

TECHNOLOGY OVERVIEW

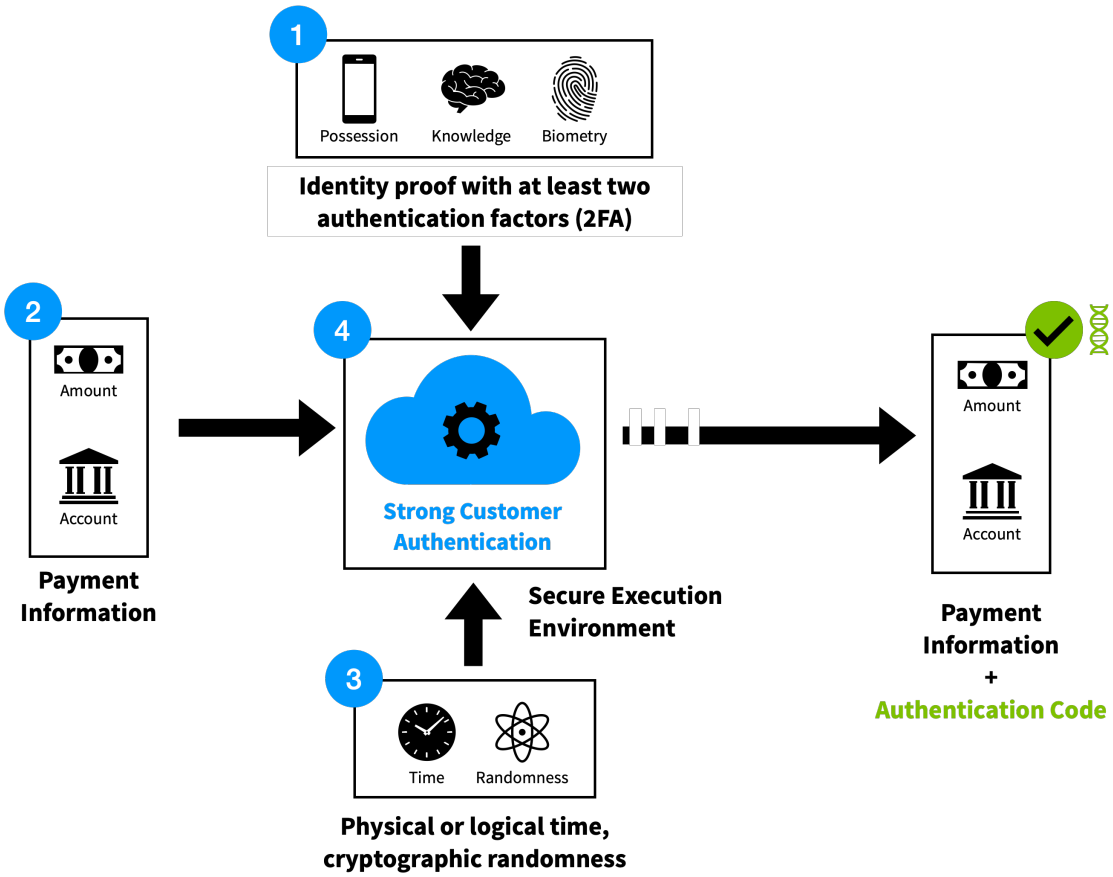
Compliance of Mobile-First Authentication with Regulatory Technical Standards (RTS) for Strong Customer Authentication (SCA) in the Context of the PSD2 Legislation

wultra

Introduction

Mobile-first authentication by Wultra helps the bank meet compliance with PSD2 Regulatory Technical Standards for Strong Customer Authentication (RTS/SCA). This document describes a basic overview of points that elaborate on this compliance.

Mobile-first authentication establishes the mobile device as a strong customer authentication (SCA) element via the process of activation – a secure device binding. After the device is bound to the user account, the SCA element computes compliant authentication code using the authentication factor keys, operation data, and other assistive cryptographic values, such as random nonce values and counters.



The main requirements for SCA-compliant authentication technologies are defined primarily in Chapter 2 of RTS ("CHAPTER II SECURITY MEASURES FOR THE APPLICATION OF STRONG CUSTOMER AUTHENTICATION").

Secure SCA Element Activation

The secure device binding process called activation in our systems is designed to establish the mobile app as a personalized SCA element which supports all three authentication factors, and which calculates authentication codes using strong cryptography.

To enable the secure device binding process, the application contains built-in (compiled-in) keys `MASTER_PUBLIC_KEY`, `APP_KEY`, and `APP_SECRET`. These values are used to initialize cryptographic algorithms used during the activation.

During the activation flow, the mobile app registers its ECDSA-P384-SHA3-384 and ML-DSA-65 public keys with the server and receives the factor specific symmetric keys and the identifier of the device over the AEAD-256 (AES-CTR/KMAC-256) encrypted channel (application-level encryption using a key established via hybrid key exchange scheme based on ECDHE-P384 and ML-KEM-768). The activation process then derives authentication factor-related keys, which are detailed in the next chapter.

You can find out more about how the mobile token is activated in the following documentation:

- https://developers.wultra.com/components/powerauth-crypto/develop/documentation/Activation

Authentication Factors Used

After the mobile application is activated using a secure pairing process, the authentication factor-related keys associated with the individual authentication factors are stored on the device. The keys are then used to calculate the authentication code for a particular transaction.

These are the following types of authentication factor-related keys:

- ***Possession*** – the relevant factor key is stored on the device using a key encryption key (KEK) derived from the device identifiers (device fingerprint).
- ***Knowledge*** – the relevant factor key is stored on the device using a key encryption key (KEK) derived from the user's password or PIN.
- ***Inherence*** – the relevant factor key is stored on the device using a randomly generated key encryption key (KEK) stored in the HW of the biometric sensor.

All authentication factor-related keys are stored on the device in the secure storage of the operating system – in the case of iOS in the iOS Keychain and in the case of Android in the storage secured by Encrypted Shared Preferences (with encryption key stored in Android KeyStore).

In both cases, security is also strengthened by the application isolation provided by the mobile operating system (sandboxing).

Detailed information about the keys used can be found here:

- https://developers.wultra.com/components/powerauth-crypto/develop/documentation/List-of-Used-Keys

Independence of Elements

Because all elements enter the algorithm for calculating the authentication code at the same time and considering that the resulting authentication code must be verified in cooperation with the server due to the algorithm design, it is not possible to recognize (from the attacker's point of view) which authentication factor failed during the particular authentication. Thanks to the design of the cryptographic algorithm, the compromise of one element (e.g., the loss of a mobile device) does not compromise other elements (especially the password / PIN code).

Authentication Code Calculation

The mobile app uses a “transaction signing” concept to calculate a cryptographic authentication code for each operation, whether it is an authentication or authorization of an active operation. The resulting authentication code is then either sent in an HTTP header (online mode) or converted to a numeric form that must be rewritten (offline mode). The authentication code has the following properties:

- Contains operation data:
  - In the case of an active operation, it is firmly linked to the operation data, such as transaction amount, currency and counterparty's account number.
  - In the case of login, it is calculated in the same way as an active operation with empty data.
- Ensures strong uniqueness reinforcement:

    - It contains a pseudo-random element in the form of a cryptographic nonce.
    - It is one-time, thanks to the use of a hash-based counter that is incremented on the device with each computed signature and on the server after each successful authentication.

      Thanks to the counter and the inclusion of a pseudo-random element, it is not possible to determine the value of another signature (or to derive the value of the counter retrospectively) from the knowledge of one signature.
- Due to the MAC algorithm (KMAC–256) used, it is impossible to determine from the authentication code alone what data was used for its calculation. Similarly, it is not possible to extract additional information from the signature, such as the authentication factor-related keys.
- Authentication codes cannot be verified locally. They must be verified in cooperation with the server component, which can block the SCA element in case of repeated failed authentication attempts.
- Due to the procedure for calculating the authentication code, it is not possible to determine from an external perspective which factor has caused authentication failure.

The algorithm for calculating the multi-factor authentication code is available here:

- https://developers.wultra.com/components/powerauth-crypto/develop/documentation/Computing-and-Validating-Authentication-Codes

Validity of the Authentication Code

The authentication code itself does not contain the physical time of expiration. For out-of-band approvals, the expiration time is stored for each particular operation and is checked when confirming the operation. This guarantees that the authentication code for a given operation can be used for a maximum of 5 minutes to confirm a given operation.

The authentication code is invalidated immediately after its use, or after the use of the next authentication code of the same SCA element.

Blocking the SCA Element

The SCA element is automatically blocked whenever there are five consecutive failed authentications. The value of maximum failure count can be changed, if necessary, by the server configuration.

It is also possible to block the SCA element manually, for example, via Internet banking or by calling a call center. In any case, the SCA element is blocked until the user unblocks the resource in cooperation with the bank.

It is also possible to permanently delete the SCA element, in which case the user must activate a new SCA element instance.

For more detailed information about the lifecycle of the SCA element, see:

- https://developers.wultra.com/components/powerauth-crypto/develop/documentation/Activation#activation-states

The mobile app can securely query the number of remaining authentications and show it to the user within the user interface. This way, the users always know that they have one last attempt left to successfully authenticate.

Documentation of how the mobile app queries the status of the SCA element is provided here:

- https://developers.wultra.com/components/powerauth-crypto/develop/documentation/Activation-Status#status-check-sequence

Secure Network Communication

Mobile app communication is always carried over a TLS/SSL-protected channel. In addition, the application supports TLS/SSL certificate pinning, which, if set, makes it impossible to intercept the contents of the encrypted communication (MITM) even if an unexpected certification authority (CA) is installed in the end device.

In addition, for particularly sensitive server queries, additional end-to-end encryption using the AEAD-256 (AES-CTR/KMAC-256) algorithm (with temporary key established via hybrid key exchange scheme based on ECDHE-P384 and ML-KEM-768) can be used on the application level to provide additional protection during transmission.

Dynamic Linking

The authentication code for any active operation includes the data of that operation, which is also presented to the user interface of the mobile app. By the cryptographic algorithm design, the valid authentication code changes for each attempt to confirm the operation. The hash-based counter and random cryptographic nonce are used for every new authentication code calculated on the device, which results in producing unique authentication code values even for the same operation data.

Secure Mobile Runtime

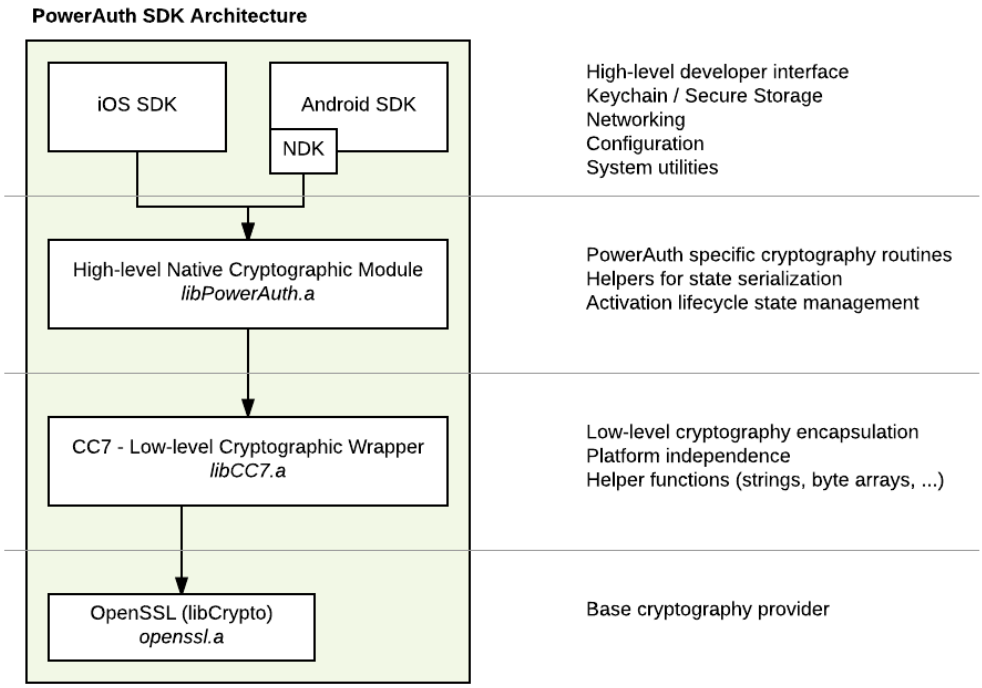
With the option to extend security with an in-app protection solution, the mobile app can offer advanced defense against a range of potential threats and attacks. In-app protection (formerly runtime application self-protection, or RASP for short) helps to meet the strictest requirements of the OWASP standard for mobile application security. This advanced solution for mobile security solves issues directly on the mobile device and optionally sends threat signals to the risk detection system.

In-app protection offers protection against a wide range of threats, including:

- Rooting/jailbreaking devices
- Overlay attacks
- Stealing keys from memory ("whitebox crypto")
- Data and application theft
- Man-in-the-app and Man-in-the-middle scenarios
- Tapjacking
- Connect the debugger to your app
- Run in an emulator or other unsupported environment
- Exchange application assets
- Damage to the integrity of the application package
- Device cloning
- Code-injection, runtime library injection, hooking frameworks
- Repackaging applications
- Use an outdated version of the operating system
- Unauthorized screenshots (system and user)
- Keylogging and screen-scraping
- External Screen Sharing
- Remote access
- Mobile Malware

Isolated Runtime Within the Mobile SDK

Authentication codes, signatures, and cryptographic algorithms are implemented in isolated low-level code (C/C++), which is available through convenient Swift/Kotlin APIs of the mobile SDK. The sensitive values are thus calculated in a well-defined component that protects the algorithm runtime to the maximum extent and meticulously enforces strict memory management over sensitive data (especially deletion of already used cryptographic material and operation data).



Audit Trails

The backend components of the mobile-first authentication store an audit trail of each individual verification attempt. The records contain the values of the authentication code used, the counter value at the time of confirmation, the timestamp, the authentication factors used, and the operation data of the operation for which the authentication code is calculated. It is thus possible to find details about the confirmed operation at any time.

Server-Side Data Storage

The keys associated with a specific activation are protected in the backend database using "transparent encryption" if the database supports it. Optionally, the keys can be also encrypted at the application level using a key derived from the "database master key" (included in the backend application configuration) and database record data. The inclusion of the record data in key derivation is also useful for consistency checking by preventing modifications of parts of the individual records (i.e., swapping ID and key).

Open-Source Components and Documentation

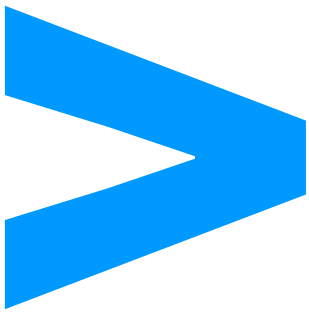
The solution is easy to audit because the implementation, including the documentation of individual components, is available online in the form of open source:

- https://github.com/wultra
- https://developers.wultra.com/tutorials/posts/Mobile-First-Authentication/

Certifications and Penetration Testing

The solution certifications and penetration testing certificates are available at:

- https://wultra.com/compliance



wultra

ABOUT US

Wultra serves as a guardian of digital finance, providing banks and fintech companies with easy-to-deploy, modern authentication solutions that genuinely improve customer experiences.

Contact our sales team:

sales@wultra.com

Visit our website:

www.wultra.com