Post-Quantum Electronic Identity: Adapting OpenID Connect and OAuth 2.0 to the Post-Quantum Era

Frederico Schardong[1,2(✉)], Alexandre Augusto Giron[2,3], Fernanda Larisa Müller[2], and Ricardo Custódio[2]

[1] Federal Institute of Rio Grande do Sul, Rolante, Brazil
frederico.schardong@rolante.ifrs.edu.br
[2] Federal University of Santa Catarina, Florianópolis, Brazil
muller.larissa@grad.ufsc.br, ricardo.custodio@ufsc.br
[3] Federal University of Technology-Parana, Toledo, Brazil
alexandregiron@utfpr.edu.br

Abstract. The quantum threat affects a multitude of network protocols, frameworks, and systems that use public-key cryptography. OpenID Connect (OIDC) and OAuth 2.0 are no exception, which means they must transition to Post-Quantum Cryptography (PQC). However, the long lifetime of access tokens necessitates this shift, if we consider the possibility of a *record-now-decrypt-later* attacker retrieving and subsequently impersonating users with these tokens. In this research, we conduct a thorough literature review to identify existing solutions to this problem, suggest modifications to OAuth 2.0 and OIDC and their underlying protocols to make them quantum-safe, then implement and evaluate the effects of PQC on a realistic OIDC study case. Our findings reveal that PQC-based OIDC has the same or better performance in low-latency settings, but the handshake of PQC-based TLS accounts for fifty percent of the overall duration in high-latency scenarios.

Keywords: Post-Quantum Cryptography · PQC · OpenID Connect · OAuth · OAuth 2.0 · Electronic Identity · Identity.

1 Introduction

The OAuth 2.0 protocol introduced standard flows for a client (such as a web application or mobile app) to interact with an Identity Provider (IdP) and obtain *authorization* to access electronic IDentity (eID) attributes or perform actions on behalf of an eID [6]. However, due to its emphasis on authorization rather than authentication, crucial aspects of eID were omitted. For example, there is no regulated method for requesting common eID attributes such as name and

This study was supported by the Federal Institute of Rio Grande do Sul (IFRS) and by the Federal University of Technology - Parana (UTFPR).

A. R. Beresford et al. (Eds.): CANS 2022, LNCS 13641, pp. 371–390, 2022.
https://doi.org/10.1007/978-3-031-20974-1_20

email address, nor for requesting two-factor authentication. The OpenID Connect (OIDC) protocol was developed to address these issues [27]. In practice, it is an extension of OAuth 2.0 that introduces uniformity for common attributes (*e.g.*, name, email, address, phone number, etc.) and permits Service Providers (SPs) to request IdPs to enforce multi-factor authentication, among other features. OAuth 2.0 and OIDC are the current *de facto* standards for working with eIDs. They are utilized daily by billions of users via industry-leading IdPs like Google and Facebook (now Meta).

However, these users are susceptible to quantum attackers disclosing their private information or impersonating them. Although there is no known quantum computer with sufficient processing power to break the cryptographic protocols underlying OAuth 2.0 and OIDC, intercepted communications can be stored for when such devices become available. Specifically, a *record-now-decrypt-later* attack could be used to retrieve access tokens from the past, enabling impersonation in the future. Consequently, these protocols (and their implementations) must be updated immediately to counteract this threat.

Quantum threats target applications and network protocols reliant on classical cryptography. By classical cryptography, we mean public-key schemes based on the Integer Factorization Problem (IFP), the Discrete Logarithm Problem (DLP), and the Elliptic Curve Discrete Logarithm Problem (ECDLP). Several network protocols, including JSON Web Token (JWT) and TLS, employ these schemes, which are vulnerable to Shor's algorithm [31]. Under Grover's algorithm [5], symmetric-key schemes such as AES are also threatened by quantum computing. If their security settings can be increased, they won't need to be replaced once their security is cut in half. Maintaining the initial guarantees only requires doubling the security parameters.

In this context, classical public-key schemes will likely be replaced in the future by Post-Quantum Cryptography (PQC) or quantum-safe cryptography [21]. PQC algorithms are based on a variety of mathematical problems believed to be intractable by both non-quantum and quantum adversaries. The alternative PQC solutions are based on lattice-based cryptography, multivariate cryptography, code-based cryptography, hash-based cryptography, or isogeny-based cryptography. In this context, a global-scale transition to PQC is expected in the coming years, with systems, protocols, and applications in general beginning to use new cryptographic schemes. We anticipate that the same transition will occur for OAuth 2.0 and OIDC. One challenging part of this transition to PQC is the increased size, which can impose delays in the protocol. Besides, they can incur in other performance drawbacks (*e.g.*, increased computational time).

In this research, we analyze the essential components of OAuth 2.0 and OIDC, suggesting and empirically evaluating modifications to protect them from quantum attackers. In conclusion, our contributions are as follows:

- We conduct a rigorous and reproducible systematic literature review to determine the current state of post-quantum OIDC and OAuth 2.0;
- We propose improvements to OAuth 2.0 and OIDC, as well as their underlying protocols TLS and JWT, to make them quantum-safe;

– We provide a post-quantum implementation of OpenID Connect, built with the integration of PQC algorithms in JWT and also TLS, as well increasing parameters in symmetric primitives; and
– We conduct a series of reproducible experiments using a realistic scenario and multiple configurations to evaluate the performance of our proposal.

The rest of this article is structured as follows. Section 2 goes over OAuth 2.0, OIDC, TLS, and PQC. The rigorous systematic literature review conducted to discover and evaluate existing approaches is then presented in Sect. 3. Section 4 proposes changes to these protocols to make them more resistant to quantum attackers, and Sect. 5 introduces our evaluation strategy and the results obtained. Section 6 concludes with closing remarks.

2 Background

2.1 OAuth 2.0

The OAuth 2.0 protocol was introduced in 2012 to allow third-party applications to access data and perform actions on behalf of users [6]. It is based on the distribution of tokens and establishes four roles: (i) the end-user is the *resource owner*; (ii) the *client* is the application that requests the resource owner's data; (iii) the *authorization server*, *i.e.*, the IdP, grants *access tokens* for a client to access the resource owner's data after authenticating the resource owner and obtaining their authorization; and (iv) the *resource server* hosts the protected resources and responds to requests.

The protocol provides three standardized flows for clients to obtain resource owner data, one of which is intended to facilitate the transition from basic and digest authentication schemes, as well as other schemes, to the two major flows. The *authorization code grant*, in which tokens are exchanged between the Relying Party (RP) and the IdP over a secure back channel, and the *implicit grant*, in which all interactions occur through the user's browser, are the two flows in question. Effectively, only the *authorization code grant* should be used, as the *implicit grant* is known to have security flaws, such as the impossibility to detect replay attacks [19]. The *authorization code grant* is depicted in Fig. 1, where red text indicates OAuth's optional parameters that are mandatory on OIDC and bold red text indicates new mandatory values added by OIDC.

First, the user requests a protected resource from the RP via their user agent or attempts to access a restricted area. If there is no cookie or other storage mechanism storing a session identifier proving that the end user has previously authenticated and authorized this client to access their data, the *authorization code grant* process begins. Users are frequently presented with one or more IdPs in which the RP is registered and asked to select one.

Second, the RP directs the unknown user to the TLS-secured authorization endpoint of the selected IdP. The protocol does not specify how the RP finds the authorization endpoint. This usually occurs during the client registration

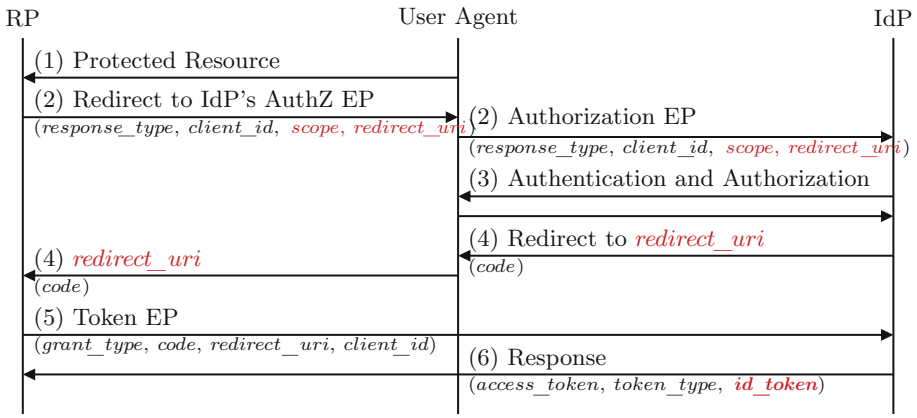

Fig. 1. OAuth 2.0's *authorization code grant* flow. Dark-colored values indicate mandatory parameters, while red-colored values are optional values made mandatory by OIDC. The bold red *id_token* is a new, mandatory OIDC value. (Color figure online)

process in the IdP, which is also out of scope. The user agent performs the redirect, which includes two mandatory parameters: `response_type`, which contains the value `token` indicating that this is the *authorization code grant* flow, and `client_id`, which contains the RP's identification. In OAuth 2.0, the `scope` and `redirect_uri` are optional parameters, whereas they are required in OIDC, as will be explained later. The `scope` parameter specifies which personal information the RP requests from the IdP. For example, the user's e-mail and postal addresses, or permission to perform an action on the user's behalf, such as updating an attribute. The `redirect_uri` parameter specifies the RP-controlled URL to which the user agent will be redirected after authentication and authorization. These parameters are optional, and if they are not specified, the values established during the client registration process are used.

Third, after receiving the request, the IdP authenticates the user and obtains their permission to access the requested resource. The protocol does not specify how users are authenticated or granted authorization. Similarly, available scopes are defined by the IdP and communicated to the RP either during the client registration process or via another mechanism. Authentication can be as simple as a username and password, or as sophisticated and secure as multi-factor authentication. This is not covered by OAuth 2.0.

Fourth, if the user is successfully authenticated and grants authorization, the IdP issues a short lived *authorization code*. The `code` parameter of the redirect response is used to send this to the RP via the user agent. If an error occurs, the IdP notifies the user and uses the `error` parameter instead of `code`.

Fifth, the client makes a TLS-secured token endpoint request. Four parameters are required: (i) `grant_type` is `authorization_code`; (ii) `code` is the authorization code from the previous step; (iii) `redirect_uri` is the same as in the second step or is omitted; and (iv) `client_id` is the same as in the second step if

no client authentication is performed. It is important to note that in OAuth 2.0, web applications are considered *confidential clients* and thus require some form of authentication to access the token endpoint. Although the protocol does not specify an authentication mechanism for clients, the OAuth 2.0 Security Best Current Practice [19], an RFC that addresses OAuth 2.0 security, recommends a challenge-response protocol specifically designed to authenticate clients [25].

In the sixth and final step, the IdP verifies that the `authorization_code` received was issued to the authenticated client and that the `redirect_uri` matches what was provided in the second step, if previously informed. An `access_token` is then issued along with a `token_type`, which instructs the RP on how to make requests to the user-authorized protected resources. In OIDC, an `id_token` is also issued. After receiving the `access_token`, the client can use it to request the protected resources from the resource server.

2.2 OpenID Connect

Even though OAuth 2.0 standardizes roles, flows, and messages, practitioners must still fill in numerous gaps. It does not, for example, address client registration, indicate the authentication factors used, or create a mechanism for clients to discover IdP endpoint URIs. OpenID Connect was presented as a solution for these and other issues [27].

OIDC adds an identity layer to OAuth 2.0. This is accomplished in a non-destructive manner by leveraging OAuth 2.0 optional parameters and the requirement to ignore unrecognized request and response parameters to/from the token and authorization endpoints. The modifications to the OAuth 2.0 three-party handshake are depicted in red in Fig. 1 and described below.

First, the `scope` parameter must contain the value `openid`. This is possible due to the fact that OAuth 2.0 permits multiple values separated by spaces for this parameter. OIDC defines four additional scopes that the RP can use to request user-related information: (i) `profile`, which requests the user's `name`, `nickname`, `picture`, `website`, `gender`, `birthdate`, and other personal data; (ii) `email` requests access to `email` address and `email_verified`, a boolean value indicating whether or not it has been verified; (iii) `phone` asks for `phone_number` and `phone_number_verified`; and (iv) `address`.

It is worth noting that in OIDC, user-related data is referred to as claims. A claim is a piece of information asserted about the user by the IdP or other attribute provider. The user claims requested by the client are accessed via the `userinfo_endpoint`, which receives a valid `access_token` and returns the claims this token is authorized to access.

Second, in OIDC, the `redirect_uri` parameter is required and must match the URI specified in the client registration. The *OIDC core* specification, like OAuth 2.0, assumes the RP is registered and knows the IdP's communication endpoints. These two challenges are addressed by two additional specifications: *OpenID Connect Dynamic Client Registration* (OIDC-DCR) [26] and *OpenID Connect Discovery* (OIDC-D) [28]. These two specifications, along with the *OIDC core*, make up what is collectively referred to as the *OIDC dynamic*. In the

OIDC-DCR specification, a client sends one or more `redirect_uri` and other optional parameters and receives a `client_id` and optionally a `client_secret`. The OIDC-D protocol defines the URI `/.well-known/openid-configuration`, which clients use to discover the endpoints and algorithms supported by an IdP.

The final and most significant modification to the OAuth 2.0 three-party handshake occurs in the response of the token request: OIDC adds a required response argument named `id_token`. This token contains information regarding the user's authentication, including the token's issuer, the intended audience (one or more `client_id`), a unique identifier of the end-user in the IdP, the date and time of the authentication, and the token's issued and expiration times. This token may also include optional information such as the *authentication context class*, which indicates the level of assurance of the authentication process [9], and an array of IdP-defined *authentication method reference*, such as password, PIN, and facial recognition.

2.3 OAuth 2.0 and OIDC Security Considerations

Known threats to these two protocols include resource owner password guessing, token fabrication, Cross-Site Request Forgery (CSRF), eavesdropping access tokens, and client impersonation of resource owner [20]. Let us roughly divide the attack surfaces into two categories: communication and stationary. To mitigate communication-related threats, both OAuth 2.0 and OIDC require the use of Transport Layer Security (TLS) [24] to authenticate RP and IdP and to ensure the integrity and confidentiality of exchanged messages [6,27]. Regarding stationary data protection, however, the original OAuth 2.0 specification does not specify how tokens (`refresh_token` and `access_token`) should be constructed; it only states that other parties should be unable to generate, modify, or guess them [6]. The OIDC protocol, on the other hand, requires the use of JWT [13] to implement the `id_token` [27].

The JWT is a JSON-based data structure consisting of a header, content, and signature. The header describes the cryptographic operations performed on the content: none, encryption, signature, or both. The JWT-based `id_token` must be signed. The IdP makes the public keys for signature verification available in a publicly accessible endpoint called `jwks_uri`, the value of which can be found using OIDC-D [28]. This URI returns a *JSON Web Key (JWK)* [11], a JSON structure containing the public key, key id, and other key-related data. It is worth noting that there is an OAuth 2.0 equivalent to the OIDC-D specification, namely the *OAuth 2.0 Authorization Server Metadata* [16], as well as a specification that standardizes the use of JWT for OAuth 2.0, namely the *JSON Web Token (JWT) Profile for OAuth 2.0 Access Tokens* [4].

2.4 Transport Layer Security 1.3

To exchange protocol messages, OpenID Connect and OAuth 2.0 require a secure and authenticated communication channel. Both standards require the use of TLS for communication security. The TLS protocol (version 1.3) is defined on

RFC 8446 [24] and is often used in this scenario. TLS consists of three components: a Handshake protocol, where an Authenticated Key Exchange (AKE) takes place; a Record protocol specifies symmetric encryption for the communication; and an Alert Protocol, which specifies error messages and conditions.

TLS Handshake. An AKE is performed in every TLS 1.3 full handshake. The parties exchange a shared secret that is further derived into a set of traffic keys, later used in the Record Protocol for encrypting application data. During the handshake, the TLS server authenticates itself to the TLS client. A x509 digital certificate is often used, but authentication with Pre-Shared-Key (PSK) is also supported. The server can, optionally, request client authentication. We describe the handshake in three steps:

1. The client send the first handshake message: `ClientHello`, among with optional extensions. Normally it sends a random nonce, protocol versions, list of supported ciphersuites, among other information. For forward secrecy, a `keyshare` is sent composed by the Elliptic-Curve Diffie-Hellman Ephemeral (`ECDHE`) public part. Although optional, at least a keyshare or a Pre-shared-Key message must be sent.
2. The server replies with negotiated parameters, such as the selected ciphersuite, and additional (optional) messages. A certificate and certificate verify message is often present part of TLS server authentication. The first is the digital certificate and the second is a signature for key confirmation. An alternative is the server authentication through the PSK, *e.g.* in a resumption session, where the certificate is not sent by the server. The authentication ends with the `finished` message, where a HMAC is computed from the transcript of the handshake context. At this point, the server can send encrypted application data to the client.
3. Upon reception, the client can complete the `ECDHE` KEX and derive symmetric keys for the communication. The handshake is completed with the client's `finished` message.

TLS Record Protocol. After the authentication and exchange of symmetric keys, the parties then communicate application data using encrypted records. The ciphersuite in use is the one agreed during the handshake; if they do not agree on a ciphersuite they abort the communication (specified in the Alert Protocol). Examples of ciphersuites specified by IANA include [8]: `TLS_AES_128_GCM_SHA256`, `TLS_CHACHA20_POLY1305_SHA256`, among others.

TLS Alert Protocol. It defines error situations and actions that TLS peers have to take. There are two types of alerts: closure or error. Closure alerts are important to avoid the "truncation attack" [3], and they occur voluntarily, whereas an error alert causes both parties to immediately close the connection. All alert messages are encrypted as specified by the current connection state.

2.5 Post-Quantum Cryptography

At the time of writing, algorithms part of Post-Quantum Cryptography (also called Quantum-Safe cryptography) are under scrutiny by the cryptographic community. Designed to be resistant to a Cryptographically Relevant Quantum Computer (CRQC), PQC algorithms can execute in classical computers to protect data against both classical and quantum attackers. The protection is based on mathematical assumptions with no (known) quantum or classical solution.

NIST is conducting a notable PQC standardization process [21], and several international agencies have declared that they will follow the NIST PQC process [7]. The proposals are for Key Exchange (KEX) and digital signatures. Currently at Round 4, NIST selected CRYSTALS-Kyber for KEX, and Dilithium, denoted as the primary choice, Falcon, and SPHINCS+ for digital signatures. Even though some PQC algorithms belong to the same group (or type), their public keys and output sizes are different. This information is presented in Table 1. The sizes are given in bytes, and correspond to NIST security levels 1 through 5. Level 5 increases both security and the size of public keys and outputs.

Table 1. PQC algorithms sizes (NIST's Process Round 4 finalists)

Type	Based on	Algorithm	Public key size	Output size
KEX	*Lattice*	Kyber	800 to 1568	768 to 1568
Signature	*Lattice*	Dilithium	1312 to 2592	2420 to 4595
		Falcon	897 to 1793	690 to 1330
	Hash	SPHINCS+	32 to 64	7856 to 49856

The differences between the algorithms compose some of the important factors to consider when deploying PQC in network protocols. For instance, Sikeridis et al. [32] showed that the increased size of PQC can slowdown network performance due to TCP congestion control mechanisms. In addition to sizes, computational cost is also an important factor, but it can vary significantly on different hardware and specific implementations.

Post-Quantum Adoption in TLS. One of the weaknesses of TLS 1.3 is that it does not prevent from quantum attacks. To address this issue, researchers start to develop and test quantum-safe algorithms integrated in the protocol's handshake. One notorious initiative is the Open-Quantum Safe (OQS) project, which provides PQC libraries and integrates it in a variety of network protocol implementations, such OpenSSL [33]. Using those libraries, literature shows different TLS benchmarks comparing post-quantum and classical implementations [22,32]. In summary, lattice-based algorithms are generally fast but with an increase in terms of size of cryptographic objects (*e.g.*, public keys). Additionally, there are disruptive proposals, which changes the TLS handshake aiming for a better performance. One example is KEMTLS [29] and KEMTLS-PDK

[30], which replaces signature schemes used in TLS handshakes by post-quantum KEMs. Still, PQC adoption in TLS remains challenging in practice, since revocation data in TLS, PKI migration issues and other situations are not yet fully explored by the literature.

3 Related Work

3.1 Method

The review protocol consists of five steps [17]: (i) research questions definition; (ii) search strategy for primary studies; (iii) definition of inclusion criteria; (iv) classification of the papers; and (v) data extraction.

The first step is to establish the scope, which we do by developing two research questions. We want to understand "*How and which PQC algorithms were used to secure OpenID Connect and OAuth 2.0?*" and "*What is the impact of replacing quantum-vulnerable cryptographic primitives with PQC alternatives?*" Our search strategy is based on these research questions and consists of using related PQC terms defined in a search string. The search string is used to query primary study sources (*e.g.*, SpringerLink). We constructed the string in accordance with the Population, Intervention, Comparison, Outcomes, and Context (PICOC) guideline [23], where the *population* consists of OpenID Connect, OAuth 2.0, and JWT; post-quantum cryptography is the *intervention* technique; and the underlying mathematical assumptions and names of PQC algorithms were used for *comparison*. It is worth noting that when we ran the search string, we discovered that the terms referring to the PICOC *outcomes* and *context* significantly reduced the number of papers returned by the search libraries. As a result, we simplified the search string in order to increase the number of papers. Figure 2 depicts the final search string.

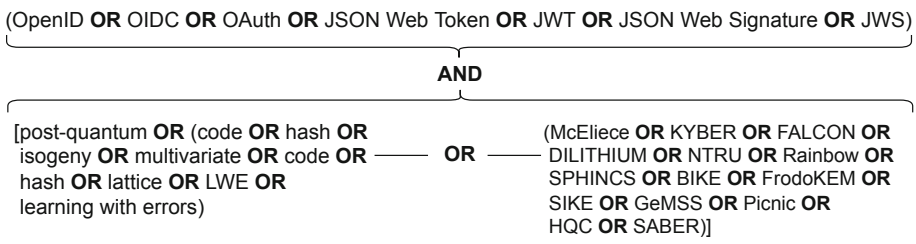

Fig. 2. Search string.

Typically, a systematic study identifies selection criteria for relevant papers, *i.e.*, papers that provide (partial or complete) answers to the research questions. In Table 2, we define one Inclusion Criterion (IC) and two Exclusion Criteria. In summary, ECs exclude papers that are not primary computer science studies, and if the paper satisfies IC-1, it will be included in our results set.

Table 2. Inclusion and exclusion criteria.

Inclusion criteria	
IC-1	The paper investigates or proposes post-quantum OAuth 2.0 or OpenID Connect.
Exclusion criterion	
EC-1	The search result is a book of proceedings.
EC-2	The research work is not in the area of computer science.

3.2 Execution

Figure 3 depicts the search execution method. We queried databases for primary studies and patents on May 4, 2022. Each action outlined eliminates unrelated search results. After applying Inclusion Criteria (IC), the final set of related works consists of a single primary study. This finding suggests that little research has been conducted on this subject to date.

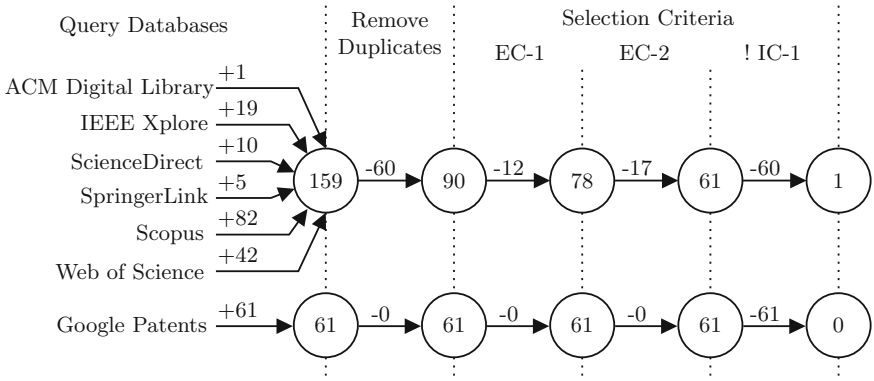

Fig. 3. The steps of our systematic literature review.

Post-Quantum JWT. Our systematic search revealed a single relevant work. Alkhulaifi and El-Alfy [1] assessed the performance of two lattice-based signature algorithms for JWT authentication. They compared two PQC algorithms (Dilithium and qTesla) to RSA in their study. They selected the NIST PQC process (Round 2) implementations using security levels one and three. The JWT body in their implementation is composed of random data. They demonstrated that Dilithium achieves superior performance when deployed in an HTTP client-server implementation, both in terms of requests per second and average response time. The comparison focuses solely on signatures, so the transfer of public keys does not factor into the evaluation. In addition, their research only considered local evaluations, *i.e.*, client and server running on the same machine, thereby excluding network conditions from the analysis.

Open Challenges. After conducting a systematic literature review, we identified open challenges. First, there are no practical OIDC or OAuth 2.0 post-quantum evaluations. A practical evaluation would include network conditions imposed by geographically distant servers and the connections required to complete authentication and authorization processes following these protocols. This void is the primary focus of our efforts. Secondly, the only post-quantum JWT evaluation does not take elliptic-curve signatures into account, resulting in superior performance and smaller sizes compared to RSA [1]. It is crucial to include elliptic curves in the analysis, particularly when comparing to PQC sizes.

4 Post-Quantum OAuth 2.0 and OIDC

OAuth 2.0 and OIDC are application-level protocols that delegate the majority of the security heavy-lifting to TLS and JWT. Aside from mandating the use of these protocols as building blocks and providing implementation guidance to prevent security flaws, few security requirements remain undefined.

OAuth 2.0 includes numerous security considerations for protocol-level messages and parameters. The only other security requirement besides the use of TLS is that the probability of an attacker guessing a token must be less than or equal to 2^{-128} and ideally 2^{-160} [6]. In other words, servers must issue tokens randomly (one token must be independent from others). Consequently, given that: (i) access tokens can have a long lifetime; and (ii) a malicious agent can perform a *record-now-decrypt-later-attack* with a quantum computer using Grover's algorithm [5]; the probability of guessing a token must be reduced to less than 2^{-256} and preferably less than 2^{-320} to account for quantum attackers.

Transporting tokens via OAuth 2.0 and OIDC requires the use of the TLS protocol. At the time of writing, the most recent version of TLS is 1.3, which is also vulnerable to quantum attacks. In the context of TLS, there are three major concerns about quantum attackers. First, the minimum block size for symmetric encryption in AES is 128 bits, which is vulnerable to Grover's algorithm and must be increased to 256 bits in order to maintain the same level of security. Second, TLS authenticates the server and, if mutual authentication is enabled, the client using x509 digital certificates. Since the authenticity of digital certificates is dependent on digital signatures and Shor's algorithm [31] poses a threat to IFP, DLP, and ECDLP-based signature algorithms such as RSA and ECDSA, quantum-safe signature schemes must be utilized instead. Third, TLS employs KEX mechanisms such as DHE and ECDHE to derive symmetric keys and for Forward Secrecy. However, these mechanisms are vulnerable to Shor's algorithm and must therefore be replaced with quantum-safe alternatives.

Quantum attackers also pose a threat to the JWT family of specifications. The JSON Web Encryption (JWE) [15] and JSON Web Signature (JWS) [12] standards describe, respectively, how to encrypt and sign JWTs. Although token

encryption is not required by OAuth 2.0 or OIDC, 256-bit AES must be used to achieve 128-bit security against quantum attackers if it is used. JWE implementations must support both 128-bit and 256-bit AES block sizes [10], in general, not a significant issue. Additionally, the same flaws that affect x509 signatures also affect JWT signatures. Consequently, PQ signature algorithms are required.

To successfully communicate, both IdP and RP must support PQ signature algorithms in their TLS and JWT implementations. Unlike TLS, where the handshake is terminated if client and server support different signature algorithms, in OIDC the "signing party MUST select a signature algorithm based on the algorithms supported by the recipient" [27]. Thus, we propose replacing "MUST" with "MAY" to allow the signing party to refrain from signing a JWT with an insecure signature algorithm if the recipient does not support PQ signature.

Finally, it is critical to consider the implications of increasing symmetric encryption block and key sizes, as well as replacing KEX and signature algorithms with quantum-resistant alternatives. At the time of writing, the NIST standardization process was not completed. Nonetheless, the finalists' KEX and signature algorithms have larger key and signature sizes than elliptic curve solutions. As a result, identity architects and implementers must be aware of the additional requirements for storing and transporting PQ-JWTs in PQ-TLS. To that end, we empirically evaluate the NIST's final signature algorithms in a real-world OAuth 2.0 and OIDC implementation with PQ-TLS, as described below.

5 Empirical Evaluation

5.1 Case Study

Although the three-party authorization handshake is emphasized in scientific and non-scientific literature, it represents only a portion of the interactions that occur when a real RP consumes user data from an IdP. Figure 4 depicts the entirety of our scenario, which includes the following interactions.

The first set of interactions is identified by the dotted rectangle with a capital letter A in the upper-right corner. As stated previously, an RP must first determine the URLs of an OIDC provider's endpoints. The OIDC-D [28] specification defines `/.well-known/openid-configuration` for this specific purpose. In response to an HTTP `GET` request to this resource, the IdP returns a JSON containing its endpoints. Figure 4 depicts the endpoints returned by the IdP that are subsequently utilized. The RP then accesses the `jwks_uri` returned in the previous step to obtain the public keys used to sign the tokens.

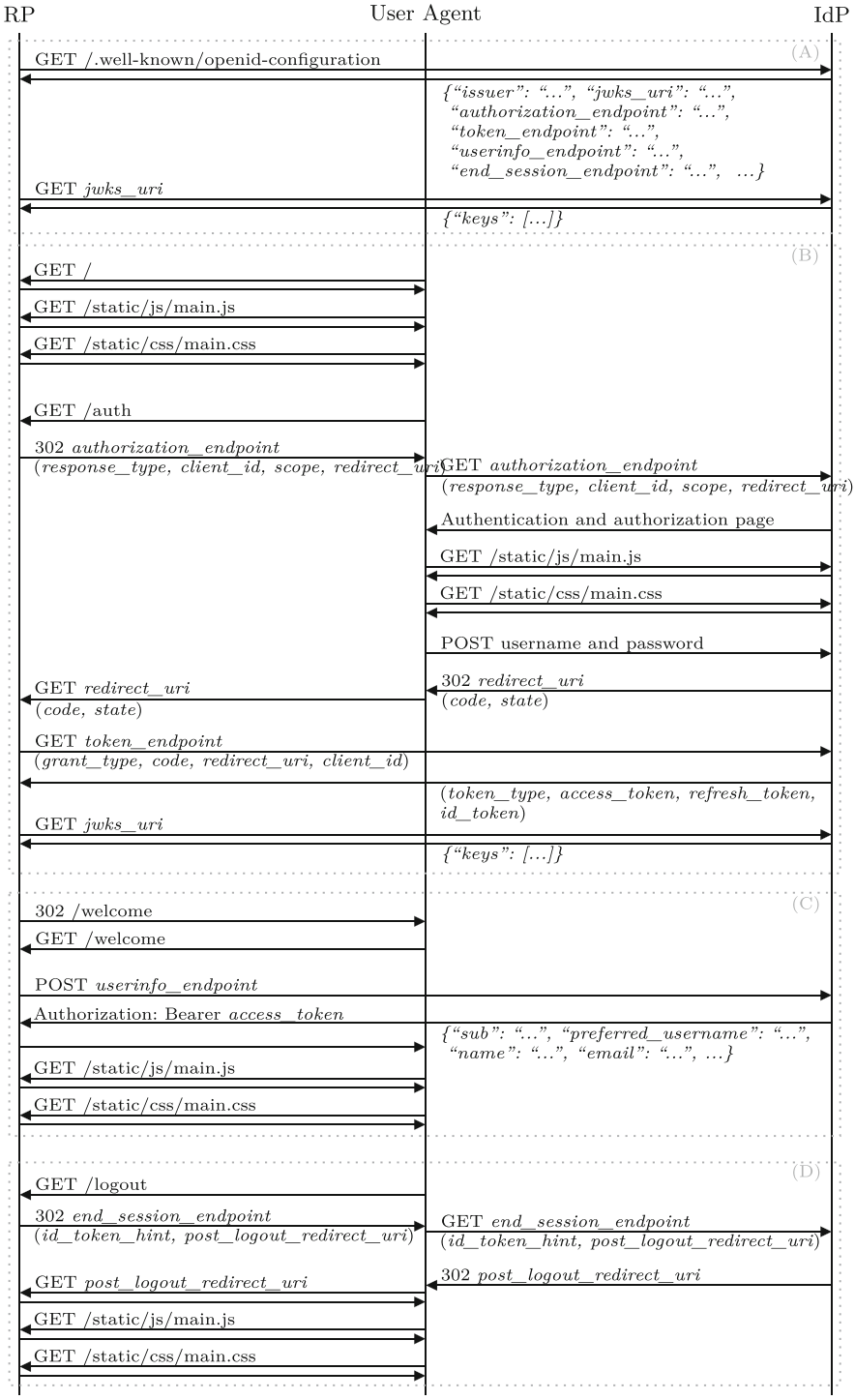

Fig. 4. The realistic evaluation scenario.

The second set of interactions, referred to as B, begins with the user accessing the RP's main page, which uses Cascading Style Sheets (CSS) and JavaScript (JS) resources. We combined all JS and CSS into a single file. The user then accesses a protected page, which requires identification and authentication of the end user as well as authorization for the RP to access their personal information. This initiates the three-party authorization handshake, which redirects the user to the IdP authentication and authorization page, which includes its own CSS and JS resources. The user then enters their username and password for authentication and, on the same page, checks the box to authorize the sharing of their nickname, full name, and e-mail.

The back-channel portion of the handshake begins after submitting the correct information and being redirected to the RP. Three tokens are created in our scenario: the OAuth 2.0 `access_token` and `refresh_token`, as well as the OIDC `id_token`. They are all JWT-based and signed with the same private key. It is worth noting that the RP then makes a second request to the `jwks_uri`. This occurs because the IdP may have rotated the keys used to sign the tokens, necessitating the acquisition of a new copy of the public key currently in use.

Following the authentication and authorization dance, the third set of requests and responses begins by redirecting the user to a welcoming page. The user agent requests the `welcome` page, causing the RP to contact the IdP and request the previously authorized user data. The RP incorporates the previously issued `access_token` into the request, which is validated by the IdP, resulting in the retrieval of the requested data. The RP assembles the welcoming page using this data and returns it to the user. This page also includes CSS and JS files.

The final group of interactions pertains to the user's logout. Beginning with the user requesting logout from the RP, the user agent is redirected to the `end_session_endpoint` URI of the IdP, which contains the `id_token`, and the `post_logout_redirect_uri` of the RP. This is the URI of the RP to which the user will be redirected after terminating her session on the IdP.

There are twenty two GET and POST requests in our real-world scenario. It is build on top of OAuth 2.0 [6], JWT Profile for OAuth 2.0 Access Tokens [4], OIDC core [27], discovery [28], RP-Initiated Logout [14] and the JWT family [10–13,15].

5.2 Prototype Implementation

We implemented the evaluation scenario in Python 3 using `pyoidc`[1], an OpenID Foundation-certified OIDC Core, OIDC-D, and OIDC-DCR implementation. JWTs are assembled, encrypted, and signed using `pyjwkest`[2]. In order to have a functional PQ-OIDC implementation, the NIST's fourt round finalist signature algorithms were added. We used the Python 3 binding of Open-Quantum Safe (OQS) to access the C implementation of these algorithms [33]. The OQS

[1] https://github.com/OpenIDC/pyoidc.
[2] https://github.com/IdentityPython/pyjwkest.

project also provides a modified `libssl` containing the PQ-KEX and PQ signing algorithms, which our Python implementation employs for PQ-TLS. Our implementation and empirical evaluation are embedded within Docker containers, making it simple to reproduce our experiments. The source code for our implementation and experiments is open-source and publicly available[3].

5.3 Experiment Plan

The experiment entails evaluating the size of messages exchanged and the elapsed time for pre-quantum and post-quantum KEX and signature algorithms in the previously described evaluation scenario on servers with increasing geographical distance. Let us concentrate on the impact of size on PQ algorithms. Except for cryptographic objects, the content of messages exchanged in our scenario remains constant, allowing us to create an estimate of the impact of cryptographic objects in pre-quantum and post-quantum OIDC. The overall cost $OIDC^{size}$, in bytes, can be defined as shown in Eq. 1.

$$\begin{aligned} OIDC^{size} = & N_{TLS} * (TLS_{KEX}^{size} + TLS_{Auth}^{size}) + \\ & N_{JWT} * JWT_{Sig}^{size} + \\ & N_{JWK} * JWK_{PK}^{size} \end{aligned} \tag{1}$$

Where N_{TLS} is the number of TLS Handshakes; TLS_{KEX}^{size} refers to the size of the TLS `ClientHello` and `ServerHello` messages; TLS_{Auth}^{size} corresponds to the size of TLS authentication considering handshake signature and certificates; N_{JWT} is the number of JWTs exchanged and JWT_{Sig}^{size} is the size of the signature present in each JWT; N_{JWK} is the number of public keys stored in JWKs exchanged in the scenario, and JWK_{PK}^{size} the size of those keys.

It is important to note that for TLS authentication, we incorporate the Certificate Authority (CA) intermediate and root certificates. Also, to better reflect real-world settings, the server certificate incorporates Signed Certificate Timestamps, a standard for publicly reporting TLS server certificates as they are issued or observed for the sake of auditing CAs' activity [18]. In addition, due to TLS connection resumption across peers, not all TLS handshakes in the real world have identical sizes. To simulate the worst-case scenario, however, we perform a new TLS handshake for each of the $N_{TLS} = 22$ connections. Consequently, this model provides an upper bound for the expected cost (in terms of size) for OIDC scenarios. Furthermore, N_{JWK} is 2 as there are two calls for the `jwks_uri`, and N_{JWT} is 8: the IdP issues one `access_token`, one `refresh_token`, and one `id_token` that are sent to the RP; the RP then sends the `access_token` to consume the `userinfo_endpoint`; and the request to `/logout` and the following redirect results in the exchange of the `id_token` four times.

Table 3 shows configurations used in each experiment, as well as the expected costs, in bytes. The first two rows show the experiments where classical cryptography is in use, while the following rows compose the post-quantum evaluations.

[3] https://github.com/fredericoschardong/post-quantum-oidc-oauth2.

We estimate the costs of the cryptographic objects present in each TLS handshake (second and fourth columns), as well as the JWT and JWK (fifth and sixth columns), which include a digital signature and the public key. Unlike the TLS handshake, which occurs for all twenty-two requests in our scenario, the JWT and JWK parts are present in some messages, as described above.

Table 3. Algorithm configurations with expected costs in terms of size

TLS_{KEX}	TLS_{KEX}^{size}	TLS_{Auth} JWT_{Sign}	TLS_{Auth}^{size}	JWT_{Sig}^{size}	JWK_{PK}^{size}	$OIDC^{size}$
ECDHE	626	ECDSA	1563	64	64	48798
		RSA 2048	3309	256	256	89130
Kyber512	1976	Dilithium 2	19419	2420	1312	492674
		Dilithium 3	26576	3293	1952	658392
		Dilithium 5	36309	4595	2592	884214
		Falcon-512	7541	690	897	216688
		Falcon-1024	13919	1330	1793	363916
		SPHINCS+128	103553	17088	32	2458406
		SPHINCS+192	215166	35664	48	5062532
		SPHINCS+256	316869	49856	64	2458406

We selected ECDHE and Kyber512 for the pre-quantum and post-quantum TLS KEX, respectively. ECDHE p256 is standardized and Kyber is the finalist of the NIST Round 4 standardization process. As we can see in Table 3, TLS authentication have a significant impact, and so we compared two commonly-used classical algorithms (ECDSA p256 and RSA2048) against three post-quantum (Falcon, SPHINCS+, and Dilithium), the latter are going to be standardized by NIST. For simplicity, the same algorithm is used for TLS authentication and JWT Signature. The overall cost is increased in the post-quantum scenarios. Regarding SPHINCS+, we selected SHAKE256 as hash function, due to the overall performance without considering hardware optimizations; and with `fast-simple` parameter set, also focusing on performance (please refer to [2] for additional information).

5.4 Experimental Results

We ran the realistic scenario a thousand times for each algorithm configuration described in Table 3 on Amazon EC2 t2.micro instances with 1 vCPU and 1 GB of RAM in five distinct locations and collected data. We begin by outlining the timings for each algorithm option. Figure 5 depicts the results of all scenarios examined, highlighting the timings of TLS handshakes in relation to the entire OIDC time. When network latencies increase between scenarios, we can see distinct behavior. Post-quantum algorithms benefit from low latencies, achieving similar (or even greater) performance than classical algorithms. When considerable latencies are present (140ms, 225ms, and 320ms scenarios),

RSA and ECDSA perform better. The increased size of PQC algorithms has a severe impact on network performance, frequently requiring additional round trips to convey data, particularly at the TCP level due to congestion management techniques. When post-quantum methods are tested at the same latency, the total duration varies little between Dilithium and Falcon, but significantly with SPHINCS+. We can also see that TLS costs account for a large portion of total time, particularly in PQC configurations.

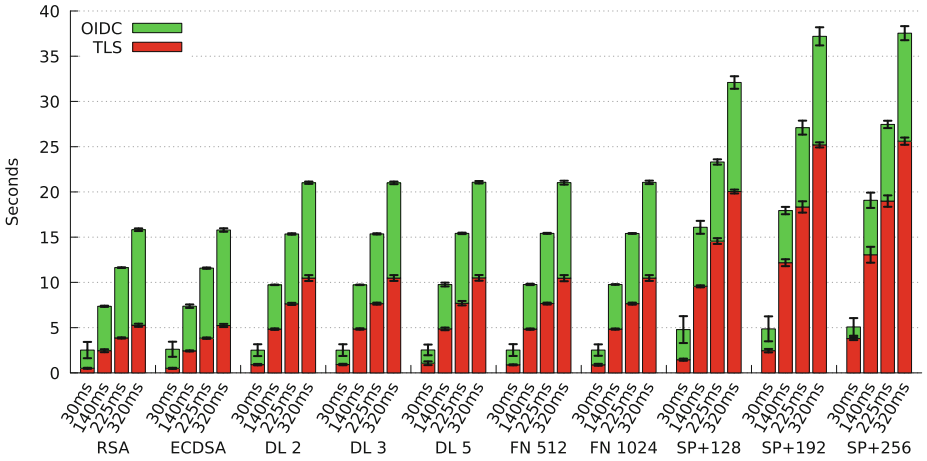

Fig. 5. Average OIDC and TLS timings for various latency settings.

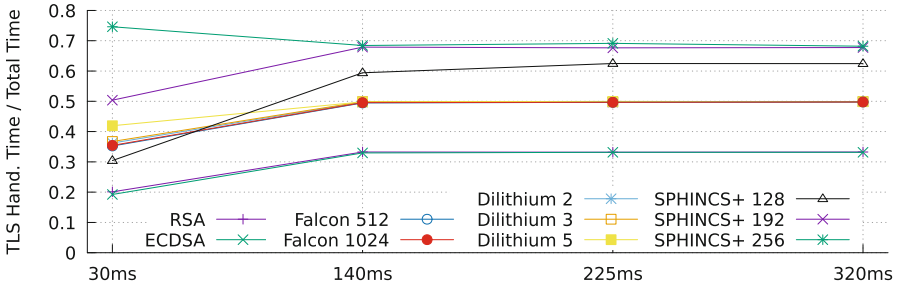

Fig. 6. The ratio between TLS handshake and overall OIDC time.

Figure 6 depicts a new perspective on TLS costs. It displays the average TLS handshake time in relation to the average total time, allowing us to compare the PQC and conventional algorithms. The ratio for Falcon and Dilithium is approximately 50% on the 140ms, 225ms and 320ms scenarios. This finding

implies that enhancing TLS handshake may result in improved OIDC overall performance.

Finally, Table 4 compares the costs in terms of bytes of the 22 TLS handshakes (KEX and Authentication) to the application payloads (including HTML, CSS, and JS content, as well as JWT and JWK sizes). It is evident that the larger application size sizes (about 1 MB) contribute to obfuscating the distinctions between PQC instances (with different parameter sets). On the other hand, when these costs are compared to the traditional techniques, the slowdowns are considerably more visible, as seen in Fig. 5. This is an interesting finding: despite increasing security parameters, the cost did not rise considerably enough to cause performance issues (from 1 to 5). However, our SPHINCS+ instances excessively increased the sizes, which explains the longer timings (Fig. 5). This suggests that SPHINCS+ should not be the first option for all TLS components (*e.g.*, OCSP Stappling, SCT Logs) or TLS-dependent protocols (*e.g.* OAuth 2, OIDC) due to the signature size.

Table 4. TLS cost versus OIDC cost, measured in bytes.

Algorithm	TLS Hand.	OIDC	Ratio
ECDHE/ECDSA	48158	1017405	0.0473
ECDHE/RSA2048	86570	1019903	0.0848
Kyber512/Dilithium2	470690	1043463	0.4510
Kyber512/Dilithium3	628144	1056528	0.5945
Kyber512/Dilithium5	842270	1072090	0.7856
Kyber512/Falcon512	209374	1025579	0.2041
Kyber512/Falcon1024	349690	1034530	0.3380
Kyber512/SPHINCS+128	2321638	1857118	1.2501
Kyber512/SPHINCS+192	4777124	2182244	2.1891
Kyber512/SPHINCS+256	7014590	2314753	3.0304

6 Conclusion and Future Works

Experimenting with PQC in protocols such as OIDC demonstrates the costs and consequences of its adoption. In this paper, we assess a Quantum-safe implementation of OIDC with realistic scenarios, including the use of PQC signatures in JWT tokens and post-quantum TLS as the underlying security protocol.

In our study of a realistic scenario, we demonstrated that the quantity and complexity of requests to use OIDC exceeds the three-party handshake established in OAuth 2.0. Nonetheless, we successfully incorporated PQC into the proposed OIDC study case, and the experiments demonstrated that the network has a considerable impact on performance. Our PQC implementation in lower latencies has demonstrated competitive performance against RSA and ECDSA.

Consequently, our empirical evaluation has demonstrated that PQC algorithms do not decrease OIDC performance in such scenarios. However, when latency increases, we discovered substantial performance issues, primarily owing to the increased size of PQC objects exchanged between the parties involved. Surprisingly, we found no significant variations in performance between PQC algorithms of varying security levels when comparing conventional to PQC methods.

We discovered that the TLS cost in OIDC constitutes a considerable portion of the costs, both in terms of time and size, particularly when network latencies are greater. Since we simulated actual setups (*e.g.*, SCTs and OCSP Stappling in certificates), we conclude that our findings provide an upper bound on TLS costs when evaluating it as a drop-in replacement for PQC in OIDC.

The scope of this work allows for several other areas of research to be pursued in the future. For example, we examine a PQC-only deployment, but another option may be hybrid PQC, which employs both PQC and classical algorithms. Using hybrids in OIDC would necessitate their deployment in TLS and JWT, the latter of which will almost certainly encounter compatibility challenges due to its list of authorized methods. Furthermore, certain techniques, such as KEMTLS [29], try to reduce the TLS costs of PQC. If TLS costs are reduced, performance benefits in PQ-OIDC can be expected.

References

1. Alkhulaifi, A., El-Alfy, E.S.M.: Exploring lattice-based post-quantum signature for JWT authentication: review and case study. In: 2020 IEEE 91st Vehicular Technology Conference (VTC2020-Spring), pp. 1–5 (2020). https://doi.org/10.1109/VTC2020-Spring48590.2020.9129505
2. Aumasson, J.P., et al.: Sphincs+: submission to the NIST post-quantum project (2021). https://sphincs.org/data/sphincs+-r3.1-specification.pdf
3. Berbecaru, D., Lioy, A.: On the robustness of applications based on the SSL and TLS security protocols. In: Lopez, J., Samarati, P., Ferrer, J.L. (eds.) EuroPKI 2007. LNCS, vol. 4582, pp. 248–264. Springer, Heidelberg (2007). https://doi.org/10.1007/978-3-540-73408-6_18
4. Bertocci, V.: JSON Web Token (JWT) Profile for OAuth 2.0 Access Tokens. RFC 9068 (2021)
5. Grover, L.K.: A fast quantum mechanical algorithm for database search. In: Proceedings of the 28th ACM Symposium on Theory of Computing, pp. 212–219 (1996)
6. Hardt, D.: The OAuth 2.0 Authorization Framework. RFC 6749 (2012)
7. Howe, J., Prest, T., Apon, D.: Sok: How (not) to design and implement post-quantum cryptography. Cryptology ePrint Archive, report 2021/462 (2021). https://ia.cr/2021/462
8. (IANA), I.A.N.A.: Transport layer security (TLS) parameters (2022)
9. Johansson, L.: An IANA Registry for Level of Assurance (LoA) Profiles. RFC 6711 (2012)
10. Jones, M.: JSON Web Algorithms (JWA). RFC 7518 (2015)
11. Jones, M.: JSON Web Key (JWK). RFC 7517 (2015)
12. Jones, M., Bradley, J., Sakimura, N.: JSON Web Signature (JWS). RFC 7515 (2015)

13. Jones, M., Bradley, J., Sakimura, N.: JSON Web Token (JWT). RFC 7519 (2015)
14. Jones, M., De Medeiros, B., Agarwal, N., Sakimura, N., Bradley, J.: Openid connect RP-initiated logout 1.0. The OpenID Foundation, p. S3 (2022)
15. Jones, M., Hildebrand, J.: JSON Web Encryption (JWE). RFC 7516 (2015)
16. Jones, M., Sakimura, N., Bradley, J.: OAuth 2.0 Authorization Server Metadata. RFC 8414 (2018)
17. Keele, S., et al.: Guidelines for performing systematic literature reviews in software engineering. Technical report (2007)
18. Laurie, B., Langley, A., Kasper, E., Messeri, E., Stradling, R.: Certificate Transparency Version 2.0. RFC 9162 (2021)
19. Lodderstedt, T., Bradley, J., Labunets, A., Fett, D.: OAuth 2.0 Security Best Current Practice. Internet-Draft 20, Internet Engineering Task Force (2022)
20. Lodderstedt, T., McGloin, M., Hunt, P.: OAuth 2.0 Threat Model and Security Considerations. RFC 6819 (2013)
21. NIST: Post-quantum cryptography (2016). https://csrc.nist.gov/Projects/Post-Quantum-Cryptography. Accessed 26 June 2021
22. Paquin, C., Stebila, D., Tamvada, G.: Benchmarking post-quantum cryptography in TLS. In: Ding, J., Tillich, J.-P. (eds.) PQCrypto 2020. LNCS, vol. 12100, pp. 72–91. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-44223-1_5
23. Petersen, K., Vakkalanka, S., Kuzniarz, L.: Guidelines for conducting systematic mapping studies in software engineering: an update. Inf. Softw. Technol. **64**, 1–18 (2015)
24. Rescorla, E.: The Transport Layer Security (TLS) Protocol Version 1.3. RFC 8446, August 2018. https://doi.org/10.17487/RFC8446
25. Sakimura, N., Bradley, J., Agarwal, N.: Proof Key for Code Exchange by OAuth Public Clients. RFC 7636 (2015)
26. Sakimura, N., Bradley, J., Jones, M.: OpenID connect dynamic client registration. The OpenID Foundation, p. S3 (2014)
27. Sakimura, N., Bradley, J., Jones, M., De Medeiros, B., Mortimore, C.: OpenID connect core 1.0. The OpenID Foundation, p. S3 (2014)
28. Sakimura, N., Bradley, J., Jones, M., Jay, E.: OpenID connect discovery. The OpenID Foundation, p. S3 (2014)
29. Schwabe, P., Stebila, D., Wiggers, T.: Post-Quantum TLS Without Handshake Signatures, pp. 1461–1480. Association for Computing Machinery, New York, NY, USA (2020). https://doi.org/10.1145/3372297.3423350
30. Schwabe, P., Stebila, D., Wiggers, T.: More efficient post-quantum KEMTLS with pre-distributed public keys. In: Bertino, E., Shulman, H., Waidner, M. (eds.) ESORICS 2021. LNCS, vol. 12972, pp. 3–22. Springer, Cham (2021). https://doi.org/10.1007/978-3-030-88418-5_1
31. Shor, P.W.: Algorithms for quantum computation: discrete logarithms and factoring. In: Proceedings 35th Annual Symposium on Foundations of Computer Science, pp. 124–134. IEEE, Santa Fe, NM, USA (1994). https://doi.org/10.1109/SFCS.1994.365700
32. Sikeridis, D., Kampanakis, P., Devetsikiotis, M.: Assessing the overhead of post-quantum cryptography in TLS 1.3 and SSH. In: Proceedings of the 16th International Conference on emerging Networking EXperiments and Technologies, pp. 149–156. Association for Computing Machinery, New York, NY, USA (2020)
33. Stebila, D., Mosca, M.: Post-quantum key exchange for the internet and the open quantum safe project. In: Avanzi, R., Heys, H. (eds.) SAC 2016. LNCS, vol. 10532, pp. 14–37. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-69453-5_2