WHITEPAPER

Authentication Protocols and Tokens: Mapping the Landscape

mobisec

Table of contents

PREPARED BY
Ahmad Zubair Zahid

PREPARED FOR
Mobisec

Introducing the paper

Authentication is a critical aspect of mobile security, but it can be complex and challenging to understand due to the abundance of similar-sounding terms and technical jargon. For instance, should we refer to "authentication protocols" or "authentication methods"? This paper aims to clarify such distinctions and bring much-needed clarity to the topic.

Understanding the available protocols, methods, and associated technologies is essential for selecting the most appropriate solution for any given context. This knowledge benefits both project managers and developers. For managers, a shared understanding and consistent terminology improve communication, minimizing the risk of misinterpretation. For developers, knowing where to start and what to prioritize enables informed decision-making and ensures the correct implementation path is followed. With these objectives in mind, we have titled this paper *Authentication Protocols and Tokens: Mapping the Landscape.*

The paper begins by demystifying commonly confused terminology and then explores authentication protocols, detailing their purposes, use cases, and implementations within the context of mobile applications. Additionally, it includes an appendix of terms to further elaborate on key concepts. This structured approach equips both managerial and technical teams with the insights needed to effectively navigate the evolving landscape of mobile authentication.

This is an approachable guide designed to help you enhance the security of your mobile applications, with a focus on authentication—the process of verifying user identities. The guide:

- Simplifies complex and often confusing concepts, making them more accessible and easier to understand;
- Maps out the relationships between authentication methods, protocols, and tokens, offering clear insights;
- Is tailored for project managers and developers, providing information to support decision-making and introduce advanced options that may not yet be on your radar.

Equip your team with practical knowledge to build secure, scalable, and future-proof applications.

PREPARED BY
Ahmad Zubair Zahid

PREPARED FOR
Mobisec

Authentication Protocols and Tokens: Mapping the Landscape

Authentication is the process of verifying that an individual or system is who they claim to be, while identification simply establishes their identity. In our daily interactions with technology, we frequently encounter terms such as *Authentication*, *Authorization*, *Authentication Protocols*, and *Authentication Methods.* Although these terms may sound similar, they serve distinct purposes, particularly in the realm of digital security.

This paper aims to clarify these concepts and highlight their importance in the context of mobile applications.

Authentication vs Authorization

Think of authentication and authorization as entering a secured building. Authentication is like showing your ID to verify who you are, while authorization is like being told what rooms you're allowed to access.

Authentication ensures that users are who they state they are, often through passwords, biometrics, or multi-factor verification. Authorization, on the other hand, controls what verified users can access within a system. These two steps work together, but serve different purposes in the digital world. While authentication answers the question "Who are you?", authorization answers the question "What are you allowed to do?

Authentication Methods vs Authentication Protocols

Authentication Method: The process by which you provide your credentials—be it through a password, fingerprint scan, or face ID—is referred to as the authentication method.

Authentication Protocol: This refers to the underlying system that securely manages, transmits, and verifies authentication information between your device, the website's servers, and other involved parties. Authentication protocols define the specific steps, message formats, and security measures required to ensure that the process is secure, reliable, and interoperable across different systems. Acting as a common language, they enable seamless and secure communication among the various components involved in authentication, regardless of their individual implementations.

In other words, authentication methods focus on how users prove their identity, such as using passwords or biometrics.

Authentication protocols, such as OAuth or SAML, define how this information is communicated securely between systems. This paper focuses on the latter, with a spotlight on mobile applications and their server connectivity.

Session vs Token-Based Authentication

When users log into an application, they expect to remain authenticated without having to re-enter their credentials. This seamless experience is enabled by either session-based or token-based authentication mechanisms.

Session-Based Authentication:
In session-based authentication, the server generates a unique session ID upon user login and stores the session data on the server-side. This session ID is then sent to the client, typically via a cookie, and included in subsequent requests to maintain the authenticated state.
This method is particularly well-suited for traditional web applications and browser-centric environments, where server-side session management provides enhanced security controls. Browsers can handle session cookies securely using attributes such as ***HttpOnly, Secure,*** and ***SameSite*** flags. These attributes prevent client-side script access, ensure that cookies are transmitted only over HTTPS, and protect against cross-site request forgery (CSRF) attacks.

Token-Based Authentication:
Token-based authentication involves issuing cryptographically signed tokens—such as JSON Web Tokens (JWTs)—to authenticated users. These tokens encode user information and claims, enabling the server to verify identity and permissions without maintaining session state. This stateless approach is particularly well-suited for mobile applications, Single Page Applications (SPAs), and distributed systems, as it enhances scalability and reduces server overhead. Tokens can be securely stored on mobile devices using hardware-backed storage solutions, such as the Android Keystore or iOS Keychain.
Protocols like OAuth 2.0 and JWTs enable efficient communication between mobile applications and APIs. Additionally, the use of refresh tokens allows for extended session durations and offline capabilities, ensuring users can stay logged in without frequent re-authentication.

Appropriateness for Mobile Applications

01 Statelessness and Scalability

The stateless nature of token-based authentication reduces server-side complexity and improves scalability, an essential benefit for mobile applications interacting with distributed APIs and microservices.

02 Performance

By eliminating the need for server-side session validation, token-based authentication reduces latency and improves the responsiveness of mobile applications, in particular in environments with unreliable networks.

03 Enhanced Security

Storing tokens in secure, hardware-backed storage significantly reduces the risk of compromise. In addition, token expiration and refresh mechanisms help mitigate the risks associated with token theft or misuse.

04 Flexibility and Fine-Grained Access Control

Tokens can embed detailed claims about user roles and permissions, enabling fine-grained access control without additional database queries.

05 Cross-Platform Consistency

Token-based systems enable seamless authentication across multiple platforms (mobile, web, desktop) because tokens are not tied to a specific server-side session store.

Token-based authentication is highly suited for mobile security due to its stateless design, performance advantages, enhanced security measures, and flexibility in distributed and cross-platform environments. It seamlessly meets the needs of today's mobile applications, which require efficient, secure, and scalable authentication mechanisms.

Common Authentication Protocols

OAuth 2.0

Description: OAuth 2.0 enables third-party applications to access an HTTP service on behalf of a user by issuing access tokens, eliminating the need to directly handle user credentials. The framework supports a variety of grant types, including authorization code, implicit, password, and client credentials, providing flexibility to address a wide range of scenarios.

Mobile App Integration: Mobile applications typically use Authorization Code Grant with PKCE, where the application generates a code verifier and challenge to securely obtain an access token after the user authenticates.

Security Considerations: OAuth 2.0 is widely adopted and supported, but improper implementations can introduce vulnerabilities such as token interception or misconfiguration. This makes rigorous security validation essential.

Use Case: OAuth 2.0 is ideal for mobile applications that require delegated access to user resources, such as integrating social media platforms without the need to manage user passwords.

OpenID Connect (OIDC)

Description: OIDC extends OAuth 2.0 by introducing an identity layer that allows clients to verify a user's identity through an authorization server and retrieve basic profile information in a standardized format.

Mobile App Integration: After the user authenticates, the application receives an ID token (a JWT) that contains claims about the user. To authenticate the user, the application validates this token by verifying its signature, issuer, audience, and expiration.

Security Considerations: OIDC inherits the complexity of OAuth 2.0 and requires careful handling of ID tokens to prevent vulnerabilities such as token replay attacks or unauthorized access.

Use Case: OIDC is ideal for applications that require user authentication and profile information from identity providers, such as Google or Microsoft, without having to manage user credentials directly.

SAML (Security Assertion Markup Language)

Description: SAML is an XML-based framework designed to facilitate the exchange of authentication and authorization data, commonly used for Single Sign-On (SSO) in enterprise environments.

Mobile App Integration: The app redirects users to a SAML identity provider for authentication. The provider returns a signed SAML assertion that the application forwards to the server for validation.

Security Considerations: SAML provides robust security for enterprise applications requiring SSO, but its XML processing and heavyweight nature can make it more resource-intensive than other protocols.

Use Case: SAML is best suited for enterprise applications that need to integrate with corporate identity systems for SSO, especially when users need to access multiple services with a single login.

Other methods

In addition to widely used user authentication protocols such as OAuth 2.0, OpenID Connect, and SAML, other methods are available for authenticating clients or endpoints in low-level technical integrations. Mutual TLS (mTLS) and API keys are two notable examples.

mTLS provides mutual authentication by requiring both the client and server to present certificates during communication. This approach provides robust security, but comes with the added complexity of certificate management.

API keys, on the other hand, are static tokens used for simple client authentication. While they are easier to implement, they lack user-specific controls and can be vulnerable if not properly secured.

Both mTLS and API keys prioritize securing client-server communication over user authentication, making them ideal for specific technical use cases.

Summary of Authentication Protocols

Use Cases, Security Features, Complexity, and Integration

Protocol	Authentication Type	Security Features	Complexity	Token Type	Mobile Integration	Common Scenarios
OAuth 2.0	Token-based	Flexible, supports scopes, PKCE for public clients	Medium	Access token	PKCE recommended for enhanced security	Social media login, access to third-party APIs
OpenID Connect	Token-based	JWT for identity, SSO	Medium	ID token (JWT)	Built on OAuth 2.0, PKCE supported	SSO, user authentication via providers (Google, Microsoft)
Mutual TLS	Certificate-based	Strong identity verification	High	X.509 certificates	Requires secure storage of certificates	Financial services, healthcare, internal enterprise apps
SAML	Assertion-based	Strong, XML-based assertions	High	SAML assertion	Redirect-based flow	Corporate SSO, enterprise identity integration
API Keys	Key-based	Simple, can be used for client and server auth	Low	Static key	Easy to implement, lightweight	Public data access, analytics, server-to-server communication

Conclusions

The landscape of authentication protocols is constantly evolving, especially in the context of mobile applications. As mobile technologies continue to evolve, the need for scalable, secure, and easy-to-use authentication methods becomes increasingly important. Token-based authentication, especially via protocols such as OAuth 2.0 and OpenID Connect, strikes a balance between security and performance, protecting sensitive data while providing a seamless user experience.

However, the choice of authentication protocol should be driven by the specific requirements of the application, considering factors such as security needs, scalability, and user experience. As digital security threats continue to evolve, so must the protocols and methods we rely on. This underscores the importance of staying informed about mobile authentication best practices and emerging standards.

Further Readings

For those looking to deepen their understanding of authentication protocols and token-based security, there is a wealth of basic and advanced resources covering both theoretical frameworks and practical implementations.

These materials cover key topics such as the OAuth 2.0 authorization framework, OpenID Connect for identity verification, and the structure and security of JSON Web Tokens (JWTs).

In addition, resources such as the NIST Digital Identity Guidelines provide authoritative best practices for implementing secure authentication mechanisms, while OWASP provides valuable insight into common mobile application security risks. Together, these resources provide a comprehensive view of the authentication landscape, enabling readers to design and maintain robust authentication systems across multiple platforms and environments.

- RFC 6749: The OAuth 2.0 Authorization Framework
- OAuth 2.0
- OpenID Connect Core 1.0 Specification
- RFC 7519: JSON Web Token (JWT)
- JWT.io: Introduction and Debugger Tool
- Apple Developer Documentation: Keychain Services
- Android Developers: Security with Credential Storage
- OWASP: Transport Layer Protection Cheat Sheet
- OWASP API Security Top 10
- NIST Special Publication 800-63B: Digital Identity Guidelines
- Android Developers: Best Practices for Security & Privacy
- Apple Developer Documentation: Secure Coding Guide
- OWASP Mobile Top 10 Risks
- CWE List: Common Weakness Enumeration

Appendix

Tokens

While protocols such as OAuth 2.0 and OpenID Connect define the framework for authentication processes, tokens serve as the practical tools that enable these operations. Acting as digital keys, tokens grant access to protected resources once a user's identity has been verified.

Understanding the structure, usage, and security considerations of tokens is essential, as they play a key role in ensuring secure, scalable, and efficient authentication across distributed systems - especially in mobile environments.

Understanding Tokens

Tokens are fundamental to modern authentication and authorization mechanisms in mobile applications. Acting as digital keys, they allow clients to access protected resources without the need to repeatedly present user credentials.

Tokens vs. Credentials: Credentials are highly sensitive pieces of information, typically consisting of a username and password used for initial authentication. They must be protected at all times to prevent unauthorized access.
Tokens, on the other hand, are issued after successful authentication and are used for subsequent requests. Usually short-lived, dynamically generated, and limited in scope, tokens reduce the risk associated with long-lived credentials.

Common Types of Token Based on Purpose

Access Tokens: Access tokens grant access to protected resources. They are short-lived to minimize risk if compromised and include scopes that define accessible resources accessible resources. In practice, they are sent in the *Authorization* header as *Bearer <token>*.

ID Tokens: ID tokens verify a user's identity. They contain user identity assertions and are commonly used in OpenID Connect implementations. Clients use these tokens to locally authenticate the user without revalidating credentials with the server.

Refresh Tokens: Refresh tokens are used to obtain new access tokens without requiring the user to re-authenticate. They are long-lived but highly sensitive and must be stored securely. Clients send refresh tokens to the token endpoint to obtain new access tokens when the current ones expire.

Types of Tokens Based on Structure

Opaque Tokens vs. Structured Tokens: Opaque tokens are random strings with no inherent meaning. The server maintains a mapping between the token and the session data, which requires server-side storage and validation. Structured tokens, such as JSON Web Tokens (JWTs), contain encoded data - often in JSON format - and are self-contained, allowing validation without server-side storage.

JSON Web Tokens (JWTs): JSON Web Tokens (JWTs) consist of three Base64 encoded parts: the header, the payload, and the signature. The header specifies the token type and signing algorithm, the payload contains user claims and metadata, and the signature ensures the integrity and authenticity of the token.

JWTs are advantageous in mobile applications because they support stateless authentication, eliminating the need for server-side session storage. Their compact size lends itself to bandwidth-constrained mobile networks, making them efficient in distributed systems.
For security, it's important to verify the token's signature and enforce short-lived tokens. Validating expiration, audience, and issuer claims helps prevent token misuse.

Appendix

Secure Token Handling Practices in Mobile Applications

Secure Storage of Tokens:
On iOS devices, use the Keychain Services API to store tokens securely.
On Android devices, use the Android Keystore System or Encrypted Shared Preferences.
NOTE: You should use backup rules to exclude all EncryptedSharedPreferences from backup.

Secure Transmission: Always send tokens over HTTPS to prevent interception by unauthorized parties. Implementing SSL pinning adds an extra layer of security by ensuring that the application communicates only with trusted servers, preventing man-in-the-middle attacks.

Token Renewal and Revocation: When access tokens expire, securely use refresh tokens to obtain new tokens. Implement server-side mechanisms to check token revocation lists or opt for short-lived tokens to minimize risk if a token is compromised.

Preventing Token Leakage: Avoid logging tokens in application logs or crash reports, as these can be accessed by unintended recipients. Ensure that tokens are not exposed to other applications through secure inter-process communication mechanisms.

Handling Token Expiry: Detect token expiration by checking the "exp" claim in JWTs or by handling server responses indicating token expiration. Implement logic to automatically refresh tokens with refresh tokens, providing a seamless user experience without unnecessary re-authentication prompts.

References

OAuth 2.0 Authorization Framework - RFC 6749
Reference:
Hardt, D. (2012).
The OAuth 2.0 Authorization Framework (RFC 6749). Internet Engineering Task Force (IETF).

Available at: https://tools.ietf.org/html/rfc6749

OpenID Connect Core 1.0
Reference:
Sakimura, N., Bradley, J., Jones, M., de Medeiros, B., & Mortimore, C. (2014).
OpenID Connect Core 1.0 incorporating errata set 1. OpenID Foundation.

Available at: https://openid.net/specs/openid-connect-core-1_0.html

JSON Web Token (JWT) - RFC 7519
Reference:
Jones, M., Bradley, J., & Sakimura, N. (2015).
JSON Web Token (JWT) (RFC 7519). Internet Engineering Task Force (IETF).

Available at: https://tools.ietf.org/html/rfc7519

NIST Digital Identity Guidelines
Reference:
National Institute of Standards and Technology (NIST). (2017).
Digital Identity Guidelines (SP 800-63B).

Available at: https://pages.nist.gov/800-63-3/sp800-63b.html

OWASP Mobile Top 10 Risks
Reference:
OWASP Foundation. (2016).
Mobile Top 10 Risks.

Available at: https://owasp.org/www-project-mobile-top-10/

staysafe@mobisec.com

Mobisec

Viale della Repubblica 22 / III
Villorba (Treviso) - Italia

mobisec.com