A Comprehensive Approach to Cryptographic and Biometric Authentication from a Mobile Perspective

Patents Pending

Francisco Corella, PhD
fcorella@pomcor.com

Karen Lewison, MD
kplewison@pomcor.com

Last updated on April 20, 2013

Executive Summary

User authentication on the Internet is widely acknowledged to be broken. Ordinary passwords have many vulnerabilities. Third-party login with a password adds a privacy problem while arguably making the security problem worse. Security questions are an invasion of privacy, and their answers can be easily discovered online. One-time passwords generated by a soft or hard token or communicated via email or text or voice messaging add only limited security at the cost of substantial user inconvenience. Cryptographic or biometric authentication is only used in special cases, such as employee or contractor authentication to information systems in US government agencies.

Mobile devices make the authentication problem worse. It is difficult to enter a high-entropy password on the touchscreen keyboard of a smart phone, and characters are echoed by the keyboard as they are typed. It is also difficult to protect the private key component of a cryptographic key pair against an adversary who captures the device because data protection mechanisms currently available on mobile devices are not effective. But at the same time mobile devices have brought us a new app-centric computing paradigm, in which both native and web-based applications can communicate with each other within the device, paving the way for innovation in the area of authentication architecture.

All of this has led us to rethink user authentication. We have identified a useful distinction between *closed-loop* and *open-loop* authentication, and defined the concept of a *protocredential*.

In *closed-loop authentication*, the party that issues or registers a credential is the same party that verifies possession of the credential at authentication time, whereas in *open-loop authentication* user attributes are asserted by a party that is not directly involved in the authentication process. Today, most authentication on the web is closed-loop, including traditional two-party authentication with username and password, and third-party login where a relying party redirects the browser to an identity provider who authenticates the user and redirects the browser back to the relying party, conveying the user's identity.

A *protocredential* consists of cryptographic parameters stored in the user's computing device, at least one of them secret, that are used to regenerate an uncertified

key pair in conjunction with one or more additional secrets provided by the user, such as a PIN and/or a biometric sample. We propose a protocredential-based technique for implementing multifactor closed-loop authentication. Without requiring tamper-proof storage, the technique provides protection against offline attack by an adversary who gains access to the user's device, as well as protection against an adversary who breaches database or network confidentiality.

One benefit of protocredential-based authentication is that it provides an effective data protection method for data stored on mobile devices, without relying on tamper resistance or special encryption hardware. The data is encrypted under a data encryption key that is entrusted to a key storage service or cryptographically distributed over multiple services. To retrieve the key, the user authenticates to the key storage service(s) using protocredential-based authentication.

Cryptographic closed-loop authentication provides full privacy when only two parties are involved, but open-loop authentication provides more privacy than closed-loop authentication when an authoritative third party is relied upon to assert user attributes. Credentials that can be used in open-loop authentication include a traditional PKI certificate coupled with its associated private key, a U-Prove token, or an Idemix anonymous credential. Protocredential-based authentication is applicable to multifactor closed-loop authentication, but not to open-loop authentication or one-factor closed loop authentication; but the proposed data protection method can be used to safeguard credentials used in open-loop authentication and one-factor closed-loop authentication when the device containing those credentials is captured by an adversary. A collection of such credentials can be encrypted under the same data-encryption key, so that they can all be decrypted at once and then used without further user intervention, thus providing a form of single sign-on.

We also propose an architecture for both closed-loop and open-loop authentication on mobile devices, encompassing both web-based applications and applications with a native front-end, that insulates application developers from the complexities of cryptographic and biometric authentication. We show how login sessions can be implemented in the framework of the architecture, and we describe a second form of single sign-on based on the sharing of login sessions among a group of applications, such as applications deployed by an enterprise. Finally, we show how the proposed mobile architecture can be adapted for use on a traditional personal computer by means of a browser extension that does not require modification of core browser functionality.

Contents

List of Figures

List of Tables

1 Introduction

User authentication on the Internet is widely acknowledged to be broken.

Ordinary passwords have many vulnerabilities and drawbacks. 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]. Calls for the demise of the password can be heard [5, 6].

Third-party login, where the user is redirected to an identity provider that requests the user's password, is an invasion of the user's privacy, since the identity provider is informed of the user's logins to relying parties. Social login, where the identity provider is a social network that grants the relying party access to the user's account and receives information from the relying party about the user's activities, is even more invasive. Identity providers and social networks claim that they increase security because the user has to remember fewer passwords, but security is arguably decreased, because prompting for a password after redirection facilitates phishing attacks [7, 8].

Security questions, whether posed automatically or by a human operator, provide little security because their answers can usually be found among the wealth of personal information that is collected and published by social networks, collected by advertising networks and potentially available to hackers, and directly collected by hackers using spyware. Forcing users to provide answers to such questions at many different sites further increases the likelihood that an adversary will obtain the answers.

A one-time password (OTP), numeric [9, 10] or graphic [11], is sometimes used instead of a long term password, or in conjunction with a long term passcode, which may be a short PIN or a higher entropy password, for two-factor authentication. A numeric OTP can be generated by a hard or soft token in the user's possession, or sent to the user in a text or voice message. But the inconvenience caused to the user by having to obtain and use the OTP buys only limited additional security. OTPs are not easy to guess and are not vulnerable to reuse, but they share other vulnerabilities with ordinary passwords, and have vulnerabilities of their own. OTPs are vulnerable to phishing attacks like ordinary passwords; the attacker who phishes an OTP has to use it while it is valid, but the validity period usually lasts several

minutes. Just as databases containing password hashes can be breached, so can databases containing seeds used to generate OTPs, and a breach of a database containing OTP seeds may allow the adversary to generate the OTPs herself; at least one such breach has actually occurred [12, 13, 14]. An adversary may be able to intercept an OTP sent in a text or voice message, since text and voice messaging do not provide strong end-to-end security. No matter how the OTP is obtained, an adversary may be able to observe both the OTP and the long term passcode used in conjunction with the OTP over the user's shoulder.[1] After capturing the OTP and the long term passcode, the adversary may be able to block the user's Internet connection (e.g. if the connection goes through a WiFi access point set up by the adversary, which masquerades as a legitimate public access point) and use the OTP and long term passcode while it is valid. Also, authentication by OTP plus long term passcode is vulnerable to a man-in-the-middle attack by an adversary who succeeds in spoofing a TLS server certificate, which is difficult but not unheard of, as discussed below in Section 4 below.

Strong authentication technology exists, but is not being widely deployed.

Secure authentication can be achieved using public key cryptography, by demonstrating knowledge of the private key component of a key pair. This is the method that underlies web authentication with TLS client certificates [15], WebID [16], BrowserID/Persona [17], and Origin-Bound Certificates [18]. But public key cryptography has failed to achieve broad adoption for user authentication. In the US, it is rarely used outside the Federal Government, which requires its agencies to use PIV cards [19] containing public key certificates and their associated private keys for authentication of federal employees and contractors to information systems. One reason for the lack of adoption is the failure of browser vendors and standards bodies to create an identity ecosystem where cryptographic credentials can be used conveniently and securely.

Biometrics do not provide strong security by themselves, and they raise privacy concerns. But they do strengthen security when used as a third authentication factor in addition to a cryptographic key and a passcode; and research in the last ten or fifteen years has addressed most of the privacy concerns. Yet biometrics are rarely used for user authentication on the Internet.

Mobile devices have created additional obstacles for user authentication.

The shortcomings of passwords are amplified on mobile devices. It is difficult to enter a password accurately on a small touchscreen keyboard, and repeated mistakes may result in an account lockout that requires the password to be reset by relying on alternative and even less secure authentication methods such as security questions. Entering digits and punctuation symbols requires switching keyboards; this, combined with the difficulty of typing on a tiny keyboard, discourages users from choosing high-entropy passwords. To aid in typing, characters are echoed by the touchscreen keyboard as they are being typed, thus defeating the security provided by a password input box that echoes characters as dots and making it easy for an attacker to observe the password by looking over the user's shoulder as it is being typed.

[1] Especially on a smart phone or tablet whose touchscreen keyboard echoes characters as they are being typed, as discussed below.

Authentication based on possession of a key pair also faces an additional difficulty on a mobile device, viz. the problem of how to protect the private key if the device is lost or stolen. Today's mobile devices do not have internal tamper resistant modules, and it is undesirable, for usability reasons, to store the private key in external tamper-resistant hardware, such as a smart card or fob that interacts with the mobile device. A traditional means of protecting a key pair without tamper resistance is to encrypt the private key under a key-encryption key derived from a password; but that requires entering a high-entropy password each time the private key needs to be decrypted, which is not practical on a small touchscreen keyboard, as discussed above. One could think of using the data protection mechanism available on the iPhone and iPad since iOS 4, based on a hardware encryption key [20]; but that data protection technique has proved to be ineffective [21, 22]. One could also think of relying on a remote data wipe offered by a Mobile Device Management (MDM) product; but an adversary who captures the device can easily circumvent that countermeasure by placing the device in a Faraday cage.

On the other hand, however, mobile devices have brought us a new computing paradigm, featuring native applications that can be developed quickly in high-level object-oriented languages by taking advantage of a rich framework of classes made available by the operating system. In this new software architecture, native applications are insulated from each other by sandboxing, but can communicate with each-other and web-based applications via novel inter-application communication mechanisms, available on Android and iOS. We shall see how, by changing the role played by the browser, this new software architecture makes it possible to circumvent a major obstacle to the deployment of cryptographic and biometric authentication.

All of this has led us to rethink user authentication. We have identified a useful distinction between *closed-loop* and *open-loop* authentication, defined the concept of a *protocredential*, and articulated a comprehensive approach to user authentication based on these concepts that provides both security and user convenience.

We say that authentication is *closed-loop* when the same party that issues or registers a credential is later responsible for verifying possession of the credential at authentication time. Closed-loop authentication is very commonly used. Authentication to a web site by username and password without third party involvement is one example. Another example is third-party login using double-redirection protocols such as SAML Browser SSO [23], Shibboleth [24], OpenID [25], OAuth [26] or OpenID Connect [27], where a relying party redirects the browser to an identity provider who authenticates the user before redirecting the browser back to the relying party.[2]

On the other hand, we say that authentication is *open-loop* when the party that issues or registers the credential is not the party that later verifies the credential (although it may assist with verification, e.g. by providing the status of a certificate in response to an OCSP [28] request). Examples of open-loop authentication include submission of a traditional X.509 public key certificate [29] together with proof of knowledge of the corresponding private key, and presentation of a privacy-enhancing credential such as a U-Prove token [30, 31, 32] or

[2]Some of these protocols use POST redirection achieved by using JavaScript code to submit a POST request, instead of or in addition to GET redirection, which uses an HTTP status code such as 302.

an Idemix anonymous credential [33, 34].

A *protocredential* is a data structure stored in the user's device that can be used to regenerate a credential in conjunction with user input. The protocredential contains one or more stored secrets and serves as a primary authentication factor, while the user input supplies a set of one or more non-stored secrets, such as a PIN or a biometric sample, which serve as additional authentication factors.

In this paper we are concerned with protocredentials that regenerate credentials comprising uncertified[3] key pairs pertaining to public key cryptosystems; a protocredential can also be used to regenerate a symmetric secret for closed-loop authentication, but that results in a weaker security posture, as discussed in Section 4.1 below.

We propose to use a protocredential-based authentication technique for multifactor closed-loop authentication that provides crucial benefits for mobile devices that can be easily lost or stolen. An adversary who gains access to the user's device and is able to read the protocredential cannot impersonate the user without the non-stored secrets. Furthermore, the adversary cannot mount an offline guessing attack against the non-stored secrets, nor against cryptographic parameters derived from the non-stored secrets, because all sets of non-stored secrets, or at least most of them, yield well-formed credentials when combined with the protocredential; those credentials can only be tested against an online party, which can limit the number of guesses. Additional security benefits are discussed in Section 4.

Two-party closed-loop cryptographic authentication is a perfect solution for replacing passwords on the web. But whereas cryptographic closed-loop authentication provides full privacy when only two parties are involved, it provides less privacy than open-loop authentication when an authoritative third party is relied upon to assert user attributes, because that third party observes the authentication transactions in closed-loop authentication. Protocredential-based authentication is intended for multifactor closed-loop authentication, not for open-loop authentication or one-factor closed-loop authentication. However, we have devised a data protection technique mediated by protocredential-based authentication that can be used to protect credentials used in open-loop authentication or one-factor closed-loop authentication when a device containing them is captured by an adversary.

Research on privacy-preserving biometrics has produced several techniques for biometric authentication that can shield the user's biometric features from an adversary who captures the user's device or breaches the user database. We explain how some of these techniques, for example [36, 37, 38], can allow a biometric sample to be used for protocredential-based authentication.

A recent paper [39] has documented how difficult it is to implement, document and use a cryptographic API. We propose a flexible architecture for both closed-loop and open-loop user authentication that insulates application developers from the dangers of cryptographic APIs by encapsulating the complexities of cryptographic (and biometric) computations in a Prover Black Box (PBB) located in the user's device and a Verifier Black Box (VBB) located online. The architecture was originally designed for mobile devices, where it takes advantage of the inter-application communication mechanisms of Android and iOS, but it

[3] We use the term *uncertified* to refer to a public key not bound to the user's identity or attributes by a public key certificate, or to a key pair whose public key component is not so bound. The term *raw public key* has been used to refer to an uncertified public key [35].

can be adapted for web-based applications accessed from traditional personal computers (PCs), by means of browser extensions.

The rest of the paper is organized as follows:

- In Section 2 we describe protocredential-based authentication, in general terms and in the particular case where a passcode such as a PIN is used to regenerate the uncertified key pair from the credential. We provide specific key-pair regeneration techniques for DSA, ECDSA, and RSA credentials, as well as a generic but often less efficient technique applicable to any public key cryptosystem.

- In Section 3 we explain how privacy-preserving biometrics can be used for protocredential-based authentication.

- In Section 4 we analyze the security of protocredential-based two-party authentication and compare it to the security provided by alternative approaches to two-party authentication.

- In Section 5 we explain how protocredential-based authentication can be used to provide effective data protection, contrasting our approach with the data protection mechanism available on iOS. We also show how data protection provides a form of single sign-on (SSO).

- In Section 6 we explain how open-loop authentication provides more privacy than closed-loop authentication when a third-party is relied upon to assert user attributes, we describe the privacy features provided by public key certificates, U-Prove tokens and Idemix credentials, and we explain how credentials used for open-loop authentication can be securely stored without tamper resistance by relying on data protection mediated by protocredential-based authentication.

- In Section 7 we describe the authentication architecture that we are proposing for mobile devices, showing how it supports web-based applications and applications with a native front-end; one-factor and multifactor authentication; as well as two-party closed-loop authentication, third-party closed loop authentication including social login, and open-loop authentication with public key certificates, U-Prove tokens and Idemix anonymous credentials. We also show how login sessions are implemented in the architecture, and how shared login sessions provide a second form of SSO.

- In Section 8 we describe the security provided by the mobile architecture.

- In Section 9 we show how the mobile authentication architecture of Section 7 can be adapted for use on traditional PCs by means of browser extensions.

- In Section 10 we discuss the privacy benefits provided by different types of credentials in the framework of our approach and the privacy implications of the proposed use of biometrics.

- In Section 11 we recapitulate and argue that, by focusing on mobile devices, our approach to authentication has the potential to enable widespread adoption of cryptographic and biometric techniques for authentication on the Internet.

2 Protocredential-Based Authentication

A protocredential can be used to implement multifactor closed-loop authentication, with possession of the credential serving as one authentication factor, and the non-stored secrets supplied by the user serving as additional factors.[4] In two-party multifactor closed-loop authentication, the protocredential is used to regenerate a credential that authenticates the user to an application back-end[5] as a returning user. In third-party multifactor closed-loop authentication, the regenerated protocredential authenticates the user to an identity or attribute provider, which in turn conveys the user's identity or attributes to an application back-end playing the role of relying party.[6] This section is concerned with credential regeneration and use in the context of two-party authentication. Credential regeneration and use is identical in the third-party case, with the identity or attribute provider substituted for the application back-end. How identity or attributes are conveyed to a relying party in third-party closed-loop authentication is discussed in Section 7.

2.1 Overview

Figure 1 illustrates the data structures used in two-party protocredential-based authentication. The user's device is shown with rounded corners to suggest a smart phone or a tablet, but the technique is applicable to any device, including traditional laptops and desktops.

Multifactor authentication is achieved by using one or more non-stored secrets supplied by the user to regenerate the uncertified key pair and thus enable the mobile device to authenticate to a mobile application back-end. Use of one non-stored secret, such as a PIN or a biometric sample, amounts to two-factor authentication, one factor being possession of the mobile device containing the protocredential, the other factor being the knowledge of the non-stored secret or the ability to produce it. Use of both a PIN and a biometric sample amounts to three-factor authentication.

The application maintains user accounts. The user accounts may be kept, for example, in a relational database, which may be internal to the application as illustrated in the figure,

[4] In this paper we are only concerned with user authentication, so the non-stored secrets are supplied by the user; but the approach could also be used for authentication of autonomous devices, using non-stored secrets produced by Physical Unclonable Functions (PUFs).

[5] By "application back-end" we mean the back-end of a mobile application, which the user accesses through a native front-end, or a web-based application, whose front-end consists of JavaScript and HTML code hosted by a web browser running on a mobile device or a traditional PC, or a web service accessed through an API, or so on. The mobile architecture of Section 7 supports two-party closed-loop authentication, as well as third-party closed-loop authentication, and open-loop authentication.

[6] By "identity" we mean we mean one or more attributes that intentionally identify the user in a particularly context. We also use the more specific term "identifier" to refer to a single attribute that uniquely identifies the user.

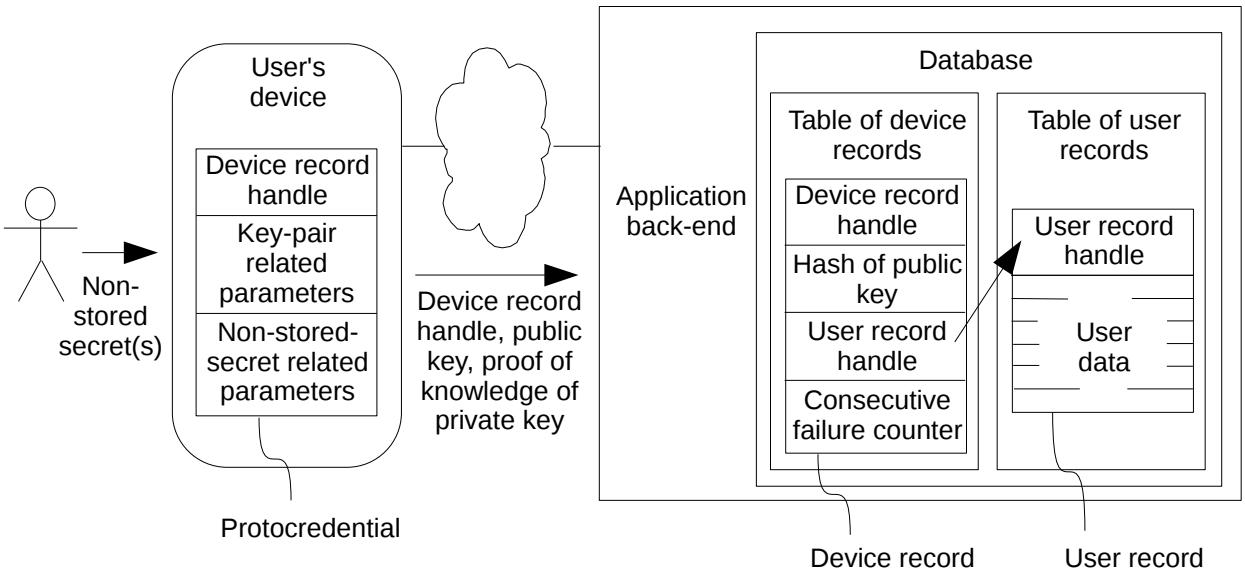


Figure 1: Multifactor two-party authentication using a protocredential.

or external and shared by multiple applications, as in the case of an enterprise directory shared by multiple enterprise applications.

The database contains user records and device records. Each user record is uniquely identified by a user-record handle, and each device record by a device-record handle; we use the word *handle* instead of the database term *primary key* to avoid confusion between database keys and cryptographic keys.

In addition to the device-record handle, each device record contains:

- A user-record handle that identifies the record of the user who owns the device. Multiple device records may reference the same user record, since a user may use, e.g., both a smart phone and a tablet.

- The hash of the public key component of an uncertified key pair that the device uses to authenticate to the application back-end. Different key pairs are used for different applications to ensure unlinkability across applications.

- A counter of failed authentication attempts, which serves as a countermeasure against an attack by an adversary who captures the device, reads the protocredential, and tries to regenerate the uncertified key pair by guessing the non-stored secrets or cryptographic parameters derived from the non-stored secrets. When the counter reaches a low limit, such as 3, 5 or 10, the device record is deleted or disabled.

A user record may contain user identifiers, user attributes, and any other data pertaining to the user. The database may contain session records (not shown in the figure), each session record being used to keep track of a login session on a particular device and referring to the corresponding device record via its device-record handle.

The mobile device authenticates to the application back-end using a credential that consists of a device-record handle and an uncertified key pair. The device contains a protocredential that comprises the device-record handle, cryptographic parameters related to the key pair, and cryptographic parameters related to the non-stored secrets. The key pair is regenerated at authentication time from the cryptographic parameters stored in the protocredential and the non-stored secrets provided by the user.

The device-record handle plays a role similar to the role that the username plays in a username-and-password credential: it makes a device-identity claim by identifying a particular device record in the database. We emphasize that the claimed device identity is relative to the database, and cannot be correlated across applications that use different databases.

The key-pair related parameters depend on the cryptosystem being used. For example, in the case of an RSA key pair, the key-pair related parameters are $\lambda = \mathrm{LCM}(p-1, q-1)$ where p and q are the factors of the modulus and $\mathrm{LCM}(p-1, q-1)$ is the least common multiple of $p-1$ and $q-1$, as well as p and q themselves if the Chinese Remainder Theorem (CRT) is used to improve signing performance. (See Section 2.6.4 below.) When a PIN is used as the only non-stored secret there is one parameter related to the non-stored secret, a salt used to compute a randomized hash of the PIN.

In general terms, authentication proceeds as follows. The non-stored secrets and the parameters in the protocredential are used to regenerate the key pair as described below in Section 2.6. The device establishes a secure connection to the application back-end, which it uses to send the device-record handle and the public key component of the key pair and demonstrate knowledge of the private key. (By *secure connection* we mean a channel providing confidentiality, message integrity, and destination authentication, e.g. a TLS connection. A secure connection has a source and a destination, but it carries bidirectional traffic. The term "source" refers to the endpoint that initiates the connection—the "client" in a TLS connection—and the term "destination" to the other endpoint—the "server" in a TLS connection. In this case the source of the connection is the device, and the destination is the application back-end.) The application back-end uses the device-record handle to locate the device record, computes the hash of the public key, and verifies that the computed hash agrees with the hash stored in the device record. It then uses the user-record handle in the device record to locate the user record and obtain user data.

If the device record cannot be located or is disabled, or the hashes do not agree, or the user record cannot be located or is disabled, then authentication fails, the counter of consecutive failed attempts is incremented, and if the counter has reached its limit, the device record is deleted or marked as disabled.

The public key and its hash must be kept secret to prevent an offline brute-force attack by an adversary who tries to regenerate the key pair by guessing the non-stored secrets or cryptographic parameters derived from the non-stored secrets after having captured the device and read the protocredential. (Since the public key must be kept secret, the reader may ask whether a symmetric secret could be used instead of a key pair. A symmetric secret could indeed be used, but the resulting security posture would be substantially weaker, as discussed below in Section 4.1.)

The device demonstrates knowledge of the private key by performing a private key operation on a challenge. From now on, for simplicity, we shall only consider public key cryptosystems that provide digital signatures, such as DSA, ECDSA or RSA; a private key

operation on a challenge is then a signature on that challenge. The challenge includes a nonce sent by the application back-end and, if the secure connection is a TLS connection, the TLS server certificate used by the application back-end in the TLS handshake. (If a different kind of secure connection is used, some equivalent data presented by the back-end for authentication during connection establishment is included in the challenge instead of the TLS server certificate.)

The secure connection protects the confidentiality of the public key and prevents man-in-the-middle attacks. If the secure connection is a TLS connection with server authentication, a man-in-the-middle attack is prevented even if the attacker is able to use a spoofed TLS server certificate signed by a compromised CA [40], or by a rogue or fictitious CA trusted by the browser. The middleman can relay the nonce from the back-end to the device, and the signature on the challenge from the device to the back-end; but the challenge signed by the device includes the spoofed server certificate, while the back-end expects a signature on a challenge that includes its own legitimate certificate.

An alternative to including the nonce and the certificate in the challenge, in the case of a TLS connection, is to include the TLS master secret.[7] However this would make it impractical for the application back-end to verify the challenge if the TLS connection is terminated at a reverse proxy.

2.2 Two-Factor Authentication with Protocredential and PIN

Figure 2 illustrates the particular case where there is one non-stored secret, which is a passcode such as a PIN entered by the user.

A randomized hash of the passcode is computed using a salt, which is included in the protocredential as a non-stored-secret related parameter and is itself a stored secret. The randomized hash can be computed using a key derivation function (KDF) such as HKDF [41], or the PRF algorithm of TLS [15, §5], or PBKDF2 [42], as discussed below at the end of Section 4.2. The randomized hash is used to regenerate the key pair, playing the role of the variable c of Sections 2.6.1, 2.6.2, 2.6.3 and 2.6.4 below.

2.3 Credential Revocation

The credential used by a device can be revoked by deleting or disabling the device record.

2.4 Preserving Access to the User's Account

The device records pointing to a given user record are alternative means available to the user for accessing her account; if one device is lost and/or the credential used by the device is revoked, the user can continue accessing her account from another device. Additional authentication means may also be available. For example, the application may generate a

[7] Including the master secret and the TLS server certificate in the challenge would be equivalent to TLS client authentication with an uncertified public key as specified in [35], except for the fact that the public key is sent in the clear in [35], while, in our approach, it is sent encrypted after the TLS connection has been established. Providing confidentiality for client credentials has been considered several times by the TLS working group (it was first proposed in August 2000 by the first author of this paper) but never adopted.

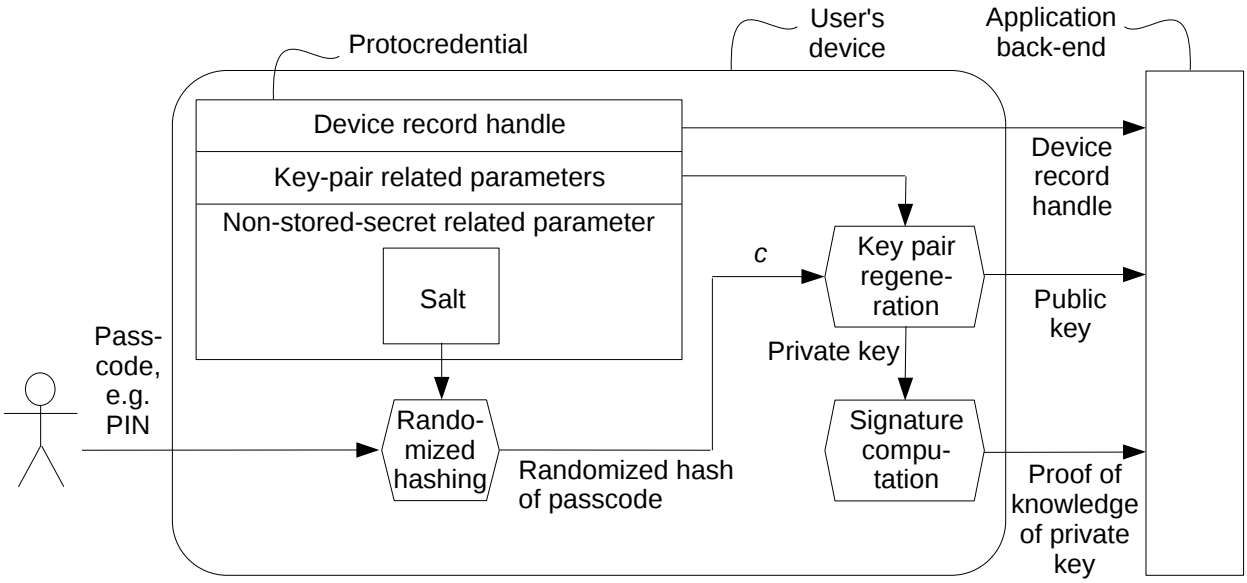


Figure 2: Two-factor authentication with a PIN.

random high-entropy master password when the user account is created, which the user may print and store in a safe place as a last-resort means of accessing the account. The user could also have an ordinary username-and-password credential for accessing the account from a desktop. Data supporting each of such additional authentication means (such as a password hash and salt) may kept in a separate database record pointing to the user record, analogous to a device record, or in the user record itself. Having multiple authentication means, it is easy to recover from the loss of any one of them, simply by using an alternative one. This obviates the need for ad-hoc recovery methods such as security questions posed by the system or by a help desk, which impinge on the user's privacy and are a weak link in the security chain.

2.5 Device Registration

A device may be registered as a means of accessing a user account, either as the account is created, or after the account has been created. During the registration process, the device obtains non-stored secrets from the user, such as a PIN and/or a biometric sample, and uses them to generate an uncertified key pair. Then the device sends the public key to the application back-end over a secure connection and demonstrates knowledge of the private key. Finally the device creates a protocredential containing a device record handle and parameters produced by the key generation process, which will be later used to regenerate the key pair at authentication time in conjunction with non-stored secrets provided anew by the user. The device record handle may be created by the application back-end and sent to the device, or it may be created by the device and sent to the back-end; in the latter case, the possibility of duplicate record handles in the database can be ignored if the device handle is a random high-entropy string.

If the account already exists when the device is registered, the user must authenticate as the owner of the account. This is preferably done by supplying a short-lived device registration code acquired after authenticating on a desktop or laptop, or on a different mobile device; the code may be, for example, a random 16-decimal-digit code with a validity period of one minute.[8] It can also be done by authenticating with a long term non-cryptographic credential such as a username-and-password credential or a master password.

2.6 Credential Regeneration

A credential used in protocredential-based authentication consists of a device-record handle and an uncertified key pair. Credential regeneration amounts to regeneration of the key pair using parameters stored in the protocredential, including at least one secret parameter, and non-stored secrets supplied by the user. There is a generic credential regeneration algorithm that is applicable to any public key cryptosystem, but is often inefficient. We describe the generic algorithm first, then we describe specific efficient algorithms for the three cryptosystems that are most commonly used for cryptographic user authentication: DSA, ECDSA and RSA; DSA and ECDSA are NSA Suite B cryptosystems [43].

2.6.1 Generic Algorithm

A key pair of any public key cryptosystem is generated by an algorithm that uses random inputs. Those random inputs are provided by a random number generator or an entropy accumulator.[9] If instead we use a pseudo-random number generator whose output is determined by a seed, then the same key pair can be regenerated later by providing the same seed.

Therefore the following is a generic approach to credential regeneration from a protocredential, applicable to any public key cryptosystem:

- Initial key pair generation:

    1. Obtain the non-stored secrets from the user.
    2. Generate one or more random high-entropy strings that will serve as non-stored-secret related parameters, and will be stored secrets kept in the protocredential.
    3. Derive a pseudo-random value c from the non-stored secrets and the non-stored-secret related parameters, as described in Section 2.2 for the case where a passcode such as a PIN is the only non-stored secret, in Section 3.3 for the case where a biometric sample is the only non-stored secret, and in Section 3.4 for the case

[8] On a mobile device it is best to use a numeric code, since it is much easier to enter digits on a numeric touchscreen keypad than mixed alphanumeric characters on small touchscreen keyboard.

[9] An entropy accumulator can be viewed as a deterministic random bit generator [44] that is reseeded from any number of sources of entropy whenever new entropy becomes available from those sources. A "blocking" entropy accumulator estimates the amount of entropy that it has received and the amount that it has "spent", and blocks the caller if no entropy is left, until more is obtained. A non-blocking accumulator produces pseudorandom output as needed without keeping track of the entropy that has been received and spent. Linux and other Unix-like systems provide a blocking entropy accumulator at /dev/random, and a non-blocking one at /dev/urandom.

where there are two non-stored secrets, one being a biometric sample and the other being a PIN.

4. Initialize a deterministic pseudo-random number generator with c as the seed.
5. Run the key pair generation algorithm for the cryptosystem using the deterministic pseudo-random number generator as the source of randomness.
6. Obtain or generate the device record handle as described above in Section 2.5.
7. Create a protocredential comprising the non-stored-secret related parameters, and the device record handle.

- Subsequent key pair regeneration:

1. Obtain anew the non-stored secrets from the user. If a biometric sample is one of the non-stored secrets, the authentication sample provided by the user for key pair regeneration will not be identical to the reference sample provided by key pair generation; but the value c derived in the next step will be the same if the two samples are not too different, as explained in Section 3.2.
2. Obtain the non-stored-secret related parameters from the protocredential.
3. Derive the pseudo-random value c from the newly obtained non-stored secrets and the non-stored-secret related parameters in the same manner as during initial generation.
4. Initialize the same deterministic pseudo-random number generator used during initial generation, with c as the seed.
5. Run the key pair generation algorithm for the cryptosystem using the deterministic pseudo-random number generator as the source of randomness.

This generic method is often inefficient because generating a key pair is computationally costly; for example, generating an RSA key pair includes generating the prime factors p and q. A big performance gain may often be achieved by retaining in the protocredential some intermediate results produced during initial generation, so that they do not have to be recomputed. In the case of RSA, for example, we retain $\lambda = \mathrm{LCM}(p-1, q-1)$, or p and q if the Chinese Remainder Theorem (CRT) is to be used to improve signing performance. In the case of DSA and ECDSA, the intermediate results that we retain are the *domain parameters* [45], which are public parameters that may (or may not) be common to a group of public keys. If the domain parameters are fixed, and their generation is not part of key pair generation, then the specific regeneration methods described below for DSA and ECDSA are essentially equivalent to the generic method (but simpler).

2.6.2 DSA

Using the notations of the Digital Signature Standard (DSS) [45, §4], the DSA cryptosystem comprises the following parameters: a prime modulus p, a prime q that divides $p-1$, a generator g of the cyclic subgroup of order q of the group $((\mathbb{Z}/p\mathbb{Z})^*, \times)$, an integer x in the range $1 \leq x \leq q-1$, and $y = g^x \bmod p$; x is a private parameter; p, q, g and y

are public parameters, of which p, q and g are domain parameters. The private key is x. Sometimes the public key is considered to include all the public parameters [46, §11.5.1] (p, q, g, y), sometimes it is considered to be just y [45, §4], excluding the domain parameters. For our purposes, the public key consists of those public parameters that the device sends to the application back-end and are hashed into a field of the device record; it will thus be (p, q, g, y) unless the mobile device and the application operate within a domain where the same domain parameters p, q and g are used by all mobile devices and configured into the application.

During initial key pair generation and construction of the protocredential, the domain parameters are generated as specified in [45, Appendix A], unless they are fixed; one or more non-stored secrets are obtained from the user; one or more non-stored-secret related parameters are generated as random high-entropy strings; x and y are computed from the non-stored secrets and the non-stored-secret related parameters; the device record handle is obtained and generated as described in Section 2.5; and the protocredential is created, comprising the device-record handle, the domain parameters, and the non-stored-secret related parameters.

During credential regeneration, non-stored secrets are obtained anew from the user, and x and y are computed from the non-stored secrets and the non-stored-secret related parameters. The resulting credential consists of the device record handle, the private key x, and the public key, which is (p, q, g, y) or simply y if the domain parameters are configured in the application back-end.

The following process is used to compute x and y during initial generation as well as during regeneration:

1. N being the bitlength of q, derive a pseudo-random value c of bitlength $N + 64$ from the non-stored secrets and the non-stored-secret related parameters as described in Section 2.2 for the case where a passcode such as a PIN is the only non-stored secret, in Section 3.3 for the case where a biometric sample is the only non-stored secret, and in Section 3.4 for the case where there are two non-stored secrets, one being a biometric sample and the other being a PIN. (We follow [45, §B.1.1] in making c be 64 bits longer than q. The purpose of this is to minimize the bias in the distribution of x caused by the mod operation in the following step. If c were roughly uniformly distributed over the strings of length N, x would be substantially more likely to belong to $[1, 2^N - q]$ than to $[2^N - q + 1, q - 1]$.)

2. Compute $x = (c \bmod (q - 1)) + 1$, so that $1 \leq x \leq q - 1$.

3. Compute $y = g^x \bmod p$.

2.6.3 ECDSA

Using the notations of the DSS [45, §6] and the Certicom white paper [47], the ECDSA cryptosystem comprises the private key d, the public key Q, and the domain parameters q, FR, a, b, σ, G, n and h, where q is the size of the field, FR is an indication of the field representation, a and b are the coefficients used in the curve equation, σ is the domain parameter seed, used for public key validation, G is the point of the curve used as the

generator of a cyclic group, n is the order of G, h is the cofactor, d is an integer in the interval $[1, n-1]$, and $Q = dG$.

As in the case of DSA, the protocredential comprises the device-record handle, the domain parameters unless they are configured into the application, and the non-stored-secret related parameters. Credential generation and regeneration are as in the case of DSA, *mutatis mutandis*. The following process is used to compute d and Q during initial generation as well as during regeneration:

1. N being the bitlength of n, derive a pseudo-random value c of bitlength $N + 64$ from the non-stored secrets and the non-stored-secret related parameters as described in sections 2.2, 3.3 and 3.4.

2. Compute $d = (c \bmod (n-1)) + 1$, so that $1 \leq d \leq n-1$.

3. Compute $Q = dG$. (The multiplicatively-written operation dG involving the integer d and the point G is the one induced by the additively written operation in the group of points of the elliptic curve.)

2.6.4 RSA

With the notations used in the DSS [45, §5][10] and in sections 8.2 and 11.3 of [46], the parameters of the RSA cryptosystem include the prime factors p and q, the modulus $n = pq$, the decryption/signature exponent d, and the encryption/verification exponent e; p, q and d are private parameters while n and e are public parameters. The public key is (n, e). The private key is often said to be d; but signing requires (n, d), or (p, q, d) if the CRT is used, so the private key may also be said to be (n, d) or (p, q, d) depending on the circumstances. We shall also use the additional private parameter $\lambda = \mathrm{LCM}(p-1, q-1)$, and we let l be the bitlength of n.

Key-pair regeneration is more complicated for RSA than for DSA or ECDSA for two reasons:

1. d must be relatively prime with λ, so that the encryption exponent e can be computed as the inverse of d modulo λ, $e = d^{-1} \pmod{\lambda}$.

2. d should not be too small, to avoid attacks against small decryption exponents; the DSS [45, §B.3.1] requires d to have at least $l/2$ bits.[11]

These requirements can be met by using the following process for initial key-pair generation, after obtaining one or more non-stored secrets from the user and generating one or more non-stored-secret related parameters:

1. Generate p and q as described in [45, §B3].

[10] The DSS specifies the use of RSA as a signature scheme, in addition to specifying DSA and ECDSA.

[11] A small encryption exponent attack allows the adversary to compute the private key from the public key. Since our approach calls for keeping the public key secret, it provides some protection against such attacks. But allowing vulnerable small decryption exponents is tantamount to using a symmetric secret rather than a key pair, which results in a weaker security posture as explained in Section 4.1.

2. Compute $\lambda = \text{LCM}(p-1, q-1)$.

3. Derive a pseudo-random value c of bitlength $l+64$ from the non-stored secrets and the non-stored-secret related parameters as described in sections 2.2, 3.3 and 3.4.

4. Compute $c' = (c \bmod (\lambda - 1)) + 1$, so that $1 \leq c' \leq \lambda - 1$.

5. Compute $c'' = \text{GCD}(c', \lambda)$.

6. Compute the decryption exponent d as $d = c'/c''$.

7. If the bitlength of d is not greater than $l/2$, start over at step 1.

8. Compute the encryption exponent e as $e = d^{-1} \pmod{\lambda}$.

The probability of having to start over at step 7 depends on how c, p and q are generated, but we argue in Appendix A.1 that it is likely to be negligible.

The protocredential includes the device-record handle, the key-pair related parameter λ, and the non-stored-secret related parameters; it also includes p and q if the CRT is used to improve signing performance. At authentication time, after obtaining anew the non-stored secrets from the user, the key-pair is regenerated by running again steps 3–8 of the key-pair generation process, except that authentication fails at step 7 if the bitlength of d is not greater than $l/2$.

The key-pair regeneration process includes two expensive operations: a run of the euclidean algorithm to compute c'', and a run of the extended euclidean algorithm to compute e. It would be tempting to include c'' in the protocredential to avoid its computation during key-pair regeneration. But since c'' is derived from the non-stored secrets, that would facilitate an offline attack by an adversary who has captured the device and obtained the protocredential; for example, if a PIN is the only non-stored secret, it would help the attacker guess that PIN.

A good way of eliminating the run of the euclidean algorithm to compute c'' is to check during initial key-pair generation that c' and λ have no large prime factor in common, e.g. no common prime factor greater than 100, starting over at step 1 if the condition is not met; the list of small factors of λ can be kept in the protocredential, and during key-pair regeneration we can then compute d by repeatedly dividing c' by elements of the list until it is no longer divisible by any of them. The probability of having to start over during initial generation depends on the probability distributions of c' and λ. However, it should be similar to the probability of two independently generated random numbers having a common prime factor greater than 100. We show in Appendix A.3 that this probability is less than 0.2%. This means that it is very unlikely that there will be multiple restarts. It also means that, at authentication time, an adversary who has obtained the protocredential and is guessing sets of non-stored secrets will find that only 0.2% of the guesses fail to produce a well-formed key pair (actually, fail to produce a key pair at all, because e cannot be computed from d and λ). The remaining 99.8% of the guesses will produce a well-formed key pair that can only be tested online and will be subject to a small limit on the number of online guesses.

3 Using Biometrics for Protocredential-Based Authentication

3.1 Security, Privacy and Usability Concerns with Biometrics

Biometric authentication allows an individual to demonstrate her identity by exhibiting a physical feature, such as the likeness of her face, the pattern of ridges on the skin of a finger, or the texture and/or pigmentation of on of her irises. This can be a very strong form of authentication when there is *assurance of liveness*, i.e. assurance that the biometric sample submitted for authentication belongs to the individual seeking authentication. Assurance of liveness is relatively easy to obtain in person-to-person authentication, e.g. when a shopper shows a photo ID while making an in-store credit card purchase, or when a US government employee submits to a fingerprint scan in the presence of a guard to gain entry to a secure building. But assurance of liveness is difficult to achieve for unattended person-to-machine authentication, and even more difficult to achieve for remote authentication, when a biometric sample is acquired by a device in possession of the user, who may be the adversary.

When there is no assurance of liveness, security must rely on the relative secrecy of biometric features, which depends on the circumstances. It would not be wise to rely on a fingerprint as an authentication factor for authentication to an application back-end from a smart phone that may have been captured by an adversary, since the phone will be covered with fingerprints of the legitimate user; but it may be reasonable to use an iris scan.

Biometric authentication also raises privacy concerns. If no precautions are taken, the use of biometric features for authentication to an application could be linked not only to other online uses, but also to the offline everyday activities of the user. Unlinkability is therefore a stronger requirement for biometric than non-biometric authentication.

Confidentiality of the user's biometric features is also a strong requirement, for three reasons. First, the linking of the user's biometric features to the user's identity could link the user's identity to both the user's online and offline activities. Second, it would allow an adversary to impersonate the user vis-a-vis an application that relied exclusively on biometric features for authentication. Third, since raw biometric features are not revocable, the loss of the a biometric feature as a secure authentication factor would be irreversible.

Much work has been done to address these privacy and security concerns using techniques that combine biometrics, error correction and cryptography [36, 48, 49, 37, 50, 38, 51]. An overview can be found in [52] and a survey in [53]. A common starting point for all these techniques is the observation that, although raw biometric data is linkable and not revocable, a randomized derivative of biometric data may be unlinkable and revocable.

Privacy risks remain, however. For example, if malware is present in a mobile device while the device is being used by the legitimate user, such malware may be able to surreptitiously acquire raw biometric data from the legitimate user.

Yet another concern with biometrics is the usability drawbacks of non-negligible false rejection rates. Soldiers, for example, may not want to use biometric authentication on a mobile device during combat operations, when false rejection may have dire consequences.

All these concerns mean that biometric authentication should be used sparingly. But they do not negate the fact that biometrics add a unique dimension to security by asking the

user to demonstrate, as is often said, "something that the user is" in addition to "something that the user knows" and "something that the user has". Biometrics have a role to play as a second or third authentication factor when extra security is required.

We propose to incorporate biometrics into protocredential-based authentication authentication using a popular approach that has been codified in the ISO/IEC 24745 standard [54, Figure 5]. The approach consists of deriving a key, sometimes called a *biometric key*, from a genuine biometric sample and a string that plays the role of auxiliary data. The auxiliary data is itself derived at registration time (a.k.a. enrollment time) from a random string and a reference sample that is discarded. Biometric samples that are similar enough to the reference sample consistently produce the same biometric key. (The biometric key may just be the random string, in which case the biometric key is said to be "locked" by the reference sample, and "unlocked" by the authentication sample.) The auxiliary data is not a traditional biometric template containing a description of biometric features derived from the reference sample. It is supposed to be infeasible to derive any information about biometric features from the auxiliary data.

3.2 Consistently Producing the Biometric Key

One method of consistently producing the same biometric key from genuine but varying biometric samples is to view the samples as *noisy data* [55] and use error correction techniques. For example, an error correction codeword C can be created by adding redundancy to the random string, and the auxiliary data can be computed by bitwise x-oring the codeword with a suitably processed reference sample R:

$$A = C \oplus R.$$

When an authentication sample S is obtained, it is x-ored with the auxiliary data to produce

$$(C \oplus R) \oplus S = C \oplus (R \oplus S)$$

Since R and S are similar, $D = R \oplus S$ only has a few 1 bits, and its effect when x-ored with C is to flip a few bits of C, the same effect that could be produced by transmitting C over a noisy channel. Error correction can therefore be applied to recover the random codeword C from $C \oplus D$. The random string can in turn be recovered from the codeword and serve as the biometric key. This method and closely related ones are used, among others, by Juels et al. [36], Dodis et al. [37], and Hao et al. [38]. We find the work of Hao et al. particularly promising for authentication on mobile devices such as smart phones and tablets because it uses an iris scan, which could be taken with a device camera and may be deemed relatively secret, and because it claims a false rejection rate (FRR) of only 0.47% with a false acceptance rate of 0%.

3.3 Two-Factor Authentication with Biometrics

Any method of producing a biometric key from a genuine biometric sample and auxiliary data can be combined with our credential regeneration technique. Figure 3 illustrates how the combination can achieve two-factor authentication, one factor being possession of the

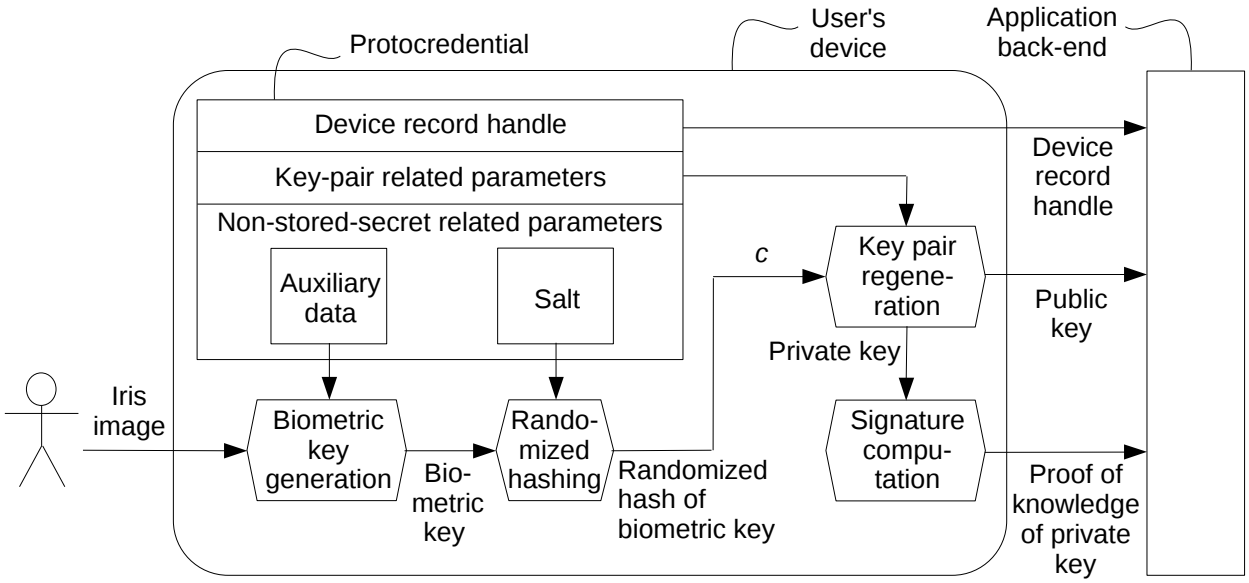


Figure 3: Two-factor authentication with an iris scan.

device that contains the protocredential, the other factor being the genuine biometric sample, e.g. an iris image.

There is one non-stored secret, viz. the iris sample. The auxiliary data is included in the protocredential as a non-stored-secret related parameter, and is itself a stored secret. The iris sample is combined with the auxiliary data to produce the biometric key. A randomized hash of the biometric key is computed using a salt, which is included in the protocredential as a second non-stored-secret related parameter and is itself a stored secret. The randomized hash is used to regenerate the key pair, playing the role of the variable c of Sections 2.6.1, 2.6.2, 2.6.3 and 2.6.4.

The purpose of the randomized hashing is to extend the bitlength of the biometric key and make its distribution roughly uniform before using the resulting hash to regenerate the key pair. Suppose for example that the biometric key is produced by the method of Hao et al. [38], and the cryptosystem being used is RSA with a 2048 bit modulus. The bitlength of the biometric key is 140 bits, and the randomized hashing extends it to $2048+64 = 2112$ bits.

3.4 Three-Factor Authentication

Figure 4 illustrates how a method of producing a biometric key from a genuine biometric sample and auxiliary data can be combined with our credential regeneration technique to achieve three-factor authentication, one factor being possession of the device that contains the protocredential, a second factor being an iris image as above, and the third factor being a PIN. At registration (a.k.a. enrollment) time, the PIN is used to encrypt the auxiliary data. A simple way of doing this is to x-or the auxiliary data with a randomized hash of the PIN of same bitlength as the auxiliary data, computed using a salt (Salt 1 in the figure), which is included in the protocredential as a non-stored-secret related parameter.

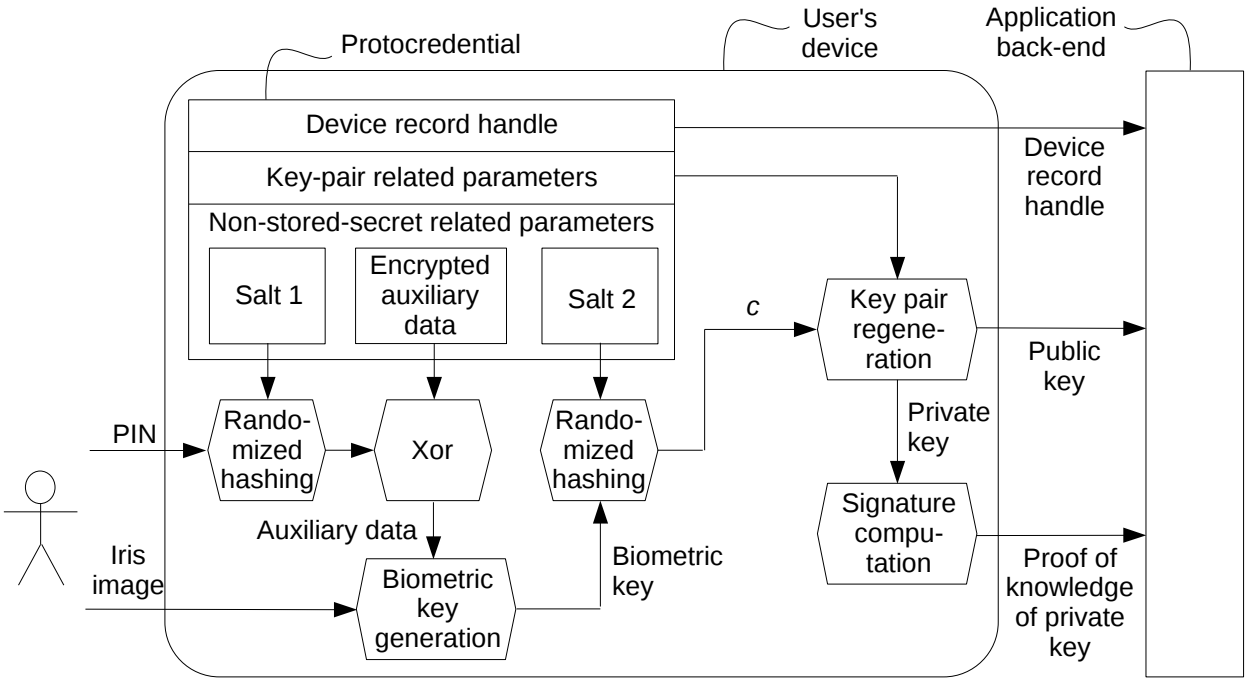


Figure 4: Three-factor authentication with an iris scan and a PIN.

At authentication time, the user supplies the PIN and the iris image. The randomized hash of the PIN with the salt is computed and x-ored with the encrypted auxiliary data to decrypt it. The decrypted auxiliary data is then combined with the iris image to produce the biometric key. A randomized hash of the biometric key is computed using a second salt (Salt 2 in the figure), which is also included in the protocredential as a non-stored-secret related parameter and is itself a stored secret, and the randomized hash is used to regenerate the key pair, playing the role of the variable c of Sections 2.6.1, 2.6.2, 2.6.3 and 2.6.4.

4 Security Analysis of Two-Party Protocredential-Based Authentication

In the following security analysis we assume that no malware is running in the device while the device is in the legitimate user's possession; otherwise no security can be expected, no matter what authentication technique is used. We do consider cases in which the adversary captures the device and installs malware on the device to read the protocredential. Our authentication methods provide protection in those cases, as long as the legitimate user does not use the device after it has been captured without first doing a factory reset.

We assume that an adversary who captures the device is able to read the protocredential contained in the persistent storage of the device (the flash memory of a smart phone, tablet, or Mac Air laptop, or the disk of a traditional laptop or desktop). We assume that the implementation will regenerate the key pair from the secrets stored in the protocredential

and the non-stored secrets supplied by the user immediately before use, and will erase it from virtual memory immediately after use, so that it is not present in the device when the device is captured by the adversary.

A discussion of side-channel attacks, such as observing the electromagnetic emissions of the device, and of countermeasures against such attacks, is outside of the scope of this paper.

4.1 Secrets and Their Purpose

Protocredential-based authentication relies on three secrets or sets of secrets:

1. One or more non-stored secrets such as a PIN or a biometric sample or both, supplied by the user.[12]

2. One or more stored secrets included in the protocredential such as salts used to generate randomized hashes, biometric auxiliary data, or the prime factors of an RSA modulus.

3. The hash of the public key stored in the device record.

The stored and the non-stored secrets are both needed to regenerate the uncertified key pair credential. The need to know the stored secrets included in the protocredential prevents an offline guessing attack against the non-stored secrets or against the cryptographic parameters derived from the non-stored secrets by an adversary who breaches the database and obtains the hash of the public key:

- In two-factor authentication with a PIN, without the need to know the salt it would be easy to crack a 4-digit or 6-digit PIN by brute force.

- In multifactor authentication with a biometric sample, the need to know the auxiliary data and the need to know the salt used to compute a randomized hash of the biometric key are two separate barriers that prevent a brute force guessing attack against the biometric key. Without any of those barriers, an adversary with great computing power might be able to crack the biometric key, which has much less entropy than a biometric sample; in the system of Hao et al. [38]), for example, a biometric sample has 147 bits of entropy, but the biometric key may have as little as 44 bits of entropy if the auxiliary data is known.

Conversely, the purpose of keeping secret the hash of the public key stored in the device record is to prevent an offline attack against the credential by an adversary who captures the device and reads the protocredential stored in the device.

Of course if the hash of the public key is to be a secret, the public key must be kept secret as well, which seems paradoxical. If the public key has to be secret, why not use a symmetric secret rather than a key pair for authentication? A symmetric secret could be used, but the security posture would be weaker from a viewpoint of defense in depth:

[12] As we said above in footnote 4, the non-stored secrets could be supplied by a PUF, but we are not considering autonomous devices in this paper.

- The device could present the symmetric secret to the application back-end as a bearer token[13] and the back-end would compute its hash and compare it with a hash stored in the device record. But then an attacker who defeated the confidentiality provided by the secure connection and obtained the secret could impersonate the user.

- Alternatively, the symmetric secret could be stored in the clear in the device record, and the device could demonstrate knowledge of the secret by using it as a symmetric signature key. But then the secret would have to be stored in the database, and an attacker who breached the database and obtained the symmetric secrets stored in the device records of all the users could impersonate any user.

It could be objected to our approach that the party that controls the application back-end knows the public key, and could thus mount an offline attack to obtain the private key. However, that would require capturing the device and extracting the protocredential. Furthermore, the private key could only be used to impersonate the user vis-a-vis the application itself, or vis-a-vis other applications that share the same database. The concern in such case is loss of non-repudiation; but we argue that, if non-repudiation of a transaction is needed, it should be achieved by a signed description of the transaction itself, rather than by non-repudiable authentication of the connection over which the transaction is conducted.

Notice that only the hash of the public key is stored in the device record, not the public key itself. One reason for this, of course, is that the hash is shorter. There is no security reason for not storing the public key when the cryptosystem used for authentication is DSA or ECDSA; however, there is a security reason when the cryptosystem is RSA. If the RSA public key were stored in the database, an adversary who breached database security and furthermore captured a device and read its protocredential would be able to compute the private key from the public key and the private parameter λ without having to mount an offline attack.

4.2 Security Posture of Authentication with a Key Pair Regenerated from a Protocredential and a Passcode

Table 1 summarizes the security posture of two-factor protocredential-based authentication using a key pair regenerated from a protocredential and a passcode such as a PIN, from a point of view of defense in depth. Each cell in the table shows to what extent the authentication method remains secure, in the sense of resisting user impersonation, when an adversary is able to achieve a variety of security breaches, or combinations of breaches. Specifically, columns 2, 3 and 4 and row B refer to different breaches, and each cell in the table refers to the case where the breach referenced by its column, if any takes place, but not the breaches referenced by other columns; and the breach referenced by its row, if any, takes place *after* the breach referenced by the column, if any, when the order matters.

- Cells in column 2 refer to cases where the adversary has breached the confidentiality of the database and read the hash of the public key stored in the device record.

[13] A bearer token is a secret that authenticates a party that presents it without requiring any further evidence. A password is a bearer token. A session cookie is a bearer token. A certificate is not a bearer token, since it only authenticates a party that is able to prove knowledge of the associated private key.

	1. Adversary cannot breach database or network security	2. Adversary breaches database confidentiality	3. Adversary breaches connection confidentiality	4. Adversary spoofs back-end certificate
A. Adversary cannot read device storage	Secure	Secure	Secure	Secure
B. Adversary captures device, reads its storage	Secure	Security depends on passcode entropy	Security depends on passcode entropy	Security depends on passcode entropy

Table 1: Security posture of protocredential plus passcode.

- Cells in column 3 refer to cases where the adversary has breached the confidentiality of the secure connection from the device to the application back-end and passively observed connection traffic, including the public key. Such a breach could happen, for example, if the back-end is hosted in an infrastructure-as-a-service cloud, the secure connection is a TLS connection that is terminated at a reverse proxy, the adversary is a cloud tenant, and a cloud security breach allows the adversary to read the clear-text traffic that the reverse proxy forwards to one or more web servers of the application back-end.

- Cells in column 4 refer to cases where the secure connection is a TLS connection and the adversary is able to spoof the TLS server certificate used by the application back-end. A spoofed certificate could be obtained from a compromised CA, or from a rogue or fictitious CA whose root certificate has been installed in a root certificate store within the device, e.g. as a result of a prior social engineering attack. We also assume that the adversary is able to divert the underlying TCP connection in order to conduct a TLS handshake with the spoofed certificate.

- Cells in row B refer to cases where the adversary has been able to capture the user's device and read its storage, including the protocredential.

The table shows how authentication remains secure, even with low passcode entropy, if the adversary achieves any single one of the four security breaches being considered.

An adversary who captures the device and reads its storage without having previously breached database or network security (case B1) cannot authenticate without knowledge of the passcode. Furthermore, as pointed out in Section 2.1, the adversary cannot mount an offline guessing attack against the passcode, because most or all passcodes yield well-formed credentials when combined with the protocredential; the credentials can only be tested online against the application back-end, which limits the number of guesses.

An adversary who breaches database or network security in cases A2–A4 can obtain the hash of the public key (in case A2) or the public key itself (in cases A3–A4) but cannot use that information to mount an offline guessing attack against the passcode without knowing the stored secrets contained in the protocredential. Of course knowledge of the public key by itself does not allow the adversary to authenticate without knowledge of the private key, this being an essential benefit of public key cryptography. Furthermore, case A4 is protected against a man-in-the-middle attack because the TLS certificate of the application back-end is included in the challenge signed by the device, as explained in Section 2.1. By contrast authentication with username and password or with an OTP coupled with a PIN or password, is vulnerable to a man-in-the-middle attack in that case.

If the adversary captures the device and obtains the protocredential after breaching database or network security (cases B2–B4), then she can mount an offline guessing attack against the passcode. Each passcode guess can be tested by regenerating the credential from the protocredential and the guess, and checking whether the public key component of the credential agrees with the hash of the public key found in the database (in case B2) or the public key obtained from the network (in cases B3–B4). Authentication security (i.e. prevention of user impersonation) can still be achieved if the passcode has high enough entropy to withstand the offline guessing attack.

If authentication security in cases B2–B4 is a requirement, then the randomized hash of the passcode should be computed using a password-based key derivation function such as PBKDF2 [42] with sufficient iterations, in order to help withstand an offline attack. However, if the probability that an adversary will be able to capture the device after breaching database or network security is deemed negligible, a key derivation function that is not purposefully slow, such as HKDF [41], or the PRF algorithm of TLS [15, §5], should be used instead of PBKDF2 to reduce the amount of computation required. Reduced computation means reduced latency and reduced power consumption; and in mobile devices, reduced power consumption means increased battery life.

If the passcode has sufficient entropy, a man-in-the-middle attack is prevented in case B4 as in case A4.

4.2.1 Countermeasures against Back-End Spoofing

Column 4 refers to cases where the connection to the back-end is a TLS connection and the adversary is able to divert the underlying TCP connection to itself (which is relatively easy to do using for example a rogue WiFi access point [56]) and use a spoofed TLS certificate to conduct the TLS handshake successfully. As explained above in Section 2.1, a man-in-the-middle attack against the user authentication process is prevented in such cases by including the server certificate in the challenge signed by the device, which keeps the back-end from accepting the user credential if forwarded by a middleman. However, the adversary may still

be able to spoof the back-end using the spoofed TLS certificate. Even though that does not allow user impersonation, spoofing the back-end can do much damage; for example, the user can be lured into entering confidential data into the spoofed back-end, thus delivering it to the attacker.

Back-end spoofing can be prevented altogether by using a secondary, non-cryptographic, means of server authentication such as showing a secret image and/or phrase that the user is trained to expect from the legitimate application back-end. This secondary authentication should take place only after the user's device has successfully authenticated by signing a challenge that includes the TLS server certificate of the back-end, ruling out a man-in-the-middle attack.

Secondary server authentication is used today by Bank of America using SiteKey [57]. The secondary authentication takes place after the user's device has authenticated as a registered device, presumably by presenting one or more bearer tokens stored in the browser as cookies, or stored in the persistent storage provided by HTML5 or by a Flash plug-in or a Silverlight plug-in. However, because device authentication relies on bearer tokens, secondary server authentication does not help prevent man-in-the-middle attacks [58].

4.3 Comparison with Other Security Postures

We now compare the protocredential-plus-passcode authentication method to five authentication methods that are similar to our method and/or popular. A summary can be found below in Section 4.3.6.

4.3.1 Encrypted Private Key

Table 2 shows the security posture of two-factor authentication using a key pair whose private key component is encoded as specified by PKCS #8 [59] and related or similar formats, and encrypted under a symmetric data-encryption key derived from a passcode. This kind of two-factor authentication is often used, for example, to manage cloud-hosted virtual servers over SSH.

The security posture of this approach is weaker than that of our passcode-plus-protocredential method in case B1, because an adversary who reads the device storage and obtains the encrypted private key can mount an offline attack against the passcode, testing each passcode guess by decrypting the private key and checking if the result of the decryption is properly formatted.[14] The user must therefore choose a high-entropy passcode (a short

[14]The PKCS #8 structure is encoded in ASN.1 notation, typically using the Distinguished Encoded Rules (DER) as recommended by [59], and may contain other cryptographic parameters in addition to the private key. It typically includes the public key or a certificate containing the public key. To test each passcode the adversary derives a data-encryption key from the passcode, uses the derived key to decrypt the encrypted structure, uses an ASN.1 parser to decode the structure, verifies that the decoded structure is as specified by PKCS #8 and, if the decoded structure contains a public key, verifies that the private key matches the public key. For a passcode other than the correct one, the test will fail with very high probability even if there is no public key in the structure. (It is likely to fail at the very first step, ASN.1 parsing.) A private key encrypted by itself without first encoding it in ASN.1 or in any other format that would add structure (and hence redundancy) to the plaintext, with additional cryptographic parameters left in the clear, could be considered a protocredential and used with our authentication method. The public key would have to be

	1. Adversary cannot breach database or network security	2. Adversary breaches database confidentiality	3. Adversary breaches connection confidentiality	4. Adversary spoofs back-end certificate
A. Adversary cannot read device storage	Secure	Secure	Secure	Secure
B. Adversary captures device, reads its storage	Security depends on passcode entropy	Security depends on passcode entropy	Security depends on passcode entropy	Security depends on passcode entropy

Table 2: Security posture of private key encrypted under passcode.

PIN is out of the question). This makes this method impractical on mobile phones, because entering a high-entropy passcode on the small touchscreen keyboard of a mobile phone is cumbersome. The security posture is the same as that of our passcode-plus-protocredential method in the other cases.[15]

omitted and treated as a shared secret between the device and the application back-end. After decrypting the private key, the public key could be computed from the private key and additional cryptographic parameters, and sent to the back-end along with the proof of knowledge of the private key and the device record handle. Alternatively, in cases where the public key could not be computed (e.g. in the RSA case if the only additional parameter were the modulus) the public key would not be sent the back-end. The back-end would then have to store the public key instead of a hash of the public key in the device record, and retrieve the public key from the database using the device record handle in order to verify the proof of knowledge.

[15]In particular, this method is resistant to a man-in-the-middle attack in cases A4 and B4 if knowledge of the private key is demonstrated by using the private key to sign a challenge that includes a TLS server certificate. SSH does not use TLS, of course, but when the client authenticates with a key pair it signs a hash that includes the server's Diffie-Hellman ephemeral public key and the server's long term RSA public key, which similarly prevents a man-in-the-middle attack. The security considerations section of the SSH specification [60, §9.3.4] misses this point. It warns that that the protocol is vulnerable to a man-in-the-middle attack in the usual case where the server public key is not validated, failing to make the distinction between client authentication with a bearer token, which allows the man-in-the-middle attack, and client authentication with a key pair, which does not.

	1. Adversary cannot breach database or network security	2. Adversary breaches database confidentiality	3. Adversary breaches connection confidentiality	4. Adversary spoofs back-end certificate
A. Adversary cannot read device storage	Secure	Secure	Secure	Secure
B. Adversary captures device, reads its storage	Somewhat secure	Security depends on passcode entropy	Not secure	Not secure

Table 3: Security posture of key pair plus independent passcode.

4.3.2 Passcode Independent of Key Pair

Table 3 shows the security posture of two-factor authentication using a key pair, plus an independent passcode that is verified separately by the application back-end. The key pair authenticates the device and the passcode authenticates the user.

In the cases where the adversary does not capture the device (A1–A4) the method is secure in the sense of not allowing user impersonation. However, although not apparent in the table, the security posture in cases A2 and A3 is weaker than that of the methods of Sections 4.2 and 4.3.1. In case A2, a database security breach allows the adversary to obtain the salted hashes of the passcode associated salts for all users. The adversary can then mount an offline attack against the passcodes, which many will not withstand. In case A3, a breach of network confidentiality allows the adversary to obtain the passcode. In either case, the adversary may be able to exploit the passcode if the user has used it for other purposes. Case A4 is further discussed below together with case B4.

If the adversary captures the device and reads its storage, thus obtaining the private key in the clear, but has not previously breached the security of the database or the network (case B1), she needs the passcode to authenticate, and she cannot mount an offline guessing attack against the passcode. But an independent passcode is vulnerable to phishing attacks and attacks that exploit passcode reuse, so the adversary may have obtained the passcode before capturing the device; we thus rate the authentication method in this case as only "somewhat secure".

If, before obtaining the private key, the adversary has breached database confidentiality

(case B2) and obtained the salted hash of the passcode and associated salt, she can mount an offline guessing attack against the passcode; if that attack succeeds, she can authenticate with the private key and the passcode.

If, before obtaining the private key, the adversary has has breached connection confidentiality (case B3) and obtained the passcode, she can authenticate.

If the secure connection is a TLS connection and the adversary is able to spoof the TLS server certificate of the application back-end and divert the underlying TCP connection (cases A4–B4), the adversary may try to spoof the back-end or mount a man-in-the-middle attack. In case A4 the latter can be prevented by including the TLS server certificate in the challenge that the device signs with the private key, and the former by further using secondary non-cryptographic server authentication to the user as explained above; to avoid passcode capture, the passcode should be sent only after successful secondary server authentication. In case B4, where the adversary reads the private key from the device storage, a man-in-the-middle attack cannot be prevented.

4.3.3 Smart Card with Key Pair Programmatically Enabled by a PIN

Yet another way of implementing two-factor authentication with a key pair and a passcode, typically a PIN, is to let the device verify the passcode and programmatically allow the use of the key pair if the passcode is correct. This approach is used in devices that provide some degree of tamper resistance, such as PIV cards [19]. The device stores the passcode or, preferably, a cryptographic hash of the passcode, for the purpose of verification.

The security posture is as shown in Table 4. The authentication method is secure if the adversary does not capture the device or is not able to break its tamper resistance (cases A1–A4).

On the other hand, if the adversary captures the device and is able to read its storage (cases B1–B4), she can obtain the private key in the clear and use it to authenticate.[16]

Cases B1–B4 are made less likely to occur by making the device or some of its storage tamper resistant. Table 5 illustrates the security posture when the tamper resistance is assumed to be strong enough to be unbreakable by the attacker.

The table does not consider the case where the adversary captures the device but cannot break the tamper resistance. Security is achieved in that case because the program that checks if the passcode is correct can limit the number of guesses.

4.3.4 OTP

There are many OTP-based authentication methods. The OTP may be generated by a fob such as a SecurID Token [9], or by an application running on the user's device using for example the HOTP algorithm [10], which works by incrementing a counter at regular intervals and computing the HMAC signature of the counter with the secret seed. The OTP may also be generated by asking the user to remember a set of image categories and

[16] She does not need the passcode to authenticate to the application back-end, but she may want to obtain the passcode for use in other attacks, in case the user uses the same passcode for multiple purposes; to that end, the attacker is able to mount an offline guessing attack against the passcode using the hash of the passcode stored in the device.

	1. Adversary cannot breach database or network security	2. Adversary breaches database confidentiality	3. Adversary breaches connection confidentiality	4. Adversary spoofs back-end certificate
A. Adversary cannot read device storage	Secure	Secure	Secure	Secure
B. Adversary captures device, reads its storage	Not secure	Not secure	Not secure	Not secure

Table 4: Security posture of smart card with key pair gated by PIN.

presenting the user with a time-varying grid of images, from which the user selects those that belong to the memorized categories; the one-time password is the concatenation of the characters that label those images in the grid [11]. In other variations, the OTP is sent to the user by text or voice messaging [61]. An OTP is often used in conjunction with a long term passcode, which may be a short PIN or a higher entropy password. Table 6 shows the security posture of an OTP used in conjunction with a passcode, in the most common and arguably most secure case where the OTP is generated by a fob. In that case the user's computing device does not store any data that is used in the authentication process, so the second row of the table is concerned with cases where the adversary captures the fob rather than the user's computing device.

In case A2, a database breach may allow the adversary to obtain the secret seeds used to generate the OTPs of all the users, as well as the salts and salted hashes of their passcodes. The adversary will then be able to impersonate users whose passcodes do not have high enough entropy. At least one such breach has actually occurred [12, 13, 14].

In case A3, an adversary who is able to breach connection confidentiality is able to see both the OTPs and the passcodes. However, if the same OTP is submitted twice during the validity period, the application back-end may reject the second submission. In that case an adversary who passively sniffs the connection will not be able to impersonate the user, unless she is able to block the connection before the OTP and the passcode reach their destination, or she is able to submit the sniffed OTP and password through a faster channel so that they reach the application back-end before the OTP and passcode sent by the user. So OTP plus

	1. Adversary cannot breach database or network security	2. Adversary breaches database confidentiality	3. Adversary breaches connection confidentiality	4. Adversary spoofs back-end certificate
A. Adversary cannot read device storage	Secure	Secure	Secure	Secure
B. Adversary captures device, reads its storage	Case prevented by tamper resistance	Case prevented by tamper resistance	Case prevented by tamper resistance	Case prevented by tamper resistance

Table 5: Security posture of smart card with key pair gated by PIN and strong-enough tamper resistance.

	1. Adversary cannot breach database or network security	2. Adversary breaches database confidentiality	3. Adversary breaches connection confidentiality	4. Adversary spoofs back-end certificate
A. Adversary has no access to fob	Secure	Security depends on passcode entropy	Usually secure	Not secure
B. Adversary captures fob	Somewhat secure	Security depends on passcode entropy	Not secure	Not secure

Table 6: Security posture of OTP plus passcode

passcode is usually secure in case A3.

In case A4, an adversary who spoofs the back-end certificate and diverts the underlying TCP connection is able to mount a man-in-the middle attack.

If the adversary captures the fob but is not able to breach database or network security (case B1), she needs the passcode to authenticate. But since the passcode is sent to the authentication back-end, it is vulnerable to phishing attacks, passcode reuse, etc.; so we rate the authentication method in this case as only "somewhat secure".

Case B2 is as A2, because capturing the fob does not help the adversary with guessing the passcode.

If, before capturing the fob, the adversary has breached connection confidentiality (case B3) and obtained the passcode, she can authenticate.

In case B4 the adversary can mount a man-in-the-middle attack as in case A4. Furthermore, after capturing the passcode as a result of the man-in-the-middle attack, she can use it later in conjunction with the captured fob to authenticate, without having to mount a man-in-the-middle attack again.

4.3.5 Ordinary Password

The security posture of authentication by username and password cannot be summarized in a table similar to the above.

Capturing the device does not matter, unless the password is stored on the device, e.g. by a browser.

Breach of database confidentiality is devastating because it exposes the passwords of all the users of the application to offline attack. If passwords are hashed and salted, the adversary must mount a separate offline attack for each password; but many if not most of the passwords will not be strong enough to withstand the attack. By contrast, with our protocredential-plus-passcode authentication method of Section 4.2, after breaching the database the adversary can only mount offline attacks after capturing user devices and reading their protocredentials.

Breach of connection confidentiality allows the adversary to capture the password and impersonate the user. By contrast, in protocredential-plus-passcode authentication, the passcode is not sent to the authentication back-end, and breach of connection confidentiality can only reveal the public key, which only matters if the adversary later captures the device and reads the protocredential.

If the device accesses the application over a TLS connection and the adversary is able to spoof the TLS server certificate and divert the underlying TCP connection she can mount a man-in-the-middle attack. Furthermore, after capturing the password as a result of the man-in-the-middle attack, she can use it later to authenticate, without having to mount a man-in-the-middle attack again.

Passwords are vulnerable to phishing attacks. By contrast, in protocredential-plus-passcode authentication, phishing attacks are not a threat because neither the passcode nor the private key are sent to the database back-end.

A big security problem with passwords is password reuse. Many users use the same password for many different sites. A malicious site can use the password submitted by the user at that site to impersonate the user at other sites. A site with weak security may

allow hackers to obtain the passwords of its users through breaches of database or network confidentiality, and hackers can then use those passwords to impersonate users at other sites, no matter how strong the security of those other sites. By contrast, in our protocredential-plus-passcode authentication method, passcodes are not sent to the application back-end, so passcode reuse is not an issue unless a passcode used as second factor is also used as an ordinary password elsewhere. Even in that case, an adversary who obtains the passcode by harvesting it at a site where it is used as an ordinary password can only use it in conjunction with a protocredential after capturing the device that contains the protocredential.

Passwords are vulnerable to *shoulder surfing*, i.e. to being observed as they are being typed, particularly in mobile devices where characters are prominently displayed by the keyboard as they are typed. In protocredential-plus-passcode authentication the passcode can be observed as well, but the adversary can only use the passcode for authentication to the application back-end after capturing the device that contains the protocredential.

4.3.6 Summary

To recapitulate, our method of credential regeneration from a protocredential and a passcode eliminates the vulnerability to attacks based on password reuse that affect ordinary authentication by username and password. It is also resistant to phishing attacks and breaches of the back-end database, a substantial advantage over both ordinary passwords and one-time passwords.

These benefits are simply a consequence of the use of public key cryptography for authentication; but our method also features a stronger security posture than other methods based on public key cryptography, in cases where the adversary is able to capture the user's device.

In such cases our method protects the credential without tamper resistance, whereas the method of storing the private key in the clear and making its use programmatically conditional on a passcode (Section 4.3.3, Table 4), often used in smart cards, requires tamper resistance.

Furthermore our method provides protection against an adversary who reads the device storage even when the passcode is a low-entropy PIN, whereas the method of encrypting the private key (Section 2, Table 2) requires a high entropy passcode for protection against such an adversary.

Our method allows the user to protect herself, by choosing a high entropy passcode, against an adversary who captures the device and reads its storage after breaching network security, whereas the method of using a key pair and an independent passcode separately verified by the application back-end (Section 4.3.2, Table 3) provides no security in that case.

4.4 Security and Privacy Provided by Two-Party Protocredential-Based Authentication with Biometrics

As we saw in Section 3.1, the usage of biometrics for authentication establishes three security and privacy requirements: security of the authentication process, unlinkability of uses of biometric features, and confidentiality of the biometric features. In Section 4.4.1 we discuss

how these requirements are met by the two-factor authentication method of Section 3.3. Then in Section 4.4.2 we discuss how the use of a PIN as a third authentication factor described in Section 3.4 strengthens the security posture. Comparison with an open-loop biometric authentication approach suggested by Boyen [50] can be found in Section 6.2.

4.4.1 Security and Privacy Provided by Two-Factor Authentication with Biometrics

Security of the Authentication Process. Security of biometric authentication is a concern at two different levels.

First, as pointed out above in Section 3.1, biometric features are only relatively secret. This is addressed by our two-factor authentication method simply by virtue of using two factors. A biometric sample is not useful without the auxiliary data included in the protocredential that is stored in the user's device; so, to use a biometric sample obtained surreptitiously, an adversary must capture the device and read the protocredential.

Second, the biometric key has relatively low entropy if the auxiliary data is known, as we saw above in Section 4.1. So an adversary who captures the device and obtains the auxiliary data may try to mount an offline brute force guessing attack against the biometric key. But our method prevents an offline attack against the biometric key just like it prevents an offline attack against a PIN in two-factor non-biometric authentication, by the fact that most or all candidate biometric keys produce well-formed key pairs, and therefore testing of candidates cannot be done offline as long as the adversary does not obtain the public key or a hash of the public key by breaching database or network security.

Unlinkability of Uses of Biometric Features. In the two-factor biometric authentication method of Section 3.3 and Figure 3, the biometric sample (the iris image in the figure) is seen by the user's device but not by the application back-end. The back-end only sees the regenerated public key. Therefore an adversary controlling the application back-end cannot link the authentication transaction to any biometric sample that may have been obtained offline. Furthermore authentication transactions the use the same biometric sample cannot be linked because they use different key pairs regenerated from the same sample. Note, however, that code running on the device may have access to the biometric sample. See Section 8.3.

Confidentiality of Biometric Data. In traditional biometric authentication, the user's biometric sample is compared to a template derived from a reference sample at enrollment time; but that requires storing the template in a device or a database, which puts the user's biometric data at risk of being captured. If the template is present in a device carried by the user, it must be stored in tamper-resistant storage. But while some smart cards may provide a certain amount of tamper-resistance, today's smart phones and tablets do not.

Our two-factor biometric authentication method, on the other hand, does not compare a biometric sample to a template. Instead of a template, it uses auxiliary data that is computed at enrollment time by x-oring a reference sample with a random codeword. Because the codeword contains redundancy, the auxiliary data leaks some amount of information about

the reference sample, from an information-theoretic viewpoint.[17] However, no methods are known for relating the leaked information to the user's biometric features.

4.4.2 Additional Security Provided by Three-Factor Authentication

The main purpose of using a PIN in addition to a biometric key as described in Section 3.4 is to implement three-factor authentication. But the PIN also provides two additional security benefits. First, the PIN adds its entropy to the entropy of the biometric key, for protection against an adversary who has both captured the protocredential and breached database or network security; this is because the biometric key has low entropy only if the attacker knows the auxiliary data, and the PIN encrypts the auxiliary data. Second, the PIN eliminates the leakage of information about the reference sample from the auxiliary data, again by encrypting the auxiliary data.

5 Data Protection Mediated by Protocredential-Based Authentication

Protocredential-based authentication is motivated by the lack of effective data protection on today's mobile devices. The purpose of storing a protocredential rather than a full-fledged credential in the user's device is to protect the credential against an adversary who captures the user's device. In this section we show how, besides protecting that credential, protocredential-based authentication can be used to implement effective data protection for any data stored on the device, including credentials used for open-loop and one-factor closed-loop authentication, which cannot be readily protected by the credential regeneration technique.

5.1 State of the Art of Data Protection

Two methods are used today to protect data stored in a computer device in case the device is lost or stolen. One is to store the data in tamper resistant storage, the other is to encrypt the data.

Tamper resistance is rarely used in ordinary computing devices such as smart phones, tablets, laptops or desktops, because it would increase the cost of the device. An exception is the use of an NFC secure element within a smart phone to store credit card data used in some payment applications. However the tamper resistance strength of such secure elements is unspecified, and, to our knowledge, none has been certified by NIST as tamper resistant (FIPS 140-2 physical level 3 or 4 [62]).

Encryption is used much more frequently. Corporate laptops are often protected with full-disk encryption combined with pre-boot authentication. Since iOS 4, data stored in the iPhone and other iOS devices is encrypted under a hierarchy of keys derived in part from a PIN or a password that the user enters to unlock the phone.

[17] No information would be leaked if the reference sample were x-ored with a random string rather than a random codeword; but of course the result would not be useful because it could not be error-corrected as discussed in Section 3.2.

The simplest way of encrypting data at rest is to use a symmetric key derived from a passcode, such as a PIN, a password, or a passphrase. But an attacker who gains physical access to the device and can extract the encrypted data from the device can mount an offline passcode-guessing attack, trying passcodes until one is found that produces a key which successfully decrypts the data. Withstanding an offline passcode-guessing attack requires a high-entropy passcode.

Requiring the user to enter a high-entropy password to boot a laptop may be reasonable, but requiring a user to enter one each time he or she unlocks a smart phone is impractical due to the difficulty of entering high-entropy passwords on a small touchscreen keyboard. In iOS, Apple addressed this difficulty by using a *hardware key* in addition to a PIN to derive the key hierarchy and protect the PIN. The hardware key is hardcoded in a hardware encryption chip and cannot be extracted by a casual attacker. However, various ways have been found of running custom code on an iOS device (with the primary purpose of *jailbreaking* the device), and custom code can make use of the hardware key even though it cannot extract it. By making use of the hardware key it is possible to mount an offline attack against the PIN using the processor in the phone. The processor is relatively slow, but most people lock the iPhone with a 4-digit PIN, and an exhaustive brute-force attack against a 4-digit PIN takes only 40 minutes on the device [63].

5.2 Effective Data Protection Using Protocredential-Based Authentication

For effective data protection we propose to encrypt the data under a random high-entropy symmetric data-encryption key; but rather than encrypting the data-encryption key under a key-encryption key derived from a password, which would make it vulnerable to offline attack by an adversary who captures the device, we propose to store it outside the device, entrusting it to a key storage service accessible over a network. The key storage service could be a web application accessible over the Internet or, if the device is a mobile device connected to a carrier network, it could be provided by the carrier and accessible through the carrier network. To retrieve the data-encryption key the device authenticates to the key storage service with an uncertified key pair regenerated from a protocredential and one or more non-stored secrets provided by the user, such as a PIN, a biometric sample, or both. As discussed above, such a key pair is not vulnerable to an offline attack by an adversary who captures the device.

Figure 5 illustrates the technique.

The protected data is encrypted under a data-encryption key k, which is kept by the key storage service. The service has a database containing stored-key records, one for each key stored by the service, each record containing a stored-key record handle, the hash of the public key component of the key pair, the key stored in the record,[18] and a counter of consecutive failed authentication attempts. The figure shows a record containing the data encryption key k. The stored-key record plays the role that the device record plays in the general case of protocredential-based authentication illustrated in Figure 1. Since the

[18]The key is stored in the clear, but it can be protected against a breach of database security by one or both of the mechanisms described below in sections 5.3 and 5.4.

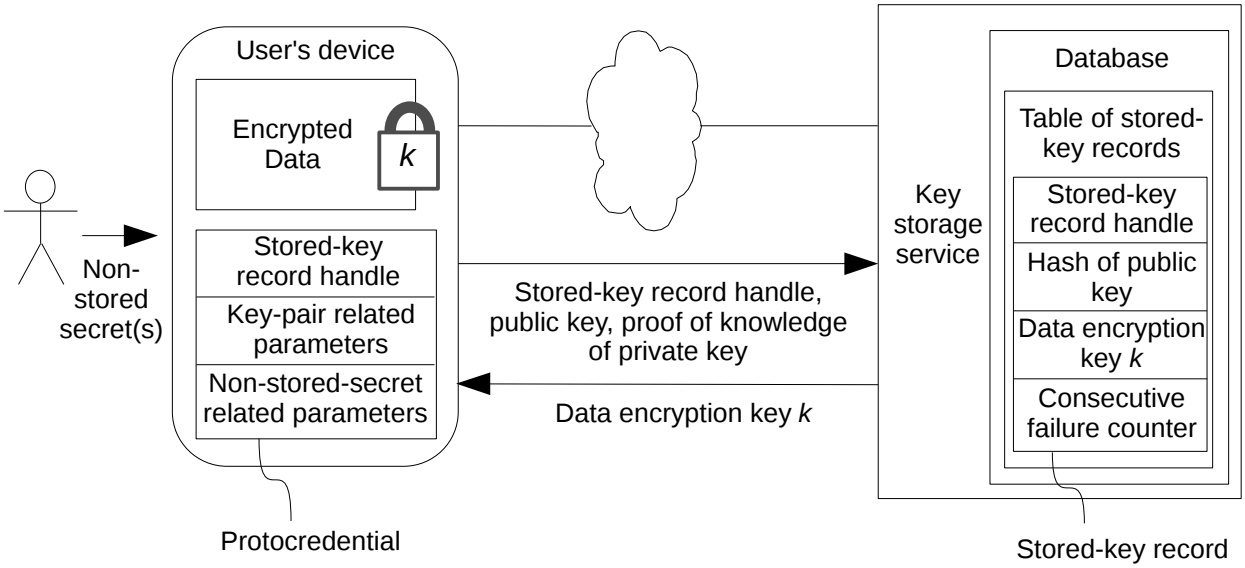


Figure 5: Data protection mediated by protocredential-based closed-loop authentication

purpose of the database is to store particular keys rather than to authenticate the user, there need not be user records.

The device authenticates to the key storage service using a key pair regenerated from a protocredential and from one or more non-stored secrets such as a PIN or an iris image supplied by the user, as described in Sections 2 and 3. The protocredential contains a stored-key record handle referring to the stored-key record that contains the data encryption key k, key-pair related parameters, and parameters related to the non-stored-secret secrets.

To authenticate, the device sends to the key storage service the stored-key record handle, the public key component of the key pair, and a proof of knowledge of the private key component of the key pair, over a secure connection. The secure connection may be provided by the carrier within the carrier network, or it may be a TLS connection made by the device to the key storage service over the Internet. In the latter case the proof of knowledge may consist of a signature on a challenge including a nonce sent by the key storage service and the TLS server certificate used in the TLS handshake. The counter of failed authentication attempts is used as described in Section 2.1 for the general case. After the device has successfully authenticated, the data encryption key k is retrieved over the secure connection.

The user can be prompted for the non-stored secrets, and the data encryption key k can be retrieved from the key storage service, either when the device is unlocked, or later when the key is needed to decrypt the data.

5.3 Using Multiple Storage Services

Shamir's secret sharing technique [64] can be used to divide the data encryption key k into N pieces that are kept by N different key storage services, in such a way that the device can reconstruct k from any K of those N pieces, but any set of $K-1$ pieces reveals no

information about k. This increases security, because even if $K-1$ servers collude they cannot reconstruct k. It also increases reliability, because the device can reconstruct k even if $N-K$ services are down. And it may increase performance, because the device can request all N pieces and use the first K that arrive.

5.4 Using a Protokey

When using one or more storage services, security can be further increased by entrusting a *protokey* to those services, rather than the data encryption key itself; the data encryption key is then derived from the protokey and the same non-stored secrets that are used to regenerate the authentication key pair from the protocredential.

When the protokey is retrieved using the two-factor non-biometric authentication method of Section 2.2, i.e. when the authentication key pair is regenerated from a passcode such as a PIN, the data encryption key can be obtained by x-oring the protokey with a randomized hash of the passcode (using a different salt for randomization than the one used for computing the randomized hash of the passcode for key pair regeneration). When the protokey is retrieved using the multifactor biometric authentication methods of sections 3.3 and 3.4, i.e. when the authentication key pair is regenerated from a biometric key, the data encryption key can be obtained by x-oring the protokey with a randomized hash of the biometric key (again, using a different salt than the one used for computing the randomized hash of the biometric key for key pair regeneration).

This prevents an adversary who captures the device and breaks into the database of the key storage service, if one service is used, or to the databases of a sufficient number of services, if Shamir secret sharing is used, from obtaining the data encryption key and immediately decrypting the data. When a protokey is used, the adversary has to mount a successful offline attack to guess the passcode or the biometric key in addition to capturing the user's device and breaking into the database(s).

The data encryption key could also be computed as a randomized hash of the passcode or the biometric key using the protokey as a salt. But this would result in a weaker security posture because it would allow the adversary to mount the offline attack without capturing the device if the encrypted data is available outside the device, e.g. in a back-up of device data stored in a PC.

5.5 Credential Protection and SSO Based on Data Protection

The proposed data protection technique can be used to protect the same kind of data that is protected in mobile devices today by traditional techniques, such as email messages or confidential enterprise data; or to protect all the data on a device, just as disk encryption can be used to protect all the data in a laptop.

It can also be used to protect credentials used in open-loop authentication and one-factor closed-loop authentication, which cannot be used in protocredential-based authentication. Credentials used in open-loop authentication include public key certificates and privacy-enhancing credentials such as U-Prove tokens and Idemix anonymous credentials.

Protecting multiple credentials by this method amounts to a form of single sign-on (SSO) based on data protection. After the device authenticates to the key storage service(s) and

retrieves the data encryption key, all the credentials protected by the data encryption key can be used without further user intervention. Enterprise SSO can be implemented by protecting the cryptographic credentials used by all enterprise applications under the same data encryption key. Global SSO can be implemented by encrypting all the credentials in the device besides the one used to authenticate to the key storage service(s); in the case of a mobile device, the user may be prompted for the non-stored secrets and the data encryption key may be retrieved when the device is unlocked.

A second form of SSO based on shared login sessions is discussed below in Section 7.5.

6 Open-Loop Authentication

From a privacy viewpoint, closed-loop authentication is not well suited for demonstrating to a relying party that a user possesses an identity or attribute asserted by a third-party. In closed-loop authentication, the third-party is responsible for verifying possession of the credential; therefore it has to participate in the authentication protocol; and by participating in the protocol, it usually learns the identity of the relying party. That impinges on the user's privacy if the user does not fully trust the third party.[19] In open-loop authentication, on the other hand, the relying party is responsible for verifying possession of the credential, so the third-party does not need to be involved in the authentication process and does not need to be informed of the identity of the relying party.[20] Hence open-loop authentication provides more privacy than closed-loop authentication when a third party plays the role of identity or attribute provider.

Open-loop authentication typically uses purely cryptographic credentials, but credentials with a biometric component have been considered as well.

6.1 Cryptographic Open-Loop Authentication

Cryptographic open-loop authentication may use credentials that provide a broad range of privacy features. X.509 public key certificates [29] provide the basic feature that a certificate issuer is not informed of how the certificate is used. Microsoft's U-Prove tokens [30, 31, 32] also provide *issue-show unlinkability*, which means that even if the token issuer colludes with a relying party, they cannot tell whether the token presented to the relying party is the same token that was issued to a particular user. IBM's Idemix anonymous credentials [33, 34] further provide multi-show unlinkability, which means that even if two relying parties collude they cannot determine if a credential presented to one of them is the same credential presented to the other. Another privacy feature of some types of credentials used in open-loop authentication is selective disclosure. Structured certificates [67, 68] bind a public key to the root of a tree of hashes of attributes, allowing the user to choose what attributes are

[19] It is possible to hide the identity of the relying party from the third-party behind the browser [65] or a trusted proxy [66] at the expense of greatly increased complexity.

[20] In the case of a public key certificate, the certificate issuer becomes indirectly involved if it assists in the authentication process by providing the status of a certificate in response to an OCSP [28] request. It may then be able to identify the relying party by observing the IP address from which the OCSP request is sent. Certificate revocation lists (CRLs) are preferable to OCSP for the sake of privacy.

disclosed when the certificate is presented.[21] U-Prove also allows the user to choose which of the attributes in a U-Prove token are disclosed when the token is presented. Idemix goes a step further, allowing the user not only to disclose a subset of the attributes in an Idemix anonymous credential, but also allowing the user to prove properties of attributes, such as being greater or less than a numeric constant, without disclosing those attributes.

6.2 Biometric Open-Loop Authentication

In [50, §8], Boyen suggests an authentication method that can be characterized as biometric open-loop authentication. Couching his method in our terminology, a biometric key is produced from a biometric sample and auxiliary data, and used to regenerate a key pair (Boyen does not explain how) which is used for authentication, as in our protocredential-based approach.

A key difference with our approach is that the public key is certified by a trusted party that may be viewed as a certificate authority (CA), and is publicly available. The public key certificate together with the private key and the auxiliary data can be viewed as a certified biometric credential. This makes Boyen's method a three-party authentication method, and, because the CA is not responsible for verifying the credential, an open-loop authentication method.

Another key difference is that Boyen lacks the concept of a protocredential. Nothing is stored in the user's machine, and the auxiliary data is considered public data. Since both the public key and the auxiliary data are publicly available, anybody can mount an offline attack at any time against the biometric key that is used to regenerate the key pair, using only publicly available data. Since the biometric key is entropy-poor, as we saw in Section 4.4.1, Boyen's method does not provide strong security.

The exposure to offline attack could be partially remedied by storing the auxiliary data in the user's device as we do, and keeping it secret. However, that would still leave the biometric key vulnerable to an offline attack by an adversary who captures the device and reads the auxiliary data. This could be remedied by protecting the auxiliary data with the data protection method of Section 5.

Another security problem with Boyen's method is that it amounts to one-factor unattended biometric authentication. We saw in Section 3.1 that biometrics only provide strong security when there is assurance of liveness, which is difficult to achieve in unattended authentication. Biometrics should only be for used unattended authentication in conjunction with at least one other factor, as in our methods of sections 3.3 and 3.4.

There is also a privacy problem with Boyen's method. The user has to disclose a biometric reference sample to the CA, so that the CA can verify that the public key in the certificate derives from the user's biometrics. This problem is inherent in the fact that Boyen's method is an open-loop authentication method. In our closed-loop methods of sections 3.3 and 3.4, on the other hand, the user does not reveal any biometric sample to any party other than her own device.

[21] To our knowledge, structured certificates have not yet been implemented or used anywhere. We mention them here because they have the remarkable property of providing selective disclosure without the mathematical complexity of systems such as U-Prove and Idemix, and the difficulties related to revoking credentials and preventing fraudulent credential sharing in those systems.

7 Mobile Authentication Architecture

Before the advent of mobile computing, users accessed web sites and applications mostly via browsers installed in personal computers. Naturally, most proposals for cryptographic authentication were browser centric, and the few that were not failed to gain traction. Unfortunately, browser vendors and standards bodies failed to create an identity ecosystem where cryptographic credentials could be deployed and used conveniently and securely. Use of SSL client certificates, which date back to 1995, remained confined to very specific contexts, such as the authentication of US government employees and contractors to the information systems of Federal agencies. More sophisticated privacy-enhancing credentials were never deployed.

But the computing landscape has changed with mobile computing. Many web applications are now accessible via a native application front-end. User or device authentication is then a private matter between the native front-end and the application back-end, without browser involvement. (Even though the native front-end and the application back-end are conceptually part of the same application, the back-end cannot trust the front-end since the front-end is under user control. Therefore the front-end must authenticate to the back-end.)

This by itself is not enough to enable widespread use of cryptographic authentication, for two reasons. First, there are web-based applications without a native front-end, and even if an application offers both a native front-end and web-based access, the user may prefer web-based access to avoid or delay downloading the native front-end. Second, as illustrated in [39], it is difficult to integrate cryptographic functionality in an application because of the complexity of cryptographic APIs.

But mobile computing has brought a second innovation: a URL-based inter-application communication mechanism that allows native applications to interact with each other within a mobile device. In this section we propose a mobile authentication architecture that allows an application to offload the complexities of cryptographic and biometric authentication to a Prover Black Box (PBB) located in the user's device and a Verifier Black Box (VBB) located in the cloud. The PBB authenticates to the VBB, the VBB creates a record of the authentication transaction, which we call the *authentication object*, and the VBB provides the PBB with a bearer token that refers to the object, which we call the *authentication token*. The PBB delivers the authentication token to the application back-end (via the browser or the native application front-end), which uses it to obtain the hash of a public key, or user attributes, or an access token (in the case of social login) from the VBB. The VBB may be part of the application back-end, in which case it is the non-VBB portion of the back-end that receives the authentication token and uses it to obtain the authentication data from the VBB.

We hope that the proposed architecture will enable the broad deployment of cryptographic authentication on mobile devices by removing the stumbling block of browser adoption. How about authentication on desktops and laptops? We show below in Section 9 how the mobile authentication architecture that we are proposing can be adapted for use in traditional personal computers via browser extensions. But maybe it is best to wait until traditional desktops and laptops are replaced with new personal computers that adopt the same computing paradigm as mobile computers, including inter-application communication mechanisms such as those available today in Android and iOS.

7.1 Configurations and Use Cases

The proposed architecture is very flexible. Many different kinds of credentials are supported. Credentials can be stored in the PBB or regenerated from protocredentials stored in the PBB. There may be one PBB in the user's device used by all applications, or several PBBs, each used by a different application or group of applications. The VBB may be part of an application back-end or it may be provided by a third-party identity or attribute provider or by a social network that supports social login functionality. When the VBB is part of the application back-end, it may provide purely cryptographic functionality, or it may access the device and user records stored in the database of the application back-end, in which case we say that the VBB is database aware.

This flexibility makes it possible to support a broad variety of use cases, including:

- **Authentication to consumer applications, with shared PBB and global SSO.** The PBB is a native application. It can be used by any application, whether web-based or having a native front-end. The PBB contains a different uncertified key pair for each application. Each application back-end has its own VBB, which may be a virtual cryptographic appliance made available in an infrastructure-as-a-service cloud. All the credentials in the PBB are encrypted under the same data encryption key, preferably using the data protection scheme of Section 5. They are decrypted when the mobile device is unlocked and used without further user interaction, thus providing global SSO.

- **Multifactor login to high-security consumer application having a native front-end.** The consumer application could be used, e.g. for mobile banking or for patient access to a health care institution. The PBB is dedicated to the application. (In Android it may be bundled in the same package as the application, as discussed below in Section 8.3.) It contains a protocredential and prompts the user for a PIN and or a biometric sample to regenerate the credential, which is deleted after login.[22] The VBB may be specific to the application, or shared by different applications deployed by the same organization, and may or may not be database aware.

- **Bring-your-own-device (BYOD) access to enterprise applications with enterprise SSO.** Enterprise applications, both web-based and with native front-ends, share an enterprise PBB implemented as a native application, and an enterprise VBB, which is database aware. The enterprise PBB contains a single protocredential. When required by one of the enterprise applications, the PBB prompts the user for a PIN and/or a biometric sample, regenerates the credential, authenticates the user, and creates a login session, which is shared by all the enterprise application, thus implementing enterprise SSO.

- **Social login.** A social network supplies the PBB and the VBB. The PBB may be built into in a native front-end of the social network, or it may be a separate native application. It contains a single credential or protocredential. Any application,

[22] Subsequent authentication during the login session is up to the application. It could be implemented using a bearer token analogous to a login session cookie.

whether web-based or with a native front-end, can ask the PBB for limited access to the user's account at the social network. The PBB asks the user's consent and, in the protocredential case, prompts the user for a PIN and/or a biometric sample to regenerate the credential. After cryptographic authentication of the PBB to the VBB, the application back-end uses the authentication token received from the PBB to obtain an access token from the VBB, which grants the requested limited access to the user's account. The application may use the access token to obtain the user's identity and/or attributes by accessing the user's account at the social network.

- **Third-party closed-loop authentication using a trusted personal data repository service (PDRS) with a shared PBB.** The PBB is a native application containing multiple credentials or protocredentials of different types used for different purposes. One credential or protocredential is intended for third-party closed-loop authentication with a PDRS freely chosen by the user. Any application, whether web-based or with a native front-end, may ask the PBB to authenticate to a PDRS that will supply requested user attributes, without specifying a particular PDRS. The PBB locates the credential or protocredential for the user-selected PDRS and asks the user's consent to disclose the requested attributes to that PDRS. In the case of a protocredential, the PBB prompts the user for a PIN and/or a biometric sample to regenerate the credential. The PBB authenticates to a VBB belonging to the PDRS, which obtains the requested attributes from the personal data repository maintained by the PDRS and delivers them to the application back-end upon presentation by the back-end of the authentication token.[23]

- **Open-loop authentication with proof of age using an Idemix anonymous credential.** The PBB is a native application containing multiple credentials or protocredentials of different types used for different purposes. One credential or protocredential is an Idemix anonymous credential issued by a party that is trusted to know the user's date of birth (e.g., in the US, the department of motor vehicles of some state). The credential contains the date of birth and may also contain a variety of other attributes. An application, either web-based or with a native front-end, that displays content unsuitable for children asks the PBB to prove that the user is at least thirteen. The PBB looks for an appropriate credential, finds the Idemix credential, and uses it to prove to a VBB belonging to the application that the user is at least thirteen, without revealing the user's date of birth, nor any other attributes, nor any information that may link the authentication transaction to the issuance of the credential or to other authentication transactions. The VBB stores the derived boolean attribute "age at least thirteen" in the authentication object and delivers it to the application back-end upon presentation of the authentication token.

- **Authentication with Multiple Public Key Certificates.** A candidate applies for a job online (in the US), presenting a public key certificate issued by the Social Security Administration that provides the candidate's Social Security Number (SSN),

[23]The social login use case above could also allow the user to freely choose among social networks that would implement a standard API for providing access to a user's account with delegated authorization.

and a public key certificate issue by a state that contains the same data as a driver's license. The PBB is a native application containing various credentials, including the SSN certificate and the driver's license certificate. The enterprise application that collects job applications asks the PBB to present the two certificates. The PBB sends the certificates to a VBB supplied by the enterprise application and demonstrates knowledge of the associated private keys. The VBB stores the attributes supplied by the two certificates in the authentication object and later delivers them to the non-VBB portion of the application back-end upon presentation of the authentication token.

7.2 Communication between Native Applications

Mobile operating systems including iOS and Android provide inter-application 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 targeting 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.

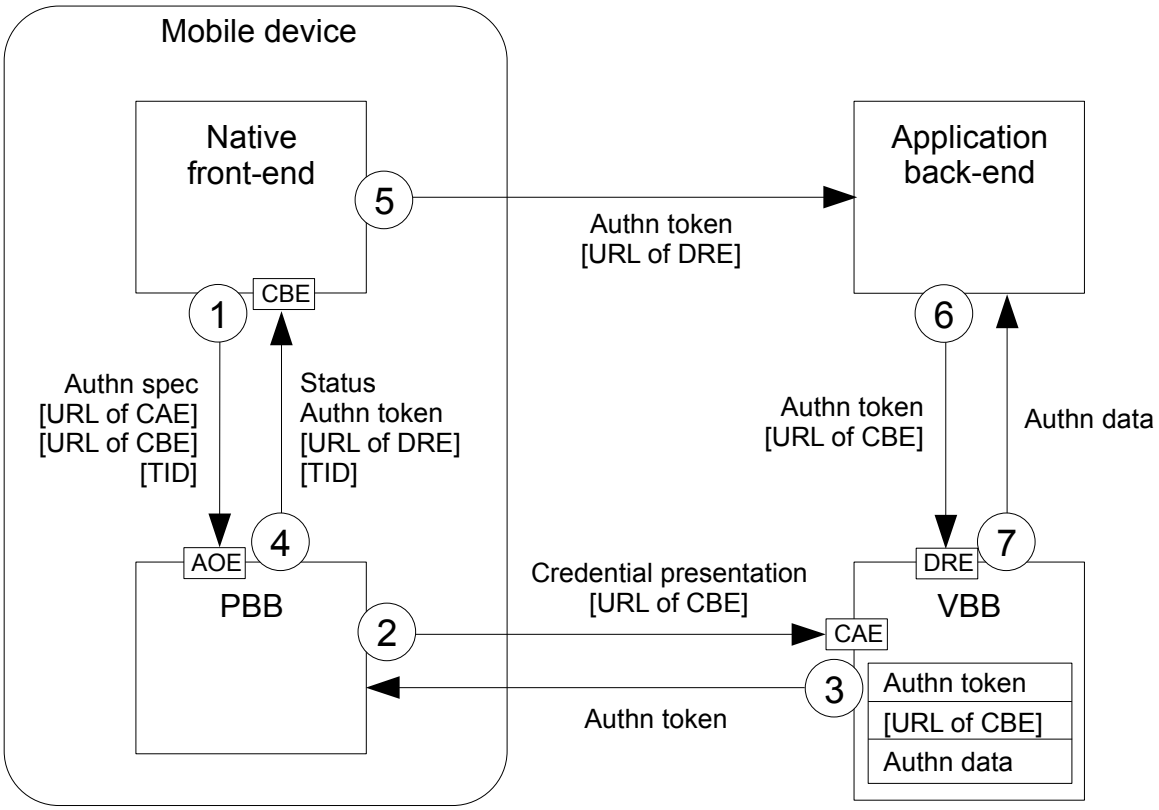
Both iOS [69] and Android [70] implement native URLs using *custom schemes*, a custom scheme identifying the native application to which the message is to be sent. As in an HTTP GET request, a native URL is both the address and the contents of the message. The native application parses it to extract 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.

7.3 Native Authentication Protocol

Figure 6 shows in general terms the process by which a native application front-end on a mobile device authenticates to the application back-end, offloading the complexities of cryptographic and biometric authentication to a PBB and a VBB.

We 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 requests, 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



PBB = Prover Black Box
VBB = Verifier Black Box
Authn = Authentication
Authn spec = Specfication of credentials and/or attributtes and/or access
TID = Transaction ID

CBE = Callback endpoint
AOE = Authentication Offloading Endpoint
CAE = Cryptographic authentication endpoint
DRE = Data retrieval endpoint

Square brackets […] indicate optional parameters

Figure 6: Mobile authentication through native front-end

parameter name followed by an equal sign ("=") and a parameter value.[24] The PBB has an authentication offloading endpoint (AOE in the figure) where the native front-end sends an authentication offloading request, and the native front-end has a callback endpoint (CBE) where the PBB sends a reply to the offloading request. The VBB has a cryptographic authentication endpoint (CAE) where the PBB sends a cryptographic authentication request, and a data-retrieval endpoint (DRE) from which the application back-end retrieves user data upon presentation of the authentication token.

The PBB may contain any number of credentials used for closed-loop one-factor authentication, protocredentials used for closed-loop multifactor authentication, and credentials used for open-loop authentication.

If the PBB contains several protocredentials used for two-factor biometric authentication as described in Section 3.3, they may share the same Auxiliary Data shown in Figure 3, so that the user has to enroll a reference biometric sample only once. The auxiliary data produces the same biometric sample for all those protocredentials, but should be randomized with different salts when used in conjunction with different protocredentials. Similarly, if the PBB contains several protocredentials used for three-factor authentication as described in Figure 3.4, they may share the same Auxiliary Data and Salt 1 of Figure 4, but they should be randomized with a different Salt 2.

The authentication process comprises seven steps, indicated by circled numbers in the figure.

Step 1

The native front-end asks the PBB to authenticate to the VBB. The request may specify one or more credentials to be used by name. Credential names are unique URIs chosen by the credential issuers. The request may also specify authentication data to be delivered by the VBB to the application back-end, such as user attributes. Attributes are also named by unique URIs, but there need not be a consistent global nomenclature for attributes: different standards for naming attributes may be developed for different contexts or by different communities.

The native front-end may specify what attributes should be supplied by what credentials, or may specify attributes without specifying credentials. In the latter case the PBB may use credential metadata to choose credentials that may deliver the attributes.

For social login, the native front-end may specify the scope of access to the user's account at the social network that the application wishes to obtain.

The request is sent as a native URL message to the AOE of the PBB, with parameters specifying:

1. One or more credentials to be used, and/or the authentication data to be delivered, and/or the scope of access to be obtained in the case of social login.

2. Optionally, the VBB to be used, specified by the URL of its CAE. In third-party closed-loop authentication the VBB is part of the third party, and the native front-end

[24] 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.

may choose not to specify the VBB to be used, allowing the PBB to choose the VBB of a third party which the PBB has a credential or protocredential, the CAE of that VBB being available as credential metadata. Freedom of choice of the third party is a matter of privacy, as discussed below in Section 10.

3. Optionally, the URL of the CBE of the native front-end to which a response is to be returned. The URL of the CBE may be omitted in two-party authentication, when it should be recorded in the credential metadata when the credential or protocredential is created.

4. Optionally a transaction ID (TID) that the native front-end may use internally to remember the context of the authentication transaction.

Step 2

The PBB asks the user's consent to use the requested credentials or disclose the requested identity or attributes, if necessary, as explained below in Section 8.2; for social login, it asks the user's consent to grant access to the user's account at the social network to the application, to the extent requested by the native front-end. Then the PBB prepares the credentials needed to satisfy the user's request. If one of the credentials is an uncertified key pair that must be regenerated from a protocredential, the PBB prompts the user for the non-stored secrets needed to regenerate the credential, such as a passcode and/or a biometric sample. Finally, the PBB establishes a TLS connection to the CAE of the VBB, which it uses to present the credentials to the VBB. In third-party closed-loop authentication, where the same credential and the same VBB are used by multiple relying parties that do not trust each other, the PBB also sends the URL of the CBE, which is needed by the VBB to prevent an attack described below in Section 8.1.

The CAE of the VBB authenticates itself with a TLS server certificate. The certificate may be backed by a certificate chain terminating at a root certificate that can be found in a root certificate store provided by the operating system or by the PBB. Alternatively, the certificate may be trusted because the certificate itself or a hash of the certificate may be known to the PBB or may be associated with a particular credential used for closed-loop authentication; in that case, the certificate may be self-signed.

Presenting an uncertified key-pair credential to the VBB means sending the device record handle and the public key, and signing a challenge with the private key. When the key pair is used for closed-loop authentication to a third-party identity or attribute provider, presentation of the credential also includes sending the names of the user attributes that the PBB wants the VBB to include in the authentication data.[25] When the key pair is used for social login, presentation of the credential also includes specifying the desired scope of access.

Presenting a public key certificate means sending to the VBB the certificate and possibly a certificate chain backing the certificate, and signing a challenge with the associated private key.

[25] We consider the specification of the desired attributes as being part of credential presentation by analogy with the presentation of a credential that supports selective disclosure.

Presenting a U-Prove token or an Idemix anonymous credential means constructing and sending a proof of possession of the credential that discloses the requested attributes, while keeping others hidden. Construction of the proof takes as input a challenge that includes a VBB nonce.[26]

In all the above kinds of credential presentation, a man-in-the-middle attack is prevented by the fact that the PBB verifies the TLS server certificate presented by the VBB during the TLS handshake. Additional security can be provided by hashing the TLS server certificate into the challenge(s) that are used to demonstrate possession of the credential(s). A man-in-the-middle attack is then prevented even if the adversary has successfully used a spoofed TLS certificate during the TLS handshake, as explained above in Section 2.1.

Step 3

The VBB generates a random high-entropy one-time[27] authentication token and creates in its own internal volatile memory a short-lived authentication object, retrievable by the authentication token, where it stores the URL of the CBE, if it was sent by the PBB, and authentication data.

If a public key certificate was presented by the PBB, the VBB includes the attributes conveyed by the certificate in the authentication data.

If a selective-disclosure credential was presented by the PBB, the VBB includes the disclosed attributes in the authentication data.

In two-party closed-loop authentication to a purely cryptographic VBB, using an uncertified key pair (possibly regenerated from a protocredential), the authentication data consists only of the hash of the public key component of the uncertified key pair and the device record handle. In two-party closed-loop authentication to a database-aware VBB, the VBB uses the device record handle received from the PBB to look up the device record and verify that the hash contained in the record is the hash of the public key received from the PBB; if that is the case, it obtains user data from the user record referenced by the device record and stores the user data in the authentication object as authentication data.

In third-party closed-loop authentication to an identity or attribute provider the VBB is part of the provider. The VBB uses the device record handle obtained from the PBB to identify a device record and a user record in the database of the provider. Then it places the attributes specified by the PBB in the authentication object as authentication data.

In social login the VBB is part of the social network. The VBB uses the device record handle obtained from the PBB to identify a device record and a user record in the database of the social network. Then it creates an access record specifying the scope of access provided and an access token that serves as an access record handle, and stores the access token in the authentication object as authentication data.

After creating the authentication object, the VBB sends the one-time authentication token to the PBB through the TLS connection.

[26] In U-Prove, the challenge is called a *message* and consists of a nonce sent by the verifier before the proof is constructed. In Idemix, the challenge is a hash of several components, including a nonce sent by the verifier.

[27] The token is used only once because it may leak in a "Referer" header, as explained in the security analysis (Section 8).

Step 4

If authentication was successful in step 3 the PBB sends a native URL message to the CBE of the native front-end, with parameters including a status code indicating success, the authentication token, and the optional TID, if one was sent in step 1. In third-party closed-loop authentication the relying party may not know the URL of the DRE of the VBB; the URL is recorded in the credential metadata and included in the message as an additional parameter.

Step 5

The native front-end authenticates to the online back-end, without cryptography, by sending the authentication token over a secure connection. The secure connection is a matter internal to the application. It may be implemented as a TLS connection from the front-end to the back-end, with TLS server authentication. The TLS server certificate may be self-signed, known to the native front-end, and updatable as needed in the native front-end by the same mechanism used to update the native front-end software. If the URL of the DRE of the VBB was received in step 4, it is sent to the back-end along with the authentication token.

Step 6

The application back-end (or its non-VBB portion, if the VBB is part of the application back-end) establishes a TLS connection to the DRE of the VBB with server authentication.[28] The application back-end uses the connection to send the one-time authentication token to the VBB in the body of an HTTP POST request. In third-party closed-loop authentication, where the same credential and the same VBB are used by multiple relying parties that do not trust each other, the back-end also sends the URL of the CBE, to identify itself to the VBB and thus prevent an attack described below in Section 8.1. The back-end knows the URL of the CBE because the CBE belongs to the native front-end of the application.

Step 7

The VBB uses the one-time authentication token to find the authentication object in volatile memory. If the URL of a CBE is present in the authentication object, the VBB checks that the same URL was received in step 6. If the object is found and the check succeeds or is not applicable, the VBB responds to the HTTP POST request by sending the authentication data found in the object, then it deletes the authentication object.

In two-party closed-loop authentication with a purely cryptographic (not database aware) VBB, the authentication data consists only of the device record and the hash of the public key. The application back-end uses the device record handle to locate the device record, verifies that the hash of the public key coincides with the one stored in the record, locates

[28] When the VBB is part of the application back-end, the communication between the the non-VBB portion of the back-end and the VBB is a matter internal to the application, and a secure connection other than a TLS connection could be used. In particular, if the connection is made over an intranet deemed secure, a TCP connection could be used without TLS protection. If a TLS connection is used, the DRE may use a self-signed certificate, known to the non-VBB portion of the back-end.

the user record from the device record, and obtains any desired user data from the user record.

After the user has authenticated, the application may want to create a login session. Section 7.5 below describes a method for implementing shared login sessions that provide a form of single sign-on. If sessions are not shared, session implementation is a matter internal to the application. One way of implementing non-shared sessions is by letting the application back-end create a session record containing a random high-entropy session token that serves as session record handle. The back-end sends the session token to the native front-end, which uses it for intra-session authentication very much like a session cookie is often used today for intra-session authentication on the web.

7.4 Web-based Authentication Protocol

Figure 7 shows the process of authenticating to a web-based application on a mobile device. Instead of the native front-end of Figure 6, the application uses as its front-end a web browser running on the mobile device, which displays HTML pages and runs JavaScript code pertaining to the application. The application back-end is accessible through one or more web servers. The callback endpoint is now located on the application back-end.

The web-based authentication process is similar to the one described in Section 7.3 for the case of an application with a native front-end (the "native process"). Neither the PBB nor the VBB need to be aware of which process is being used. In this section we explain the differences between the two processes.

The web-based authentication process comprises six steps, indicated by circled numbers in the figure. It is initiated as a result of an HTTP request sent by the browser to the application back-end, e.g. an HTTP request that requires authentication and is not accompanied by a login session cookie.

In step 1, the application back-end issues an HTTP redirection response to the HTTP request,[29] redirecting the browser to a native URL that targets the AOE of the PBB, with the same parameters as in the request sent by the native front-end to the PBB in step 1 of the native process, except that:

- The URL of the CBE, if included, is a web URL instead of a native URL, since the CBE is now in the cloud. More specifically, it is a URL that uses the https scheme; use of the http scheme is not allowed.
- The TID is now compulsory, and must be a random high-entropy string. The reason for this is that the TID now plays a security role, explained below in Section 8.1, in addition to helping the application remember the context of the authentication transaction.

The HTTP response also sets a cookie in the browser containing the TID, for reasons explained below in Section 8.1.

Steps 2 and 3 are the same as in the native process.

Step 4 replaces steps 4 and 5 of the native process. If step 3 was successful, the PBB constructs a callback URL by augmenting the URL of the CBE (received in step 1) with

[29] Specifically, a response with an HTTP status code equal to 302.

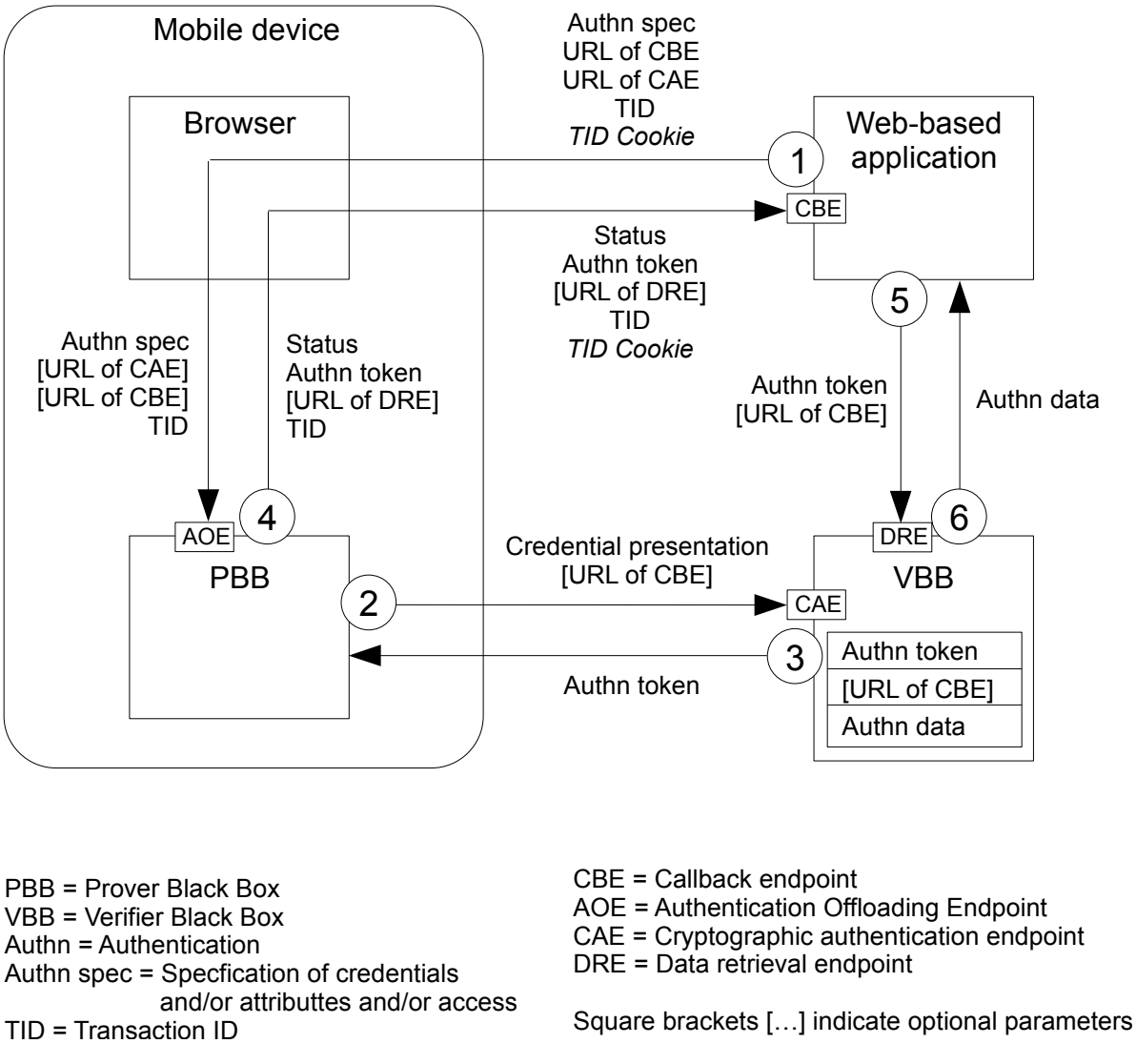


Figure 7: Mobile authentication to a web-based application

a query-string that comprises parameters including a status code indicating success, the authentication token, the now-compulsory TID, and the URL of the DRE of the VBB in the case of third-party closed-loop authentication.

The PBB asks the operating system of the mobile device to deliver the callback URL to its destination. Since the callback URL is a web URL, the operating system delivers it as a message to the default browser. This causes the browser to send an HTTP GET request to that URL, adding to the request a cookie HTTP header that contains the TID set as a cookie in step 1.

In step 5, the application back-end checks that the status parameter indicates success and the TID parameter in the callback URL coincides with the TID in the cookie header. The TID check protects against *Login CSRF* as discussed below in Section 8.1. If both checks succeed, the application back-end sends an HTTP POST request with the one-time authentication token and the URL of the callback endpoint (without a query string) to the data retrieval endpoint of the VBB as in step 6 of the native process.

Step 6 includes all of the tasks performed at step 7 of the native process.

Section 7.5 below describes a method for implementing shared login sessions, which provides a form of single sign-on among a group of applications that may include both web-based applications and applications with a native front-end. Non-shared sessions can be implemented using session cookies as is usually done today for web-based applications on both mobile devices and traditional PCs.

7.5 SSO Based on Shared Login Sessions

In Section 5.5 we discussed a kind of SSO based on data protection. The proposed mobile architecture provides a second kind of SSO based on shared login sessions, that has some performance and security advantages. The first kind is well suited for global SSO and for enterprise SSO with devices dedicated to enterprise use. The second kind is well suited for enterprise SSO in a bring-your-own-device (BYOD) context.

The second kind of SSO allows a group of applications, including both web-based applications and applications with a native front-end, to share a common login session. The applications in the group share a dedicated PBB that contains one uncertified key pair protocredential for multifactor closed-loop authentication. They also share a database and a database-aware VBB. The URL of the CAE of that VBB can be configured into the PBB so that it does not need to be included as a parameter in each authentication offloading request to the PBB.[30]

The technique used for providing shared login sessions is similar to two-party closed-loop authentication techniques and third-party closed-loop authentication techniques, but cannot be properly classified as either. We refer to it as *groupwise closed-loop authentication*.

Figure 8 shows the shared database. It contains a table of user records and a table of device records that refer to user records, as in Figure 1. In addition it contains a table of shared session records, which represent login sessions. Each shared session record contains a one-time session token, a reusable session token, a device record handle that references

[30]Multiple database-aware VBBs could be used, all accessing the same enterprise database; in that case the CAE could be specified by each authentication offloading request.

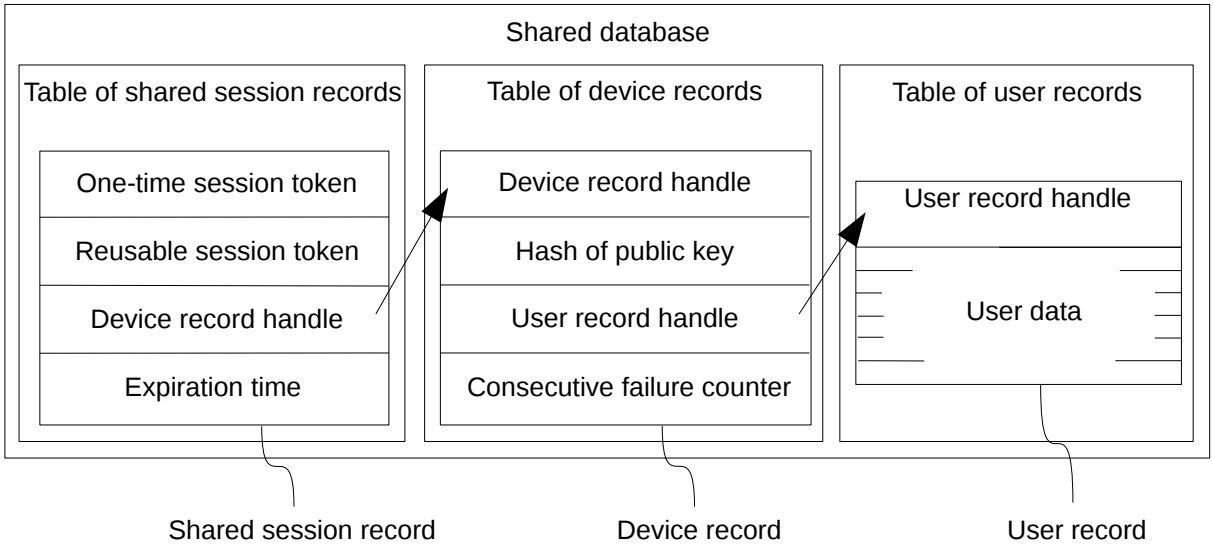


Figure 8: Shared database containing shared session records.

a device record, and an expiration time. Each of the session tokens is a record handle, which uniquely identifies the shared session record and can be used to retrieve it efficiently. The one-time session token is used the first time that one of the web-based applications in the group uses the shared session record, after which it is erased from the record.[31] The shared session record is created by the VBB after successful cryptographic authentication. The VBB sends the one-time and reusable session tokens in the shared session record to the PBB, which retains one or both of them in a session object.

A shared session record may be deleted before or after its expiration time is reached. The shared session it represents and the one-time and reusable session tokens it contains are deemed to have expired when the expiration time is reached or the session record is deleted, or either the device record referenced by the session record or the user record referenced by the device record are deleted or invalidated.

An application does not request cryptographic authentication explicitly. Instead a web-based application requests a one-time session token or an application with a native front-end requests a reusable session token, and the PBB performs cryptographic authentication if it cannot satisfy the request with an existing session token that has not expired.

7.5.1 Shared Session Usage by a Native Front-End

Figure 9 shows the process that takes place when an application with a native front-end requests a reusable session token from the PBB. The PBB contains a protocredential. As in Figure 6, the native front-end has a callback endpoint (CBE), the PBB has an authentication offloading endpoint (AOE) and the VBB has a cryptographic authentication endpoint (CAE).

[31] The token is used only once because it may leak in a "Referer" header, as explained in the security analysis (Section 8).

The AOE is used by the native front-end to request a reusable session token, which triggers cryptographic authentication if no login session is in progress. The token is sent to the CBE of the native front-end.

The application back-end uses the reusable session token to obtain user data from the shared database; so the VBB need not store authentication data, and the data retrieval endpoint (DRE) of Figure 6 is not needed. The VBB does not need the URL of the CBE either, for reasons discussed below in Section 8.2.2, so there is no use for the authentication object of Figure 6.

In step 1, the native front-end sends a native URL request to the AOE of the PBB, with parameters including the URL of the CBE, an optional TID used as in the process of Section 7.3, and an optional expired reusable session token. The purpose of including the token is to inform the PBB that it has expired and a more recent one is needed, so that the PBB does not have to keep track of session expiration.

The PBB has a session object in volatile memory that may contain a reusable session token. If the session object contains a reusable session token different from the expired reusable session token included in the request as a parameter, if any, then steps 2, 3 and 4 are skipped, and at step 5 the PBB sends a native URL message to the callback endpoint of the native front-end with parameters comprising a successful status, the reusable session token found in the session object, and the TID if one was received at step 1. If there is no reusable session token in the session object, or there is one that coincides with the expired reusable session token that was included in the request as a parameter, the process continues at step 2.

In step 2, the PBB interacts with the user to obtain the non-stored secrets, such as a passcode and/or a biometric sample, needed to regenerated an uncertified key pair from the protocredential. Then it establishes a secure connection to the VBB and performs cryptographic authentication, sending the device record handle and the public key and signing a challenge as described in Section 2.1.

In step 3, the VBB verifies the signature and checks that the device record handle references a valid device record that itself references a valid user record. If so, the VBB generates a random high-entropy one-time session token and a random high-entropy reusable session token, and creates a session record in the shared database containing those tokens as well as the device record handle and an expiration time.

In step 4, the VBB sends the one-time session token and the reusable session token to the PBB over the secure connection.

In step 5, if cryptographic authentication was not skipped, the PBB places the two tokens in the session object before sending a native URL message to the callback endpoint of the native front-end with parameters comprising a successful status, the reusable session token, and the TID if one was received at step 1.

The native front-end stores the reusable session token in volatile memory and uses it repeatedly for intra-session authentication to the application back-end; this is shown as step 6 in the figure. Each time the back-end receives the token, it uses it to locate the session record, the device record referenced by the session record, and the user record referenced by the device record; this is shown as step 7 in the figure. If the back-end cannot find a session record containing the reusable session token, or the expiration time in the record has been reached, or the device or user records have been deleted or invalidated, then intra-session

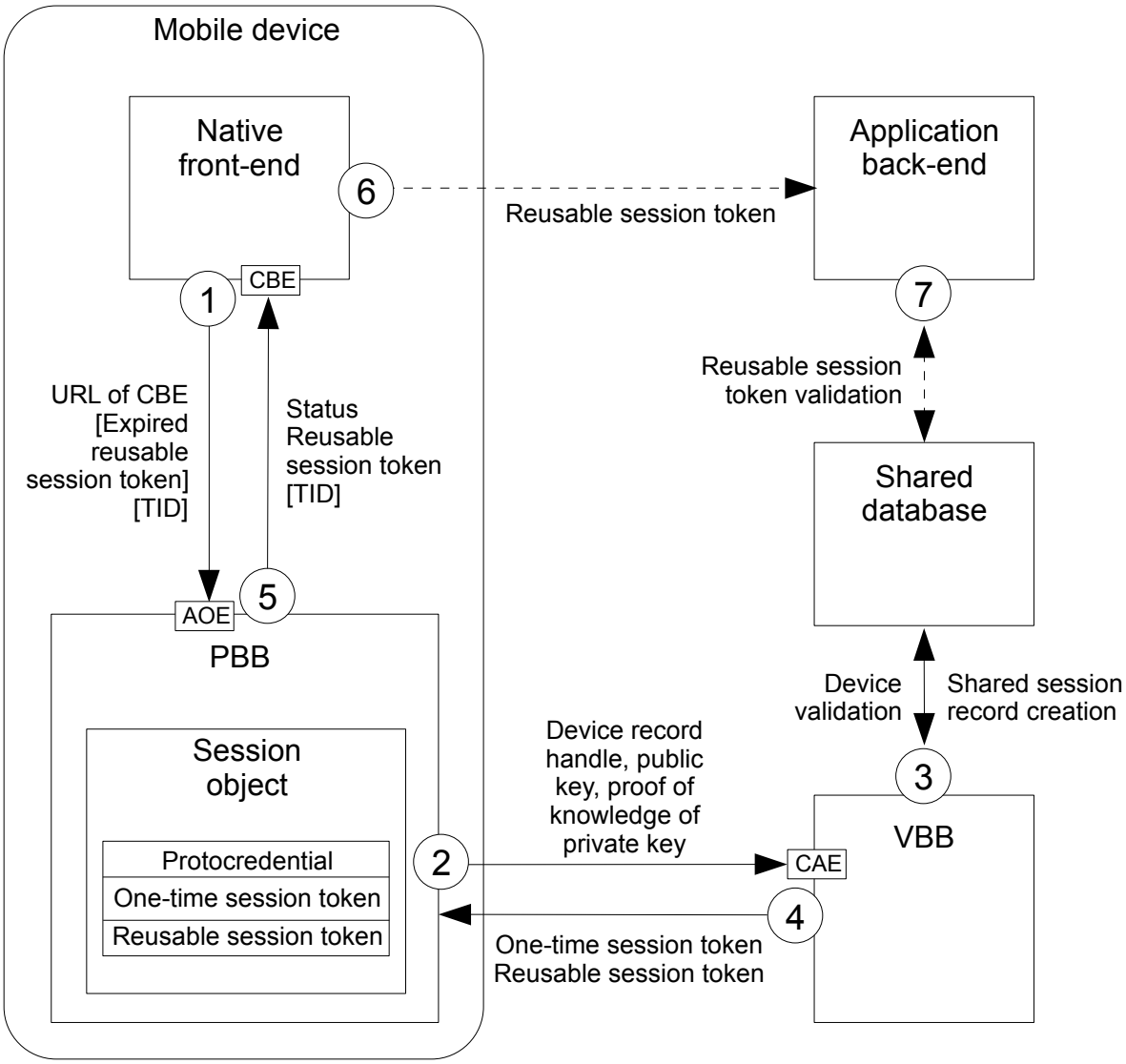


Figure 9: Creation of shared login session by application with native front-end.

authentication fails, the reusable session token is deemed to have expired, and the native front-end sends a second reusable session token request to the PBB, containing the expired reusable session token.[32]

7.5.2 Shared Session Usage by a Web-Based Application

Figure 10 shows the process that takes place when a web-based application requests a one-time session token. It is similar to the process described in Section 7.5.1 by which an application with a native front-end obtains a reusable session token (the "native process").

The process is initiated as a result of an HTTP request sent by the browser to the application back-end that requires authentication.

In step 1, the application back-end issues an HTTP redirection response to the HTTP request,[33] redirecting the browser to a native URL that targets the AOE of the PBB, with parameters including the URL of the CBE, a TID, and optionally either an expired one-time session token or an expired reusable session token. (Different parameter names are used for the two kinds of token.) The redirect request sets a cookie containing the TID in the browser. See sections 7.4 and 8.1 regarding the usage of the TID parameter and cookie.

If the session object in the PBB is populated by a one-time session token and a reusable session token, and no expired session token was included in the request as a parameter, or a one-time session token was included but it is not the one in the session object, or a reusable session token was included but it is not the one in the session object, then steps 2, 3 and 4 are skipped, and at step 5 the PBB sends the one-time session token in the session object to the application back-end via the browser as a parameter of an HTTP GET request, together with a successful status parameter an a parameter containing the TID; the browser adds the TID cookie to the request; the PBB erases the one-time session token from the session object after sending it. Otherwise the process continues at step 2.

Steps 2, 3 and 4 are the same as in the native process.

In step 5, if cryptographic authentication was not skipped, the PBB places the one-time and reusable session tokens received from the VBB in the session object before sending the one-time session token in the session object to the application back-end via the browser as a parameter of an HTTP GET request, together with a successful status parameter and a parameter containing the TID; the browser adds the TID cookie to the request.

At step 6, the application back-end uses the one-time session token to locate the session record, retrieve the reusable session token, and then erase the one-time session token from the record. The application back-end checks that the session record has not expired and the device record handle in the session record points to a valid device record that contains a user record handle pointing to a valid user record. If the session record cannot be found or any of the checks fails, the one-time session token is deemed to have expired, and the application back-end sends a second one-time session token request to the PBB, containing the expired one-time session token, by means of an HTTP redirect response to the HTTP GET request

[32] If the device or user records have been deleted or invalidated, the cryptographic authentication triggered by the second reusable session token request will fail after the user is needlessly prompted for the non-stored secrets. For a more user friendly failure in those cases, the native front-end may let the user know that the device registration or the user account are no longer valid and not send the second request.

[33] Specifically, a response with an HTTP status code equal to 302.

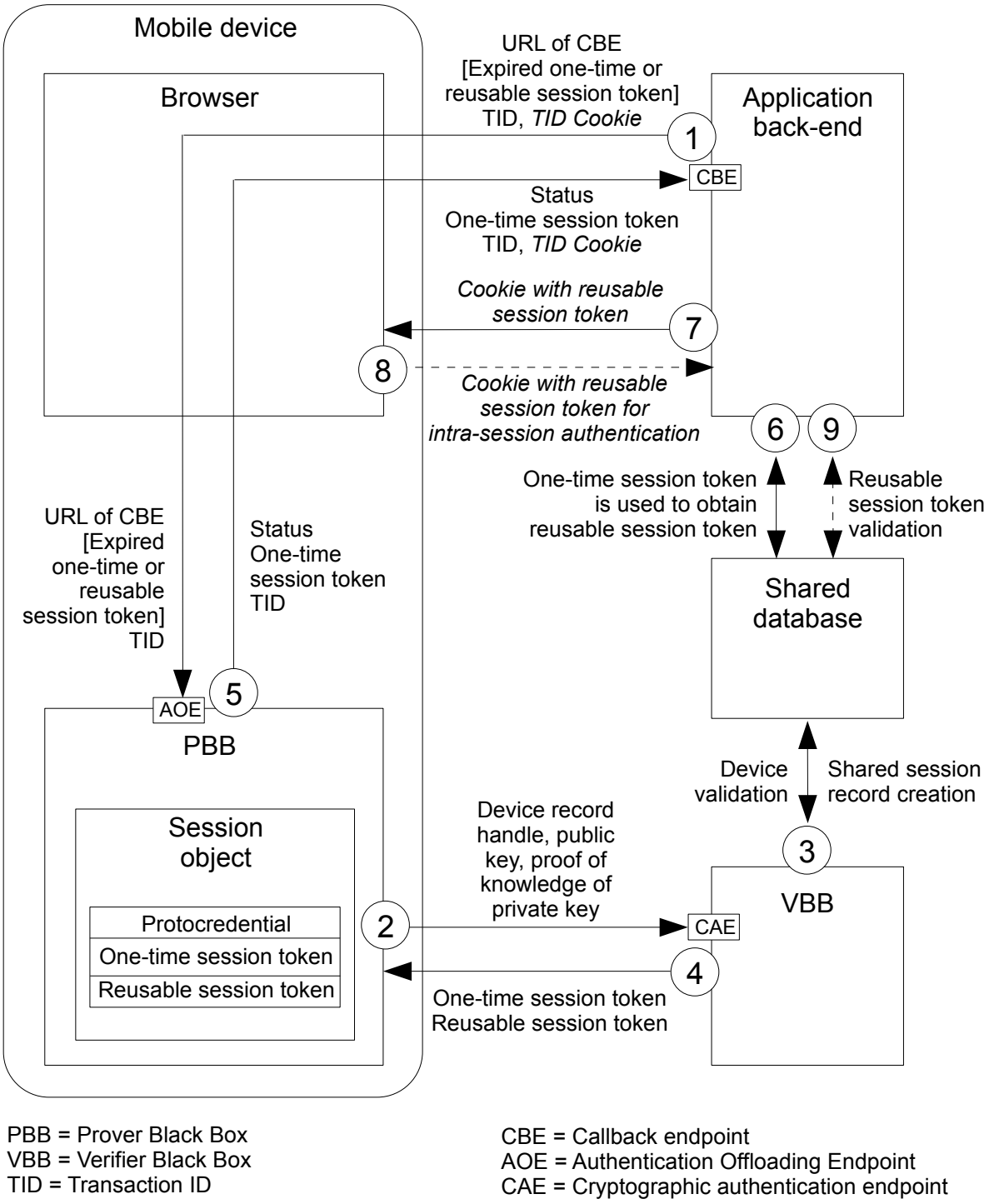


Figure 10: Creation of shared login session by web-based application.

received from the PBB.[34] Otherwise, at step 7, the application back-end sets a secure session cookie in the browser, containing the reusable session token as its value, and with (implicit or explicit) domain and path attributes broad enough so that all web-based applications in the group receive the cookie.

The session cookie is used as usual for intra-session authentication. The browser automatically sends the cookie along with HTTP requests targeting web-based applications in the group; this is shown as step 8 in the figure.

The back-end of an application that receives the cookie uses the reusable session token in the cookie to locate the session record, the device record referenced by the session record, and the user record referenced by the device record; this is shown as step 9 in the figure. As in the native process, if the back-end cannot find a session record containing the reusable session token, or the expiration time in the record has been reached, or the device or user records have been deleted or invalidated, then intra-session authentication fails and the reusable session token is deemed to have expired. In that case the application back-end sends a one-time session token request to the PBB, containing the expired reusable session token in the cookie, by means of an HTTP redirect response to the HTTP GET request that brought the cookie.[35]

It should be noted that, when the web-based application finds that the reusable session token has expired, it sends a one-time session token request that contains the expired reusable session token, not an expired one-time session token. The purpose of sending the expired reusable session token is to allow the PBB to decide whether a new cryptographic authentication is needed. If there is a reusable session token in the session object that coincides with the expired reusable session token in the request, then the shared session referenced by the token is the latest one, and it has expired; therefore a new cryptographic authentication is needed, and, as specified above, one is initiated (steps 2, 3 and 4 are not skipped). Otherwise, if the session object contains a one-time session token (in which case it must also contain a reusable session token) then, as specified above, steps 2, 3 and 4 are skipped and the one-time session token is sent to the application back-end (and erased from the session object). In some corner cases there may be no one-time session token in the session object even though there is a reusable session token in the session object different than the expired reusable session token in the request; this could happen, e.g., if a web-based application has received a one-time session token but has failed to set a cookie. In those cases, a cryptographic authentication is initiated.

[34] As noted with respect to the native process, if the device or user records have been deleted or invalidated, the cryptographic authentication triggered by the one-time session token request will fail after the user is needlessly prompted for the non-stored secrets. For a more user friendly failure in those cases, the application back-end may respond to HTTP GET request from the PBB with an HTML page informing the user that the device registration or the user account are no longer valid.

[35] As noted with respect to the native process, and earlier in this section, if the device or user records have been deleted or invalidated, the cryptographic authentication triggered by the one-time session token request will fail after the user is needlessly prompted for the non-stored secrets. For a more user friendly failure in those cases, the application back-end may respond to the HTTP request that brought the cookie with an HTML page informing the user that the device registration or the user account are no longer valid.

7.5.3 Benefits of SSO Based on Shared Sessions

SSO based on shared sessions is less flexible and more difficult to implement than SSO based on data protection, but it has advantages for performance and security.

The performance advantage comes from the fact that full-fledged cryptographic authentication is avoided while the common login session is in effect.

The security advantage comes from the fact that, in SSO based on data protection, the symmetric key used for data encryption and decryption is available in the clear for a long period of time. If an adversary captures the mobile device and is able to run custom code on it, she may be able to obtain the key and use it to decrypt all the long term credentials that are protected by it. By contrast, in SSO based on shared sessions, if all applications share a protocredential for closed-loop authentication, the key pair regenerated from the protocredential is only present on the device while cryptographic authentication is taking place and the adversary will most likely be able to obtain only the short term session tokens.

7.6 Credential Creation

The PBB and the VBB encapsulate not only the complexities of cryptographic and biometric authentication, but also the even greater complexities of credential creation. Credentials are created by the PBB in conjunction with the VBB, with various degrees of VBB involvement depending on the kind of credential.

Credential creation falls into two broad categories:

1. Device registration for closed loop authentication.

2. Issuance of cryptographic credentials by an authoritative identity or attribute provider for use in open loop authentication.

7.6.1 Device Registration in the Mobile Architecture

Device registration was discussed in general terms in Section 2.5. In this section we are concerned with how the registration process is implemented in the context of the proposed mobile architecture. The details vary depending on whether the VBB is database aware or not, and whether or not a dedicated PBB provides shared login sessions for a group of related applications. We discuss the details for the case where the VBB is not database aware and a general purpose PBB is used by applications unrelated to each other.

When user and device registration take place at the same time, the application sends a request to a device registration endpoint of the PBB, either through a native front-end or through a browser, using the inter-application communication mechanism of the mobile device as in the authentication process. The PBB generates a key pair, after obtaining non-stored secret(s) from the user if multifactor authentication is desired. The PBB also generates a random high-entropy device record handle.[36] The PBB establishes a secure connection to

[36]The device record handle could instead be created by the application back-end or the database management system, and sent to the PBB as part of the device registration request, but that complicates the protocol unnecessarily. The risk of ending up with duplicate handles in the database when the PBB generates the handle is negligible because of the high entropy of the handle.

the cryptographic authentication endpoint of the VBB, sends the public key and the device record handle, and demonstrates knowledge of the private key. The VBB verifies the proof of knowledge, creates an authentication object retrievable by an authentication token which contains the hash of the public key and the device record handle as authentication data, and sends the authentication token to the PBB. Note that the role of a database-unaware VBB is exactly the same during registration as during authentication. A database-unaware VBB appliance programmed for authentication can be used for registration without programming additional registration functionality.

The PBB creates a credential for one-factor authentication, or a protocredential for multifactor authentication containing parameters produced by the key generation process that will later be used to regenerate the key pair at authentication time in conjunction with non-stored secrets provided anew by the user. The PBB also includes the device record handle in the credential or protocredential.

The PBB sends the authentication token to the application back-end, either through the native front-end or through the browser, using the inter-application communication mechanism of the mobile device as in the authentication process. The application back-end uses the authentication token to retrieve the hash of the public key and the device record handle from the VBB, and creates a user record and a device record that references the user record and contains the device record handle. The application may also create a login session as described above in sections 7.3 and 7.4.

The application back-end populates the user record with self-asserted user data. The user data may be obtained from the user during the login session. Alternatively, in the case of an application with a native front-end, the front-end may have obtained the user data before creation of the user and device records and sent it to the back-end along with the authentication token.

A user may also register a device to be used for accessing an existing user account that was created on a different mobile device or on a traditional PC. In that case the user must demonstrate ownership of the account using a non-cryptographic credential, such as a short-lived device registration code obtained on a different device as described above in Section 2.5, a username-and-password credential, or a master password.

The user enters the non-cryptographic credential in the same interaction in which she requests device registration. In the case of an application with a native front-end, the front-end retains the non-cryptographic credential and sends a request to the device registration endpoint of the PBB. The process continues as in the case where there is no preexisting user record until the front-end receives the authentication token. Then the front-end sends the non-cryptographic token and the authentication token together to the back-end. In the case of a web-based application, the back-end receives the non-cryptographic credential directly from the user in an HTTP POST[37] request and sets it in the browser as the value of a cookie as it sends an HTTP redirect response that conveys the device registration request to the PBB. Then the back-end discards the non-cryptographic credential. The process continues as in the case where there is no preexisting user record until the PBB sends an HTTP GET

[37]It must be a POST request because a GET request could expose the non-cryptographic credential to a leak through a "Referer" header. A normal response to the request should be a redirect response (targeting the credential registration endpoint of the PBB) which would not expose the credential. But an error response will download an HTML page that may contain links.

request that conveys the authentication token to the back-end, to which the browser adds the cookie containing the non-cryptographic credential.[38] In either case, once the application back-end has received the non-cryptographic credential and the authentication token, it uses the authentication token to obtain the hash of the public key and the device record handle from the VBB, and it uses the non-cryptographic credential to locate the existing user record. Then it creates a device record containing the device record handle and the hash of the public key, and it stores a reference to the existing user record in the device record.

7.6.2 Issuance of Cryptographic Credentials for Open-Loop Authentication

As discussed above in Section 6.1, cryptographic credentials used in open-loop authentication include public key certificates, U-Prove tokens, and Idemix anonymous credentials. Such credentials are issued by an authoritative identity or attribute provider after the user has demonstrated possession of the attributes asserted by the credential. They are stored by the PBB as full-fledged credentials; protocredentials are not used for open-loop authentication.

To demonstrate possession of the attributes to be included in a credential, the user may first establish an anonymous account at an application supplied by the identity or attribute provider, then provide evidence of possession of the attributes either online or by mail or in person. The attributes are recorded in the user record of the initially-anonymous account, either automatically or by an account administrator.

The application supplied by the identity or attribute provider includes a VBB that is database aware and can access the user record. Once the verified attributes are present in the user record, the credential is created as follows. The application sends a credential creation request to the PBB, either through a native front-end or through a browser, using the inter-application communication mechanism of the mobile device as in the authentication process.

The PBB establishes a secure connection to the VBB and creates the credential in conjunction with the VBB, in a manner dependent on the type of credential:

1. If the credential consists of a public key certificate and its associated private key, the PBB generates the key pair, sends the public key to the VBB and demonstrates knowledge of the private key. The VBB issues a certificate binding the public key to the verified user attributes. The VBB sends the certificate to the PBB which stores it together with the private key as a full-fledged credential.

2. An Idemix anonymous credential [33, 34] is jointly computed by the PBB and the VBB using the Idemix Issuing Protocol [34, §6.1] and stored by the PBB as a full-fledged credential.

3. A U-Prove token [30, 31, 32] is jointly computed by the PBB and the VBB using the U-Prove Issuance Protocol [32, §2.4], and stored in the PBB as full-fledged credential.

[38] Thus the authentication token and the non-cryptographic credential are received by the back-end in the same HTTP request. If no cookie were set and the back-end simply used the non-cryptographic credential received in the POST request, the back-end would have to use some other way of linking the POST request and the GET request as originating from the same device. Discarding the non-cryptographic credential after setting the cookie is a countermeasure against a possible denial-of-service attack by storage exhaustion.

Once one or more public key certificates or privacy-enhancing credentials are installed in the PBB, they can be used to provide evidence for the issuance of other such credentials by other identity or attribute providers. We note, in particular, that Idemix makes it possible to issue a credential upon presentation of one or more existing credentials, with privacy features such as inclusion in the new credential of an attribute asserted by the existing credential without learning the value of the attribute, or verification that two attributes of different existing credentials have the same value without learning the value.

8 Security Analysis of the Mobile Authentication Architecture

In this section we discuss the implications of using bearer tokens for the security of the proposed mobile authentication architecture; then we carry out a case-by-case security analysis according to whether and how the user is asked for consent before the use of a credential; finally we discuss the risk of PBB spoofing.

8.1 Secondary Authentication with a Bearer Token

In Section 4 we saw that multifactor closed loop authentication with credential regeneration from a protocredential provides very strong security, even against an adversary who captures the device or breaches database or network security. But that strong security may be compromised by implementation errors due to the difficulty of using cryptographic APIs correctly [39]. The mobile architecture of Section 7 reduces the risk of implementation errors by encapsulating cryptographic complexity in the PBB and the VBB, so that developers do not have to use a cryptographic API. After the PBB executes a primary cryptographic authentication to the VBB, the native front-end or the browser executes a secondary, non-cryptographic authentication to the application back-end by presenting a bearer token resulting from the primary authentication. The bearer token may be the authentication token of figures 6 and 7, or the one-time or reusable session tokens of figures 9 and 10.

Use of a bearer token has the following implications:

1. Confidentiality protection is needed for the bearer token as it travels in a loop, from the VBB, to the PBB, to the native front-end or browser, to the application back-end, and back to the VBB. Confidentiality is achieved by the use of secure connections between machines, and by entrusting the token to the operating system within the mobile device, as described in previous sections.

2. A bearer token that is included in the URL of an HTTP GET request may leak in a "Referer" header to the target of a link included in a web page downloaded to the browser in the HTTP response to the GET request. Hence such a bearer token should not be used after the HTTP response has been sent. This is the reason why the authentication token is a one-time token in Section 7.4, and the session token sent to the CBE of a web-based application is a one-time token in Section 7.5.[39] ("Referer" leaks

[39]The reason why a one-time authentication token is also used for an application with a native front-end

can be avoided by sending the authentication token in the body of an HTTP POST request instead of a GET request. But that substantially complicates implementation.)

3. At each stage in the loop, the token must be sent to the correct destination. This is discussed case-by-case below in Section 8.2.

4. In third-party closed-loop authentication, an application back-end playing the role of relying party must be prevented from using a bearer token sent to it to impersonate the user vis-a-vis another relying party. This requires making the authentication token specific to each relying party. The VBB makes the token specific by recording the URL of the CBE in the authentication object, and delivering the authentication data to the application back-end only if the back-end uses the same URL to self-assert its identity when presenting the token.

   Misuse of a bearer token is not an issue in two-party authentication or when relying parties can be assumed to trust each other, as in the case of a group of applications that share login sessions, discussed in Section 7.5.

5. A web-based application must prevent a Login CSRF attack where malicious user (the attacker) causes another user (the victim) to log in to the attacker's account. The attacker can carry out the attack by setting up an unrelated web site and luring the victim into visiting the site and clicking on a link whose URL is the URL of the callback endpoint augmented with a query string that contains the attacker's bearer token. If the victim does not realize that she is accessing the wrong account she may enter confidential information that will be captured by the attacker. The attack is prevented by setting the TID cookie as described in Section 7.4 and shown in figures 7 and 10. The cookie causes the attack to fail because the attacker cannot inject the cookie into the victim's browser. Discussion of CSRF attacks in other settings can be found in [71].

8.2 Case Analysis

Security in the proposed mobile architecture depends to a great extent on whether and how the user is asked for consent before using a credential and disclosing attributes. The request may specify the party to which the credential will be presented (the party that owns the VBB) and/or the party to which attributes will be disclosed (the application), if different. We distinguish four cases.

8.2.1 Two-Party Closed Loop Authentication

In this case there is no need to ask the user for consent. The URL of the CAE of the VBB is recorded in the credential metadata when the credential is created and the PBB uses it at authentication time to establish a secure connection to the VBB. The URL of the CBE is also recorded in the credential metadata, and the PBB uses it at authentication time to send the authentication token.

in Section 7.3 is so that the VBB does not have to be concerned with whether the application is web-based or has a native front-end.

There are two security issues related to sending the bearer token to the CBE:

1. If the application has a native front-end, the URL of the CBE uses a custom scheme that identifies the application. However, a malicious application may register the same custom scheme.

   In iOS, all applications are distributed through the Apple app store after they have been inspected and approved by Apple. Security depends on the assumption that no malicious applications are running on the phone.

   In Android, the PBB can determine the package name of the application that handles the scheme (i.e. receives messages that target URLs constructed with that scheme). It should record the package name in the credential metadata when the credential is created, and verify that it has not changed at authentication time.

2. If the application is web-based, the URL of the CBE uses the https scheme. In Android, a malicious application may be able to register to receive messages that use the https scheme. As a countermeasure, the PBB should record at credential time in the credential metadata the package name of the application that handles https messages (which we have been calling the default browser), verify that it has not changed at authentication time, and warn the user if it has. No such countermeasure is needed in iOS because applications cannot register the https scheme.

8.2.2 Shared Login Sessions

In Section 7.5 we saw how a form of SSO suitable for enterprise applications can be implemented by allowing a group of applications to share login sessions, in what we called groupwise closed-loop authentication. Since the goal is to provide SSO the user should not be asked for consent each time she wants to access an application. So the PBB cannot rely on the user to decide whether an application is a member of the group entitled to receive a session token.

Instead, for web-based applications, the PBB can decide based on the hostname of the URL of the CBE endpoint. As in the two-party case, in Android the PBB should check that the package name of the application that handles https messages has not changed since the credential was created.

For applications with a native front-end, however, the PBB cannot decide based on the URL of the CBE with sufficient security, due to the fact that a malicious application may register the custom scheme used in the URL of the CBE. In Android, the scheme registration problem was addressed in the two-party case by recording in the credential metadata the package name of the application handling the scheme; this is not possible in the group case, however, since each application in the group uses a different scheme. In iOS we relied on the vetting of applications by the Apple app store; but this may not be deemed a sufficient basis for the security of enterprise applications in a BYOD context.

Instead, the PBB should limit the native applications that it accepts requests from. In Android this can be achieved by using the same code-signing key pair to sign the PBB and each application in the group, and using the signature-based permission mechanism to limit access to the PBB to native applications signed by that key pair. In iOS, it can be achieved

by using the same team ID for the PBB and all the applications in the group, thus allowing them to share keychain data; the PBB may then limit access to native applications that demonstrate knowledge of a symmetric key stored in the key chain by using it to sign a challenge.

8.2.3 Third-Party Closed-Loop Authentication

In third-party closed-loop authentication, the primary, cryptographic authentication is to an identity or attribute provider, while the secondary, non-cryptographic authentication is to a relying party. The URL of the CAE of the VBB is recorded in the credential metadata when the credential is created, but the URL of the CBE is supplied to the PBB by the relying party.

There is no need to ask the user for consent to perform the primary authentication, but consent is needed before disclosing user attributes to the relying party (or allowing the relying party to access the user's account in the case of social login). When asking for consent, the relying party must be identified to the user.

A web-based application can be identified based on information in the TLS certificate presented by the CBE. When a relying party requests authentication for the first time, the PBB should access the CBE and obtain the certificate before asking for consent. The authentication token is sent to the CBE via the browser. As in the two-party case, the PBB should verify that the package name of the application that handles https messages has not changed since the credential was created.

In Android, when the URL of the CBE is native, the PBB can obtain from the operating system the icon and human-readable label of the native application targeted by the URL, and use them to identify the relying party to the user. In iOS there is no equivalent functionality at present, so the user should be told that the relying party is a native application, without specifying its identity.

8.2.4 Open-Loop Authentication

In open-loop authentication, the primary and secondary authentications are typically to the same party, in the sense that the VBB is part of the application back-end, or at least trusted by the back-end. However, security cannot rely on that assumption. For example, an adversary could use a malicious application to make an authentication request to the PBB specifying a CBE that the user trusts, but a VBB controlled by the adversary. If the user gave consent to the authentication transaction based on the fact that she trusts the CBE, the adversary would obtain user attributes when the PBB executes the primary authentication to the VBB. Therefore when asking the user's consent to authenticate and disclose user attributes, the PBB should separately identify the party that owns the CAE (of the VBB) and the party that owns the CBE (of the application), unless it can determine that they are the same party.

The party that owns the CAE can be identified based on information in the TLS certificate presented by the CAE. The PBB should obtain the certificate by making a connection before asking for consent. The session resumption feature of TLS is likely to eliminate the need for

a second TLS handshake when the PBB makes a second connection to execute the primary authentication transaction.

If the application is web-based, the PBB can similarly identify the party that owns the CBE using information in the TLS certificate presented by the CBE. In Android, as in the two-party and third-party closed-loop cases, the PBB should verify that the package name of the application that handles https messages has not changed since the credential was created.

If the application has a native front-end, the PBB should identify the native front-end as in the third-party closed-loop case.

8.3 Risk of PBB Spoofing

In the mobile architecture security depends, of course, on using a legitimate PBB. An adversary could spoof the PBB by tricking the user into installing a malicious application that purports to be the legitimate PBB, or a seemingly unrelated malicious application that registers the custom scheme of the AOE of the PBB.

In iOS, these risks are mitigated by the fact that all applications are distributed through the Apple app store after they have been inspected and approved by Apple.

In Android, on the other hand, there is at present no way of verifying that a native application is legitimate. In fact, malware is often distributed by embedding it in modified versions of legitimate applications [72]. Android applications are signed, but the code-signing certificate is typically self-signed. Symantec offers code-signing certificates [73], but Android app stores do not provide a mechanism for making trust decisions based on the code-signing certificate of an application and the CA who has issued the certificate; and the Android operating system does not give an application access to the code-signing certificate of a different application installed on the same mobile device.

Thus, in Android, for the time being, security depends on the user being careful and installing a legitimate PBB rather than a fake one. This should not be a problem when the PBB is used to provide cryptographic authentication for a group of enterprise applications as described in Section 7.5, because the PBB can be supplied by the IT department; but it may be a problem for consumer applications.

However, Android allows multiple applications to be bundled within the same package. The problem can be solved for a consumer application by bundling the PBB in the same package as the application, as illustrated in Figure 11. Figure 11 is the same as Figure 6, except that:

1. The native front-end and the PBB are bundled into the same package.

2. The URL of the DRE is not shown as optionally being sent in steps 4 and 5 because it is known to the application back-end.

3. The URL of the CBE is not shown as optionally being sent in step 1 because it is recorded in the credential metadata.

4. The URL of the CBE is not shown as optionally being sent in steps 2 and 6 and stored in the authentication object within the VBB because the PBB cannot be used for third-party closed-loop authentication.

The URL of the CAE is shown is optional in step 1 of Figure 11 as in step 1 of Figure 6, but for different reasons. In Figure 6 it was optional to allow the relying party in third-party close-loop authentication to not specify the identity or attribute provider to be used. In Figure 11 it is optional because the PBB is only used by one application, and thus the CAE of the VBB could be configured in the PBB.

In particular, high security Android applications (used, e.g., for online banking or patient access to a health care institution) should bundle the PBB to avoid using a fake PBB present in the user's device.[40]

Notice, however, that if the application uses biometric authentication, embedding the PBB in the native front-end may allow the application to obtain the user's biometric sample at enrollment time and authentication time. This is a privacy drawback if the user trusts the application less than it would a separate PBB application downloaded from a trusted source.

9 Authentication on traditional PCs

Mobile devices have produced a new computer paradigm whose ingredients include gesture-based interfaces, native apps downloadable from app stores, a native-app/web-based app duality, sandboxing, and URL-based inter-app communication facilities. There is no doubt that laptops and desktops will sooner or later migrate to the new paradigm. Migration is beginning to happen as witnessed by the introduction of gestures in Mac computers and more recently by the dual launch of Windows 8 and Windows RT.[41] Once it has been completed laptops and desktops will be able to take advantage of the authentication architecture of Section 7. (Notice that the protocredential-based multifactor closed-loop authentication technique of sections 2 and 3 is not specific to mobile devices and can be implemented on traditional PCs today, by integrating cryptographic and biometric functionality directly into browsers and applications.)

But if full migration takes time, it may be desirable to emulate some elements of the architecture within traditional PC computing. In particular, web application developers would benefit from being able to use the same authentication methods when a web application is accessed through a browser running on a mobile device as when it is accessed through a browser running on a traditional PC. We propose to make this possible by implementing the PBB as a browser extension.[42] A redirect HTTP response that would be forwarded to a PBB implemented as native application in a mobile device is intercepted and handled by the browser extension in a traditional PC. Figure 12 illustrates the technique.

An important benefit of this technique is that it can be implemented without in-depth knowledge of browser internals. The only knowledge of browser software required is how an extension can intercept an HTTP redirect response. The functionality of the PBB extension

[40]The high security application itself will remain vulnerable to spoofing, but a bank or health care institution should be able to provide out-of-band instructions to users on how to install its mobile application.

[41]It is unfortunate that, while Microsoft is championing the convergence of mobile and desktop computing, neither Windows 8 nor Windows RT provide a mechanism for communication between native applications equivalent to those of Android and iOS.

[42]Most browsers have facilities that allow developers to create functionality extensions, variously called plug-ins, add-ons, helper objects, etc.

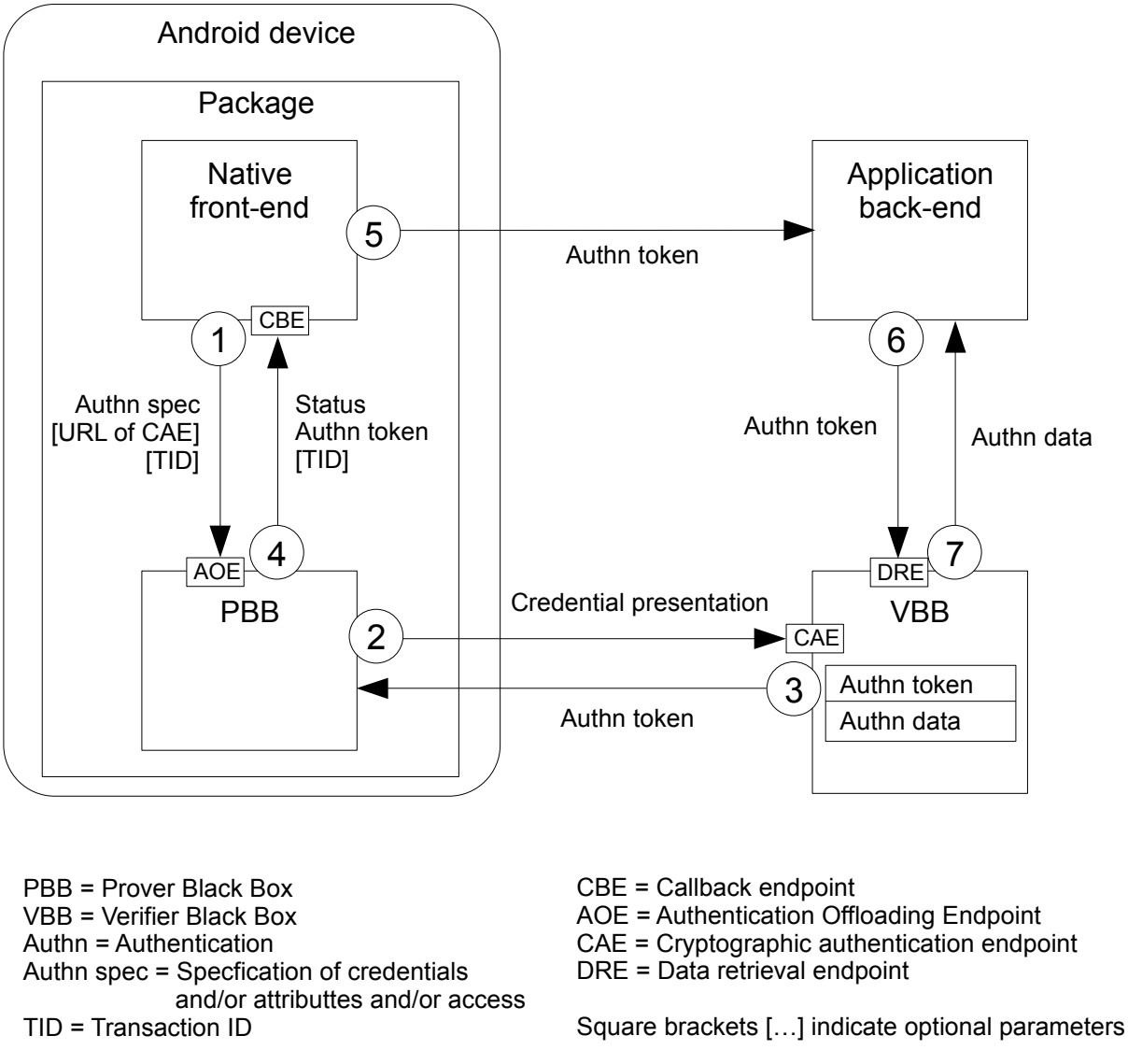
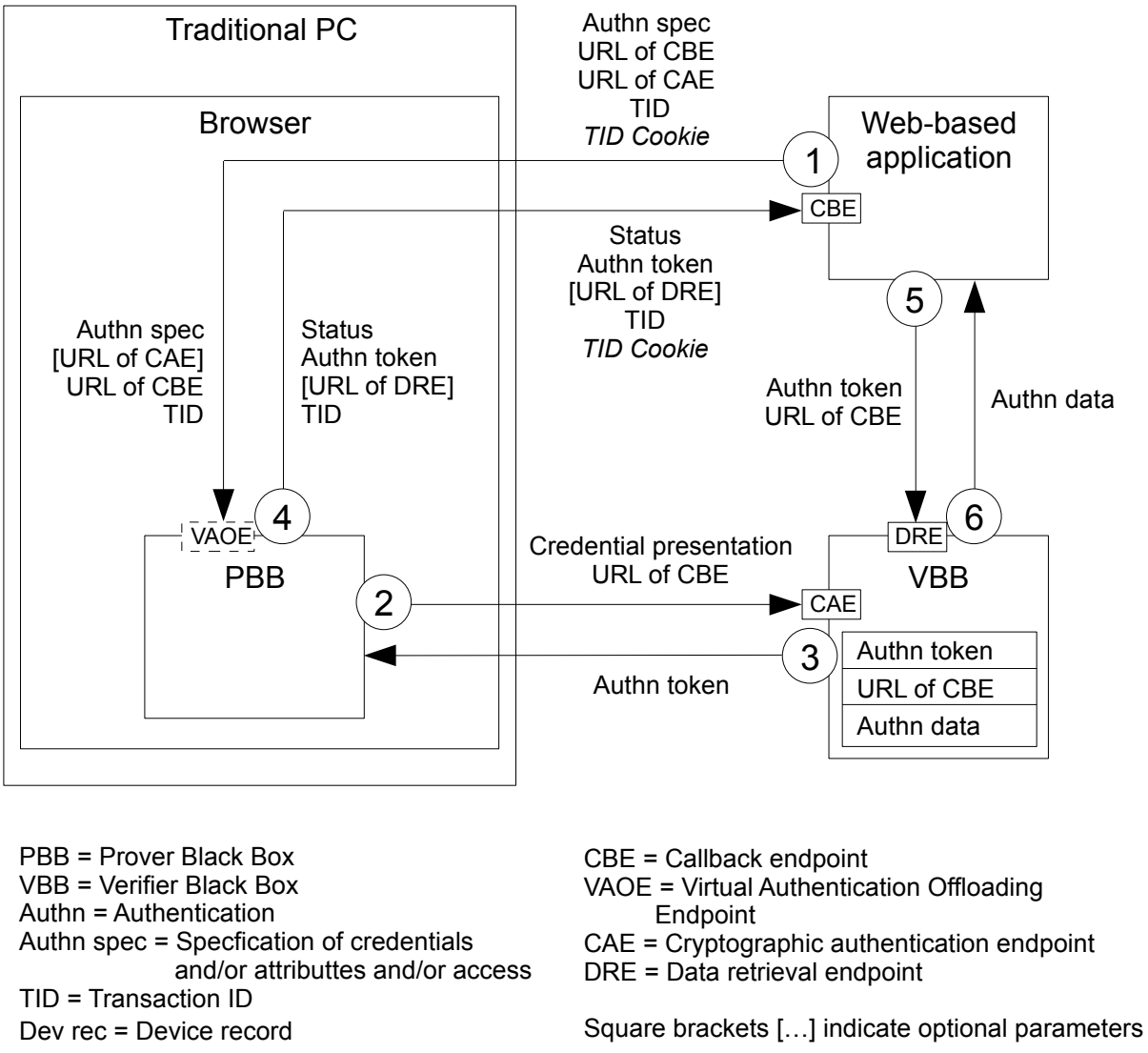


Figure 11: PBB bundled into same package as an Android native application front-end



PBB = Prover Black Box
VBB = Verifier Black Box
Authn = Authentication
Authn spec = Specfication of credentials and/or attributtes and/or access
TID = Transaction ID
Dev rec = Device record

CBE = Callback endpoint
VAOE = Virtual Authentication Offloading Endpoint
CAE = Cryptographic authentication endpoint
DRE = Data retrieval endpoint

Square brackets […] indicate optional parameters

Figure 12: PBB implemented as browser extension in traditional PC

is implemented separately and independently from the browser internals, and is identical to the functionality of a PBB implemented as a native application in a mobile device.[43]

10 Privacy Considerations

Traditional passwords have an important benefit that is often overlooked: they can provide full privacy. When a user logs in to a web site with username and password, if the username was freely chosen by the user during registration (if it is not, e.g., a verified email address), then the user can be anonymous, and no third party is informed of the login. This privacy is lost in third-party login, using a protocol such as OpenID or OAuth, because the identity provider is informed of the login [75]; often, the user is only offered a limited choice of identity providers by the relying party, and is thus forced to surrender her privacy to an identity provider that she may not trust.

Our method of two-party closed-loop authentication eliminates the security drawbacks of logging in with username and password while preserving full privacy and anonymity. This is so even when a biometric is used as a factor in multifactor authentication. (The privacy and security implications of using a biometric were discussed above in Sections 4.4.1 and 4.4.2.)

Our method of third-party closed-loop authentication on mobile devices, in addition to eliminating the vulnerability to phishing attacks inherent in password-based third-party login, provides freedom of choice of identity or attribute provider, because the relying party can specify the attributes it needs rather than the identity providers it supports.

Third-party closed-loop authentication should only be used in a limited number of cases. It is convenient for social login and for implementing a trusted repository of self-asserted personal data [76]. It should not be used for attributes for which the relying party requires stronger evidence than self-assertion. Such attributes should be asserted by open-loop authentication using public key certificates or anonymous credentials issued by authoritative parties, without involvement of non-authoritative identity providers as middlemen.

Open-loop authentication with a public key certificate provides a great deal of privacy because the certificate issuer is not told how the certificate is used (although it may learn the identity of the relying party from its IP address if OCSP is used for certificate revocation checking). When the relying party only requires an attribute, such as age or nationality, that does not uniquely identify the user, then an Idemix anonymous credential provides additional privacy by preventing user tracking even if relying parties collude, and by preventing user identification even if a relying party colludes with the credential issuer; a U-Prove token also provides the latter feature.[44]

While our mobile architecture should make it easier to deploy privacy-enhancing credentials, we do not underestimate the deployment difficulties that remain. We have discussed

[43] By contrast, we proposed earlier an authentication architecture for the NSTIC ecosystem [74] that facilitated the use of cryptographic credentials, but required substantial modifications of browser software and of the TLS protocol. No modification of browser internals or the TLS protocol are required by the present approach.

[44] Notice that a privacy-enhancing credential that discloses an attribute that uniquely identifies the user provides no more privacy than a public key certificate that binds a public key to that same attribute, since disclosure of the attribute makes presentations of the credential linkable. .

some of these difficulties in the past in the Pomcor blog [77, 78, 79]. It is inherently difficult to revoke privacy-enhancing credentials. At this time Idemix does not provide a means of revoking credentials. Instead, it provides an efficient protocol for *updating a credential*, which facilitates issuing short-time credentials whose expiration time is postponed on a regular schedule. U-Prove credentials are not revocable by the issuer. The U-Prove Technology Overview [31, §5.2] suggests issuing U-Prove tokens *on-demand*, including a nonce supplied by the verifier as an attribute not seen by the issuer to ensure freshness. It is also inherently difficult to prevent the fraudulent sharing of credentials.

The privacy implications of using a biometric sample have been discussed in several earlier sections. To recapitulate: Neither biometric samples or biometric templates are stored in the device nor in any database, and they are not presented to applications or to a certificate authority. Uses of the same biometric for authentication to different applications are not linkable. Malware running on the user's device after an adversary has captured the device cannot obtain data that can be related to the user's biometric features. However malware running on the device while the user is using it could prompt the user for a biometric sample, and embedding the PBB in the native front-end of a web application may allow the front-end to obtain a biometric sample.

11 Conclusion

We have defined the concept of *closed-loop authentication*, where the same party that issues or registers a credential is later responsible for verifying possession of the credential, whereas in *open-loop authentication* the party that issues or registers a credential is not directly involved in the authentication process. Two-party authentication is necessarily closed-loop, but a third party can assert the user's identity or attributes either in open-loop, by issuing a public key certificate or a privacy-enhancing credential such as a U-Prove token or an Idemix anonymous credential, or in closed-loop, by communicating the user's identity or attributes to a relying party after verifying possession of the credential by the user. Open-loop provides more privacy than closed-loop when a third party is involved, because the third-party observes the authentication transactions in closed-loop authentication.

We have proposed a method of multifactor closed-loop authentication where an uncertified key pair is regenerated at authentication time from a protocredential stored in the user's device and one or more non-stored secrets, such as a passcode or a biometric sample, supplied by the user; the verifier stores the hash of the public key in a database record and keeps it secret. We have compared protocredential-based two-party authentication against five other two-party authentication methods, and shown that it provides a stronger security posture against an adversary who may be able to capture the protocredential or breach database or network security.

The protocredential-based authentication technique cannot be used directly to implement open-loop or one-factor closed-loop authentication methods, but we have proposed a data protection technique that can be used to protect credentials used in such methods. Data is protected by encrypting it under a data-encryption key that is entrusted to a key storage service or distributed over several services using Shamir's secret sharing technique,

and protocredential-based two-factor authentication is used to retrieve the key. The data protection technique can be used to implement global single sign-on by encrypting multiple credentials used for one-factor closed-loop authentication under the same data-encryption key, and to extend the security benefits of protocredential-based authentication to open-loop authentication.

We have proposed an architecture for authentication on mobile devices that allows both web-based applications and applications with native front-ends to offload both closed-loop and open-loop authentication to a prover black box (PBB) and a verifier black box (VBB), taking advantage of an inter-application communication mechanism available in Android and iOS. The PBB and the VBB encapsulate the complexities of cryptographic and biometric authentication, making its benefits easily available to application developers.

We have shown how the architecture allows a group of applications to share login sessions, thus providing a second form of single sign-on, which is well suited for enterprise SSO in a BYOD context. We have shown how the architecture can be adapted for use on traditional PCs using browser extensions, while waiting for the app-centric computing paradigm introduced by mobile devices to fully migrate to desktops and laptops.

Skeptics may argue that many earlier attempts at using cryptography for user authentication have failed to displace passwords on the Internet. We respond by pointing out that our approach removes many of the obstacles that have faced earlier approaches. The mobile architecture can be used by web-based applications on mobile devices without browser modification, and by applications with native front-ends without browser involvement. No modifications of the TLS protocol are required. No special hardware is required. No tamper resistance is required. And no knowledge of cryptography or biometrics is required from application developers. We hope that the approach we are proposing will facilitate the broad deployment of cryptographic and biometric authentication on the Internet.

A Appendices

A.1 Bit Length of the RSA Decryption Exponent

The algorithm for initial generation of an RSA key pair specified in Section 2.6.4 checks the bitlength of d at step 7 and starts over if it is no greater than $l/2$. We argue here that this is very unlikely.

Using the notations of Section 2.6.4, recall that $d = c'/c''$, where $c' = (c \bmod \lambda - 1) + 1$ and $c'' = \mathrm{GCD}(c', \lambda)$.

If c is roughly uniformly distributed over $[0, 2^{l+64})$, then c' is roughly uniformly distributed over $[1, \lambda - 1]$. This means that, l' being the bitlength of λ, the bitlength of c' is no less than $l' - k$ with probability roughly 2^k; for example, the bitlength of c' is no less than $l' - 128$ with negligible probability $\approx 2^{-128}$.

Recall that $\lambda = (p-1)(q-1)/\mathrm{GCD}(p-1, q-1)$. The bitlength of $(p-1)(q-1)$ is the bitlength l of the modulus. The bitlength of $\mathrm{GCD}(p-1, q-1)$ depends on the distributions of $p-1$ and $q-1$, but is unlikely to be more a few bits. Indeed, we prove in Appendix A.2 that the expected value of $\log_2(\mathrm{GCD}(u, u'))$ is, somewhat surprisingly, less than 1 if u and u' are random numbers independently and uniformly distributed over $[1, k]$ and $[1, k']$ respectively

(for any $k, k' > 0$); $p-1$ and $q-1$ are not uniformly distributed over initial segments, but their specific distributions should not make a big difference as long as they are independent.

Finally, $d = c'/c''$ with $c'' = \mathrm{GCD}(c', \lambda)$, and c'' is unlikely to be more than a few bits long because c' and λ are independently generated, based again on the result of Appendix A.2.

To recapitulate, the bitlength of $(p-1)(q-1)$ is l, λ is likely to be no more than a few bits shorter, c' is no more than 128 bits shorter than λ with negligible probability $\approx 2^{-128}$, and d is likely to be no more than a few bits shorter than c'. So d is very likely to be greater than $l/2$ if l has one of the bitlengths specified for the RSA modulus in [45], viz. 1024, 2048 or 3072.

A.2 Size of the GCD of two Random Numbers

Let u and u' be two random integers independently and uniformly distributed over $[1, k]$ and $[1, k']$ respectively ($k, k' \geq 2$). We want to prove the somewhat surprising result that the expected value of

$$E = \log_2(\mathrm{GCD}(u, u'))$$

is less than 1.

Let p be a prime number, and let $\mathrm{prob}(p, i)$, $i \geq 1$, be the probability that p^i divides $\mathrm{GCD}(u, u')$. We have

$$\mathrm{prob}(p, i) \leq \frac{1}{p^{2i}}.$$

The expected value of $\log_2$ of the largest power of p that divides $\mathrm{GCD}(u, u')$ is

$$\begin{aligned}
&\sum_{i \geq 1} \log_2(p^i) \cdot (\mathrm{prob}(p, i) - \mathrm{prob}(p, i+1)) \\
&= \log_2(p) \cdot \sum_{i \geq 1} i(\mathrm{prob}(p, i) - \mathrm{prob}(p, i+1)) \\
&= \log_2(p) \cdot \sum_{i \geq 1} \mathrm{prob}(p, i) \\
&\leq \log_2(p) \cdot \sum_{i \geq 1} \frac{1}{p^{2i}} \\
&= \frac{\log_2(p)}{p^2 - 1}
\end{aligned}$$

So the expected value of $\log_2(\mathrm{GCD}(u, u'))$ is

$$E = \sum_{j \geq 1} \frac{\log_2(p_j)}{p_j^2 - 1}$$

where p_j is the j-th prime. Let

$$F(x) = \frac{\log_2(x)}{x^2 - 1}.$$

so that

$$E = \frac{1}{\ln 2} \sum_{j \geq 1} F(p_j).$$

Let $\pi(x)$ be the number of primes less than or equal to x and

$$g(x) = 1.25506\frac{x}{\ln x}.$$

It is known [46, Fact 2.96] that $\pi(x) < g(x)$ for $x > 1$. The function $g(x)$ is monotonically increasing for $x \leq e$. Let $h(x)$ be the inverse of the restriction of g to the interval $[e, +\infty)$, i.e. the monotonically increasing function defined over the interval $[g(e), +\infty)$ by $h(g(x)) = x$. Since $g(e) < 4$ and $\pi(x) < g(x)$ we have, for $j \geq 4$, $j = \pi(p_j) < g(p_j)$ and $h(j) < h(g(p_j)) = p_j$. Because h is monotonically decreasing over over the range of h we have, for $j \geq 4$,

$$F(p_j) < F(h(j))$$

and

$$\sum_{j\geq 5} F(p_j) < \sum_{j\geq 5} F(h(j)) < \int_{u=4}^{+\infty} F(h(u)du.$$

Letting $u = g(x)$ and $du = g'(x)dx$,

$$\begin{aligned}
\int_{u=4}^{+\infty} F(h(u)du &= \int_{x=h(4)}^{+\infty} F(x)g'(x)d \\
&= \int_{x=h(4)}^{+\infty} \frac{(\ln x) - 1}{(x^2-1)\ln x}dx \\
&= \int_{x=h(4)}^{+\infty} \frac{1}{x^2}\frac{1-\frac{1}{\ln x}}{1-\frac{1}{x^2}}dx \\
&< \int_{x=h(4)}^{+\infty} \frac{1}{x^2}dx \\
&= \left[-\frac{1}{x}\right]_{h(4)}^{+\infty} \\
&= \frac{1}{h(4)}
\end{aligned}$$

Since $g(5) = 3.8990\ldots$, we have $g(5) < 4$, and $5 < h(4)$, and

$$\frac{1}{h(4)} < \frac{1}{5}.$$

Therefore

$$\begin{aligned}
\sum_{j\geq 1} F(p_j) &= F(p_1) + F(p_2) + F(p_3) + F(p_4) + \sum_{j\geq 5} F(p_j) \\
&< F(2) + F(3) + F(5) + F(7) + \frac{1}{5} \\
&= 0.676...
\end{aligned}$$

and

$$\begin{aligned}
E &= \frac{1}{\ln 2}\sum_{j\geq 1} F(p_j) \\
&= 0.975... \\
&< 1.
\end{aligned}$$

A.3 Probability of two Random Numbers Having a Common Prime Factor Greater than 100

The probability that two random numbers u and v independently and uniformly distributed over intervals $[1, N]$ and $[1, N']$ have a common prime factor belonging to a set P of prime numbers is no greater than

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

In particular, if p_i is the i-th prime number, the probability that u and v have a common prime factor greater than the j-th prime number is no greater 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... [80]. There are 25 prime numbers less than 100, and

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

Therefore the probability that u and v have a common prime factor greater than 100 is less than $0.452248 - 0.450428 = 0.0018 < 0.2\%$.

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] Zack Whittaker. Let's get rid of usernames and passwords for good. November 23, 2009. http://www.zdnet.com/blog/igeneration/lets-get-rid-of-usernames-and-passwords-for-good/2498.

[6] Mat Honan. Kill the Password: Why a String of Characters Cant Protect Us Anymore. November 15, 2012. http://www.wired.com/gadgetlab/2012/11/ff-mat-honan-password-hacker/.

[7] David P. Kormann and Aviel D. Rubin. Risks of the Passport Single Signon Protocol. *Computer Networks*, 33:51–58, 2000. http://avirubin.com/passport.html.

[8] Ben Laurie. OpenID: Phishing Heaven, January 19, 2007. http://www.links.org/?p=187.

[9] EMC. RSA SecurID Hardware Authenticators. http://www.emc.com/security/rsa-securid/rsa-securid-hardware-authenticators.htm.

[10] D. M'Raihi, M. Bellare, F. Hoornaert, D. Naccache, and O. Ranen. HOTP: An HMAC-Based One-Time Password Algorithm, December 2005. http://tools.ietf.org/html/rfc4226.

[11] Confident Technologies Delivers Image-Based, Multifactor Authentication to Strengthen Passwords on Public-Facing Websites. October 2010. http://www.marketwire.com/press-release/Confident-Technologies-Delivers-Image-Based-Multifactor-Authentication-Strengthen-Pa htm.

[12] Arthur W. Coviello, Jr. Open Letter to RSA Customers. http://www.rsa.com/node.aspx?id=3872.

[13] John Markoff. SecurID Company Suffers a Breach of Data Security. March 17, 2011. http://www.nytimes.com/2011/03/18/technology/18secure.html.

[14] Elinor Mills. Attack on RSA used zero-day Flash exploit in Excel. April 5, 2011. http://news.cnet.com/8301-27080_3-20051071-245.html.

[15] T. Dierks and E. Rescorla. The Transport Layer Security (TLS) Protocol Version 1.2, August 2008. http://tools.ietf.org/html/rfc5246.

[16] WebID Incubator Group. WebID - Universal Login and Identity for the Web. At http://webid.info/.

[17] Mozilla. BrowserID Protocol Overview. https://developer.mozilla.org/en-US/docs/Persona/Protocol_Overview.

[18] M. Dietz, A. Czeskis, D.S. Wallach, and D. Balfanz. Origin-Bound Certificates: A Fresh Approach to Strong Client Authentication for the Web, 2012.

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

[20] Apple Inc. iOS: Understanding data protection. http://support.apple.com/kb/HT4175.

[21] Andrey Belenko. Overcoming iOS Data Protection To Re-Enable iPhone Forensics. Presentation at BlackHat 2011. https://media.blackhat.com/bh-us-11/Belenko/BH_US_11_Belenko_iOS_Forensics_WP.pdf.

[22] 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.

[23] J. Hughes et al. Profiles for the OASIS Security Assertion Markup Language (SAML) V2.0. OASIS Standard, March 2005. http://docs.oasis-open.org/security/saml/v2.0/saml-profiles-2.0-os.pdf.

[24] Tom Scavo and Scott Cantor. Shibboleth Architecture Technical Overview, Working Draft 02, June 2005. http://shibboleth.internet2.edu/docs/draft-mace-shibboleth-tech-overview-latest.pdf.

[25] OpenID Foundation. OpenID Authentication 2.0 Final, December 5, 2007. http://openid.net/specs/openid-authentication-2_0.html.

[26] OAuth Working Group Documents. http://datatracker.ietf.org/wg/oauth/.

[27] OpenID Foundation. Welcome to OpenID Connect. http://openid.net/connect/.

[28] 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.

[29] 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/.

[30] Microsoft Corporation. U-Prove Home Page. http://www.microsoft.com/u-prove.

[31] Christian Paquin. U-Prove Technology Overview V1.1 Draft Revision 1, February 2011. There is no http URL for this document, but it can be downloaded by following links from http://www.microsoft.com/u-prove.

[32] Christian Paquin. U-Prove Cryptographic Specification V1.1 Draft Revision 1, February 2011. There is no http URL for this document, but it can be downloaded by following links from http://www.microsoft.com/u-prove.

[33] J. Camenisch, P. Bichsel, and T. Gross. Idemix Blog. http://idemix.wordpress.com/.

[34] Jan Camenisch et al. Specification of the Identity Mixer Cryptographic Library, Version 2.3.4, February 10, 2012. Downloadable from https://prime.inf.tu-dresden.de/idemix/.

[35] 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.

[36] Ari Juels and Martin Wattenberg. A fuzzy commitment scheme. In *Proceedings of the 6th ACM conference on Computer and communications security*, CCS '99, pages 28–36, New York, NY, USA, 1999. ACM.

[37] 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.

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

[39] Martin Georgiev, Subodh Iyengar, Suman Jana, Rishita Anubhai, Dan Boneh, and Vitaly Shmatikov. The most dangerous code in the world: validating ssl certificates in non-browser software. In *ACM Conference on Computer and Communications Security*, pages 38–49, 2012.

[40] Phillip Hallam-Baker. The Recent RA Compromise. March 23, 2011. http://blogs.comodo.com/it-security/data-security/the-recent-ra-compromise/.

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

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

[43] NSA. Suite B Cryptography. http://www.nsa.gov/ia/programs/suiteb_cryptography/index.shtml.

[44] Elaine Barker and John Kelsey. Recommendation for Random Number Generation Using Deterministic Random Bit Generators, January 2012. NIST Special Publication 800-90A.

[45] NIST. Digital Signature Standard (DSS), June 2009. FIPS PUB 186-3, http://csrc.nist.gov/publications/fips/fips186-3/fips_186-3.pdf.

[46] 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/.

[47] D. Johnson, A. Menezes, and S. Vanstone. The Elliptic Curve Digital Signature Algorithm (ECDSA), 2001. Certicom whitepaper, http://cs.ucsb.edu/~koc/ccs130h/notes/ecdsa-cert.pdf.

[48] Nalini K. Ratha, Jonathan H. Connell, and Ruud M. Bolle. Enhancing security and privacy in biometrics-based authentication systems. *IBM Systems Journal*, 40(3):614–634, 2001.

[49] Ari Juels and Madhu Sudan. A fuzzy vault scheme. In *In International Symposium on Information Theory (ISIT*, page 408. IEEE Press, 2002.

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

[51] W. J. Scheirer and T. E. Boult. Bipartite biotokens: Definition, implementation, and analysis. In *in Proc. of IEEE/IAPR Int. Conf. on Biometrics*, pages 775–785, 2009.

[52] Ann Cavoukian and Alex Stoianov. Biometric Encryption: A Positive-Sum Technology that Achieves Strong Authentication, Security and Privacy, 2007. Discussion paper of the Office of the Information and Privacy Commissioner of Ontario, http://www.ipc.on.ca/images/Resources/bio-encryp.pdf.

[53] 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.

[54] ISO/IEC. Biometric Information Protection, 2011.

[55] P. Tuyls, B. Skoric, and T. Kevenaar, editors. *Security with Noisy Data: Private Biometrics, Secure Key Storage and Anti-Counterfeiting*. Springer, 2007. ISBN 978-1-84628-983-5.

[56] P. Moceri and T. Ruths. Cafe Cracks: Attacks on Unsecured Wireless Networks. http://www1.cse.wustl.edu/~jain/cse571-07/cafecrack.htm.

[57] Bank of America. Site Key Authentication, an Additional Layer of Online and Mobile Banking Security. https://www.bankofamerica.com/privacy/online-mobile-banking-privacy/sitekey.go.

[58] Stuart E. Schechter, Rachna Dhamija, Andy Ozment, and Ian Fischer. The Emperor's New Security Indicators. In *IEEE Symposium on Security and Privacy*, pages 51–65, 2007.

[59] S. Turner. Asymmetric Key Packages, August 2010. http://tools.ietf.org/html/rfc5958.

[60] T. Ylonen. The Secure Shell (SSH) Protocol Architecture, January 2006. http://tools.ietf.org/html/rfc4251.

[61] Google. Advanced sign-in security for your Google account. February 10, 2011. http://googleblog.blogspot.com/2011/02/advanced-sign-in-security-for-your.html.

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

[63] 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/.

[64] Adi Shamir. How to share a secret. *Commun. ACM*, 22(11):612–613, November 1979.

[65] F. Corella and K. P. Lewison. Achieving the Privacy Goals of NSTIC in the Short Term, May 2011. http://pomcor.com/whitepapers/NSTICWhitePaper.pdf.

[66] Anil John. Challenges in Operationalizing Privacy in Identity Federations - Part 2. October 8, 2012. http://info.idmanagement.gov/2012/10/challenges-in-operationalizing-privacy.html.

[67] Francisco Corella. Structured certificates and their applications to distributed systems. Presented at the RSA Conference, San Jose, California, January 16-20, 2000.

[68] Francisco Corella. Method and apparatus for providing field confidentiality in digital certificates, October 2004. US Patent 6,802,002.

[69] 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.

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

[71] 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.

[72] Symantec. Protecting Android Applications with Secure Code-Signing Certificates. https://www.symantec.com/content/en/us/enterprise/white_papers/b-code_signing_for_android_WP.pdf.

[73] Symantec. Symantec Code Signing for Android. http://www.symantec.com/verisign/code-signing/android.

[74] F. Corella and K. Lewison. A Proposed Architecture for the NSTIC Ecosystem, July 17, 2011. http://pomcor.com/whitepapers/ProposedNSTICArchitecture.pdf.

[75] Anil John. Challenges in Operationalizing Privacy in Identity Federations. September 30, 2012. http://info.idmanagement.gov/2012/09/challenges-in-operationalizing-privacy.html.

[76] Personal Data Ecosystem Consortium (PDEC). http://personaldataecosystem.org/.

[77] Francisco Corella. Pros and Cons of U-Prove for NSTIC, October 2011. http://pomcor.com/2011/10/04/pros-and-cons-of-u-prove-for-nstic/.

[78] Francisco Corella. Pros and Cons of Idemix for NSTIC, October 2011. http://pomcor.com/2011/10/10/pros-and-cons-of-idemix-for-nstic/.

[79] Francisco Corella. Credential Sharing: A Pitfall of Anonymous Credentials, December 2011. http://pomcor.com/2011/12/19/credential-sharing-a-pitfall-of-anonymous-credentials/.

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