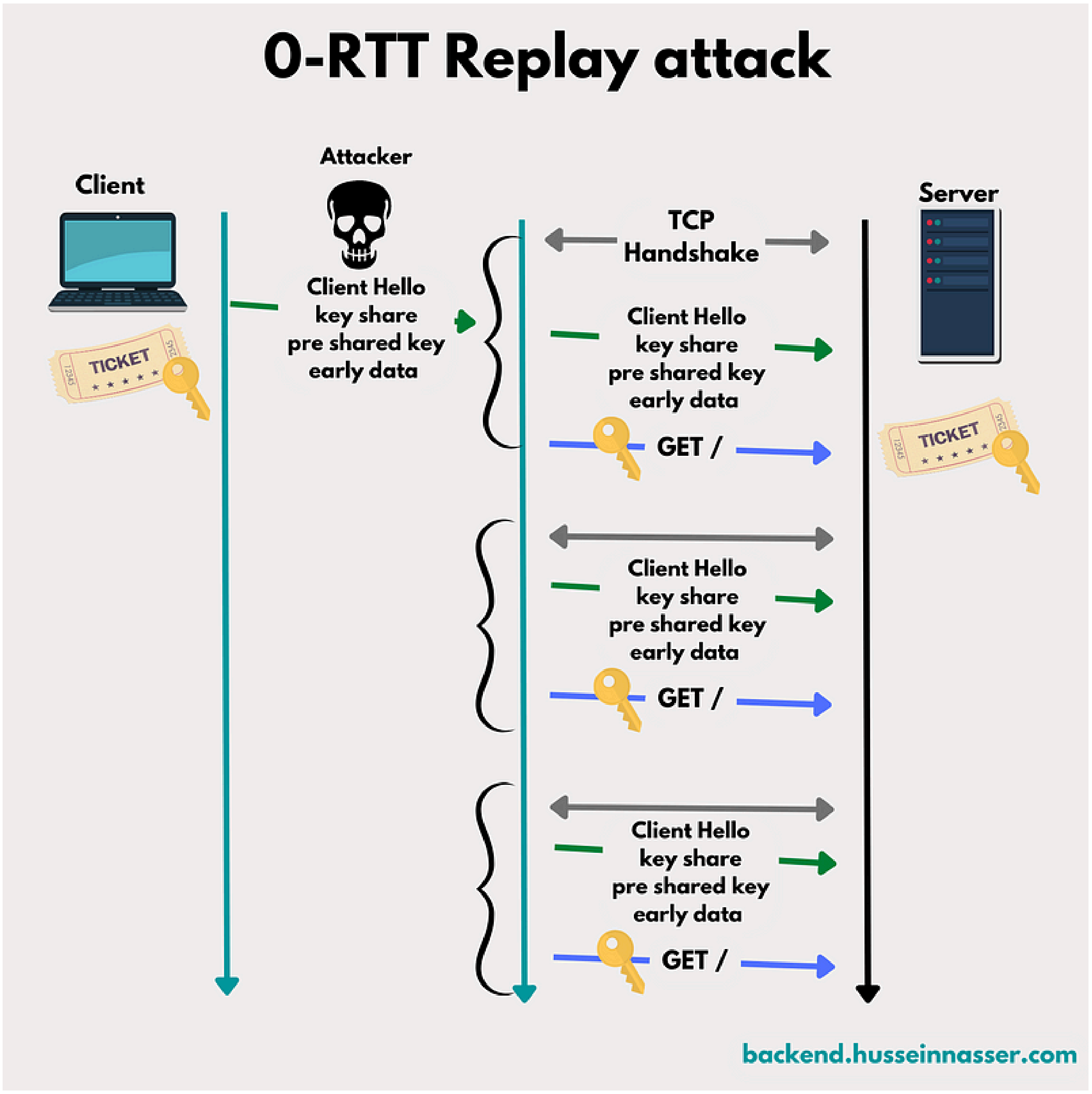
The danger of TLS Zero RTT

Why early data is subject to Replay attacks



TLS 1.3 had the clever idea to separate the single negotiation of a cipher suite into three negotiations for key exchange, symmetric cipher and digital signature. This allowed the TLS 1.3 to shave an entire round trip reducing it to 1-RTT (round trip time).

The client proposes its own key share parameters, symmetric ciphers and signiture algorithm. The server picks the preferred key exchange and cipher, the server sends back its parameter to the client. both the server and the client has the full secret that is only known by the server and the client. The secret is then fed to the selected symmetric key cipher to produce the session key that will be used for encryption.

The PSK (Pre-shared-key) extension allows both client and server to skip the authentication part where the server sends its certificate and just perform the handshake to get a new session key. Optionally, they may choose skip the handshake itself to save some CPU cycles by reusing the same secret they already know sacrificing forward secrecy.

TLS 1.3 allows you to combine early data extension with the PSK shaving the remaining round trip down to what is known as 0-RTT. This basically means the client can send data immediately as I talked about this in another post.

0-RTT does come at a security cost of replay attacks, and the main question I try to answer in this post is why is replay attacks a problem with 0-RTT requests but not a problem with normal TLS traffic?

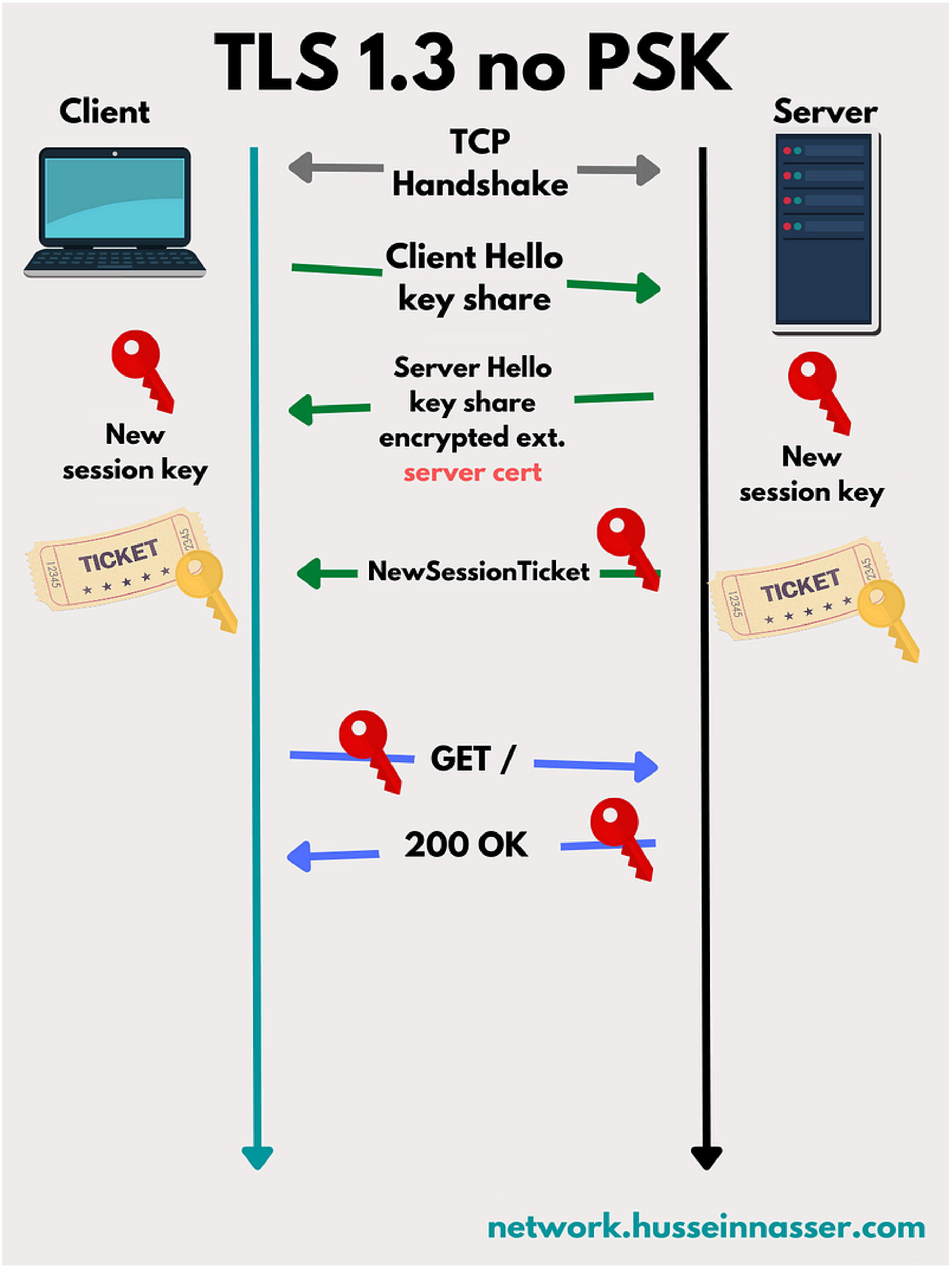
Pre-Shared-Key and Tickets

Exchanging keys takes time and resources. The Pre-Shared-Key allows both the client to store a pre-shared key for use in future sessions just in case the connection (be it TCP or QUIC) is lost or if the client wants to establish more than one connection for demultiplexing requests as its the case with HTTP/1.1.

The problem is state. A client need to remember one key per server while a server with tens of thousands of clients need to remember far many keys. Sure you can have a backend database and cache keys to oblivion but that adds scaling and complexity and we know how to deal with that with our backend engineering skills. But is it possible to skip this all together?

We can let the server forget about the key and have only the client remember it. The client will then send the key back to the server when it wants to resume the TLS session and server can verify if that key is indeed correct. The key—of course—cannot be sent in plain text it needs to be encrypted by the server such that only the server can decrypt it.

Both approaches are possible in TLS 1.3 with New Session Ticket option, the new session ticket will have a payload called ticket label, you can choose to embed an encrypted key or a session id that the server can look up in a database.

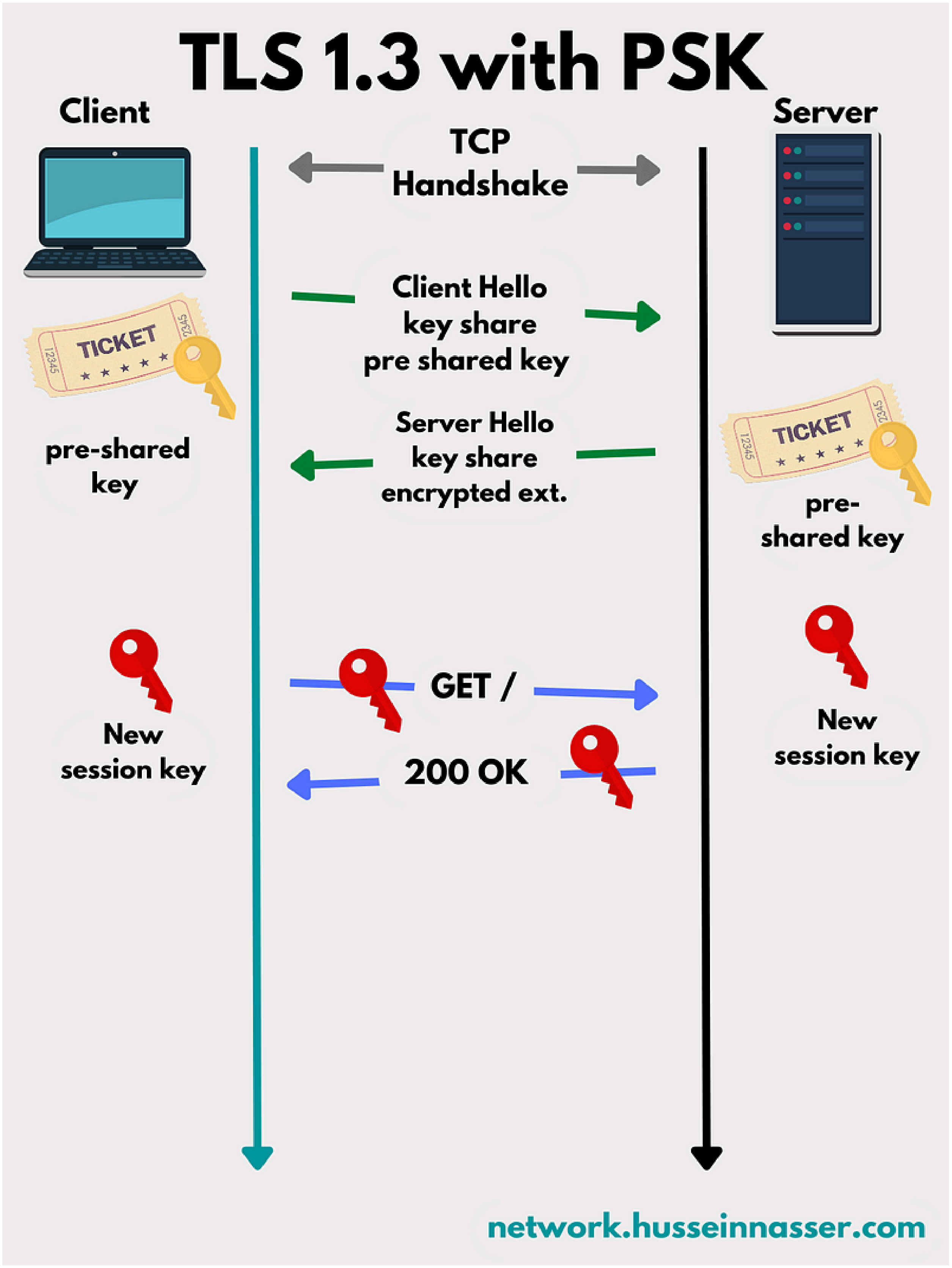


Session Ticket generation—Full handshake

When choosing to embed the resumption key inline in the ticket label, that key must be encrypted with a key known only to the server. It could be the server's private key but that isn't a good idea, if server private key is leaked all past recorded resumed sessions from all clients can be decrypted.

Alternatively, the server can generate temporary keys and rotate them every hour or so, thats how Cloudflare does it. Worth noting that the new session ticket message is encrypted with the normal session key.

Regardless how the server gets the pre-shared key, that knowledge alone is enough for the server to trust the client and cryptographically link the new connection to old session. This allows the server to skip parts of the TLS handshake such as sending the certificate which we know can be huge. Optionally the server and the client can agree to skip even the handshake which will save more CPU cycles speeding up resumption.



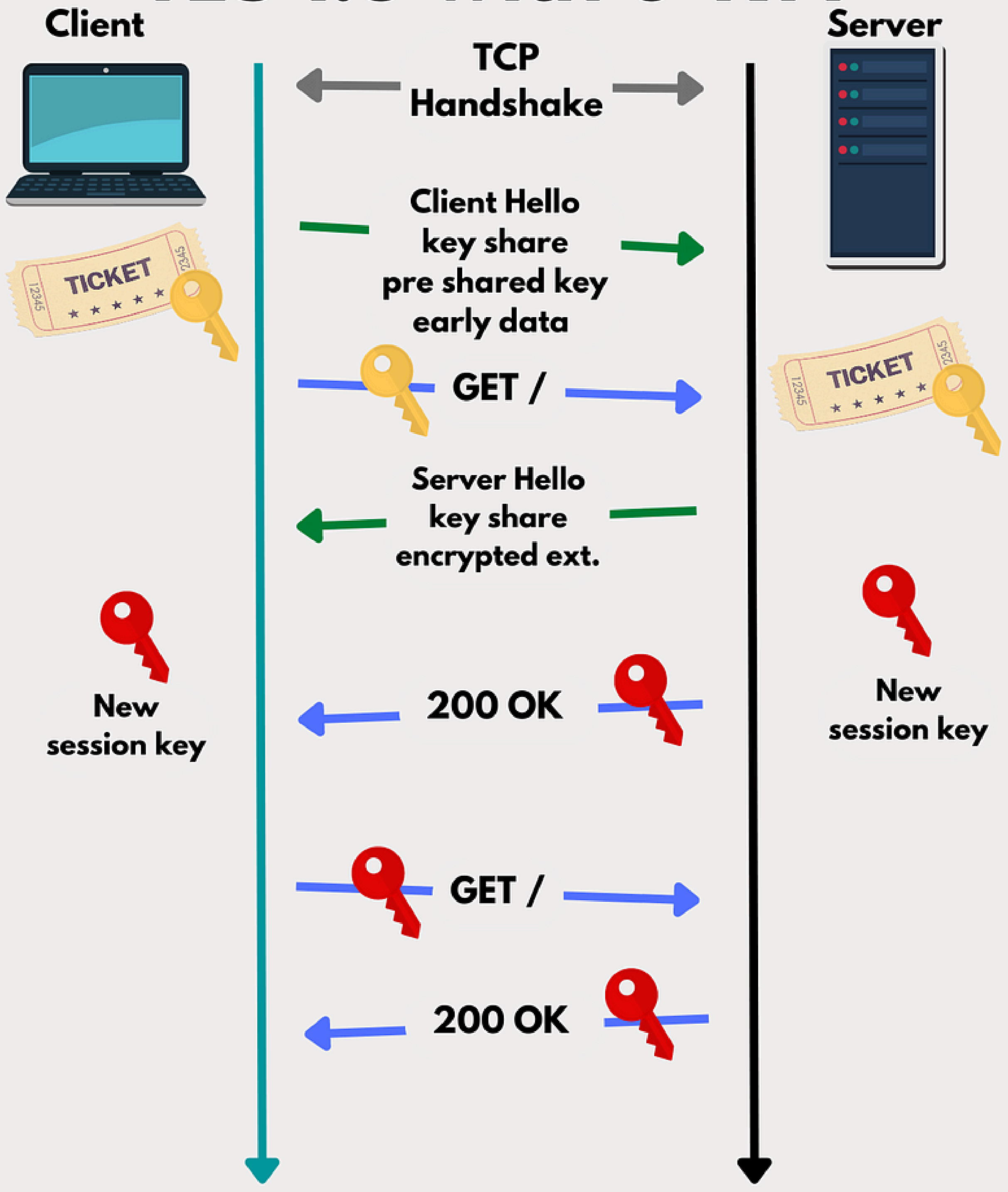
TLS Session Resumption

Theoretically you can have the client do the full handshake and get a session ticket and then resume 10 other connections with the same session ticket. I see value in that with HTTP/1.1 connections but not much for HTTP/2 and HTTP/3 as we can simply create new streams in the same connection.

0-RTT vs 1-RTT Request

PSK is useful for resuming TLS sessions and speeding up connections. However, we can do more. What if we used that known shared secret that both the client and server know to encrypt the first flight request and send it right after the client hello in a new session resumption?

TLS 1.3 with 0-RTT



network.husseinnasser.com

This allows for the application to immediately send its request before the TLS session is even established, yes we still need to establish the TCP connection in case of HTTP/2 or HTTP/1.1 but with HTTP/3 it is a true 0-RTT as QUIC connection establishment and TLS is done in same round trip.

Replay attacks

A replay attack happens when a MITM attacker can see the encrypted traffic but can't really change it. If the attacker knows that an encrypted TLS segment has an HTTP request, they can replay the same segment to the server causing the HTTP request to re-execute.

Now I really don't like when I read this simplistic statements made on security web sites "oh attacker simply replays the request". Replaying a packet as a passive observer requires work. The attacker has the transport layer to consider, sequences to sync up and hashes and checksums to compute. If the attacker knows that a TCP segment contains an encrypted TLS segment with a request, the attacker can copy the data portion of that segment but then need to craft a new TCP segment with same port/ips and basically repopulate the new TCP sequence numbers as if its a new packet sent from the client. This can only happen of course if the MITM is the path.

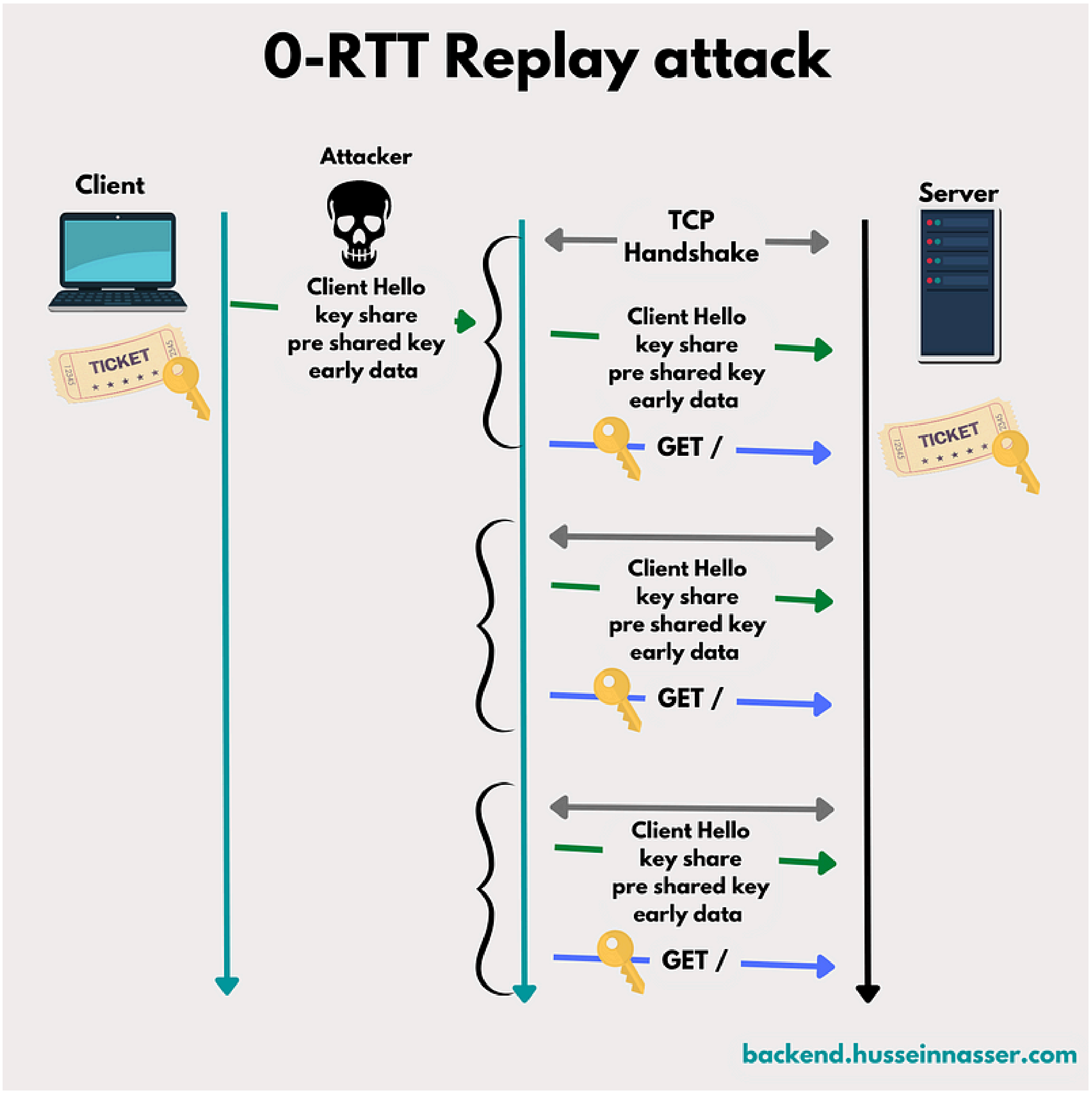
However, even if a sophisticated attacker manage to craft and replay an encrypted request the server will still reject it. You might say why? That is because each TLS segment is labeled with a unique sequence number that is monotonically increasing. This 64 bit sequence number is completely different than sequence number in the TCP transport layer. If the server received a sequence number that it already processed it will simply discard the packet, this is a feature in TLS, you can't replay TLS segments.

> You might say how expensive it is to check if a sequence was processed? Does the server has to keep track of all sequences it ever processed for this connection and do a lookup? Fortunately no, the server can keep the latest sequence and compare the incoming TLS segment sequence numbers against it, if the incoming sequence is larger then the current one its a new one, if its smaller its a replay or duplicate. Out of order segments will be handled by TCP in this case. This does leave UDP with DTLS ? Which I think is a problem in that case. I just didn't research DTLS this deep enough.

0-RTT Replay Attacks

0-RTT is a different beast and that is why everyone talks about Replay attacks in the context of 0-RTT and not normal TLS. When the client sends the client-hello with PSK and early data, it already encrypted and signed the request with PSK-binder to prevent attackers changing the client-hello.

However, the attacker can still intercept the packet, turn around and establish a brand new connection as the attacker to the server, copy the client hello and send it to the server, then copies the 0-RTT request and send it to the server. The server upon accepting the PSK will process the 0-RTT request because guess what, the sequence number of TLS starts with 0 for new connections. As long as the attacker creates new connection, replays the client-hello and the 0-RTT data, the sequence will always be 0 and the server will have no idea that this is a replayed segment.



If the attacker does this multiple times it will cause the server to re-execute requests sent in 0-RTT. What is wrong with that? If it is a GET request, nothing, but if it is a non-idempotent POST request that changes the state in the database than it may be a problem. This is why making your backend APIs idempotent is critical.

Servers are smart to only allow GET with no parameters to be executed when in 0-RTT data.

Many solutions have been proposed to solve this:

- Build your backend APIs such that they are idempotent
- Reject anything and only allow GET with no query params
- Allow one session ticket redemption
- Hash and store x number of client-hellos and check if the server ever seen the client hello.

RFC 8470 is dedicated for this here.

Summary

0-RTT is a powerful feature. Sending request at the same breath with establishing the TLS session, this makes encrypted session almost as fast as 2000s unencrypted traffic. It does come with its cost of replay attacks just because the attacker early data is the first request which has a TLS sequence of 0. RFC8470 tries to mitigate this.

I see great value in IOT and low resources devices enabling 0-RTT. The other value shines really when clients move from one interface to another (WIFI-5G) and causes a disconnect, QUIC handles this better then TCP but the value of 0-RTT in both is valuable. Not to minimize the dangers though.

If you like this post consider checking out my courses.