## SLIDE 1 Authentication methods and protocols in web applications

Today we will look at the use of various authentication methods for web applications, including password authentication, certificate authentication, one-time password authentication, access keys, and token authentication.

## SLIDE 2 Terminology

Let’s take a look on some basic terms.

Identification — it is a statement about who you are. Depending on the situation, this could be a name, email address, account number, etc.

Authentication — providing evidence that you really are who you identified yourself with

Authorization — checking that you are allowed to access the requested resource.

For example, if you try to enter a private club, you will be identified (asked for your first and last name), authenticated (asked to see your passport and photo checked), and authorized (verified that your last name is on the guest list) before being allowed inside.  
  
Similarly, these terms are used in computer systems, where traditionally, identification means getting your account (identity) by username or email; under authentication - verification that you know the password for this account, and under authorization - verification of your role in the system and the decision to grant access to the requested page or resource.

However, in modern systems, there are more complex authentication and authorization schemes.

## SLIDE 3 Types of authentication

1. Authentication by password

This method relies on the fact that the user must provide a username and password in order to successfully identify and authenticate with the system. The username/password pair is specified by the user when registering in the system, while the username can be the user's e-mail address.  
  
In relation to web applications, there are several standard protocols for password authentication, which we will discuss below.

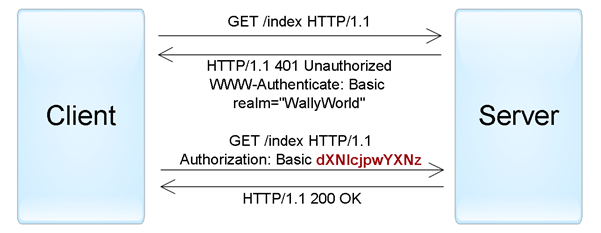
1.1 HTTP authentication **http-authentication**

1. 1. The server, when an unauthorized client accesses a protected resource, sends the HTTP status “401 Unauthorized” and adds the “WWW-Authenticate” header indicating the authentication scheme and parameters.
2. 2. The browser, upon receiving such a response, automatically displays the username and password input dialog. The user enters their account details.
3. 3. In all subsequent requests to this website, the browser automatically adds the HTTP header “Authorization”, which transmits user data for authentication by the server.
4. The server authenticates the user according to the data from this header. The decision to grant access (authorization) is made separately based on the user's role, ACL, or other account data.

## SLIDE 4 **http-authentication-schemes**

The whole process is standardized and well supported by all browsers and web servers. There are several authentication schemes that differ in the level of security:

**Basic** — the simplest scheme, in which the username and password of the user are passed in the Authorization header in unencrypted form (base64-encoded). However, when using the HTTPS (HTTP over SSL) protocol, it is relatively secure.



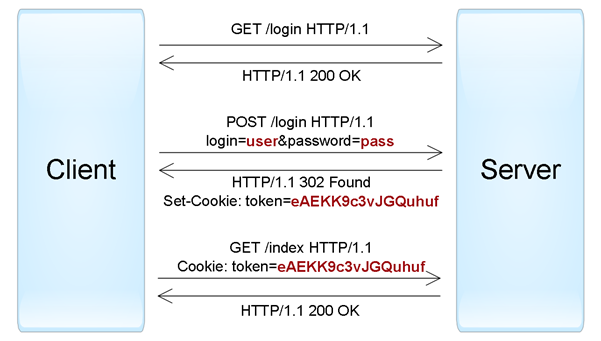
**Digest** — challenge-response- a scheme where the server sends a unique nonce and the browser sends an MD5 hash of the user's password computed using the specified nonce. A more secure alternative to the Basic scheme for insecure connections, but subject to man-in-the-middle attacks (with the scheme replaced by basic). In addition, using this scheme does not allow using modern hash functions to store user passwords on the server.

**NTLM** (known as Windows authentication) — is also based on the challenge-response approach, where the password is not passed in its pure form. This scheme is not an HTTP standard, but is supported by most browsers and web servers. Primarily used to authenticate Windows Active Directory users in web applications. Vulnerable to pass-the-hash attacks.

**Negotiate** — another scheme from the Windows authentication family that allows the client to choose between NTLM and Kerberos authentication. Kerberos is a more secure protocol based on the Single Sign-On principle. However, it can only function if both the client and the server are in the intranet zone and are part of a Windows domain.

It is worth noting that when using HTTP authentication, the user has no standard option to log out of the web application other than to close all browser windows.

## SLIDE 5 Forms authentication **forms-authentication**

There is no specific standard for this protocol, so all its implementations are specific to current systems, or rather, to authentication modules of development frameworks.  
  
It works according to the following principle: an HTML form is included in the web application, in which the user must enter their username / password and send them to the server via HTTP POST for authentication. If successful, the web application creates a session token, which is usually placed in browser cookies. On subsequent web requests, the session token is automatically passed to the server and allows the application to obtain information about the current user to authorize the request.

## SLIDE 6 Other password authentication protocols **id="other-password-authentication"**

The two protocols described above have been successfully used to authenticate users to websites. But when developing client-server applications using web services (for example, iOS or Android), along with HTTP authentication, non-standard protocols are often used in which authentication data is transmitted in other parts of the request.

There are only a few places where you can pass username and password in HTTP requests:

**URL query** — considered unsafe as URL strings can be remembered by browsers, proxies and web servers.

**Request body** — safe option, but only applicable for requests containing a message body (such as POST, PUT, PATCH).

**HTTP header** — the best option, in this case, both the standard Authorization header (for example, with the Basic schema) and other arbitrary headers can be used.

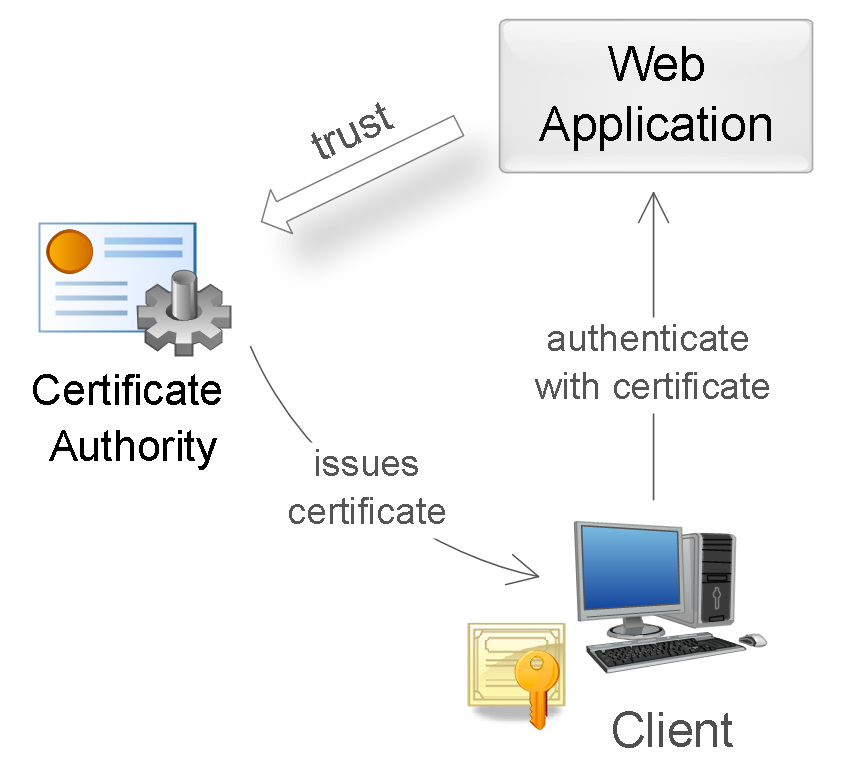
Password authentication is not considered a very reliable method, since the password can often be guessed, and users tend to use simple and identical passwords in different systems, or write them down on scraps of paper. If an attacker was able to figure out the password, then the user often does not know about it. In addition, application developers can make a number of conceptual errors that make it easier to hack accounts.

## SLIDE 7 Authentication certificate

A certificate is a set of attributes that identify the owner, signed by a certificate authority (CA). The CA acts as an intermediary that guarantees the authenticity of certificates (similar to the FMS issuing passports). Also, the certificate is cryptographically associated with a private key, which is stored by the owner of the certificate and allows you to unambiguously confirm the fact of ownership of the certificate.

On the client side, the certificate together with the private key can be stored in the operating system, in the browser, in a file, on a separate physical device (smart card, USB token). Typically, the private key is additionally protected by a password or PIN.

Web applications traditionally use X.509 certificates. Authentication using an X.509 certificate occurs at the time of connection to the server and is part of the SSL/TLS protocol. This mechanism is also well supported by browsers, which allow the user to select and apply a certificate if the website allows this method of authentication.



~~During authentication, the server performs certificate validation based on the following rules:~~

* ~~Сертификат должен быть подписан доверенным certification authority (проверка цепочки сертификатов).~~
* ~~Сертификат должен быть действительным на текущую дату (проверка срока действия).~~
* ~~Сертификат не должен быть отозван соответствующим CA (проверка списков исключения).~~

## SLIDE 8 One Time Password Authentication

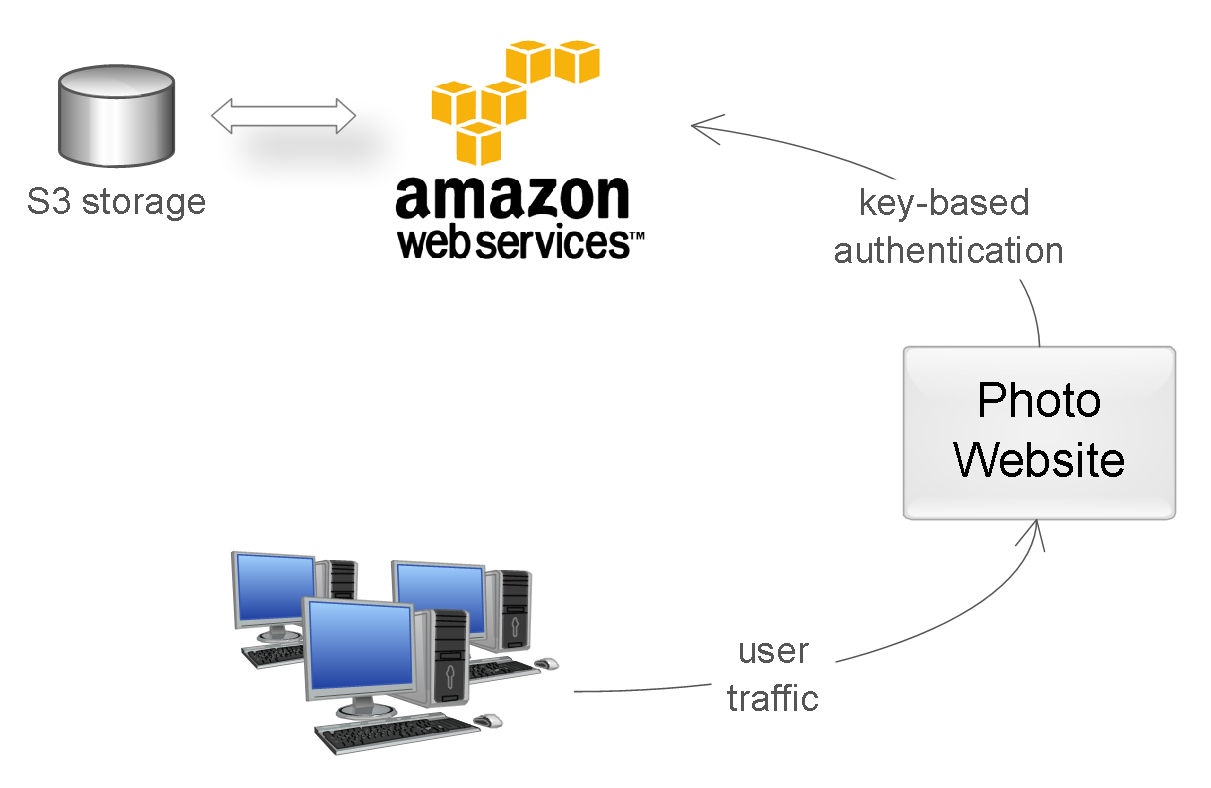
One-time password authentication is usually used in addition to password authentication to implement two-factor authentication (2FA). In this concept, the user needs to provide two types of login data: something he knows (eg a password) and something he owns (eg a device for generating one-time passwords). The presence of two factors can significantly increase the level of security, which can be required for certain types of web applications.

1. Hardware or software tokens that can generate one-time passwords based on the secret key entered into them and the current time. Users' secret keys, which are a factor of ownership, are also stored on the server, which allows verification of the entered one-time passwords. An example of hardware implementations of tokens is RSA SecurID; software - Google Authenticator application.

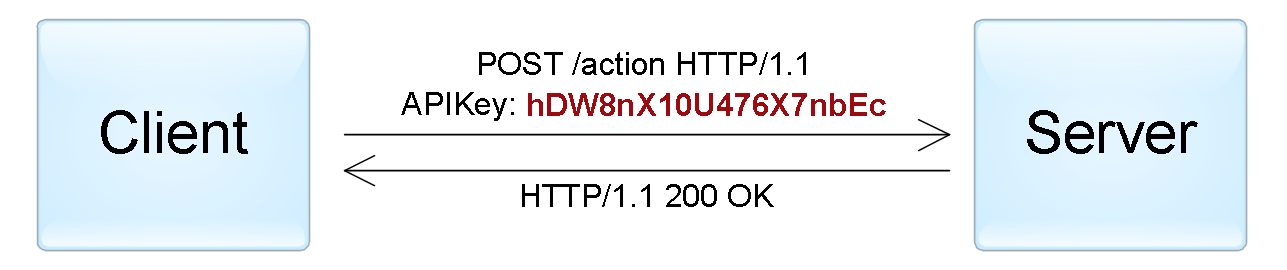
2.Randomly generated codes transmitted to the user via SMS or other communication channel. In this situation, the ownership factor is the user's phone (more precisely, a SIM card tied to a specific number).

3.Printing or scratch card with a list of pre-generated one-time passwords. For each new login, you need to enter a new one-time password with the specified number.

## SLIDE 9 Access Key Authentication



This method is most commonly used to authenticate devices, services, or other applications when accessing web services. Here, access keys (API keys) are used as a secret - long unique strings containing an arbitrary set of characters, essentially replacing the username / password combination.

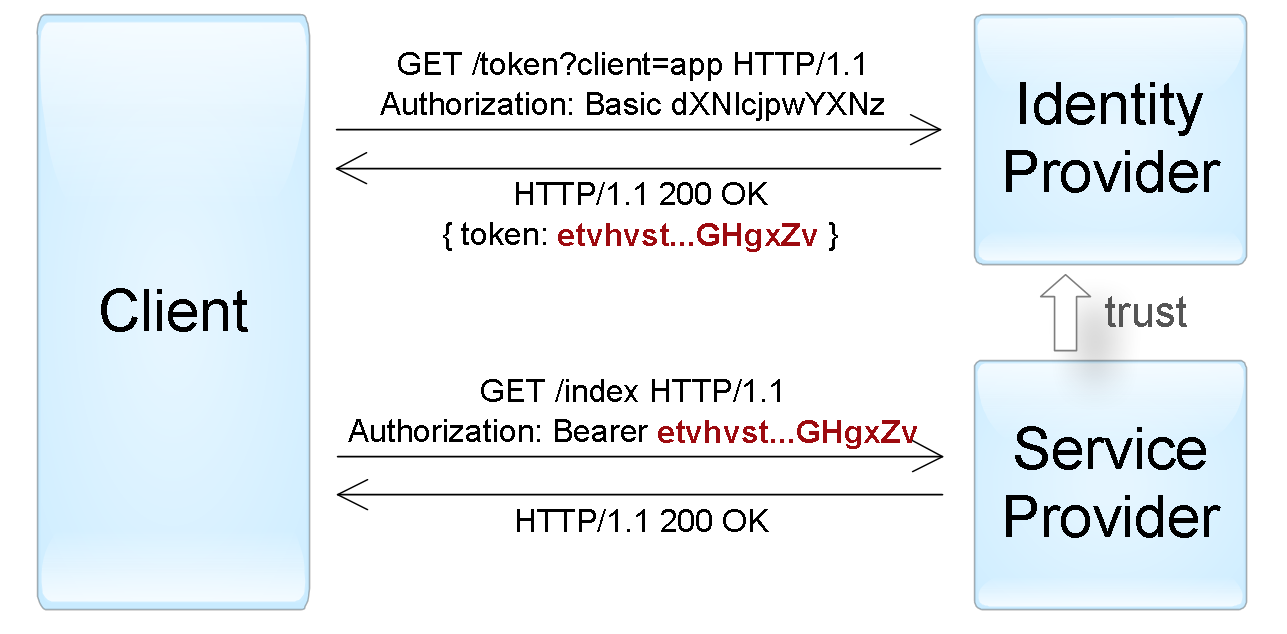


An example of authentication using an access key passed in the HTTP header.

## SLIDE 10 Token authentication

This authentication method is most often used when building distributed Single Sign-On (SSO) systems, where one application (service provider or relying party) delegates the user authentication function to another application (identity provider or authentication service). A typical example of this method is logging into an app with a social media account. Here, social networks are authentication services, and the application trusts the user authentication function to social networks.

The implementation of this method is that the identity provider (IP) provides reliable information about the user in the form of a token, and the service provider (SP) application uses this token to identify, authenticate and authorize the user. On a general level, the whole process looks like this:

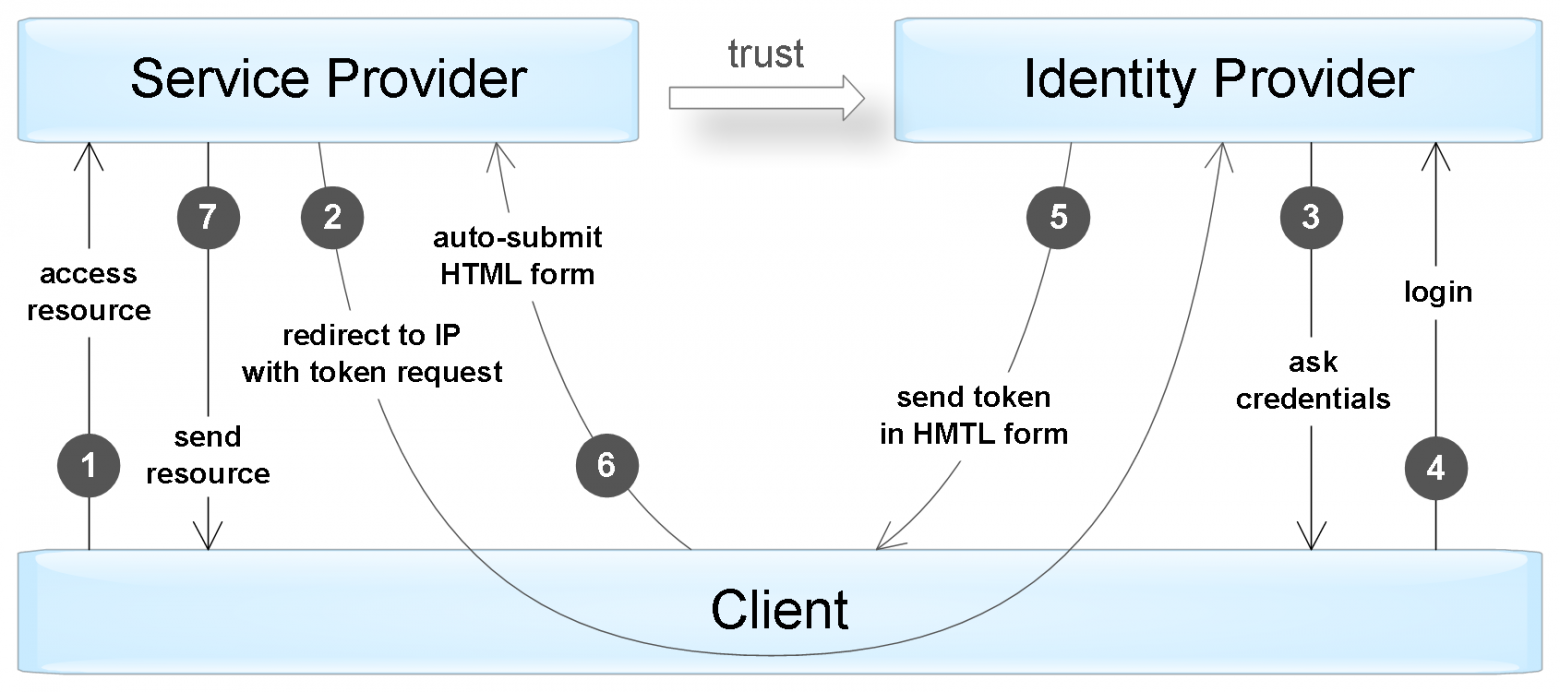


The client authenticates to the identity provider in one of the ways specific to it (password, access key, certificate, Kerberos, etc.).

The client asks the identity provider to provide it with a token for a specific SP application. The identity provider generates a token and sends it to the client.

The client authenticates to the SP application using this token.

The process described above reflects the mechanism for authenticating an active client, i.e. one that can perform a programmed sequence of actions (for example, iOS / Android applications). The browser, on the other hand, is a passive client in the sense that it can only display the pages requested by the user. In this case, authentication is achieved by automatically redirecting the browser between the identity provider and service provider web applications.



## SLIDE 12

There are several standards that define exactly the protocol for interaction between clients (active and passive) and IP / SP applications and the format of supported tokens. Among the most popular standards are OAuth, OpenID Connect, SAML, and WS-Federation.

The token itself is usually a data structure that contains information about who generated the token, who can be the recipient of the token, the expiration date, a set of information about the user himself (claims). In addition, the token is additionally signed to prevent unauthorized changes and guarantee authenticity.

When authenticating with a token, an SP application must perform the following checks:

1. The token was issued by a trusted identity provider application (checking the **issuer** field).
2. The token is assigned to the current SP application (**audience** field check).
3. The token has not yet expired (checking the **expiration** date field).
4. The token is genuine and has not been modified (signature verification).

If validation is successful, the SP application authorizes the request based on the user data contained in the token.

## SLIDE 13 Token formats

1. Simple Web Token (SWT) - the simplest format, which is a set of arbitrary name / value pairs in the HTML form encoding format. The standard defines several reserved names: Issuer, Audience, ExpiresOn, and HMACSHA256. The token is signed with a symmetric key, so both IP and SP applications must have this key to be able to create/verify the token.
2. JSON Web Token (JWT) — contains three blocks separated by dots: a title, a set of fields (claims) and a signature. The first two blocks are in JSON format and additionally encoded in base64 format. The field set contains arbitrary name/value pairs, and the JWT standard defines several reserved names (iss, aud, exp, and others). The signature can be generated using both symmetric encryption algorithms and asymmetric ones. In addition, there is a separate standard that unsubscribes the format of an encrypted JWT token.
3. Security Assertion Markup Language (SAML) — defines tokens (SAML assertions) in XML format, including information about the issuer, about the subject, the necessary conditions for checking the token, a set of additional statements (statements) about the user. SAML tokens are signed using asymmetric cryptography. In addition, unlike previous formats, SAML tokens contain a mechanism to prove ownership of the token, which prevents tokens from being intercepted through man-in-the-middle attacks when using insecure connections.

## SLIDE 14 Conclusion

|  |  |  |
| --- | --- | --- |
| **Method** | **Main application** | **Protokols** |
| **By password** | User authentication | HTTP, Forms |
| **По сертификатам** | User authentication in secure applications; service authentication | SSL/TLS |
| **With one-time passwords** | Additional user authentication (to achieve two-factor authentication) | Forms |
| **By access keys** | Service and Application Authentication | - |
| **By token** | Delegated user authentication; delegated application authorization | SAML, WS-Federation, OAuth, OpenID Connect |

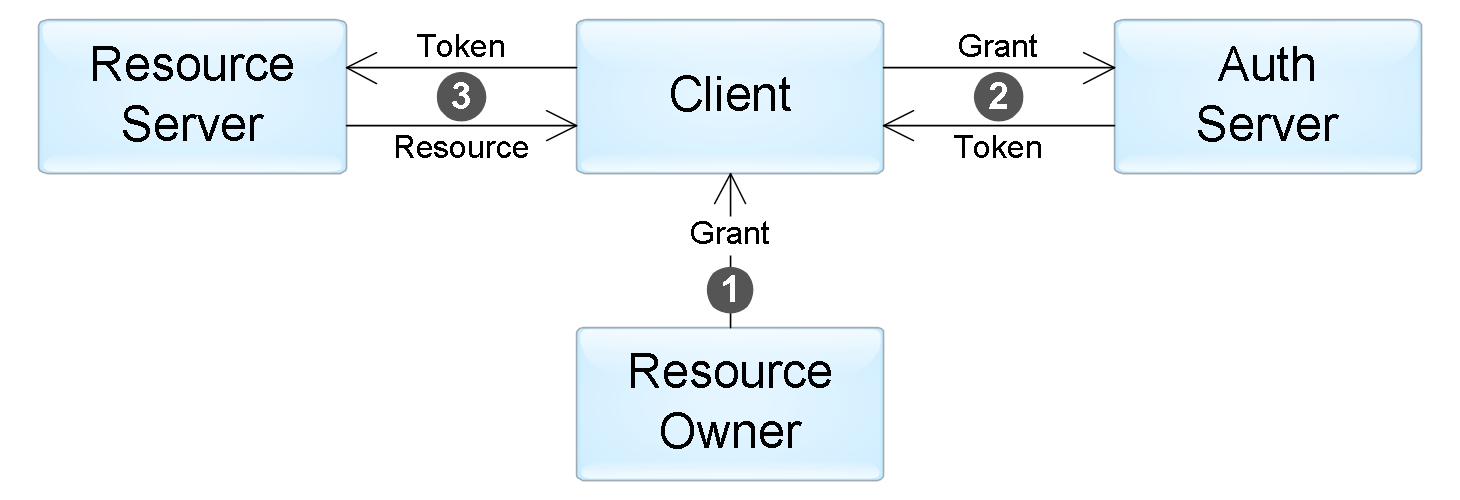
## SLIDE 14 OAuth and OpenID Connect standards

Unlike SAML and WS-Federation, the OAuth (Open Authorization) standard does not describe a user authentication protocol. Instead, it defines a mechanism for one application to gain access to another on behalf of a user. However, there are schemes that allow user authentication based on this standard (more on that below).

In general, the whole process consists of several steps:

1. The user (resource owner) gives permission to the application (client) to access a specific resource in the form of a grant. What is a grant, we will consider a little lower.
2. The application contacts the authorization server and receives a resource access token in exchange for its grant. In our example, the authorization server is Google. When called, the application is additionally authenticated using the access key given to it during pre-registration.
3. The application uses this token to obtain the required data from the resource server (in our case, the Gmail service).

## SLIDE 15 Взаимодействие компонентов в стандарте OAuth.



## SLIDE 16 Types of grants

The standard describes four types of grants that define possible application scenarios:

**Authorization Code** — the user can receive this grant from the authorization server after successful authentication and confirmation of consent to grant access. This method is most commonly used in web applications. The grant process is very similar to the passive client authentication mechanism in SAML and WS-Federation.

**Implicit** — used when the application has no way to securely obtain a token from the authorization server (for example, a JavaScript application in a browser). In this case, the grant is a token received from the authorization server, and step #2 is excluded from the script above.

**Resource Owner Password Credentials** — used when the application has no way to securely obtain a token from the authorization server (for example, a JavaScript application in a browser). In this case, the grant is a token received from the authorization server, and step #2 is excluded from the script above.

**Client Credentials** - in this case, there is no user, and the application gets access to its resources using its access keys (step number 1 is excluded).

The standard does not define the format of the token that the application receives: in the scenarios addressed by the standard, the application does not need to parse the token, since it is only used to access resources. Therefore, neither the token nor the grant itself can be used to authenticate the user. However, if the application needs to get reliable information about the user, there are several ways to do this:

1. Often, the resource server API includes an operation that provides information about the user itself (for example, /me in the Facebook API). The application can perform this operation every time it receives a token to identify the client. This method is sometimes called pseudo-authentication.
2. Use the OpenID Connect standard designed as a credential layer on top of OAuth (published 2014). In accordance with this standard, the authorization server provides an additional identity token in step #2. This JWT token will contain a set of specific fields (claims) with information about the user.