UDC 004.056.55 **doi:** https://doi.org/10.20998/3083-6298.2025.02.01

Ostap Vasylenko[1]

[1]Aimprosoft, Lviv, Ukraine

COMPARATIVE ANALYSIS OF MODERN AUTHENTICATION AND AUTHORIZATION PROTOCOLS FOR WEB APPLICATIONS

Abstract. Topicality. In the context of rapid digital transformation, secure access control to information resources has become critically important. Authentication and authorization mechanisms form the basis of information security, ensuring user identity verification and assigning appropriate access rights. The expansion of web applications, cloud services, and distributed systems requires the implementation of flexible and attack-resistant protocols. **The subject of study** in this article is a set of widely used authentication and authorization protocols, including Basic Authentication, OAuth 1.0, OAuth 2.0, Single Sign-On (SSO), and Multi-Factor Authentication (MFA). **The purpose of the article** is to compare these technologies in terms of security, scalability, integration complexity, and suitability for various IT environments. The research is based on the analysis of official standards (RFCs, NIST guidelines, OASIS specifications) and industry practices. **The results** demonstrate that traditional approaches, such as Basic Auth and OAuth 1.0, are becoming obsolete due to limited security, while OAuth 2.0 combined with SSO provides a balance between user convenience and security. The introduction of MFA significantly enhances protection against account compromise but increases implementation complexity. **Conclusion.** No single protocol is a universal solution; the highest security level is achieved through a combination of modern authorization protocols, multi-factor authentication, token encryption, and Zero Trust principles. These findings can be applied by developers, system architects, and cybersecurity specialists to design and implement robust authentication and authorization systems capable of addressing current and emerging digital security challenges.

Keywords: authentication; authorization; OAuth; Basic Auth; Single Sign-On; Multi-Factor Authentication; digital security.

Introduction

Problem statement and relevance. In the context of rapid digital transformation, secure access control to information resources has become critically important. Authentication and authorization are fundamental security mechanisms that ensure the verification of a user's identity and the definition of their access rights. The growing number of web applications, cloud services, and distributed information systems requires the use of flexible and attack-resistant protocols.

Literature review. Research on authentication and authorization mechanisms has evolved significantly in recent years, reflecting the growing complexity of information systems and the need for robust access control. According to RFC 7617 [7], Basic Authentication remains one of the earliest HTTP-based authentication schemes, but it lacks encryption and is recommended only in conjunction with TLS. OAuth protocols, described in RFC 5849 [8] and RFC 6749 [9], introduced delegated authorization, enabling third-party applications to access resources without exposing user credentials. OAuth 2.0 further simplified token handling and is now a de facto standard for web and mobile application security. Single Sign-On (SSO) solutions, implemented via SAML 2.0 [10] or OpenID Connect [11], provide centralized authentication and improve user experience while reducing password fatigue, though they introduce a single point of failure. Multi-Factor Authentication (MFA), recommended by NIST SP 800-63B [12] and CISA guidelines [22], strengthens security by requiring multiple verification factors, effectively mitigating risks such as phishing and credential theft.

Recent studies [21–25] emphasize the importance of integrating modern protocols with additional safeguards, including API gateways, rate limiting, and Zero Trust principles, to address emerging threats such as token interception and replay attacks. However, the choice of a specific authentication method remains context-dependent, balancing security requirements, user convenience, and implementation complexity.

The purpose of the research. The purpose of this article is to conduct a comparative analysis of key authentication and authorization protocols used in web applications, taking into account their security characteristics, ease of integration, and scalability potential. The study also aims to formulate practical recommendations for selecting the most appropriate technologies for corporate and public IT environments.

1. Theoretical Background

1.1. Definition of Authentication and Authorization. According to ISO/IEC 27000:2018 and the terminology defined in RFC 4949: Internet Security Glossary, authentication is the process by which a system verifies whether the claimed identity of an entity (user, device, or process) corresponds to the evidence it provides. Such evidence may include secret data (e.g., a password), cryptographic keys, hardware or software tokens, as well as biometric characteristics. The goal of this process is to ensure that access is granted to the exact entity it claims to be.

Authorization, as defined in the same sources, is the stage that follows successful authentication and involves verifying an entity’s rights to specific resources or actions. This is achieved by mapping the entity to certain roles, groups, or other access attributes and applying security rules defined in the organization’s policies.

In simplified terms, authentication establishes the identity of the user, while authorization defines the boundaries of their capabilities within the system. Although they are closely related, these processes are implemented separately and can be combined in various configurations depending on the requirements and architecture of the information system.

1.2 Access Control Models. Access control models define the logic and rules by which users, processes, or other entities are granted or restricted in their ability to interact with the resources of an information system [1–6]. They establish mechanisms for verifying permissions and regulate who may perform specific operations on objects, such as reading, modifying, deleting, or executing, and under what conditions.

The choice of a particular access control model affects the level of information security, the scalability of rights management, administrative convenience, and compliance with regulatory requirements [2,3].

The following subsections present the key characteristics, advantages, disadvantages, and typical use cases of commonly adopted models, including Discretionary Access Control (DAC), Mandatory Access Control (MAC), Role-Based Access Control (RBAC), Attribute-Based Access Control (ABAC), Identity-Based Access Control (IBAC), and Access Control Lists (ACLs).

Discretionary Access Control (DAC) **-** is a model in which the resource owner independently determines who may perform specific operations on that resource. Permissions are stored in the form of Access Control Lists (ACLs). This approach is flexible but poses the risk of unstructured rights management in large-scale systems due to the absence of a centralized policy.

Advantages: flexibility of configuration; simplicity of implementation in small-scale systems.

Disadvantages: lack of centralized management; risk of uncontrolled privilege propagation.

Use cases: UNIX file systems, NTFS; small corporate services with a limited number of users.

Mandatory Access Control (MAC) - is a model based on a strict, centralized security policy that neither users nor resource owners can modify. Access is determined according to security labels (e.g., Confidential, Secret, Top Secret) assigned to both resources and users. It is most commonly applied in military, governmental, and mission-critical systems where misconfiguration of permissions can have catastrophic consequences.

Advantages: high level of security; compliance with strict standards.

Disadvantages: low flexibility; complexity of configuration and maintenance.

Use cases: governmental and military information systems; SELinux; Trusted Solaris.

Role-Based Access Control (RBAC) - is a model in which access rights are assigned not to individual users but to specific roles (e.g., *Administrator*, *Editor*, *Viewer*). Users automatically obtain the necessary privileges when they are assigned to a corresponding role. This approach simplifies administration in large-scale systems, enables centralized access management, and reduces the risk of errors in permission assignment, particularly in corporate environments with a large number of users and a complex access hierarchy.

Advantages: scalability; centralized management; support for the principle of least privilege.

Disadvantages: complexity of managing a large number of roles; requires a well-designed role structure.

Use cases: corporate ERP systems; AWS IAM; human resource management systems.

Attribute-Based Access Control (ABAC) - is a flexible model in which access decisions are made based on a set of attributes: user properties (e.g., job title, department), resource characteristics (e.g., data type, confidentiality level), and environmental conditions (e.g., time, location of access). Access policies are defined as logical rules, enabling precise control over access conditions and consideration of the context in which the request is executed.

Advantages: high flexibility; adaptability to various scenarios.

Disadvantages: implementation complexity; requires a well-developed infrastructure and clearly defined policies.

Use cases: Google BeyondCorp; Microsoft Conditional Access; cloud services with dynamic context-based verification.

Identity-Based Access Control (IBAC) - is a model in which access rights are assigned directly to specific users rather than to roles or attributes. Identification is performed using a unique identifier (e.g., username, certificate, ID card), after which the user is granted a set of permissions linked specifically to their account. This approach is simple to implement and convenient for small systems with a limited number of users; however, in large-scale environments, it complicates administration because each permission change must be applied to individual users separately.

Advantages: easy configuration in small systems; rapid implementation.

Disadvantages: poor scalability; high administrative overhead in large environments.

Use cases: small internal company systems with few users and no defined roles.

Access Control Lists (ACLs) are an access control mechanism in which each object stores a list of subjects (users or groups) and their corresponding permissions (read, write, execute, etc.). When an object is accessed, the system checks this list to determine whether the subject has the required rights. ACLs are convenient for fine-grained configuration of access to individual resources; however, in large systems, managing these lists can become complex and may require automation. This approach offers flexibility, but with a large number of objects and users, ACL administration can be labor-intensive.

Advantages: fine-grained access control for each resource.

Disadvantages: maintenance complexity in large systems; risk of errors during large-scale changes.

Use cases: POSIX ACL, NTFS ACL, network ACLs in routers and firewalls.

1.3 The Role of Encryption and Tokens. Encryption and tokens are fundamental components of modern authentication and authorization mechanisms, playing a key role in ensuring confidentiality, integrity, and controlled access within information systems.

Encryption ensures that transmitted or stored credentials, access tokens, and other sensitive data remain inaccessible to unauthorized parties, even in cases of intercepted network traffic or compromised system nodes.

Tokens, in turn, provide a flexible and standardized way to confirm access rights without repeatedly transmitting primary credentials, thereby reducing the risk of their disclosure and simplifying integration between distributed services.

Combined, these two mechanisms form the foundation of secure interaction between users, services, and resources in corporate, cloud, and distributed environments [5,9,13-15].

Encryption - according to NIST SP 800-175B [13] and RFC 4949 [5], encryption is the process of transforming information into a form unreadable by unauthorized parties, using cryptographic algorithms and keys.

In the context of authentication and authorization, encryption performs several critical functions:

Credential protection during transmission — ensured by transport-layer protocols such as TLS 1.3, which guarantee that logins, passwords, and tokens cannot be intercepted in plain text;

Secure password storage — achieved by applying cryptographic hash functions with salting (bcrypt, scrypt, Argon2), making it impossible to recover the original password even in the event of a database breach;

Encryption of access tokens and session data — prevents unauthorized use or forgery of data that confirms the authenticity of a session. The use of modern cryptographic algorithms (AES-256, ChaCha20-Poly1305, ECC) and adherence to NIST recommendations significantly reduces the risk of man-in-the-middle attacks and credential compromise;

Tokens - as defined in RFC 6750 [14] and RFC 7519 [15], a token is a digital marker containing information about the subject's identity and their access rights. Tokens eliminate the need for repeated password transmission and provide a standardized mechanism for interaction between clients and servers.

Main types of tokens:

Bearer tokens — grant access to any subject that presents them and therefore require special protection against interception.

Proof-of-possession tokens — require cryptographic proof that the subject possesses the private key associated with the token.

Most common token formats:

JWT (JSON Web Token) [15] — a compact format that can include a signature (JWS) or encryption (JWE), allowing secure transmission of user information and access rights.

Opaque tokens — non-transparent identifiers that carry no meaningful payload and require validation on the authorization server.

Interaction between Encryption and Tokens. In modern authorization protocols such as OAuth 2.0 [9] and OpenID Connect [11], tokens are transmitted exclusively over secure channels (HTTPS/TLS), and their contents are additionally signed or encrypted. This ensures the integrity and authenticity of the transmitted data and makes forgery impossible. Improper implementation of encryption or insufficient protection of tokens often leads to critical vulnerabilities, such as session hijacking or replay attacks. Therefore, these aspects must be a top priority when designing secure systems.

2. Overview of Modern Approaches

Modern authentication and authorization systems implement a range of technologies and architectural solutions aimed at achieving a balance between security, user convenience, and administrative efficiency. This section provides an extended review of the main methods most commonly used in web applications, corporate environments, and cloud services, in accordance with official standards and specifications [7–12].

2.1. Basic Authentication. According to RFC 7617 [7], Basic Authentication is an HTTP authentication method in which user credentials are transmitted in Base64 format within the Authorization header. The method is simple to implement but does not provide encryption, and therefore should only be used in combination with HTTPS [12,24].

```
Authorization: Basic <base64(username:password)> ->
Authorization: Basic YWRtaW46MTIzNA==
```

Fig. 1. Example: The request header for a user: "admin" with password: "1234". Where:

YWRtaW46MTIzNA== is the Base64-encoded string "admin:1234"

```
GET /api/data HTTP/1.1
Host: testenv.com
Authorization: Basic YWRtaW46c2VjcmV0
```

Fig. 2. Example: HTTP request uses basic auth

Key features: Base64 is not an encryption method — it only encodes data into a text format for transmission and does not ensure confidentiality. This approach does not hide content, and anyone obtaining the Base64 string can easily decode it back; it also does not protect against tampering or interception. Additionally, with this method, the username and password are sent with every HTTP request to the server. On the positive side, it is compatible with most HTTP clients and servers and requires no additional libraries.

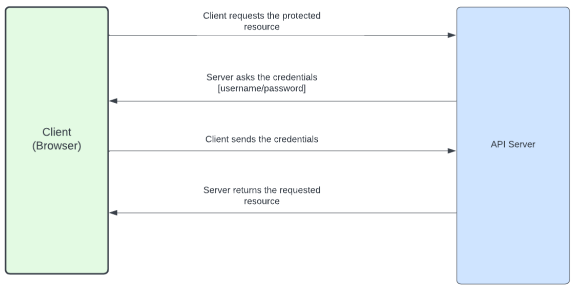
Client requests the protected resource
Server asks the credentials [username/password]
Client sends the credentials
Server returns the requested resource
Client (Browser)
API Server

Fig. 3. Basic Authentication simplified flow

Recommendations for using this protocol include mandatory HTTPS to prevent credential interception [12,24]. Passwords should never be stored in plain text on the server — hashing algorithms such as bcrypt or Argon2 must be used instead. This method should be combined with other security measures, such as API Gateway, rate limiting, and IP address filtering. Furthermore, instead of sending a username and password directly, it is preferable to use an API token to reduce the risk of account compromise.

Disadvantages:

- No protection against interception – Without TLS (HTTPS), credentials are transmitted in clear text and can be captured;
- No session mechanism – The server does not maintain state; the client must send credentials with each request;
- No password brute-force protection – Credential guessing is possible without additional safeguards.

Advantages:

- Ease of implementation – Supported by almost all web servers, browsers, and HTTP clients without extra configuration;
- Operational transparency – Does not require complex libraries or additional protocols.

Use cases: Internal APIs in secure networks, test environments, and quick prototypes. Rarely used in public services without TLS.

2.2. OAuth 1.0 / OAuth 1.0a. According to RFC 5849 [8], OAuth 1.0 is a delegated authorization protocol that enables one application to obtain limited access to the resources of another without transmitting the user's credentials. It uses cryptographic signatures to verify requests. The core idea is that the user explicitly grants the application permission to access their data, after which the service issues temporary access keys (Access Tokens) to the application. In this way, the user's login and password are neither transmitted to nor stored by the third-party application.

A typical authorization process involves the client first sending a request to the service to obtain a temporary token and secret (Request Token). The user is then redirected to a URL provided by the service, where they grant access. Afterwards, the client exchanges the Request Token for an Access Token, which is subsequently used to sign requests and gain access to the required resources.

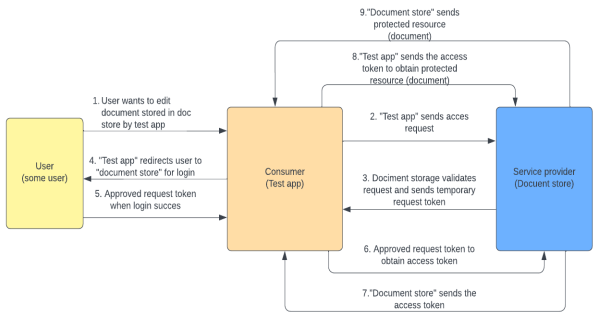
9."Document store" sends protected resource (document)
8."Test app" sends the access token to obtain protected resource (document)
1. User wants to edit document stored in doc store by test app
2. "Test app" sends acces request
User (some user)
4. "Test app" redirects user to "document store" for login
Consumer (Test app)
3. Dociment storage validates request and sends temporary request token
Service provider (Docuent store)
5. Approved request token when login succes
6. Approved request token to obtain access token
7."Document store" sends the access token

Fig. 4. OAuth 1.0 simplified flow

Key features: OAuth 1.0 does not transmit the user’s password to third-party services, relying instead on signed requests (HMAC-SHA1, RSA-SHA1, or PLAINTEXT) to prevent data tampering. Although it can technically operate without HTTPS, in modern environments the use of HTTPS is still mandatory. The protocol also protects against replay attacks by using unique nonce values and timestamp parameters for each request.

Disadvantages:

- Complex implementation (particularly signature generation and parameter handling);
- Multiple steps in the authorization process increase the risk of errors;
- Largely replaced by OAuth 2.0 due to complexity and lack of flexibility;
- Poor compatibility with modern SPA and mobile applications.

Advantages:

- Prevents password disclosure to third-party services;
- Can work without HTTPS (due to cryptographic signing);
- Protects against data interception in transit (when signature implementation is correct).

Use Cases: Service-to-service integrations in the 2008–2013 period (e.g., legacy Twitter API, older versions of LinkedIn API).

2.3. OAuth 2.0. OAuth 2.0 is a modern authorization protocol that allows third-party applications to obtain limited access to a user’s resources without the need to transmit the user’s password. It was standardized in RFC 6749 [9] in 2012 as a simplified and more flexible replacement for OAuth 1.0. The main difference from the previous version lies in abandoning the complex mechanism of cryptographic signatures in favor of mandatory use of the HTTPS protocol and simplified token handling.

Key Features: The protocol defines four key roles: Resource Owner — the owner of the resources, typically the end user, who controls access to their data; Resource Server — the server where the resources are stored and which provides them in response to requests with a valid access token; Client — the application that seeks to access resources on behalf of the user; and Authorization Server — the server responsible for authenticating the user and issuing tokens [16]. Several types of tokens are used in the authorization process.

The Access Token has a short lifespan and is used to access resources, while the Refresh Token has a longer lifespan and allows obtaining a new Access Token without reauthorization. In the context of OpenID Connect, an ID Token is additionally used, containing information about the user [17], [18]. Tokens are obtained through so-called grant types. The most secure and recommended method is the Authorization Code Flow (with PKCE support for mobile applications and SPAs) [19]. Less secure methods include the Implicit Flow for SPAs and Resource Owner Password Credentials (ROPC), where the user directly enters their login and password into the client. The Client Credentials Flow is used for machine-to-machine communication between services without user involvement [16,20].

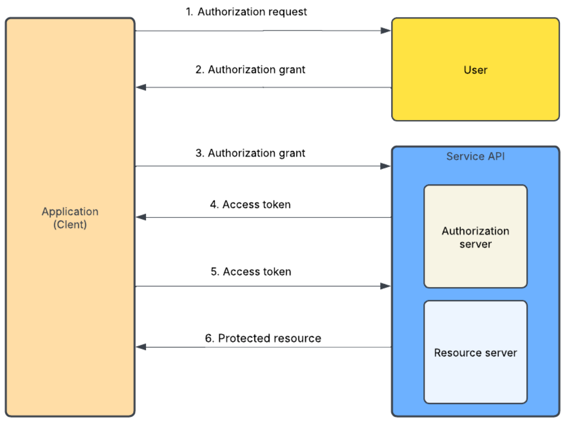

Fig. 5. OAuth 2.0 simplified flow

Disadvantages:

- Complete reliance on HTTPS for security;
- Vulnerability in case of token compromise;
- Lack of a built-in mechanism for request signing;
- Need for careful storage and protection of tokens.

Advantages:

- Delegation of access without transmitting the password;
- Support for various scenarios (web, mobile, IoT, API);
- Flexible integration with OpenID Connect for Single Sign-On (SSO);
- Simpler implementation compared to OAuth 1.0.

Use Cases: OAuth 2.0 is widely applied across various domains due to its flexibility and support for multiple authorization scenarios. In public web applications, it enables secure access to user data on external services — for example, allowing a website to retrieve a user's Google Calendar events or Facebook profile information without exposing credentials. In mobile applications, it provides secure authentication and resource access through integration with social identity providers (e.g., Google Sign-In, Apple ID). In enterprise environments, OAuth 2.0 is frequently used in combination with OpenID Connect for implementing Single Sign-On (SSO) across corporate applications, thereby improving user experience and reducing password fatigue. In API-based architectures, it facilitates secure machine-to-machine communication by issuing tokens via the Client Credentials Flow, ensuring that backend services can authenticate and authorize each other without human intervention. In IoT ecosystems, OAuth 2.0 supports secure device-to-cloud interactions, where constrained devices use delegated tokens to interact with cloud services under predefined scopes and lifetimes.

2.4. Single Sign-On (SSO). According to the *SAML 2.0 Technical Overview* [10] and *OpenID Connect Core 1.0* [11], Single Sign-On (SSO) is an authentication architecture that allows a user to complete a single login procedure and gain access to multiple independent systems without re-entering their credentials. SSO is not a standalone protocol but is implemented using existing standards and technologies such as SAML 2.0, OpenID Connect, or Kerberos. In this mechanism, a central authentication service maintains the user's session and issues tokens or assertions to other systems that trust this service. SSO is widely used in corporate, cloud, and inter-organizational environments, ensuring a balance between usability and centralized access management.

Key Features. The key principle of SSO is that a user authenticates only once via a central service, after which the system passes a token or assertion to other services confirming the user's identity. Administrators can centrally enforce unified access policies, including multi-factor authentication, IP-based restrictions, and session management. The primary advantages of this approach include a reduced number of passwords to remember, improved productivity through faster access to resources, and centralized security control. At the same time, disadvantages include a single point of failure—if the central IdP experiences an outage, access to all services is blocked; the risk of account compromise granting an attacker access to all integrated systems; and the need for configuration alignment between all participants in the system.

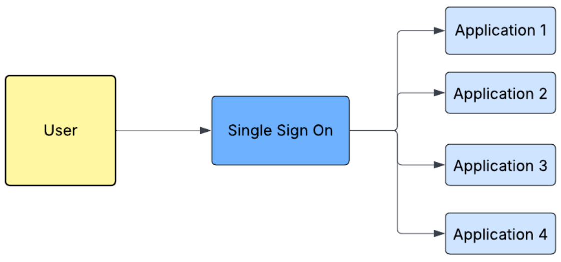

Fig. 5. SSO simplified schema

SAML 2.0. Security Assertion Markup Language (SAML) 2.0 is an XML-based standard for exchanging authentication and authorization data, developed by the OASIS Security Services Technical Committee [10]. Its primary purpose is to enable SSO in corporate and inter-organizational environments. The process works as follows: the user sends an access request to a service provider (SP), which redirects them to an identity provider (IdP) for authentication. Upon successful verification, the IdP generates a SAML Assertion—an XML document containing the user's attributes and authorization information. The SP validates the digital signature on the assertion and, if successful, grants access to the resource. The key roles in this process include: the Principal (the user) — initiates the resource access request; the Identity Provider (IdP) — authenticates the user and issues the SAML assertion; and the Service Provider (SP) — grants access to the resource upon successful validation of the assertion.

OpenID Connect (OIDC). OpenID Connect is an authentication protocol built on top of OAuth 2.0 that uses JSON Web Tokens (JWT) to transmit user information [11]. Unlike SAML, it is oriented toward modern web and mobile applications. The process works as follows: the user attempts to access a client application, which redirects them to an authorization server for authentication. After a successful login, the server returns an ID Token and an Access Token, which the application uses to access resources and retrieve user data. The key components include: the ID Token — a JWT containing information about the authenticated user; the Access Token — a token used for accessing resources; and the Authorization Server / Identity Provider — the entity responsible for authentication and token issuance.

Disadvantages of SSO:

- Single point of failure — an outage in the central IdP blocks access to all services;
- Compromise of a user account may result in an attacker gaining access to all integrated systems;
- Need for configuration consistency among all participants in the system.

Advantages of SSO:

- Reduction in the number of passwords that need to be remembered;
- Increased user productivity through faster access to resources;
- Centralized security control.

Use cases (SSO): Enterprise application suites: one login for ERP/CRM/HR/Email via a central IdP (e.g., Azure AD, Okta) using SAML/OIDC; SaaS federation: corporate SSO to Google Workspace, Microsoft 365, Salesforce, GitHub, etc., with corporate policies (MFA, conditional access); B2C "social login": customer portals allowing "Sign in with Google/Apple/Facebook" via OIDC; Education & campuses: students/staff access LMS, library, and labs through a campus IdP (eduGAIN/Shibboleth, SAML); Government e-services: citizen SSO across tax, healthcare, and licensing portals (often SAML with national eID); Healthcare: clinicians access EHR, PACS, and e-prescription systems through hospital IdP with enforced MFA; Partner/B2B federation: suppliers and contractors access selected apps via trust between organizations' IdPs (SAML/OIDC); Hybrid cloud & microservices: centralized identity for on-prem and cloud apps; service gateways validate tokens from the IdP.

2.5. MFA (Multi-Factor Authentication). Multi-Factor Authentication **-** is an authentication method that requires the use of at least two factors from different categories: something you know (password, PIN), something you have (token, smartphone), and something you are (biometric data). It significantly enhances account security by reducing the risk of compromise if one of the factors is stolen.

Key features: The key features of MFA involve using at least two of the three categories of factors [22]: something you know — a password, PIN code, or an answer to a security question; something you have — a physical token, smartphone, smart card, or USB key (e.g., YubiKey); and something you are — biometric data such as fingerprints, facial recognition, or iris scans. In some systems, this concept is extended with additional factors, such as verifying the user's location (geolocation) or restricting authorization based on time.

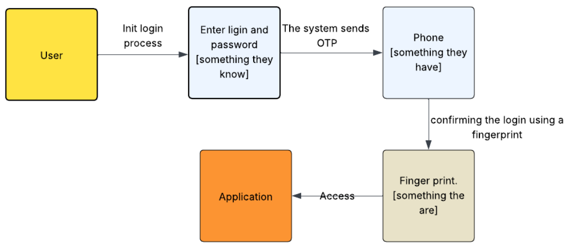
User
Init login process
Enter ligin and password [something they know]
The system sends OTP
Phone [something they have]
confirming the login using a fingerprint
Finger print. [something the are]
Access
Application

Fig. 6. MFA simplified schema

Example: A typical MFA login process may look like this: the user enters a username and password (something they know), then the system sends a one-time password (OTP) to their mobile phone (something they have), and the final step is confirming the login using a fingerprint (something they are).

Disadvantages:

- Increases the time and complexity of login for the user [22];
- Requires additional hardware or software applications;
- Potential access issues if one of the factors is lost (e.g., a phone).

Advantages:

- Significantly reduces the risk of unauthorized access in the event of a password leak [21];
- Enables adaptive verification (requiring an additional factor only in suspicious cases);
- Flexible factor selection for different scenarios.

Use cases: MFA is widely used in scenarios where a high level of security is required, such as: Online banking and financial services — to protect against unauthorized access to accounts and transactions;
365, Google Workspace, or AWS; Developer and administrative accounts — securing privileged access to production environments, servers, and code repositories (e.g., GitHub, GitLab); Healthcare systems — protecting patient data in compliance with regulations like HIPAA.

3. Comparative Analysis

This section presents a comparative analysis of the authentication and authorization technologies covered in the previous chapters. The table below evaluates each technology across several parameters: security level, performance, ease of integration, infrastructure requirements, user convenience, and typical use cases. This comparison provides a concise overview to help determine the most suitable approach for specific application scenarios.

Table 1 – **Comparison of the Main Authentication and Authorization Technologies**

Technology	*Security Level*	*Performance*	*Ease of Integration*	*Infrastructure Requirements*	*User Convenience*	*Typical Use Cases*
Basic Auth	Low (passwords are transmitted in plain text without TLS)	High	Very simple	Minimal (only an HTTP-enabled server is required)	Low (login/password must be entered each time or stored)	Prototypes, internal APIs, testing environments
OAuth 1.0	Medium (signed requests, but complex process)	Medium	Complex	Requires implementation of request-signing logic and key storage	Medium	Legacy integrations, old APIs
OAuth 2.0	High (with TLS, PKCE, short-lived tokens)	High	Medium	Requires an authorization server	High	Web and mobile applications, integration with Google, Microsoft
SSO*	High (depends on implementation — SAML, OIDC, Kerberos)	High	Medium/High	Requires IdP, integration configuration	Very high (single sign-on for multiple services)	Corporate portals, government and educational systems
MFA	Very high (multi-factor verification)	Medium	Medium	Requires support for additional factors (TOTP, SMS, hardware keys)	Medium (additional login step)	Banks, financial systems, administrative accounts

***Note:** SSO is implemented through protocols such as SAML 2.0, OpenID Connect, or Kerberos

Table 1 shows that each authentication and authorization technology offers a different balance between security, performance, ease of integration, infrastructure requirements, and user convenience. Basic Authentication, while the easiest to implement and highly performant, provides the lowest security level and is suitable only for internal or testing environments. OAuth 1.0 offers moderate security but is largely obsolete, remaining relevant only in legacy integrations. OAuth 2.0 has become the standard for modern web and mobile applications, delivering strong security and high convenience at the cost of requiring an authorization server and moderate integration effort. Single Sign-On (SSO) significantly improves user experience by enabling a single login for multiple services, though its security depends heavily on correct implementation and it requires a dedicated identity provider. Multi-Factor Authentication (MFA) delivers the highest security by adding verification layers but slightly reduces convenience due to extra login steps. Ultimately, no single approach is universally optimal—the choice depends on achieving the right balance between security, usability, and available infrastructure, tailored to the specific environment and operational needs.

3.1. Examples of Use in Corporate and Public Environments. Across corporate and public sectors, authentication and authorization technologies are adopted with varying priorities. In corporate ecosystems, security and centralized access control are paramount, leading to the predominance of modern protocols such as OAuth 2.0 and Single Sign-On (SSO), often combined with Multi-Factor Authentication (MFA) for administrative and high-privilege accounts. Legacy methods like Basic Authentication and OAuth 1.0 have been largely phased out, persisting only in niche scenarios such as isolated internal systems or legacy integrations where modernization is impractical.

In the public sphere, usability and interoperability with a wide range of devices and services play a decisive role. OAuth 2.0 with OpenID Connect has become the de facto standard for social logins and integrations with cloud platforms, while MFA is increasingly adopted by high-risk services such as online banking, payment platforms, and large-scale e-commerce or developer portals.

This divergence in adoption patterns highlights how the same technologies can fulfill different roles depending on the operational context, setting the stage for a comparative analysis of their security, performance, and implementation requirements.

3.2. Recommendations for Selecting a Technology. The choice of an authentication and authorization protocol depends on the type of environment, security requirements, and the scalability of the system.

Corporate Environments Recommended: SSO combined with MFA, integrated with corporate directories (e.g., Active Directory, LDAP). Rationale: Simplifies account management, enables centralized access control, and supports rapid scaling across a large number of users and applications. Implementation considerations: Requires a mature infrastructure, configuration of an Identity Provider (IdP), and clearly defined access policies.

Public Online Services Recommended: OAuth 2.0 with OpenID Connect, supplemented with MFA or passwordless approaches. Rationale: Ensures security when interacting with a large number of external clients and third-party applications, while reducing the risk of password storage on the service side. Implementation considerations: Adherence to RFC recommendations, proper token handling (protection against leaks, lifetime configuration), and use of PKCE for SPA and mobile applications.

Small-Scale and Internal Projects Recommended: Simplified methods such as Basic Auth over HTTPS or IBAC, in cases where risks are low and no scalability is anticipated. Rationale: Minimal development costs and rapid deployment. Implementation considerations: Ensure encryption of the communication channel and restrict access from external networks.

4. Challenges and Future Directions

Threats to Modern Authentication and Authorization Systems. Despite the evolution of security protocols and the enhancement of protection mechanisms, modern authentication and authorization systems remain vulnerable to a range of cyber threats that attackers actively exploit to gain unauthorized access.

Phishing — one of the most common threats, based on social engineering techniques. The user enters their credentials on a spoofed website or in a fake application, believing they are interacting with a legitimate resource. Even when data transmission is encrypted (HTTPS), phishing attacks remain effective if the user fails to detect the forgery.

Session hijacking — interception of an active user session by stealing the session identifier (session ID). This can occur through man-in-the-middle attacks, cross-site scripting (XSS), or unsecured communication channels. Once the session token is obtained, the attacker can act on behalf of the user without requiring re-authentication.

Token leakage — exposure of access or refresh tokens used in protocols such as OAuth 2.0 or OpenID Connect. This often results from improper storage of tokens on the client side (e.g., in the browser's localStorage) or their transmission in plain text within a URL.

Brute force and credential stuffing — automated attacks that attempt to guess passwords or use stolen username–password pairs from other services. The vulnerability is exacerbated when users reuse the same password across multiple systems.

Replay attacks — reusing an intercepted authentication request to gain access again. These attacks are particularly dangerous when systems do not validate the uniqueness and lifetime of the request.

These threats illustrate that the security of authentication and authorization systems depends not only on the chosen protocol or method but also on its proper implementation, regular updates to security policies, and a comprehensive approach to data protection.

4.1. Prospects for the Development of Authentication Technologies. The further evolution of authentication and authorization mechanisms is driven by the need to enhance security without compromising user convenience. Among the key trends are the adoption of the Zero Trust Architecture concept and the growing use of biometric and passwordless approaches.

Zero Trust Architecture (ZTA) is an approach that completely rejects the traditional "trust the internal network" model that has long dominated corporate security. The core principle of ZTA is to continuously verify the authenticity and privileges of every request, regardless of whether the traffic originates from inside or outside the network.

Key elements of ZTA include:

- Least privilege — each user or service is granted only the rights necessary to perform specific tasks.
- Network segmentation — dividing the infrastructure into isolated zones with separate access rules.
- Integration with MFA — requiring an additional authentication factor for critical operations.
- Continuous monitoring and behavioral analytics — detecting anomalies in user or service activities.

By applying these principles, ZTA reduces the impact of a single node or account compromise and increases the resilience of corporate networks to attacks.

Biometric Methods and Passwordless Approaches. One of the most promising directions is the transition to passwordless authentication, aimed at eliminating the vulnerabilities associated with weak, reused, or stolen passwords.

Core technologies include:

- FIDO2 and WebAuthn — open standards enabling the use of hardware keys, mobile devices, or biometrics instead of passwords.
- Biometrics — fingerprint, facial, or iris recognition, often combined with local data encryption in modern devices (e.g., Secure Enclave, Trusted Platform Module).
- Hybrid methods — combining biometrics with tokens or hardware keys to improve resilience against the compromise of a single factor.

Such solutions are already being adopted by large corporations, financial institutions, and government online services, which aim to reduce the risks of phishing and password-based attacks while ensuring fast and convenient access for users.

5. Discussion of results

Modern authentication and authorization systems face increasingly sophisticated threats, with phishing, session hijacking, token leakage, and automated credential attacks remaining among the most prominent. Effectively countering these challenges requires not only the improvement of existing protocols but also a strategic shift toward new approaches such as Zero Trust Architecture, which mandates continuous verification of every request, and the adoption of biometric and passwordless technologies capable of significantly reducing password-related risks. The combination of these strategies represents a promising direction for the evolution of digital security, one that addresses the challenges of the coming years while maintaining a balance between reliability and user convenience.

6. Conclusions

The analysis of modern authentication and authorization approaches has shown that the choice of a specific technology depends on a combination of factors, including the required level of security, performance requirements, integration complexity, and the specifics of the applications. Traditional methods, such as Basic Authentication and OAuth 1.0, are losing relevance due to their limited functionality and insufficient security. In contrast, OAuth 2.0 combined with Single Sign-On (SSO) provides an optimal balance between user convenience and security; however, its effective implementation requires:

- Proper handling of tokens (storage, renewal, invalidation);
- Correct processing of authorization errors;
- Secure redirect configuration;
- Adaptation of client-side logic to the specifics of the protocol (particularly in mobile and single-page applications).

The use of Multi-Factor Authentication (MFA) significantly enhances resilience against account compromise but requires additional changes to user interfaces and login processes to maintain convenience and compatibility across different devices.

The comparative analysis confirmed that there is no universal solution for all scenarios. The highest level of protection is achieved through a combination of several approaches: modern authorization protocols, multi-factor authentication, token encryption, and Zero Trust architecture principles. At the same time, the implementation of comprehensive solutions becomes more complex as the number of integrated components grows, requiring consistency between the frontend and backend, support for various client platforms, and proper interaction with external authorization services.

Future development prospects are associated with the widespread adoption of passwordless authentication and biometric technologies, which can minimize the risks of password theft or brute-force attacks. Furthermore, the large-scale implementation of Zero Trust Architecture will ensure more flexible and reliable access control in dynamic corporate environments but will require significant investment in infrastructure modernization.

Therefore, while no universal solution exists, the comparative analysis indicates that OAuth 2.0 combined with SSO is generally the most effective choice for large-scale and public-facing systems, MFA remains a critical layer for securing sensitive accounts, and Zero Trust principles provide the foundation for future-proof access control. The choice should always be adapted to the specific threat model, scale, and operational requirements of the environment.

References

1. NIST Special Publication 800-162 — Guide to Attribute Based Access Control (ABAC) Definition and Considerations (2014), *National Institute of Standards and Technology*, [online]. nist: https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-162.pdf
2. ISO/IEC 10181-3:1996 — Information technology — Open Systems Interconnection — Security frameworks for open systems: Access control framework (1996), *International Organization for Standardization*, [online]. iso: https://www.iso.org/standard/18396.html
3. NIST Special Publication 800-53 Rev. 5 — Security and Privacy Controls for Information Systems and Organizations (2020), *National Institute of Standards and Technology*, [online]. csrc: https://csrc.nist.gov/publications/detail/sp/800-53/rev-5/final
4. DNIST RBAC Standard — Role-Based Access Control (2004), *Information Technology Laboratory, National Institute of Standards and Technology*, [online]. nist: https://csrc.nist.gov/projects/role-based-access-control
5. RFC 4949 — Internet Security Glossary, Version 2 (2007), *Internet Engineering Task Force (IETF),* [online]. ietf: https://datatracker.ietf.org/doc/html/rfc4949
6. RFC 8176 — Authentication and Authorization for Constrained Environments (2017), *Internet Engineering Task Force (IETF),* [online]. ietf: https://datatracker.ietf.org/doc/html/rfc8176

7. Resnick, P. (2015), “The ‘Basic’ HTTP Authentication Scheme”, *RFC 7617, Internet Engineering Task Force (IETF),* [online]. ietf: https://datatracker.ietf.org/doc/html/rfc7617
8. Hammer-Lahav, E. (2010), “The OAuth 1.0 Protocol”, *RFC 5849, Internet Engineering Task Force (IETF)*, [online]. ietf: https://datatracker.ietf.org/doc/html/rfc5849
9. Hardt D. (2012), “The OAuth 2.0 Authorization Framework”, *RFC 6749, Internet Engineering Task Force (IETF*), [online]. Ietf: https://datatracker.ietf.org/doc/html/rfc6749
10. OASIS (2008), “Security Assertion Markup Language (SAML) V2.0 Technical Overview”, *OASIS Standard*, [online]. oasis: https://docs.oasis-open.org/security/saml/Post2.0/sstc-saml-tech-overview-2.0.html
11. OpenID Foundation (2014), “OpenID Connect Core 1.0”, [online]. openid: https://openid.net/specs/openid-connect-core-1_0.html
12. National Institute of Standards and Technology (NIST) (2017), “Digital Identity Guidelines: Authentication and Lifecycle Management”, *NIST Special Publication 800-63B*, [online]. nist: https://pages.nist.gov/800-63-3/sp800-63b.html
13. National Institute of Standards and Technology (NIST) (2020), “Recommendation for Pair-Wise Key Establishment Using Integer Factorization Cryptography”, *NIST Special Publication 800-175B*, [online]. nist: https://csrc.nist.gov/publications/detail/sp/800-175b/final
14. Jones, M., Hardt, D. and Recordon, D. (2012), “The OAuth 2.0 Authorization Framework: Bearer Token Usage,” *RFC 6750, Internet Engineering Task Force (IETF)*, [online]. ietf: https://datatracker.ietf.org/doc/html/rfc6750
15. Jones, M., Bradley, J. and Sakimura, N. (2015), “JSON Web Token (JWT)”, *RFC 7519, Internet Engineering Task Force (IETF)*, [online]. ietf: https://datatracker.ietf.org/doc/html/rfc7519
16. Hardt, D. (2012), “The OAuth 2.0 Authorization Framework”, *RFC 6749, Internet Engineering Task Force (IETF*), [online]. ietf: https://datatracker.ietf.org/doc/html/rfc6749
17. Sakimura, N., Bradley, J., Jones, M., Medeiros, B. and Mortimore, C. (2014), “OpenID Connect Core 1.0”, *The OpenID Foundation*, [online]. openid: https://openid.net/specs/openid-connect-core-1_0.html
18. Parecki, A. (2021), “OAuth 2.0 Simplified”, *2nd ed., Okta*, [online]. https://oauth2simplified.com/
19. Campbell, B., Bradley, J., Sakimura, N. (2015), “Proof Key for Code Exchange by OAuth Public Clients”, *RFC 7636, IETF*, [online]. ietf: https://datatracker.ietf.org/doc/html/rfc7636
20. OAuth.net, “Introduction to OAuth 2.0”, [online]. oauth: https://oauth.net/2/
21. National Institute of Standards and Technology (NIST) (2022), “Digital Identity Guidelines: Authentication and Lifecycle Management”, *NIST Special Publication 800-63B*, [online]. nist: https://pages.nist.gov/800-63-3/sp800-63b.html
22. CISA (2023), “Multi-Factor Authentication (MFA)”, *Cybersecurity and Infrastructure Security Agency*, [online]. cisa: https://www.cisa.gov/mfa
23. Microsoft (2023), “What is: Multifactor authentication”, *Microsoft Security Documentation*, [online]. microsoft: https://learn.microsoft.com/azure/active-directory/authentication/concept-mfa-howitworks
24. OWASP (2023), “Authentication Cheat Sheet”, *Open Worldwide Application Security Project*, [online]. owasp: https://cheatsheetseries.owasp.org/cheatsheets/Authentication_Cheat_Sheet.html
25. Google Cloud (2023), “Multi-Factor Authentication overview”, *Google Cloud Documentation*, [online]. google: https://cloud.google.com/architecture/identity/multi-factor-authentication-overview

Received (Надійшла) 12.08.2025
Accepted for publication (Прийнята до друку) 27.08.2025

ВІДОМОСТІ ПРО АВТОРІВ/ ABOUT THE AUTHORS

Василенко Остап Володимирович – спеціаліст за спеціальністю «Електронні прилади та системи», співробітник та розробник програмного забезпечення компанії Aimprosoft, Lviv, Ukraine;
Ostap Vasylenko — specialist in “Electronic Devices and Systems,” employee and software engineer at Aimprosoft, Львів, Україна;
e-mail: ostapvasylenko17@gmail.com; ORCID Author ID: https://orcid.org/0009-0006-1664-4937.

ПОРІВНЯЛЬНИЙ АНАЛІЗ СУЧАСНИХ ПРОТОКОЛІВ АВТЕНТИФІКАЦІЇ ТА АВТОРИЗАЦІЇ ДЛЯ ВЕБЗАСТОСУНКІВ

О.В. Василенко

Анотація. Актуальність. В умовах швидкої цифрової трансформації безпечний контроль доступу до інформаційних ресурсів став критично важливим. Механізми автентифікації та авторизації формують основу інформаційної безпеки, забезпечуючи перевірку ідентичності користувачів та призначення відповідних прав доступу. Розширення веб-додатків, хмарних сервісів та розподілених систем вимагає впровадження гнучких та стійких до атак протоколів. **Предметом дослідження** в даній статті є набір широко використовуваних протоколів автентифікації та авторизації, включаючи Basic Authentication, OAuth 1.0, OAuth 2.0, Single Sign-On (SSO) та Multi-Factor Authentication (MFA). **Метою статті** є порівняння цих технологій з точки зору безпеки, масштабованості, складності інтеграції та придатності для різних IT-середовищ. Дослідження базується на аналізі офіційних стандартів (RFC, рекомендації NIST, специфікації OASIS) та галузевих практик. **Результати** демонструють, що традиційні підходи, такі як Basic Auth та OAuth 1.0, застарівають через обмежену безпеку, тоді як OAuth 2.0 у поєднанні з SSO забезпечує баланс між зручністю та безпекою користувача. Впровадження багатофакторної автентифікації (MFA) значно покращує захист від компрометації облікового запису, але збільшує складність впровадження. **Висновок.** Жоден протокол не є універсальним рішенням; найвищий рівень безпеки досягається завдяки поєднанню сучасних протоколів авторизації, багатофакторної автентифікації, шифрування токенів та принципів нульової довіри. Ці висновки можуть бути застосовані розробниками, системними архітекторами та фахівцями з кібербезпеки для проектування та впровадження надійних систем автентифікації та авторизації, здатних вирішувати поточні та нові виклики цифровій безпеці.

Ключові слова: автентифікація; авторизація; OAuth; Basic Auth; Single Sign-On; Multi-Factor Authentication; цифрова безпека.