# Replay Attack in TLS 1.3 0-RTT Handshake: Countermeasure Techniques


Network Security (933II)
M.Sc. Cybersecurity
Paolo Bernardi (660944)

Università di Pisa

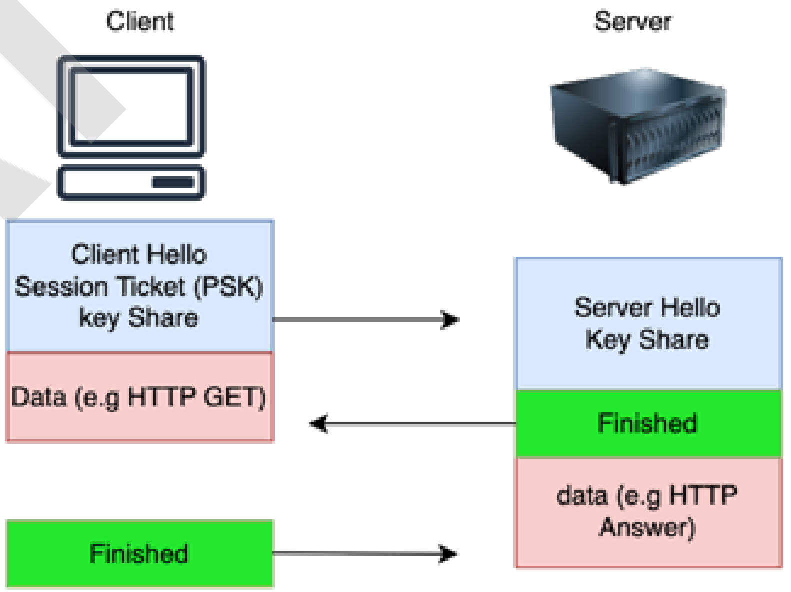

Version 1

# The Paper

- **Conference:** 2023 IEEE 6th International Conference on Electrical, Electronics and System Engineering (ICEESE)
- **Authors:** M.E Abdelhafez (Malaysia), Sureswaran Ramadass (Malaysia), Mohammed S. M. Gismallab (Saudi Arabia)
- **Goal:** review anti-replay protection techniques
- **Keywords:** TLS 1.3, replay attack, 0-RTT, handshake

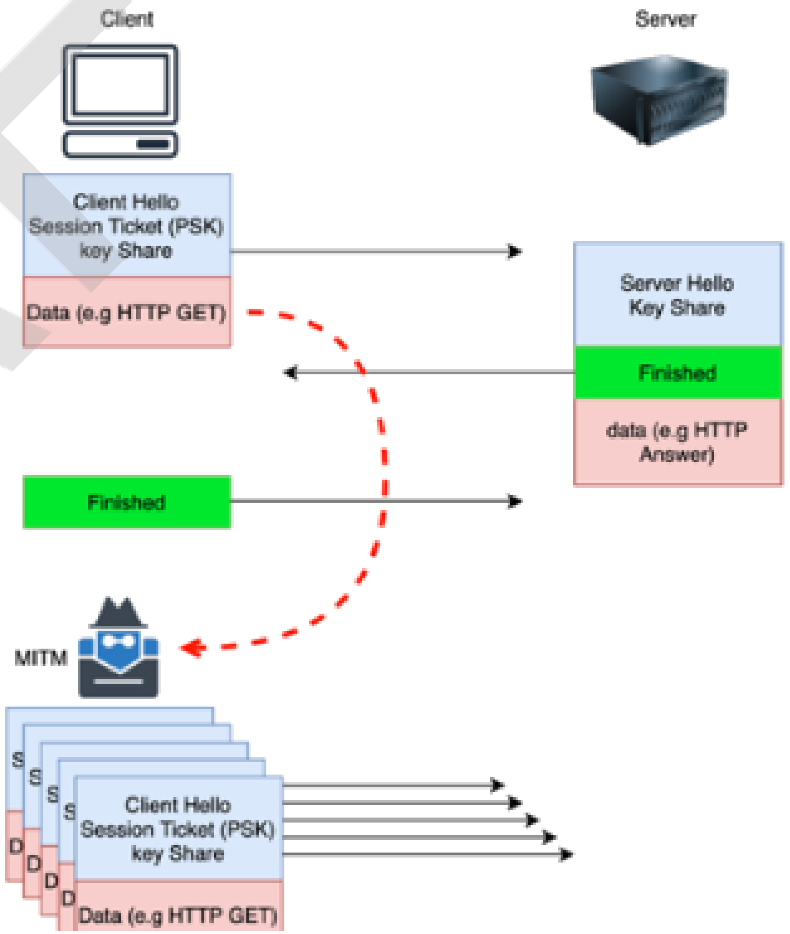
# Context

- TLS resumable connections
- TLS 1.3 introduced **0-RTT** resume mode, based on a **Session-Ticket key** created during the initial full handshake
- 0-RTT obtained by sending a **single message** that contains both the **ClientHello** (with a known Session-Ticket key) and **Application Data** (also known as **Early Data**)



# Attack Scenarios

- **Replay** attack
- Attacker intercepts and replays **ClientHello** messages with **Early Data**
- The replayed message is valid because the **ClientHello** contains a **Session-Ticket key** recognized by the server
- **ALTERNATIVE SCENARIO**: the attacker performs a **MITM** and makes the client **to believe that the 0-RTT message wasn't received**, triggering a resending



# Freshness check

Reject **ClientHello** messages whose **gmt_unix_time** too much in the past

> ✅ Simple implementation
> ❌ Can be inconvenient and there is an exploitable time window for attackers

# ClientHello Recording

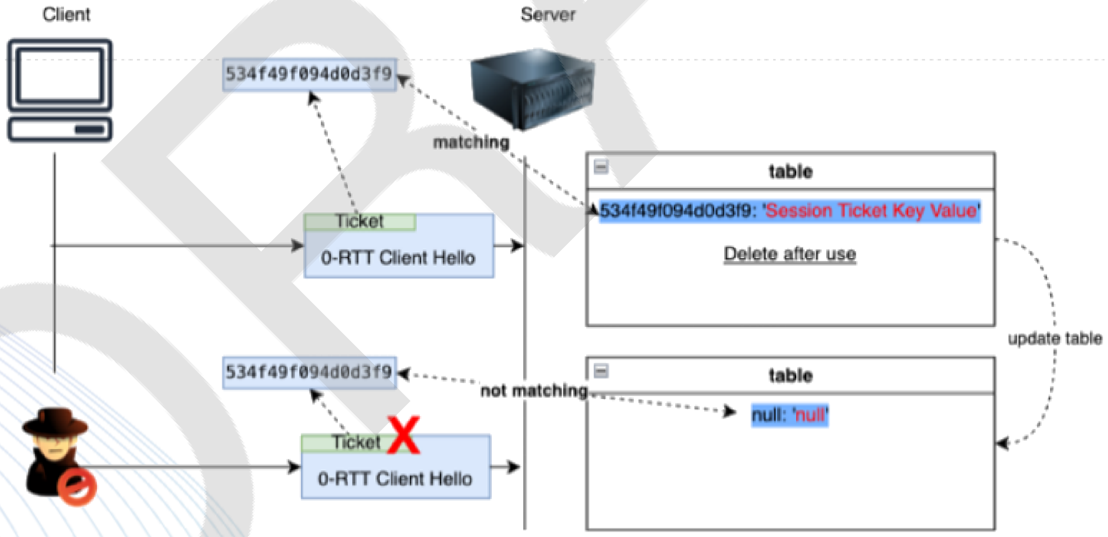
The server keeps a list of received **ClientHello** messages and uses it to detect and discard replays

> ✅ Can block all replay attacks
> ❌ Complex setup in distributed environments, complex synchronization

# Single-Use Tickets

The server **deletes** the "session ticket key" used to decrypt the early data after the first 0-RTT resume, making it impossible to decrypt replayed messages.

> ✅ Can block all replay attacks
> ❌ Complex setup in distributed environments, complex synchronization



# Application Profile

Each application should implement a specific **profile** that specifies under which conditions it will use 0-RTT (e.g. HTTP GET).

> ✅ Flexibility
> ❌ Not 100% safe, requires intervention at application level

# Separate API

Both client and servers use libraries that make 0-RTT usage **explicit**, rather than implicit and automatic.

> ✅ Explicit behaviour
> ❌ Requires TLS libs restructuring and programmers attention

# Puncture Pseudorandom Function (PPRF)

By using **PPRF** the server can decrypt 0-RTT early data only once.

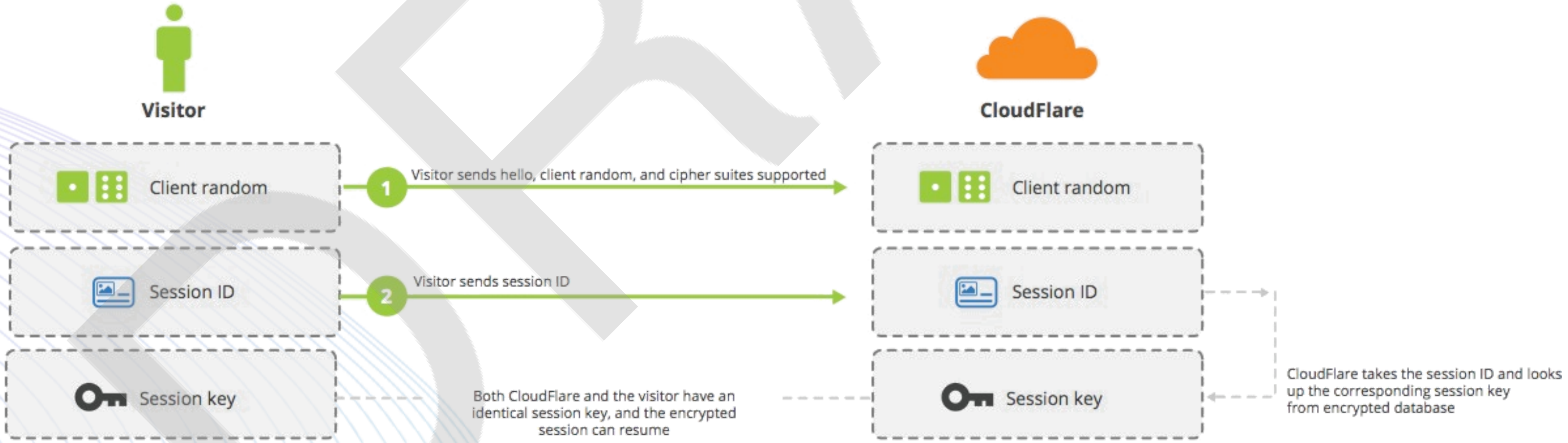
**Example approach:** a server maintains a session ticket encryption key (STEK) *k* that can decrypt any session ticket. Then it uses it to decrypt a ticket *t* and it generates a STEK *k*' that can decrypt all session tickets but *t* and so on...

> ✅ Forward secrecy
> ❌ Long processing time, not practical in distributed environments

# Universal SSL

Introduced by **Cloudflare** in 2015 (doesn't support TLS 1.3), Universal SSL stores negotiated sessions into multiple **Memcached** instances. Each session is indexed and encrypted by **Session ID**.

> ✅ Great performance
> ❌ Memcached servers are synchronized only within each Cloudflare PoP



# Just-in-Time Shared Keys (JIT-SK)

Based on a **synchronized PRNG**, dynamically changes keys for each session to secure 0-RTT messages (the same key cannot be reused multiple times, so "blind replaying" is impossible).

> ✅ Prevents replay attacks while providing provides forward secrecy
> ❌ Doesn't support distributed environments

# Conclusions

## 0-RTT is here to stay

The performance improvements are real (the paper stats that 0-RTT resume is 44.7% than 1-RTT) and the percentage of resumed TLS connections is also quite high (40% ins some applications).

## 0-RTT anti-reply protection requires trade offs:

The evaluated protections introduce overheads and/or inconveniences, especially in distributed environments (e.g. CDNs), therefore 0-RTT replay protection is still an open research topic.

# THANK YOU!