User Authentication on Mobile Devices

Amit Sethi
Technical Manager

Omair Manzoor
Senior Security Consultant

Tarun Sethi
Software Security Consultant

Page 1 of 28
1 Introduction

Authenticating users on mobile devices can be challenging, and many solutions currently being used by mobile applications either compromise security or usability. Two common solutions that are often used are:

- Requiring the user to enter a password every time the application is started. Complex passwords are difficult to enter on mobile devices, and requiring frequent password entry typically results in users either saving the passwords on their devices in plain text files (so they can be copied and pasted), or in users choosing weak passwords that they can easily enter on their devices. We will discuss this approach in detail later.

- Requiring the user to enter a password once, and storing it on the device for subsequent authentication. Even if the password is stored encrypted/obfuscated, the key required to decrypt/decode the password needs to be stored in one of two locations:
  - Somewhere on the device: the device decrypts the password and sends it to the server – if the device is lost or stolen, the user's password is compromised.
  - Somewhere on the server: the device transmits the encrypted password to the server. The encrypted password is as good as the user's actual password for authenticating to the server. If the device is lost or stolen, the user's password may be protected, but the user's account is compromised.

This whitepaper compares various approaches for authenticating users on mobile devices and highlights their pros and cons in terms of security and usability. The authors do not advocate any particular solution here because the best solution depends on your application's requirements. For example, the security requirements for an online banking application that performs funds transfers are different from the security requirements for a media streaming application.

Additionally, the ultimate security of your application will depend on many implementation details. Always consult a security expert to ensure that your design and implementation are secure.

1.1 Risks

A mobile device, user authentication solution must address three main security risks.

1.1.1 Stolen Device

If a user's device is lost or stolen, the attacker can generally get access to everything stored on the device. This is generally true because currently, most mobile devices either do not
support disk encryption, or do not have disk encryption enabled by their users. If a user’s
device is stolen, the attacker can access plaintext data on the device, brute force
passwords used to encrypt data, etc. The attacker can perform both software attacks and
physical attacks against the device.

1.1.2 Borrowed Device
The likelihood of this risk is much greater for mobile devices than for other computers
(laptops, desktops, etc.). Users often allow others to borrow their mobile devices to make
a phone call, send a SMS, etc., and mobile devices currently only support a single user
account/password. Users cannot create a “guest” account on their devices that only
offers access to a limited subset of functionality.
The risk is different than that of a stolen device because in this case, the user provides the
device in an unlocked state to somebody else. Therefore, full disk encryption is an
ineffective control in this case. On the other hand, the types of attacks that can be
conducted with a borrowed device are limited as the user borrowing the device will have
access to it for a limited amount of time and may be under the device owner’s
supervision.

1.1.3 Infected Device
Mobile devices are at least as likely to get infected with malware as any other type of
computer. In fact, there are several reasons why mobile devices may be more likely to get
infected:

- Users download applications for many purposes on their mobile devices from
  potentially untrusted sources. Most users do not have separate devices for corporate
e-mail, online banking, media streaming, gaming, etc. Most mobile platforms allow
users to download applications from large “application stores” containing hundreds
of thousands of applications. Although some application stores have stricter controls
than others, there is no way for the application stores to guarantee that the
applications they are providing do not contain something malicious.

- Currently supported desktop and server operating systems generally receive frequent
  security updates; however, due to several reasons, mobile devices’ operating systems
do not receive frequent security updates. Many mobile devices receive updates every
few months, and many run operating systems that are several years old. Many mobile
devices run operating systems with known exploitable vulnerabilities.

- Anti-virus software for mobile devices is not as mature as anti-virus software for
desktop and server operating systems. This is due to a combination of several factors
including a lack of operating system support, limited battery life on mobile devices,
etc.
Note that application sandboxing generally limits the impact of malware on mobile devices. Mobile devices run applications in sandboxes where the resources that the applications can access are limited. Unless a device is jailbroken/rooted, these protections generally limit what malware can do on mobile devices.

Overall, it is important to remember that mobile devices are not immune from malware. Malware can steal data persisted to disk, capture a user’s inputs, etc. Of the three risks discussed in this section, this one is the most difficult to mitigate; in fact, there is no perfect solution, and all we can do is sufficiently raise the bar to deter malware authors. Note that this risk is not unique to the mobile platform in any way. Malware continues to be a significant risk for many end users on their computers.

1.2 Authentication Factors

There are three authentication factors that are typically used to authenticate users: something you know, something you have, and something you are. These factors are not unique to mobile devices; however, there are mobile device-specific issues to consider for all three factors.

1.2.1 Something You Know

This is currently the most commonly used authentication factor. A user presents a user id or a username and then provides a secret value, the password, that only the user knows. The authentication scheme relies on the strength of the secret value: how difficult the password is to guess.

Strong passwords are required to protect user accounts from various types of brute force attacks. Traditionally, many applications enforce strong password policies (e.g. at least 8 characters; mix of upper case letters, lower case letters, numbers and special characters; no words from the dictionary, etc.). Passwords that conform to such policies are difficult to enter on mobile devices. Requiring users to enter such passwords repeatedly creates usability problems.

Note that passphrases may be easier to enter on mobile devices and may offer slightly better security than passwords.

1.2.2 Something You Have

A user somehow proves possession of something external to the system. Common choices for proving possession are:

---

1 https://www.cl.cam.ac.uk/~jcb82/doc/BS12-USEC-passphrase_linguistics.pdf
• Hardware tokens that generate one-time passwords such as RSA SecurID, VASCO, SafeNet SafeWord, Authenex A-Key, SecureMetric SecureOTP, ActivIdentity OTP Token, etc.

• Hardware tokens that perform cryptographic operations and connect directly to the device authenticating the user, to transmit the result of a cryptographic operation to the server. Examples include smartcards and USB tokens.

• Access to an e-mail address is often used to authenticate users, especially for password reset operations. Note that this is a weaker form of this authentication factor as multiple users can have simultaneous access to an e-mail address, whereas generally, “something you have” refers to a single object that cannot be easily cloned.

• The mobile device itself can be “registered” with an application, and then, possession of the device can be used as a “something you have” authentication factor.

The above methods can be used with mobile applications as well, and some vendors have started producing specialized hardware for mobile devices; e.g. smartcard readers that can be connected to the headphone jacks on mobile devices.

Choices for “something you know” that require a user carry an additional device are less convenient for the user. One of the reasons for the popularity of mobile devices is convenience.

1.2.3 Something You Are

This authentication factor uses biometrics to authenticate users, and is starting to become popular with mobile devices. Some mobile devices come equipped with fingerprint readers, facial recognition software, etc. However, most devices only allow the biometric verification hardware/software to be used by the operating system for unlocking the devices. Currently, most mobile devices do not allow applications to utilize specialized hardware/software to authenticate users using biometrics.

The main problem with using biometrics on mobile devices is that most devices do not support a standard interface using which biometric information can be collected. Eventually, this problem may be resolved; once applications can rely on devices to gather biometric information, usability issues associated with authentication on mobile devices will be virtually eliminated. However, the typical concerns with biometrics such as tuning false accept / false reject rates will still need to be addressed.

Typical biometrics used to authenticate users (not just on mobile devices) include facial features, speech patterns, fingerprints, iris patterns, etc. Some of these are easier to bypass than others.
2 Authentication Options

Mobile applications that require users to authenticate either need to authenticate the user to a server, or need to authenticate the user locally before allowing the user to access data stored on the device. The authentication options discussed in this whitepaper will be split into the following categories:

- Authentication options for general applications that need to authenticate to a server; this whitepaper will discuss twelve such options.
- Additional protections available for internal corporate applications meant for use by employees; corporations can enforce an additional layer of protection using Mobile Device Management solutions.
- Authentication options for applications that need to authenticate users locally; there are only two possible options in this case.

Table 1 provides a summary of the discussion in this section; the symbols used in Table 1 are defined in Table 2.

<table>
<thead>
<tr>
<th>Authentication Factor</th>
<th>Risks Mitigated</th>
<th>Appropriate for Transitions</th>
</tr>
</thead>
<tbody>
<tr>
<td>Something You Know</td>
<td></td>
<td>UDUU → UDAU</td>
</tr>
<tr>
<td>Something You Have</td>
<td></td>
<td>UDUU → ADUU</td>
</tr>
<tr>
<td>Something You Are</td>
<td></td>
<td>UDUU → ADAU</td>
</tr>
<tr>
<td>Stolen Device</td>
<td></td>
<td>UDAU → ADAU</td>
</tr>
<tr>
<td>Borrowed Device</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Infected Device</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Authenticating to Server

2.1.2.1 Require User to Enter Strong Password

2.1.2.2 Require User to Enter Weak Password

2.1.2.3 Image Based Authentication

2.1.2.4 Retrieve Password Stored on Device

2.1.2.5 Retrieve another Secret from Device

2.1.2.6 Retrieve another Secret from Device (Revocation Capability)

2.1.2.7 SMS One-Time-Passwords

2.1.2.8 Device-Generated One-Time-Passwords

2.1.2.9 Out-of-Band Authentication (Phone Call)

2.1.2.10 Hardware Tokens
### 2.1 Authentically to Server

We will examine several options available to mobile applications for authenticating users to a server along with their pros and cons. Before we discuss the authentication options in detail, we need to discuss four authentication states that a mobile application may be in. Many applications have only two authentication states: authenticated user, and unauthenticated user. Given the usability issues resulting from authenticating users frequently, many mobile applications rely on a combination of device authentication and user authentication. The authentication options that we will discuss will allow an application to transition between these states. Different authentication methods are appropriate for different transitions.

<table>
<thead>
<tr>
<th>Column</th>
<th>Symbol</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>Authentication Factor</td>
<td>☐</td>
<td>The authentication option is an instance of this authentication factor</td>
</tr>
<tr>
<td>Authentication Factor</td>
<td>☐</td>
<td>The authentication option is not an instance of this authentication factor</td>
</tr>
<tr>
<td>Authentication Factor</td>
<td>☐</td>
<td>The authentication option can be an instance of this authentication factor depending on how it is implemented</td>
</tr>
<tr>
<td>Risks Mitigated</td>
<td>☐</td>
<td>The authentication option mitigates this risk</td>
</tr>
<tr>
<td>Risks Mitigated</td>
<td>☐</td>
<td>The authentication option does not mitigate this risk</td>
</tr>
<tr>
<td>Risks Mitigated</td>
<td>☐</td>
<td>The authentication option somewhat mitigates this risk; for example, it may allow an unauthorized user to access an application for a limited amount of time, or it may make it difficult (but not impossible) for an unauthorized user to access an application</td>
</tr>
<tr>
<td>Appropriate for Transitions</td>
<td>☐</td>
<td>The authentication option can be used for this transition</td>
</tr>
<tr>
<td>Appropriate for Transitions</td>
<td>☐</td>
<td>The authentication option should not be used for this transition</td>
</tr>
<tr>
<td>Appropriate for Transitions</td>
<td>☐</td>
<td>The authentication option can be used for this transition if implemented in certain ways</td>
</tr>
</tbody>
</table>

Table 2: Notations used
2.1 Authentication States
Mobile applications often rely on a combination of user authentication and device authentication. This leads to the four authentication states that an application may be in:

- Unauthenticated Device, Unauthenticated User
- Unauthenticated Device, Authenticated User
- Authenticated Device, Unauthenticated User
- Authenticated Device, Authenticated User

Some details about the authentication states are provided below. As an example, we will discuss a hypothetical mobile banking application, and the types of functionality that may be exposed by the application in each of the four states.

2.1.1.1 Unauthenticated Device, Unauthenticated User (UDUU)
In this case, the mobile application is in a state where it cannot prove device identity or user identity to the server. The device initiating the request cannot be identified by the server, and there is nothing in the request that demonstrates that a legitimate user is using the device.

Some applications may allow users to access a subset of their functionality in this state. For example, a mobile banking application may allow users to search for branches/ATMs while a device is in this state.

If the mobile device is lost or stolen while an application is in this state, unauthorized users will not be able to access any authenticated portions of the application as the device owner. However, if the device is infected with malware, it can potentially steal any information entered by the user the next time the user attempts to authenticate.

2.1.1.2 Unauthenticated Device, Authenticated User (UDAU)
In this case, the mobile application is in a state where it cannot prove device identity or user identity to the server, but it can prove user identity to the server. The requesting device cannot be identified by the server, but the user authenticates to the application and obtains a valid session.

Most applications will allow users to access all functionality in this state. For example, a mobile banking application may allow users to initiate transfers, pay bills, etc. in this state. However, performing high risk transactions (e.g. transfers over a certain amount) may require the device to be authenticated as well (e.g. via a SMS message sent to the device). Note that some mobile banking applications do require device authentication in addition to user authentication to perform operations.

If the mobile device is lost or stolen while an application is in this state, unauthorized users will be able to access the application as the device owner. Although device locking mechanisms may provide some protection in the case of stolen devices, most mobile
applications can only advise users about security best practices and cannot require their users to enable locking on their devices. If the device is infected with malware, it can steal the user’s session or perform arbitrary operations in the application from the user’s device.

2.1.1.3 Authenticated Device, Unauthenticated User (ADUU)

In this case, the mobile application is in a state where it can prove device identity to the server, but it cannot prove user identity to the server. The application can somehow prove that the authorized user had used the application on the device before, but cannot prove the user’s identity.

Some applications allow users to access a subset of their low-risk functionality in this state. For example, a mobile banking application may allow users to view account balances and a small number of recent transactions in this state.

If the mobile device is lost or stolen while an application is in this state, unauthorized users will be able to access some parts of the application as the device owner. Although device locking mechanisms may provide some protection in the case of stolen devices, most mobile applications cannot require their users to enable locking on their devices. If the device is infected with malware, it can steal any identifiers stored on the device to authenticate the device to the server. An attacker can then impersonate the device from elsewhere.

2.1.1.4 Authenticated Device, Authenticated User (ADAU)

In this case, the mobile application is in a state where it can prove device identity as well as user identity to the server. The application can somehow show the server that the authorized user is currently using the application, and that the authorized user has used the device to authenticate to the server before.

Most applications will allow users to access all functionality in this state. For example, a mobile banking application may allow users to initiate transfers, pay bills, etc. in this state.

If the mobile device is lost or stolen while an application is in this state, unauthorized users will be able to access the application as the device owner. Although device locking mechanisms may provide some protection in the case of stolen devices, most mobile applications cannot require their users to enable locking on their devices. This risk can be partially mitigated by limiting the amount of time that devices are in this state, and automatically reverting to the “Authenticated Device, Unauthenticated User” state after a short period of time. If the device is infected with malware, it can steal the user’s session or perform arbitrary operations in the application from the user’s device.

2.1.2 Authentication Options for General Apps

The list of authentication options in this section is not meant to be exhaustive; only twelve common options are discussed here.
2.1.2.1 Require User to Enter Strong Password

**Summary**

<table>
<thead>
<tr>
<th>Authentication Factor</th>
<th>Risks Mitigated</th>
<th>Appropriate for Transitions</th>
</tr>
</thead>
<tbody>
<tr>
<td>Something You Know</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Something You Have</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Something You Are</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Stolen Device</td>
<td>☐</td>
<td>☐</td>
</tr>
<tr>
<td>Borrowed Device</td>
<td>☐</td>
<td>☐</td>
</tr>
<tr>
<td>Infected Device</td>
<td>☐</td>
<td>☐</td>
</tr>
<tr>
<td>UDUU → UDUU</td>
<td>UDUU → ADUU</td>
<td>UDAU → ADAU</td>
</tr>
</tbody>
</table>

- Pros:
  - Secure if device is borrowed or stolen

- Cons:
  - Usability
  - Users may store password on device to improve usability

**Discussion**

Requiring users to authenticate using a strong password is one of the most common approaches that applications (not just mobile applications) use today. Other than the usual problems with this approach (difficulty of remembering complex passwords, users reusing passwords for multiple applications, etc.), this approach presents a unique challenge for mobile devices: it is difficult to enter complex passwords on mobile devices. If users are required to enter complex passwords, they may use approaches such as the following:

- Store the password in a text file on the mobile device that can be copied and pasted into the application. This effectively makes this authentication mechanism similar to the one discussed in Section 2.1.2.4 with the drawback that the application has no control over how the password is stored.

- Store the password in a password manager application on the mobile device. This essentially shifts part of the authentication problem to the password manager application. The application again loses control over how the password is stored.

Users may choose other approaches to make the authentication mechanism more usable; the main point is that using this authentication approach frequently will make many users store the password on the device in a location that your application has no control over. Also, the typical mechanism for entering the password will become a copy and paste from another application. Since the clipboard is accessible by all mobile applications, a malicious application can easily steal the password, which will likely stay in the clipboard for a long period of time. Depending on the mobile platform, the application may need to be in the foreground to access the clipboard. This results in poor security and usability.
User Authentication on Mobile Devices

Authentication Options

The approach discussed in Section 2.1.2.3 may partially solve this problem (as that approach will make it impossible to copy and paste passwords from another application). However, implementing complex passwords using that approach may be difficult.

2.1.2.2 Require User to Enter Weak Password

Summary

<table>
<thead>
<tr>
<th>Authentication Factor</th>
<th>Risks Mitigated</th>
<th>Appropriate for Transitions</th>
</tr>
</thead>
<tbody>
<tr>
<td>Something You Know</td>
<td></td>
<td>UDUU → UDAU</td>
</tr>
<tr>
<td>Something You Have</td>
<td></td>
<td>UDUU → ADUU</td>
</tr>
<tr>
<td>Something You Are</td>
<td></td>
<td>UDUU → ADAU</td>
</tr>
<tr>
<td>Stolen Device</td>
<td></td>
<td>ADUU → ADAU</td>
</tr>
<tr>
<td>Borrowed Device</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Infected Device</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- Pros:
  - Usability – It is easier to enter a weak password on mobile devices
  - Can protect against lost/stolen devices in conjunction with other measures (e.g. account lockouts)

- Cons:
  - Password easier to guess/brute force

Discussion

Although having a password policy that permits weak passwords is generally considered poor practice, it can be used in specific ways where it may pose an acceptable level of risk. For example, if a mobile application initially authenticates users using a strong password (see Section 2.1.2.1) and then stores a secret on the device (see Sections 2.1.2.5 and 2.1.2.6) for subsequent device authentication, a weak password can be used to authenticate a user as long as the server can authenticate the device. If this approach is used, several security measures need to be in place:

- The secrets stored on devices must be infeasible to brute force.
- If the server receives a number of consecutive invalid login attempts with a valid device authentication secret but an invalid password (generally 3-5), it must either lock the user’s account completely, or invalidate the device authentication secret so that the only way to authenticate the user will be by using the user’s original strong password.
- The user must be required to change his/her weak password frequently to prevent slow brute force attacks where an attacker attempts 1-2 logins per week (for example) and relies on the legitimate user to log in once per week to reset the failed login counter.
- The weak password must be verified by the server (not by the device), and must not be stored anywhere on the device in any form (plaintext, encrypted, hashed, etc.).
Note that the weak password does not need to be the traditional password/PIN consisting of printable characters; it can be an image-based password (see Section 2.1.2.3), a pattern-based password (e.g. one of the options currently available on Android devices), etc.

2.1.2.3 Image Based Authentication

Summary

<table>
<thead>
<tr>
<th>Authentication Factor</th>
<th>Risks Mitigated</th>
<th>Appropriate for Transitions</th>
</tr>
</thead>
<tbody>
<tr>
<td>Something You Know</td>
<td></td>
<td>UDUU → UDAU</td>
</tr>
<tr>
<td>Something You Have</td>
<td></td>
<td>UDUU → ADUU</td>
</tr>
<tr>
<td>Something You Are</td>
<td></td>
<td>UDAU → ADAU</td>
</tr>
</tbody>
</table>

- Pros:
  - Usability – The user does not have to type and remember a password. He/she just has to remember 3 or 4 categories.
  - Fingerprint / trails on the device screen do not reveal the password as the image order is randomized for each authentication attempt.

- Cons:
  - Choosing 3 categories in a 3x3 grid only permits 84 possible “passwords”, and choosing 4 categories in a 4x4 grid only permits 1820 possible “passwords”. This may allow brute force attacks, and on its own, may be insufficient for critical transactions.

Discussion

This patented authentication scheme offered by Confident Technologies requires the user to remember visual categories chosen by the user during an initial setup process instead of passwords. Based on the categories chosen by the user, a 3x3 or 4x4 grid of images is presented to the user when authentication is required, and the user needs to select 3 or 4 images that are related to the previously chosen categories.

- During initial setup, a user chooses 3 or 4 categories like cars, fish, furniture, etc.
- When authentication is required, the device displays a grid of images that are randomly selected from a large pool of images. The user is asked to select the images related to his/her chosen categories.
- The location of images in the grid and the images (for each category) are different for each login attempt. With each failed login attempt, a new set of images is displayed.

---

2 http://www.confidenttechnologies.com/products/confident-mobile-authentication
and a limit of failed authentication attempts can be set after which the user has to go through the initial setup again.

This approach is conceptually similar to the approach discussed in Section 2.1.2.2; however, it is easier to use for end users. It also offers the advantage that any fingerprints left on the device screen by the authorized user cannot be used to determine the user’s password – as the image locations are randomized for each authentication attempt.

### 2.1.2.4 Retrieve Password Stored on Device

#### Summary

<table>
<thead>
<tr>
<th>Authentication Factor</th>
<th>Risks Mitigated</th>
<th>Appropriate for Transitions</th>
</tr>
</thead>
<tbody>
<tr>
<td>Something You Know</td>
<td>Stolen Device</td>
<td>UDUU → UDAU</td>
</tr>
<tr>
<td>Something You Have</td>
<td>Borrowed Device</td>
<td>UDUU → ADAU</td>
</tr>
<tr>
<td>Something You Are</td>
<td>Infected Device</td>
<td>UDUU → ADAU</td>
</tr>
<tr>
<td>●</td>
<td>○</td>
<td>○</td>
</tr>
</tbody>
</table>

- Pros:
  - Easy to implement
  - Usability
- Cons:
  - Does not mitigate any risks discussed in this paper

#### Discussion

This is an approach that applications often use; they will ask the user to enter his/her password the first time the application is started, and will store it on the device for subsequent authentication (either in the clear, or encrypted using a key stored on the device). This approach is easy to implement and typically does not require any server-side changes. It is also easy to use for end users who just have to authenticate to the application the first time they use it from the mobile device.

However, this approach does not mitigate any of the risks discussed in this document. If the device is lost/stolen, is borrowed, or is infected with malware, another user can get access to the stored password. If a user realizes that his/her password has been stolen, he/she can change it through a web application unless the attacker changes it first. If the user does not realize that his/her password has been stolen, the attacker can have access to the user’s account for an extended period of time. Also, many users choose the same password for multiple services. If an application stores a user’s password on a device, it can allow an attacker to not only compromise the user’s account in that application, but in other applications as well.

### 2.1.2.5 Retrieve another Secret from Device

#### Summary
User Authentication on Mobile Devices

Authentication Options

<table>
<thead>
<tr>
<th>Authentication Factor</th>
<th>Risks Mitigated</th>
<th>Appropriate for Transitions</th>
</tr>
</thead>
<tbody>
<tr>
<td>Something You Know</td>
<td>Stolen Device</td>
<td>UDUU → UDAU</td>
</tr>
<tr>
<td>Something You Have</td>
<td>Borrowed Device</td>
<td>UDUU → ADUU</td>
</tr>
<tr>
<td>Something You Are</td>
<td>Infected Device</td>
<td>UDAU → ADAU</td>
</tr>
</tbody>
</table>

- **Pros:**
  - Usability

- **Cons:**
  - The secret stored on the device is as good as the password

**Discussion**

This option is a variation of the approach discussed in Section 2.1.2.4; however, instead of storing the password on the device, another secret is stored on the device by the application. The secret can be a random ID, a cryptographic key, the user’s password encrypted using a key stored only by the server, etc. Some of these approaches are safer than others; however, in all cases, the idea is that if the user’s device is compromised, the attacker cannot obtain the user’s password.

As with the approach discussed in Section 2.1.2.4, this approach is easy for end users; they only have to authenticate the first time they use the application.

Note that if the secret stored on the device is compromised, that secret can be used by an attacker as if it were the user’s password. Therefore, this device authentication approach should not be used for highly sensitive transactions; those should require the user to authenticate explicitly.

This approach does pose a problem in that there needs to be a revocation mechanism for the secret stored on the device. Otherwise, if the secret stored on the device is stolen, the user cannot prevent the attacker from using it for an extended period of time. Combining this approach with a revocation mechanism is discussed in Section 2.1.2.6.

**2.1.2.6 Retrieve another Secret from Device (Revocation Capability)**

**Summary**

<table>
<thead>
<tr>
<th>Authentication Factor</th>
<th>Risks Mitigated</th>
<th>Appropriate for Transitions</th>
</tr>
</thead>
<tbody>
<tr>
<td>Something You Know</td>
<td>Stolen Device</td>
<td>UDUU → UDAU</td>
</tr>
<tr>
<td>Something You Have</td>
<td>Borrowed Device</td>
<td>UDUU → ADUU</td>
</tr>
<tr>
<td>Something You Are</td>
<td>Infected Device</td>
<td>UDAU → ADAU</td>
</tr>
</tbody>
</table>

- **Pros:**
  - Usability
  - Devices can be de-provisioned on the server
Cons:
- Greater implementation complexity

Discussion
This option is a variation of the approach discussed in Section 2.1.2.5; it adds the ability to revoke the secret stored on a device such that if the user realizes that his/her device secret has been stolen (e.g. in the case of a lost/stolen device), he/she can sign into a web application using his/her password (or call a customer service line) to revoke the secret stored on the device. Depending on how the web application is implemented, this can give users good visibility into and control over the device(s) that they have registered with the application. For example, each device registered by the user can be displayed in the web application along with information about the last few times each one connected to the server.

Note that revoking a secret does not require the server to communicate with the device. The server will simply have to invalidate the device secret so that any attempts to connect to it using that secret will fail.

There are several ways in which this could be implemented. For example:

- If the secret stored on the device is the user’s password encrypted with a key that only the server has access to, the revocation process could simply involve having the user change his/her password.

- If the secret stored on the device is a cryptographic key or a random ID, the server will just have to mark that value as deleted/disabled; the user will not need to change his/her password.

As with the approach in Section 2.1.2.5, if the secret stored on the device is compromised, that secret can be used by an attacker as if it were the user’s password until the user revokes the secret. Therefore, this approach should not be used for highly sensitive transactions; those should require the user to authenticate explicitly.

### 2.1.2.7 SMS One-Time-Passwords

**Summary**

<table>
<thead>
<tr>
<th>Authentication Factor</th>
<th>Risks Mitigated</th>
<th>Appropriate for Transitions</th>
</tr>
</thead>
<tbody>
<tr>
<td>Something You Know</td>
<td>Stolen Device</td>
<td>UDUU → ADAU</td>
</tr>
<tr>
<td>Something You Have</td>
<td>Borrowed Device</td>
<td>UDUU → ADAU</td>
</tr>
<tr>
<td>Something You Are</td>
<td>Infected Device</td>
<td>UDAU → ADAU</td>
</tr>
</tbody>
</table>

- Pros:
  - “Something You Have” authentication factor without having to carry separate hardware token

- Cons:
User Authentication on Mobile Devices

Authentication Options

- Usability
- OTPs are transferred over insecure channel (SMS)
- Cloned devices may receive SMS OTP
- No guarantees around delivery time
- Only suitable for step-up authentication for high-risk functionality

Discussion

This authentication scheme leverages short messaging service (SMS) to deliver a one-time-password (OTP) to a configured device. An application may require an OTP for performing highly sensitive operations such as performing funds transfers. The OTP is valid for only one use, and can be combined with other authentication approaches; for example, a user may be required to enter the OTP to perform a sensitive operation. This is similar to authentication schemes where the user enters an OTP from a soft or hard token for authentication (see Sections 2.1.2.8 and 2.1.2.10).

One of the benefits of SMS OTP over hardware tokens is that the user does not have to carry a separate token. Four to six digit OTPs should generally be used as they are easy for users to memorize and enter for authentication. Depending on the platform that the mobile application runs on, the mobile application may be able to get the permission to read incoming SMS messages and could automate this so that user intervention is not required to authenticate the device. When performing a sensitive operation, the application would wait until the server sends a SMS to the device, read the SMS message, and send the OTP back to the server without requiring user intervention.

Some issues with this approach are:

- SMS is not a secure communication channel. The SMS OTP can potentially be read by unauthorized users in transit. Given that this mechanism is generally suitable only for step-up authentication (i.e. where the user is already authenticated, but the server wants to perform an additional level of authentication), this is an acceptable risk.

- Although SMS messages are generally delivered within a few seconds, there is no guarantee that they will get delivered in a timely manner. SMS messages may sometimes get delayed for several minutes, several hours, or longer. This could cause usability issues unless used sparingly for high-risk transactions.

- Reading the SMS message may require the user to switch to the SMS application on his/her mobile device. A best practice for sensitive applications is to ensure that the user’s session is terminated as soon as the application loses focus. This obviously cannot be done if a SMS OTP approach is used where the user needs to manually read the SMS and enter it on his/her device.
User Authentication on Mobile Devices

Authentication Options

SMS OTP can be used for each high-risk transaction, or the application can use it to establish a short-term session in which the user can perform multiple high-risk transactions.

SMS OTP partially protects the application in case of a stolen device because the device will stop receiving SMS messages as soon as the user reports his/her device stolen to his/her carrier.

When SMS OTP is used, the server should terminate the existing session after a number of consecutive failed OTP step-up authentication attempts.

### 2.1.2.8 Device-Generated One-Time-Passwords

**Summary**

<table>
<thead>
<tr>
<th>Authentication Factor</th>
<th>Risks Mitigated</th>
<th>Appropriate for Transitions</th>
</tr>
</thead>
<tbody>
<tr>
<td>something you know</td>
<td>stolen device</td>
<td>UDUU → UDAU</td>
</tr>
<tr>
<td>something you have</td>
<td>borrowed device</td>
<td>UDUU → ADAU</td>
</tr>
<tr>
<td>something you are</td>
<td>infected device</td>
<td>UDAU → ADUU</td>
</tr>
</tbody>
</table>

- **Pros:**
  - “Something You Have” authentication factor without having to carry separate hardware token
  - Can be used for step-up authentication

- **Cons:**
  - Usability
  - Software token can potentially be copied to another device

**Discussion**

These are One-Time-Passwords (OTPs) generated by software running on a mobile device. A software token application on the device is responsible for generating OTPs that the user can use to authenticate to an application. Generally, the device’s time needs to be synchronized with the server’s time for this approach to work. Some software token vendors include RSA, Quest Software and ActivIdentity.

One particular example is Google Authenticator³, which is a mobile application that generates 6-digit verification codes that can be used as a second authentication factor for logging into a Google account (after a user opts in for 2-factor authentication).

Device-generated OTPs are better than SMS OTPs in some ways. There is no opportunity for an attacker to sniff the OTP in transit (unless the OTP is transmitted to

---

³ [http://support.google.com/a/bin/answer.py?hl=en&answer=1037451](http://support.google.com/a/bin/answer.py?hl=en&answer=1037451)
the server over an insecure link). There are also no issues with OTP delivery time as the
OTPs are generated on the devices themselves.
As with SMS OTPs, device generated OTPs should not be used alone for authentication;
rather, they should be used as an additional factor in an authentication scheme. They can
be used for initial authentication in the application or for step-up authentication.
Device-generated OTPs partially protect against borrowed devices; if a user lends his/her
device to somebody else, the user borrowing the device can potentially access the
application protected with a device-generated OTP while he/she is in possession of the
device. However, once he/she returns the device, he/she cannot access the user’s
application.
Note that this approach is less secure than hardware token based OTPs (see section
2.1.2.10) as these tokens can potentially be cloned.

2.1.2.9 Out-of-Band Authentication (Phone Call)

Summary

<table>
<thead>
<tr>
<th>Authentication Factor</th>
<th>Risks Mitigated</th>
<th>Appropriate for Transitions</th>
</tr>
</thead>
<tbody>
<tr>
<td>Something You Know</td>
<td>Something You Have</td>
<td>Something You Are</td>
</tr>
<tr>
<td></td>
<td>Stolen Device</td>
<td>Borrowed Device</td>
</tr>
<tr>
<td></td>
<td>Infected Device</td>
<td>UDUU → UDAU</td>
</tr>
<tr>
<td></td>
<td></td>
<td>UDUU → ADUU</td>
</tr>
<tr>
<td></td>
<td></td>
<td>UDAU → ADAU</td>
</tr>
<tr>
<td></td>
<td></td>
<td>ADAU → ADAU</td>
</tr>
</tbody>
</table>

• Pros:
  - Server can verify details of sensitive transactions in addition to performing
    step-up authentication

• Cons:
  - Usability
  - Voice channels are generally insecure
  - Cloned devices may receive phone call
  - Only suitable for step-up authentication for high-risk functionality

Discussion

Out-of-Band authentication uses a separate channel from the one that is being used for
general communication to authenticate a device. SMS OTP discussed in Section 2.1.2.7 is
a type of out-of-band authentication. Out-of-band authentication can also be performed
using a phone call. When a user attempts an operation that requires out-of-band
authentication, the server automatically calls the user’s registered mobile phone. The user
answers the phone and authenticates using some factors (PIN/password, OTP, and/or
voice biometrics). The server then automatically processes the transaction, or provides
the user with an OTP that he/she enters into the mobile application to submit the transaction.

This option has similar pros and cons as the SMS OTP approach. However, this approach is more flexible; it allows the server to verify details of the sensitive transaction with the user in addition to performing step-up authentication. For example, the server can verify the destination account number in a funds transfer request to ensure that malware on the device did not modify it.

Out-of-band authentication is not suitable for frequent authentication and may cause usability fatigue; however, it is a reasonable option for step-up authentication for high-risk transactions.

Additionally, if a mobile device is on a 3G CDMA or 2.5G GSM network (or below), it cannot use voice and data functionality simultaneously. In such situations, the user will lose data connectivity during the out-of-band authentication, which may cause usability issues with the application. Also, as with SMS OTP, if an application automatically ends a user’s session when it loses focus, it cannot use this approach.

### 2.1.2.10 Hardware Tokens

#### Summary

<table>
<thead>
<tr>
<th>Authentication Factor</th>
<th>Risks Mitigated</th>
<th>Appropriate for Transitions</th>
</tr>
</thead>
<tbody>
<tr>
<td>Something You Know</td>
<td>Something You Have</td>
<td>Something You Are</td>
</tr>
<tr>
<td>○</td>
<td>●</td>
<td>○</td>
</tr>
</tbody>
</table>

- **Pros:**
  - “Something you have” authentication factor that cannot be cloned easily.
- **Cons:**
  - Inconvenient to carry another hardware device.

#### Discussion

At least two types of hardware tokens can also be used for authentication:

- Hardware tokens that generate one-time passwords such as RSA SecurID, VASCO, SafeNet SafeWord, Authenex A-Key, SecureMetric SecureOTP, ActivIdentity OTP Token, etc.
- Hardware tokens that perform cryptographic operations and connect directly to the device authenticating the user, to transmit the result of a cryptographic operation to the server. Examples include smartcards, USB tokens and tokens that attach to the headphone jack on mobile phones.
Hardware tokens are often used for applications that require a higher level of security for critical transactions more than what passwords/passphrases/PINs/images can provide. Regardless of the type of hardware token used, these schemes rely on the following:

- Cryptographic operations are performed in a secure environment that is very unlikely to get infected with malware.
- The hardware tokens include physical and software-based protection mechanisms that make them very difficult to clone.

Hardware token-generated OTPs are generally not used alone for authentication; rather, they are generally used as an additional factor in an authentication scheme. They can be used for initial authentication in the application or for step-up authentication.

### 2.1.2.11 Biometrics

<table>
<thead>
<tr>
<th>Authentication Factor</th>
<th>Risks Mitigated</th>
<th>Appropriate for Transitions</th>
</tr>
</thead>
<tbody>
<tr>
<td>Something You Know</td>
<td>Stolen Device</td>
<td>UDUU → UDAU</td>
</tr>
<tr>
<td>Something You Have</td>
<td>Borrowed Device</td>
<td>UDUU → ADUU</td>
</tr>
<tr>
<td>Something You Are</td>
<td>Infected Device</td>
<td>UDAU → ADAU</td>
</tr>
</tbody>
</table>

- Pros:
  - Usability

- Cons:
  - Very few devices currently support this

### Discussion

Biometric authentication schemes attempt to use users’ physical characteristics or traits to identify and authenticate them. The characteristics used by biometric authentication schemes are generally physiological and not behavioral. The most common biometric currently used to authenticate users is a fingerprint. Other biometrics include facial features, speech patterns, iris patterns, etc. For example, to capture speech patterns, an application may prompt a user to read a random phrase or sentence for authentication.

The main problem with using biometrics for authentication on mobile devices is that very few devices currently support gathering users’ biometric information. A few devices contain built-in fingerprint readers, and some vendors provide add-on fingerprint readers for some mobile devices.

Unlike other authentication mechanisms based on “something you know” and “something you are” that can be verified using exact matches, biometrics require complex algorithms for matching that can generally be tuned for false accept / false reject rates.
Since biometric verification is not exact, biometrics are generally not used as a sole authentication mechanism; they are generally used in conjunction with a password/PIN. Biometric based authentication holds the greatest promise for mobile devices in the long run. It can be used in conjunction with a weak password (see Section 2.1.2.2) to strongly authenticate users without causing usability problems.

Some other issues with biometrics on mobile devices and biometrics in general are:

- If a user’s biometric (e.g. fingerprint or the corresponding template) is stolen, it may not be possible to change it. With fingerprints, a user could enroll a different finger, but that is not possible with all biometrics.

- With fingerprints in particular, the likelihood of a mobile device user’s fingerprints getting compromised is currently high. Mobile devices generally have large glossy touchscreens that users often leave their fingerprints on. An attacker may be able to use a fingerprint obtained from a touchscreen to create a fake finger and use it to authenticate.

- There are no standard APIs provided by mobile platforms that applications can use to gather biometric information.

### 2.1.2.12 Rely on another Application for Authentication

#### Summary

<table>
<thead>
<tr>
<th>Authentication Factor</th>
<th>Risks Mitigated</th>
<th>Appropriate for Transitions</th>
</tr>
</thead>
<tbody>
<tr>
<td>Something You Know</td>
<td>Something You Have</td>
<td>Something You Are</td>
</tr>
<tr>
<td>Stolen Device</td>
<td>Borrowed Device</td>
<td>Infected Device</td>
</tr>
<tr>
<td>UDUU →</td>
<td>UDUU →</td>
<td>UDAU →</td>
</tr>
<tr>
<td>ADUU →</td>
<td>ADUU →</td>
<td>ADAU →</td>
</tr>
</tbody>
</table>

- **Pros:**
  - Authentication for several applications can be delegated to single trusted application.

- **Cons:**
  - Application loses control over exactly how users are authenticated.

#### Discussion

This is not an authentication solution in itself; this mechanism delegates user authentication to another application. Several options are available:

- A standard mechanism such as OpenID can be used where an application trusts an identity provider to authenticate a user. Either the identity provider would have an application on the mobile device that other applications would communicate with using various IPC mechanisms, or the application attempting to authenticate the user would need to directly interface with the identity provider’s servers.
Some vendors have recently created solutions that wrap applications and allow them to be used only after the user authenticates to a management application on the mobile device. This approach is especially suitable for corporate applications where a single authentication application can be created, and other corporate applications can use it to authenticate users.

The authentication factors, risks mitigated and appropriate authentication transitions depend on the details of the implemented solution.

### 2.1.3 Additional Protections for Internal Corporate Apps

Given the current trend of Bring Your Own Device (BYOD), many users use their personal devices to access corporate resources including internal corporate applications. IT departments no longer retain significant control over the devices. However, corporations do have some tools at their disposal that they can use to try to secure users’ devices and expand the number of authentication options available to non-corporate applications.

In an enterprise wide infrastructure, securely authenticating users on mobile devices is essential. Some organizations require users to authenticate to their internal mobile applications using their corporate LDAP credentials, which if compromised, would allow an attacker to access a variety of resources at the corporation.

Given the greater potential for damage, device-level authentication is often enforced in addition to application-level authentication. Corporations frequently use Mobile Device Management (MDM) technologies to try to enforce security policies on devices. These technologies are not perfect, and they generally require the end user’s cooperation; if an authorized device user wants to bypass controls enforced by the MDM technology, they generally can do so because the controls are enforced on the device itself. Some security controls that are typically enforced by corporations using MDM technologies are:

- Enable device encryption
- Require strong password to unlock device
- Disable side-loading of applications
- Detect jailbroken/rooted devices

MDM technologies enroll devices for tracking purposes, and allow them to be revoked and remotely wiped. Note that remote wipe is only effective once a device connects to a network. If an attacker places a stolen device in airplane mode, he/she can prevent the device from receiving remote wipe commands. The attacker can then retrieve any secret data stored on the device.

Although MDM technologies are not perfect, corporate applications dealing with sensitive data should use MDM technologies enforce security controls on mobile devices.
As we discussed earlier in this paper, non-corporate applications generally cannot enforce controls like device encryption and device-level locking; however, corporations can enforce these controls for internal application users.

Device-level authentication is not meant to replace application-level authentication. It is part of a defense in depth strategy that leverages controls at different layers. All of the application-layer authentication options discussed in Section 2.1.2 are still applicable. If one control fails, the other control should still provide a layer of protection.

2.2 Local Authentication on Device

Many applications store sensitive data on mobile devices such that the data can be accessed even while the device is offline (e.g. in airplane mode). Although this practice should be discouraged due to security reasons, there is often a strong business requirement to allow this. Applications that need to store sensitive data on mobile devices need some way to authenticate users before allowing them to access the data. There are very few effective options available for protecting this data; these options are discussed below. Note that storing data offline in plaintext form and simply authenticating the user before displaying it is insufficient because users can directly access the plaintext data without going through the application. Therefore, the data needs to be stored in an encrypted format. The encryption can be performed using several techniques:

- If MDM technologies can be used and the mobile device supports it, the data can be encrypted by device itself. This only protects the data in case of a stolen device. It does not protect the data in case of a borrowed device or an infected device. Once the device is unlocked, all data is potentially accessible by all applications on the device.

- The application could roll its own encryption scheme. This protects the data in case of a stolen device or a borrowed device. It also offers some protection in case of an infected device.

- Some MDM vendors are releasing application wrapping technologies that wrap system calls that perform operations like file I/O and automatically encrypt/decrypt data for the application without the application needing to worry about it. Depending on the implementation details, this may or may not protect the user in any of the scenarios discussed in Section 1.1.

2.2.1 Encryption with Strong Password

<table>
<thead>
<tr>
<th>Authentication Factor</th>
<th>Risks Mitigated</th>
<th>Appropriate for Transitions</th>
</tr>
</thead>
<tbody>
<tr>
<td>Something You Know</td>
<td>Stolen Device</td>
<td>UDUU → UDAU</td>
</tr>
<tr>
<td>Something You Have</td>
<td>Borrowed Device</td>
<td>UDUU → ADU</td>
</tr>
<tr>
<td>Something You Are</td>
<td>Infected Device</td>
<td>ADAU → ADAU</td>
</tr>
</tbody>
</table>
Discussion

One way to protect data stored on a mobile device is to encrypt it with a cryptographic key and then store that key encrypted with the user’s password. The user needs to have a strong password to prevent offline brute force attacks. One way for an application to roll its own encryption scheme is described below.

The data can be encrypted using a symmetric key algorithm like AES-128 (if data integrity is required, an authenticated encryption mode like GCM can be used; otherwise, more commonly supported encryption modes like CBC can be used). The secret data encryption key (DEK) can be stored on the device encrypted using a key encryption key (KEK) derived from the user’s password using Password Based Encryption (e.g. PBKDF2).

When the application is first installed, the following steps need to be performed:

• Generate a random DEK (e.g. using the KeyGenerator API on Android devices).
• Ask the user for an offline password and generate KEK using PBKDF2. PBKDF2 requires a password, a salt (recommended length is 64 bits for PBKDF2), a cryptographic hash algorithm and an iteration count. The salt is a string that should be different for each device. The iteration count (recommended value is 1000) is the number of times the hash function will be iterated; the purpose is to slow down brute force attacks.
• Generate a challenge text (C) using unique values from the device such that it can be regenerated later. Encrypt it using DEK. DEK[C] denotes the encrypted challenge text; store this value on the device. The purpose of the challenge text is to allow the application to detect when the user has entered the correct password before attempting to decrypt the actual data that the user is trying to access.
• Now encrypt DEK using KEK and store KEK[DEK] on the device.
• Encrypt any data that needs to be accessed offline using DEK.

Subsequently, to decrypt the data for offline access, perform the following steps:

• Generate KEK again using the password entered by the user.
• Decrypt KEK[DEK] using KEK to obtain DEK.
• Decrypt DEK[C] using DEK to obtain C.
• Regenerate the challenge text and compare it with C from the last step. If they match, the entered password is correct.

4 http://www.ietf.org/rfc/rfc2898.txt
- Use DEK to decrypt data and display it to the user.

### 2.2.2 Encryption with TPM Support

#### Summary

<table>
<thead>
<tr>
<th>Authentication Factor</th>
<th>Risks Mitigated</th>
<th>Appropriate for Transitions</th>
</tr>
</thead>
<tbody>
<tr>
<td>Something You Know</td>
<td>Stolen Device</td>
<td>UDUU → ADAU</td>
</tr>
<tr>
<td>Something You Have</td>
<td>Borrowed Device</td>
<td>UDUU → ADAU</td>
</tr>
<tr>
<td>Something You Are</td>
<td>Infected Device</td>
<td>ADUU → ADAU</td>
</tr>
</tbody>
</table>

#### Discussion

Another way to protect data stored on a mobile device is to encrypt it with a cryptographic key stored inside secure hardware like a Trusted Platform Module (TPM). This requires capabilities that most mobile devices do not currently possess. When that changes in the future, the following approaches may be possible.

The TPM can authenticate the user using a weak password and/or biometrics. It can then do one of the following:

- Provide the key to the requesting application so that it can use it to decrypt the user's data. This approach helps protect the data in case of a borrowed or stolen device; however, the key can potentially be stolen by malware on the device.

- Allow the requesting application to encrypt/decrypt data; the actual cryptographic operations are performed inside the TPM. This approach also helps protect data in case of a borrowed or stolen device. It also helps protect the key from malware on the device. However, note that the malware may be able to get the TPM to decrypt the data for it.

The TPM approach is effective only if the following controls are present:

- The TPM must wipe all keys that it stores after a number of failed authentication attempts.

- The TPM must contain physical controls to wipe keys if an attacker attempts physical attacks against the TPM.
3 Designing a Solution

Combining the authentication methods properly for an application is a non-trivial exercise. Always consult a security expert when designing and implementing your mobile application. To illustrate the point, we will analyze three ways of combining some of the authentication mechanisms discussed in Section 2.1.2.

3.1 Example

Let’s consider a hypothetical mobile banking application. It stores a token on the device that it uses to authenticate the device. Additionally, if the user wants to perform a transaction, the user has to enter a PIN from a hardware token. As we will discuss below, different choices have different security properties and there are tradeoffs to be made.

3.1.1 Initial Authentication with Device and Biometrics; Step-Up Authentication with OTP

The application starts with an authenticated device using the technique discussed in Section 2.1.2.6. It then uses the user’s voice as an additional authentication factor when the application is started (see Section 2.1.2.11). When the user chooses to transfer funds to another user’s account, he/she enters an OTP generated by a hardware token as discussed in Section 2.1.2.10. The application ensures that both the device and the user are authenticated before permitting the potentially high-risk transaction.

One potential problem in this scenario is that if the device is infected, the malware can modify the beneficiary account number and transaction amount to transfer funds into the attacker’s account.

3.1.2 Initial Authentication with Device and Biometrics; Step-Up Authentication and Transaction Verification with OTP

The application starts with an authenticated device using the technique discussed in Section 2.1.2.6. It then uses the user’s voice as an additional authentication factor when the application is started (see Section 2.1.2.11). When the user chooses to transfer funds to another user’s account, he/she enters an OTP generated by a hardware token as discussed in Section 2.1.2.10. Part of the OTP generation process requires the user to enter the beneficiary account number and transaction amount into the hardware token. The generated PIN depends on these values.

In this case, malware cannot modify the beneficiary account number or transaction amount.
3.1.3 Initial Authentication with Device; Step-Up Authentication with OTP; Transaction Verification with Biometrics

Both of the above options suffer from one potential problem. If a user loses his/her device and hardware token while he/she is authenticated to the application, an unauthorized user may be able to perform transfers on the user's behalf. If we reorder the authentication steps, we can help mitigate this issue.

Let's start with an authenticated device using the technique discussed in Section 2.1.2.6. When the user chooses to transfer funds to another user's account, he/she enters an OTP generated by a hardware token as discussed in Section 2.1.2.10.

It is common for banks to have all transactions go through a fraud detection system. If there is a high likelihood of a transaction being fraudulent (as determined by the fraud detection system), the bank can have an automated system call the user and use the user's voice as an additional authentication factor (as discussed in Section 2.1.2.11) before processing the transaction.

This option uses voice authentication in a different way to solve the problem of what happens if a user loses his/her hardware token and device while authenticated to the application. However, it may make it easier for an attacker to perform some transactions that are not caught by the bank's fraud detection system if the hardware token and device are stolen while the user is not authenticated to the application.

4 Conclusion

As is clear from the above case studies, combining authentication approaches is not a trivial task. To keep it simple, the above discussion skipped over many implementation details that will have a real impact on security. For example, the token stored on the device by the server could be used in several ways:

- It can be transmitted with the authentication request. In this case, the token can be obtained by network-based attackers who can decrypt SSL traffic between the device and the server.

- It can be used as part of a challenge-response protocol. In this case, once the token is provisioned on the device, a network-based attacker cannot access it. The attacker will only see the challenge, and the response generated using the challenge and the secret token.

- It can be used to encrypt a counter value and send it to the server each time it needs to authenticate to the server. This will require the server to maintain a counter value for each device that connects to it. In this case, once the token is provisioned on the device, a network-based attacker cannot access it. The attacker will only see the encrypted counter values. This approach has the potential advantage that if the
device and server get out of sync, it may be a good indicator that an attacker has
stolen the secret and used it.

There are many other design choices that have various pros and cons, and combining
them in various ways leads to thousands of possible implementations. Designing an
authentication mechanism for mobile applications is a non-trivial exercise. Although we
discussed several options in this document, there are too many combinations and
design/implementation choices to list here. Make sure you consult with a security
professional when designing the authentication mechanism for your mobile application.