Problem Statement

HRR ambiguity

Was ECH accepted at the point where you process HRR?

- https://github.com/tlswg/draft-ietf-tls-esni/issues/233
- https://github.com/tlswg/draft-ietf-tls-esni/issues/373

RFC8446 HRR compliance

Do changes in HRR impact inner ECH, which seems to violate RFC 8446; S 4.1.2:

- https://github.com/tlswg/draft-ietf-tls-esni/issues/358

Stateless HRR

How does the client-facing server handle ECH statelessly if the back-end sends HRR?

- https://github.com/tlswg/draft-ietf-tls-esni/issues/333

Assumptions

- The double cookie problem is out-of-scope.
- Cookies may leak information about the backend server.
- Backend server changes are in scope. (The backend server is ECH-aware and must modify the ServerHello (or HRR) accordingly, to confirm acceptance, based on ClientHelloInner.)
- HRR language in RFC8446 is malleable, up to a certain extent. (HRR rules in RFC8446(bis) may change if needed.
- Changes that might cause bugs in brittle TLS stacks are out-of-scope. (E.g., LibreSSL doesn't handle HRR quite right, and we don't want ECH to knock it over.)
- HRR in the presence of ECH can “stick out.” (ECH is allowed to stick out after HRR only if HRR was inevitable for the given client and server.)
- HRR is exercised very little in practice.

Design Variants

Option (1): Signal Acceptance in HRR

Summary: Include an explicit signal in the backend-generated HRR extension when ECH is used.

Pros:

- Builds on existing machinery already in ECH (acceptance signal computation) with no new significant backend requirements.

Cons:

- Changes difficult-to-test code that is barely exercised. (The change is less invasive than (1), though.)
- Cookies may leak information about the backend server.

Option (2): Consistent ClientHelloOuter/Inner

Summary: Clients construct ClientHelloOuter/Inner such that “HRR-relevant” parameters match.

Pros:

- Most stacks will construct ClientHello messages this way.
- Simplest change (code-wise).
- Doesn’t stick out (but we don’t care about that).

Cons:

- Difficult to specify future-proof criteria for what constitutes an HRR-relevant parameter. (Will we actually have such parameters in the future?) Failure to do this correctly may cause disambiguation failures.
- Cookies may leak information about the backend server.

Assessment and Recommendations

After several discussions, the design team concluded that Option (1) was the better outcome because protocol changes that avoid the HRR ambiguity via an explicit signal (Option 1) instead of specification changes that avoid it (Option 2) were preferred. The design team was unable to come up with language that everyone felt comfortable with for Option 2.

The specifics of Option 1, i.e., using an extension that carries the acceptance signal for HRR rather than overloading HRR.Random, were also discussed amongst the team. In converging on Option 1, the team asked two fundamental questions: how is the signal computed, and where does the signal go? The team feels confident that the current signal computation achieves the desired security goals. Currently, the non-HRR acceptance signal is included in the ServerHello.random, whereas Option (1) places the HRR acceptance signal in a backend-generated “encrypted_client_hello” extension. This naturally led to the question: *why are these signals different for the HRR and non-HRR path?*

First, the client act of HRR is not a secret to passive eavesdroppers, so hiding the signal had no security benefit. Second, use of an extension in the HRR acceptance case is due to RFC8446 compliance. In particular, clients are only expected to act on HRR if the ServerHello.Random is fixed to the sentinel value specified in RFC8446. From this, we concluded that using an extension for the HRR acceptance signal was best.

This left open the question of where the non-HRR acceptance signal goes. Putting this signal in a ServerHello extension allows passive eavesdroppers to learn when a server is capable of ECH. In contrast, putting the signal in ServerHello.Random does not allow this sort of inference.

However, using ServerHello.Random risks HRR viability when used with ECH. In particular, if middleboxes react negatively to the presence of an extension, then use of HRR might break when used with ECH. (Middleboxes would not interrupt non-HRR ECH connections, but might interrupt HRR ECH connections.) Moving this signal to a ServerHello extension would reveal such middlebox issues early in deployment, allowing them to be fixed, at the cost of passively inferring whether a server is capable of ECH. (Note that whether or not a server is capable of ECH depends on the attacker's ability to actively probe, and active attackers are consistent with the ECH threat model.)

Based on the team's conclusion and discussion, the following action items were identified:

1. Merge PRs to address Option (1):
    a. https://github.com/tlswg/draft-ietf-tls-esni/pull/423
    b. https://github.com/tlswg/draft-ietf-tls-esni/pull/422
2. File a new issue to discuss HRR mitigations (within the context of ECH)
3. File a new issue to discuss the non-HRR ECH acceptance signal placement