# TLS 1.3 for Fast and Secure Edge Service Continuity in Smart Cities

Lorenzo Catoni, Carlo Puliafito, Gianluca Dini
Department of Information Engineering, University of Pisa, Pisa, Italy

## I. Introduction

Cloud computing delivers computational resources via remote data centers. Edge computing enhances this model by deploying servers across the cloud-edge continuum, placing them nearer to end users. This supports low-latency applications critical to smart cities, such as augmented/virtual reality and autonomous vehicles. The European Telecommunications Standards Institute (ETSI) is standardizing this paradigm under the name Multi-access Edge Computing (MEC).

In cloud-edge deployments, clients frequently access services from various locations, driven by factors such as mobility or load balancing. Maintaining seamless service continuity — ensuring uninterrupted, transparent, and secure access to services despite location changes — is vital. In another extended abstract submitted to I-CiTies 2025, we present a service-continuity solution leveraging a pool of distributed edge proxies. These proxies function as intermediaries between clients and edge services, regulating request routing, load balancing, and access control. Fig. 1 illustrates a mobility scenario showcasing a practical use case of that solution. Consider a drone delivering a package through an urban environment. As it moves, it may need to access various services within the cloud-edge infrastructure. We focus on a specific example: an image-filtering service, denoted as $\lambda$. The drone captures live video of its surroundings and relies on $\lambda$ to process the frames. Access to this service is mediated by distributed edge proxies, $\pi_i$, with the drone always connected to the nearest one. Initially (Fig.1(**a**)), it connects to proxy $\pi_1$, which routes requests to $\lambda$ hosted on *edge server 2*. As the drone travels (Fig.1(**b**)), the connection to $\pi_1$ becomes inefficient due to increasing network latency. Consequently, the platform must reconfigure to allow the drone to connect to a closer proxy, $\pi_2$ (Fig.1(**c**)). In this example, requests continue to reach the same $\lambda$ instance on *edge server 2*; however, the orchestrator may alternatively choose to relocate services based on system conditions or application demands.

The foundation of that solution lies in dynamic coordination of proxies. When a client switches from one proxy to another, a system orchestrator updates configuration accordingly: it transfers the per-client state to the new target proxy, allowing it to recognize the client and route its requests appropriately. Concurrently, the orchestrator directs the source proxy to inform the client of the new proxy address. This redirection leverages the HTTP *Alternative Services* (AltSvc) extension, which enables clients to discover alternative network endpoints offering the same service. Subsequently, and without requiring changes to the application logic, the client initiates a new Transport Layer Security (TLS) session with the new proxy and begins sending its requests there. However, establishing a new TLS session introduces both latency and data overhead — challenges becoming pronounced in latency-critical applications or scenarios involving frequent proxy handovers. Consequently, there is a need to optimize TLS session establishment. Furthermore, such optimizations must not compromise security, which must be robust even against emerging threats posed by advancements in quantum computing.

In this work — which is a short version of [1] — we enrich the presented service-continuity platform by integrating advanced TLS 1.3 features toward fast session resumption and post-quantum key exchange. This improved version of the platform, which is based on Envoy proxies and BoringSSL library, is publicly available at https://gitlab.com/tesi-lm/tls13resumption-pqkex-platform. The goal of this work is testing the viability of strong security while not sacrificing performance for service continuity in cloud-edge systems.

## II. Resumption and Post-quantum Key Exchange

### A. Post-quantum TLS 1.3

Migrating the Internet to post-quantum key agreement is essential to protect encrypted communications from future quantum threats. A critical concern is the *Store Now Decrypt Later* (SNDL) strategy, where adversaries capture encrypted data today to decrypt them once quantum capabilities mature. To address this, *post-quantum Key Encapsulation Mechanisms* (KEMs) are being integrated into protocols like TLS.

Among these, ML-KEM-768 — based on the Learning With Errors (LWE) problem over modular lattices — is currently the only post-quantum KEM standardized by NIST, offering an estimated 192-bit security level. This surpasses the 128-bit security of classical schemes like X25519, which is known to be vulnerable to quantum attacks.

Despite the stronger theoretical guarantees of ML-KEM-768, hybrid key exchange schemes such as X25519MLKEM768 have been defined by the IETF for TLS 1.3. These combine classical and post-quantum algorithms to hedge against potential future weaknesses in lattice-based cryptography or implementation flaws.

### B. Session Resumption in TLS 1.3

Session resumption is particularly beneficial for latency-sensitive applications deployed in dynamic cloud-edge envi-

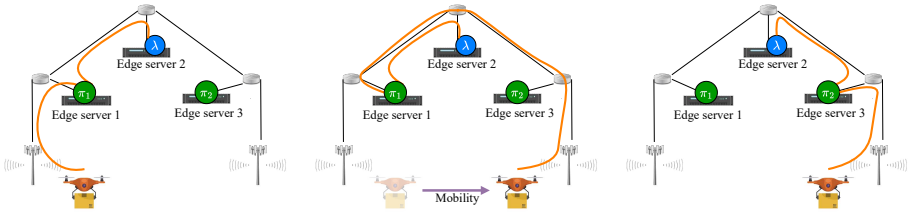


Fig. 1: The service-continuity use case of a drone for last-mile delivery.

ronments, where minimizing handshake overhead is critical. In TLS 1.3, session resumption is enabled through a session ticket sent by the server at the end of the initial handshake. This ticket encapsulates the information required to resume a session, most notably the pre-shared key (PSK), and is stored by the client. Upon reconnection, the client includes the ticket in its `ClientHello`, allowing the server to resume the session without a full handshake.

Servers may handle session tickets in either a stateful or stateless manner. Stateful resumption requires maintaining per-session data on the server side, which introduces complexity and scalability issues in distributed systems. Stateless resumption, in contrast, embeds all necessary session state within the ticket itself, which is encrypted and authenticated using a Session Ticket Encryption Key (STEK). In this work, we adopt the stateless approach by synchronizing a single STEK across all Envoy proxies, enabling cross-node ticket decryption and seamless session resumption across edge proxies. To maintain strong security guarantees, the STEK is periodically rotated, reducing the impact of potential key compromise.

## III. Performance Evaluation

Our goal is evaluating the impact of the different TLS 1.3 configurations on session establishment and ultimately on service continuity. We deployed a client and an Envoy proxy as two Docker containers, both running within a virtual machine having 2 vCPUs, 2 GB of RAM, and running Ubuntu 20.04. We conducted the experiments across four TLS 1.3 configurations, which we derived by combining two key exchange mechanisms – X25519 and X25519MLKEM768 – with the two handshake approaches, namely full and resumed. We considered the following two metrics:

- *Cryptographic overhead* — Time spent on key cryptographic operations involved in the TLS handshake.
- *Data overhead* — Size of the `Client Hello` and `Server Hello` TLS handshake messages.

We ran 35 repetitions for each experiment and show results with 95% confidence intervals, if present.

Fig. 2 depicts cryptographic overhead by client and server. Certificate validation dominates client-side cost, while key exchange operations — key generation, encapsulation, and decapsulation — have similar latency across full and resumed handshakes. X25519MLKEM768 incurs about twice the overhead of X25519, whereas certificate operations remain unaffected by the key exchange method.

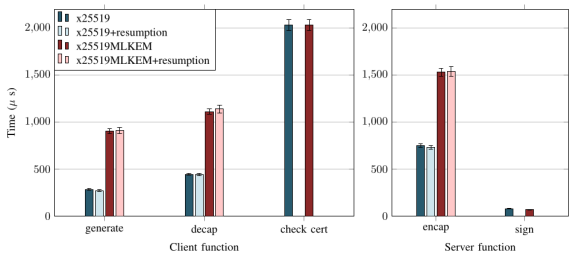


Fig. 2: Cryptographic overhead on client and server.

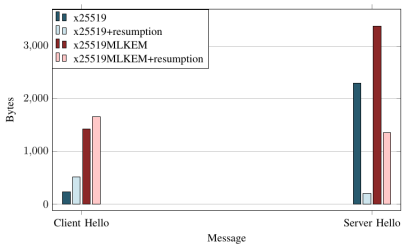


Fig. 3: Data overhead in terms of `Client Hello` and `Server Hello` messages, expressed as bytes exchanged by the different TLS 1.3 configurations.

Fig. 3 shows that full `Client Hello` is about 240 B for X25519 and over 1.4 kB for X25519MLKEM768, mainly due to public key size. Full `Server Hello` messages include a 2 kB certificate. In resumed handshakes, a 250 B session ticket is added to `Client Hello`, but the certificate is omitted from `Server Hello`, reducing total sizes to 700 B for X25519 and 3 kB for X25519MLKEM768 — 27% and 62% of the full handshake, respectively.

## Acknowledgment

This work has been in part supported by the MUR, within the framework of the PRIN 2022 project FuSeCar, and by the European Union — NextGenerationEU — within the National Sustainable Mobility Center (CN00000023, Italian Ministry of University and Research Decree n. 1033—17/06/2022, Spoke 10) and the PE SERICS HARDNESS project.

## References

[1] L. Catoni, C. Puliafito, and G. Dini, "Fast and secure service continuity in the edge-cloud continuum: A study of TLS 1.3 resumption and post-quantum key exchange," in *Proceedings of the 11th International Conference on Smart Computing*, ser. SMARTCOMP '25, 2025, pp. 1–6.