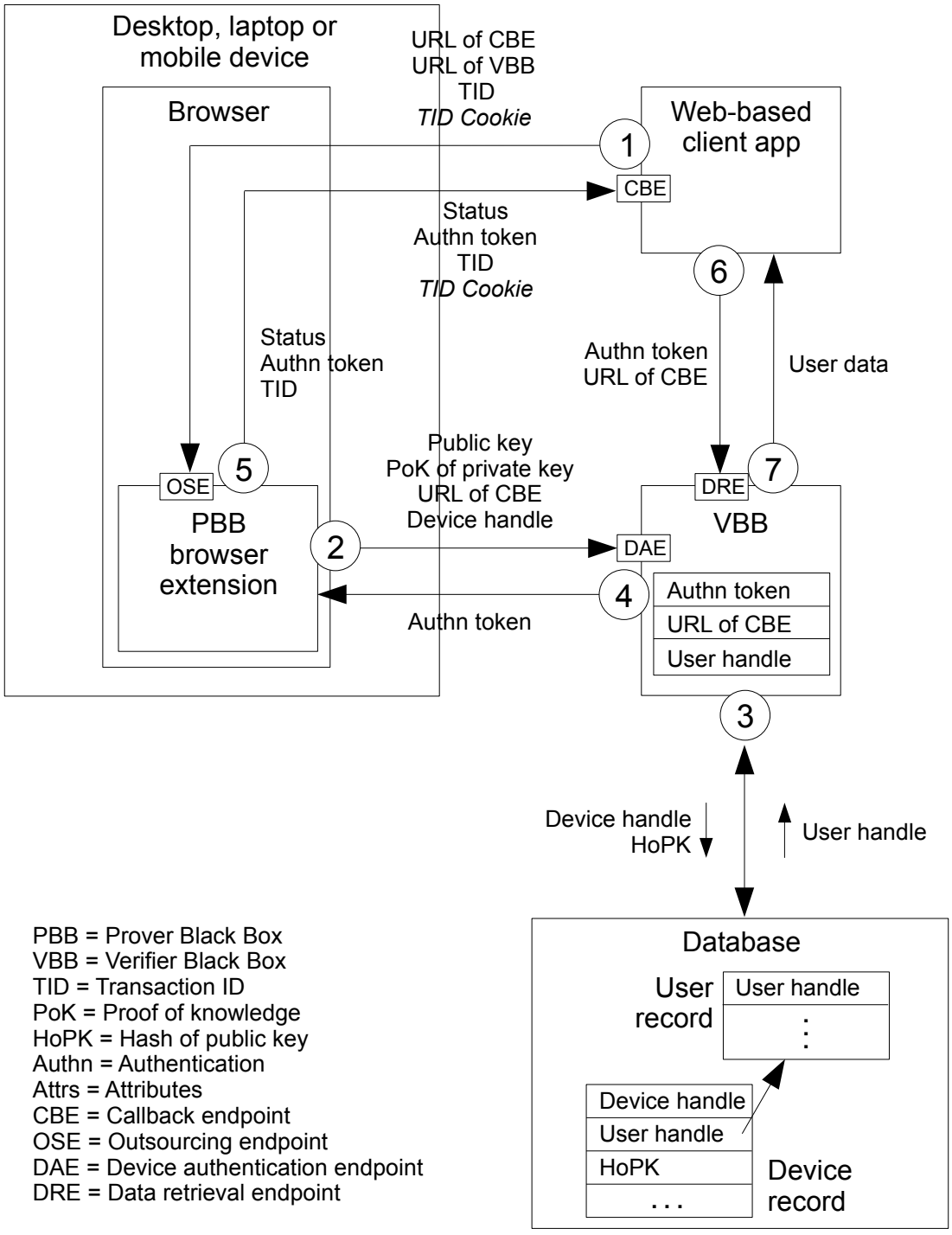

Figure 8. Web-based app, PBB browser plugin application or enterprise-specific VBB

on public key cryptography, but they do not use certificates, and their cryptographic functionality is encapsulated in two black boxes, a prover black box (PBB) and a verifier black box (VBB).

The proposed one-factor method requires no user input and provides the strong security of public-key cryptography. By contrast, the one-factor method used today consists of entering a username-and-password credential, which provides little security and is difficult to type on a small touch-screen keyboard. The proposed two-factor method only requires the user to enter a passcode such as a PIN, which is used to regenerate a key pair. By contrast, the two-factor method used today consists of entering a PIN or a password to generate or request a one-time password; obtaining the one-time password from a hard token, a soft token, a text message, or an email message; and entering the one-time password, which provides only relative security because it remains valid for several minutes. In the proposed three-factor method, the user enters a passcode such as a PIN and provides a biometric such as an iris image to generate a biometric key that in turn is used to regenerate a key pair.

In two- and three-factor authentication, the passcode is protected against offline attack from a sophisticated attacker who tampers with the device and reads its persistent memory. In three-factor authentication, neither the original biometric sample nor a biometric template are stored in the device, and both the passcode and the biometric key are protected against offline attack even in the absence of tamper resistance.

We have described several deployment scenarios that can be used with today's mobile devices, some where the PBB is embedded in the native front-end of a mobile application, some where the PBB is a separate native application, and some where the PBB is implemented as a browser extension. A PBB implemented as a native application is suitable for an enterprise setting where all applications installed in the mobile device are trusted. A PBB implemented as a browser extension extends the scope of the proposed methods beyond mobile devices to traditional desktop and laptop computers.

We have recommended mobile operating system improvements that would allow a single PBB implemented as a native application to be used securely by any number of native or web-based applications even if not all native applications installed on the device can be trusted. We have also pointed out that the functionality of the PBB could be embedded in the operating system.

References

[1] J. Yan, A. Blackwell, R. Anderson, and A. Grant. Password Memorability and Security: Empirical Results. *IEEE Security & Privacy*, 2004. http://homepages.cs.ncl.ac.uk/jeff.yan/jyan_ieee_pwd.pdf.

[2] Joseph Bonneau. Measuring Password Reuse Empirically, February 2011. http://www.lightbluetouchpaper.org/2011/02/09/measuring-password-re-use-empirically/.

[3] Ian Paul. Update: LinkedIn Confirms Account Passwords Hacked, June 6, 2012. http://www.pcworld.com/article/257045/update_linkedin_confirms_account_passwords_hacked.html.

[4] Rich Newman. Why Some Password Security is a Waste of Time, February 5, 2012. http://richnewman.wordpress.com/2012/02/05/why-some-password-security-is-a-waste-of-time/.

[5] D. Cooper, S. Santesson, S. Farrell, S. Boeyen, R. Housley, and W. Polk. Internet X.509 Public Key Infrastructure Certificate and CRL Profile, May 2008. http://datatracker.ietf.org/doc/rfc5280/.

[6] M. Myers, R. Ankney, A. Malpani, S. Galperin, and C. Adams. X.509 Internet Public Key Infrastructure Online Certificate Status Protocol - OCSP, June 1999. http://tools.ietf.org/html/rfc2560.

[7] ITU-T. Introduction to ASN.1. http://www.itu.int/ITU-T/asn1/introduction/index.htm.

[8] Government Accountability Office. PERSONAL ID VERIFICATION - Agencies Should Set a Higher Priority on Using the Capabilities of Standardized Identification Cards, September 2011. http://www.gao.gov/new.items/d11751.pdf.

[9] P. Wouters et al. TLS Out-of-Band Public Key Validation, April 25, 2012. Internet draft. Work in progress. http://tools.ietf.org/html/draft-ietf-tls-oob-pubkey-03.

[10] 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/.

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

[12] 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.

[13] 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.

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

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

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

[17] F. Corella and K. Lewison. Techniques for Implementing Derived Credentials. Pomcor whitepaper. http://pomcor.com/whitepapers/DerivedCredentials.pdf.

[18] 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.

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

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

[21] 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.

[22] 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.

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

[24] 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.

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

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