The Token Binding Protocol

V. Anupam, Google Inc.
A. Popov, Microsoft Corp.

The Problem: Bearer Tokens

- Web services generate various security tokens (HTTP cookies, OAuth tokens) for web applications to access protected resources.
- Currently these are bearer tokens, i.e. any party in possession of such token gains access to the protected resource.
- Attackers export bearer tokens from the user's machine, present them to web services, and impersonate authenticated users.
- The Token Binding protocol can be used to create long-lived TLS bindings spanning multiple TLS sessions and connections.
- Applications are then enabled to cryptographically bind security tokens to the TLS layer, preventing token export and replay attacks.

Establishing a Token Binding

- The user agent generates a private-public key pair (possibly within a secure hardware module, such as TPM). Different keys SHOULD be used by the client for connections to different servers, according to the token scoping rules of the application protocol.

- The user agent proves possession of the private key on every TLS connection to the target server.

- The proof of possession involves signing the tls_unique value [RFC5929] for the TLS connection with the private key.

- TLS Token Binding is identified by the corresponding public key.

- TLS Token Bindings are long-lived, i.e. they encompass multiple TLS connections and TLS sessions between a given client and server.

Preventing Token Theft

- When issuing a security token to a client that supports TLS Token Binding, a server includes the client's TLS Token Binding ID in the token.
- Later on, when a client presents a security token containing a TLS Token Binding ID, the server makes sure the ID in the token matches the ID of the TLS Token Binding established with the client.
- In the case of a mismatch, the server discards the token.
- In order to successfully export and replay a bound security token, the attacker needs to also be able to export the client's private key, which is hard to do in the case of a key generated in a secure hardware module.

Negotiating the Token Binding Protocol

- The client and server use ALPN protocol IDs [RFC7301] to negotiate the use of the Token Binding protocol, in addition to the actual application protocol.

- ALPN IDs are also used to negotiate the parameters (signature algorithm, length) of the Token Binding key.

ALPN ID	Protocol
h2_tb_p256	HTTP/2 with Token Binding using ECDSA key and NIST P256 curve
h2_tb_rsa2048	HTTP/2 with Token Binding using 2048-bit RSA key
http/1.1_tb_p256	HTTP/1.1 with Token Binding using ECDSA key and NIST P256 curve
http/1.1_tb_rsa2048	HTTP/1.1 with Token Binding using 2048-bit RSA key

- This negotiation does not require TLS protocol changes or additional round-trips.

The Token Binding Protocol

- The Token Binding protocol consists of one message sent by the client to the server, proving possession of one or more client-generated asymmetric keys.
- This message is only sent if the client and server agree on the use of the Token Binding protocol and the key parameters.
- The Token Binding message is sent with the application protocol data in TLS application_data records.
- A server receiving the Token Binding message verifies that the key parameters in the message match the Token Binding parameters negotiated via ALPN, and then validates the signatures contained in the Token Binding message. If either of these checks fails, the server terminates the connection.
- When a server receives a bound token, the server compares the TLS Token Binding ID in the security token with the TLS Token Binding ID established with the client. If the bound token came from a TLS connection without a Token Binding, or if the IDs don't match, the token is discarded.

Token Binding Protocol Message Format

```
struct {
    ExtensionType extension_type;
    opaque extension_data<0..2^16-1>;
} Extension;

struct {
    TokenBindingID tokenbindingid;
    opaque signature<0..2^16-1>;       // Signature over hashed ("token binding", tls_unique)
    Extension extensions<0..2^16-1>;
} TokenBinding;

struct {
    TokenBinding tokenbindings<0..2^16-1>;
} TokenBindingMessage;
```

Token Binding ID Format

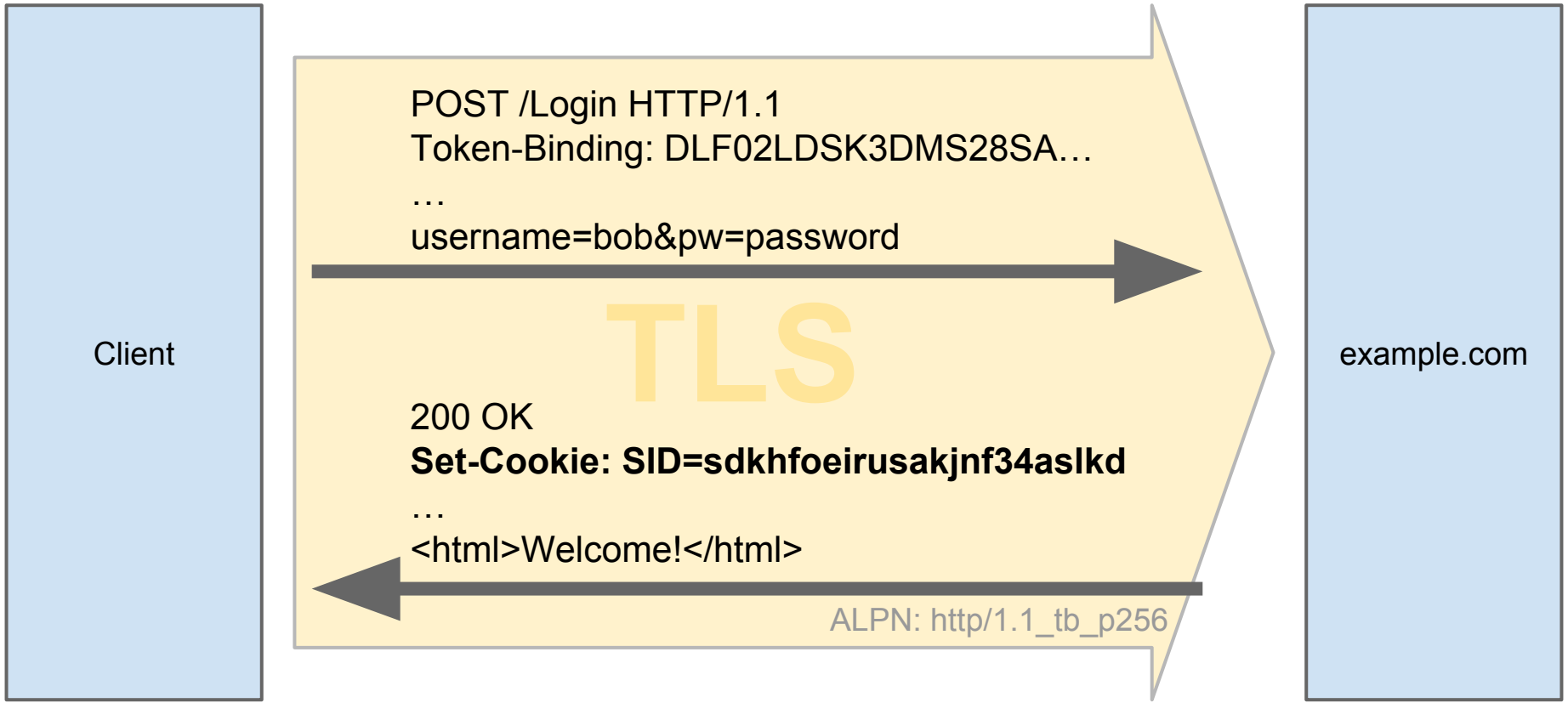
```
enum {
    provided_token_binding(0), referred_token_binding(1), (255)
} TokenBindingType;
struct {
    TokenBindingType tokenbinding_type;
    SignatureAndHashAlgorithm algorithm;
    select (algorithm.signature) {
        case rsa: RSAPublicKey rsapubkey;
        case ecdsa: ECDSAParams ecdsaparams;
    }
} TokenBindingID;
```

- provided_token_binding is used to establish a Token Binding when connecting to a server.
- referred_token_binding is used when requesting tokens to be presented to a different server.

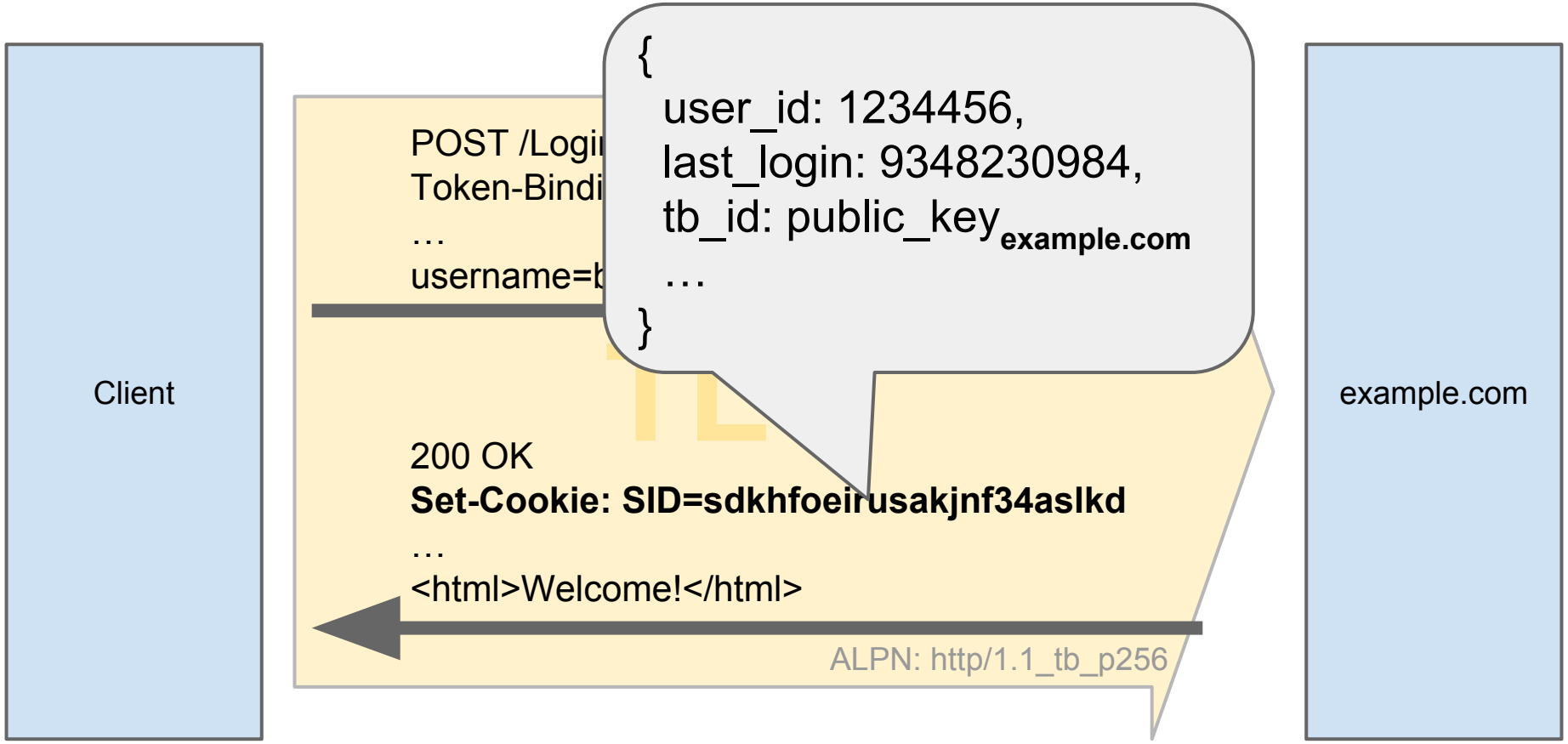
Token Binding Header

Token Binding Header

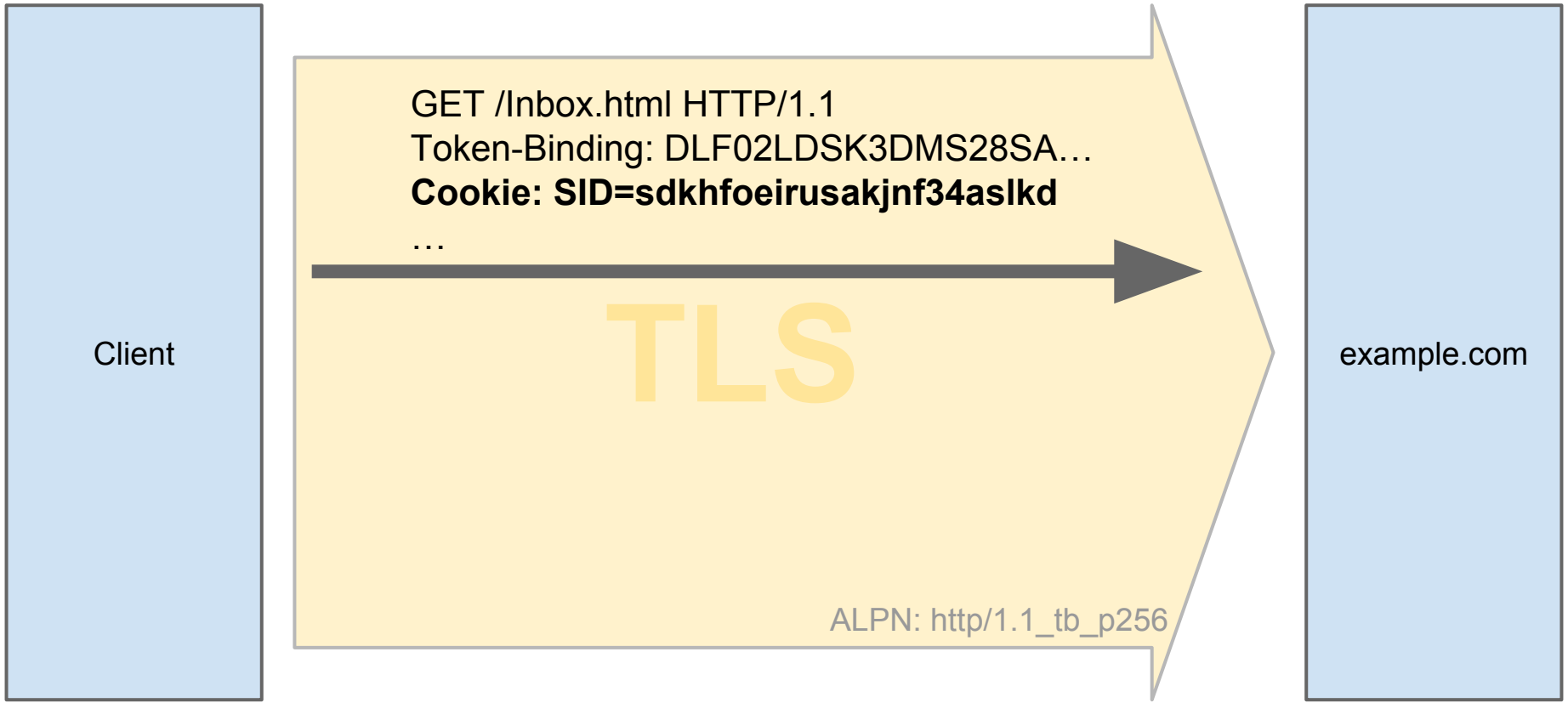
First-Party Binding



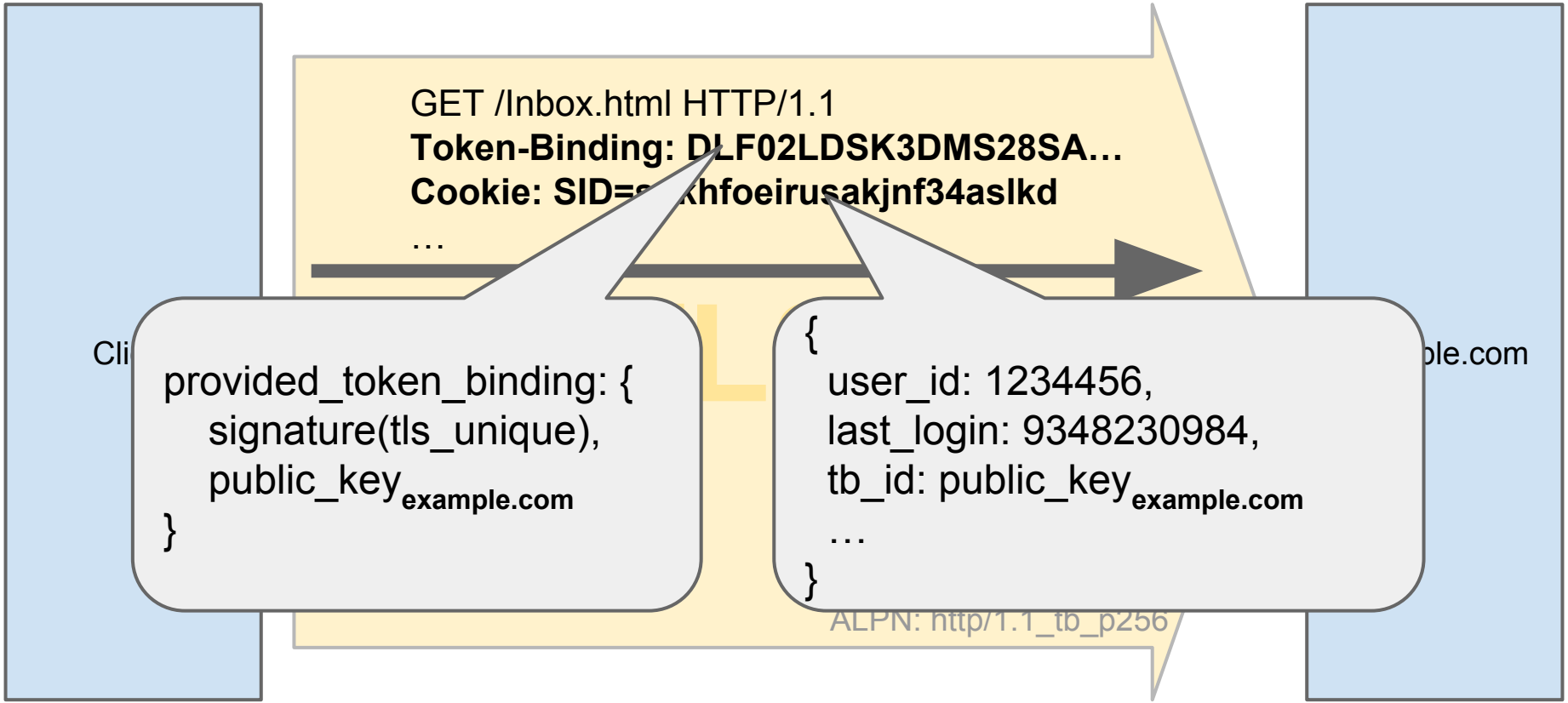
First-Party Binding



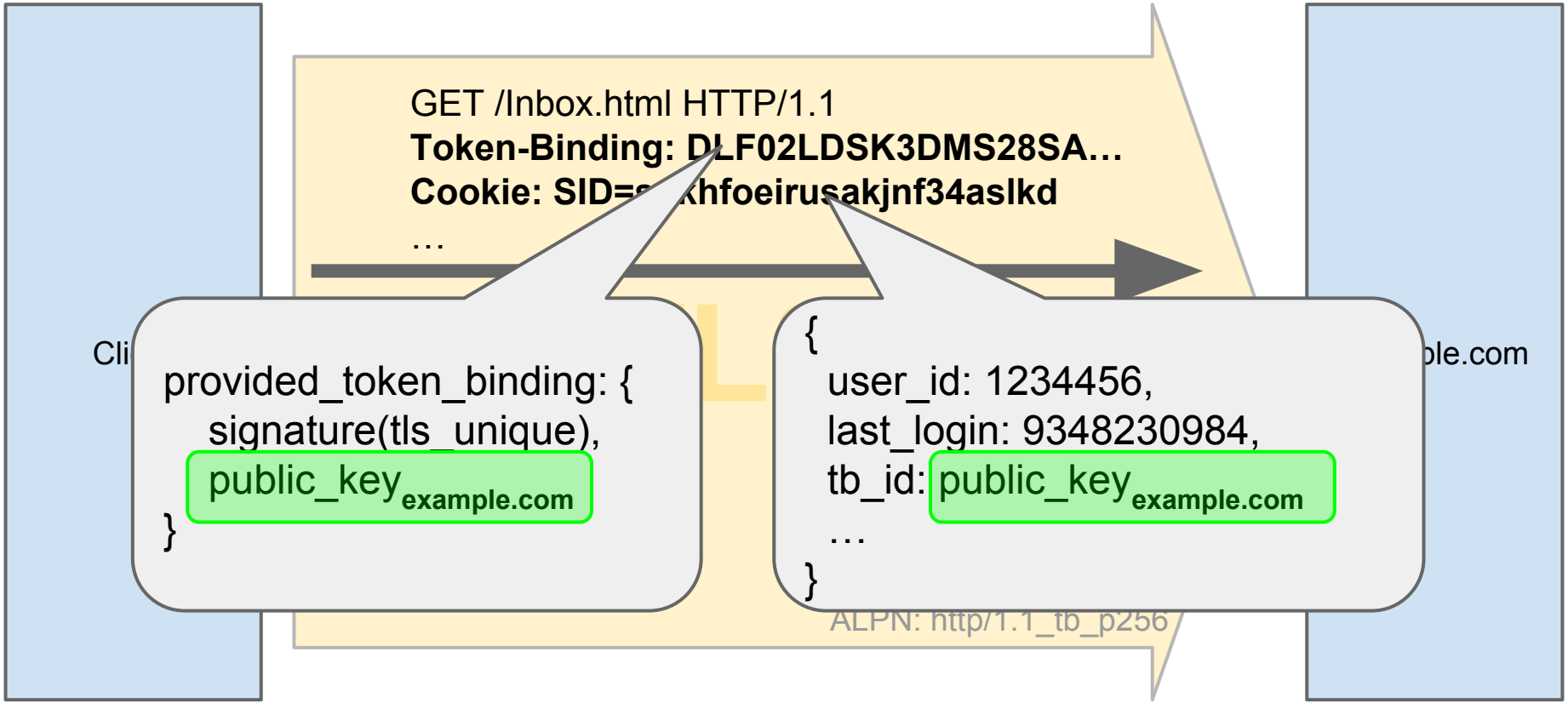
First-Party Binding



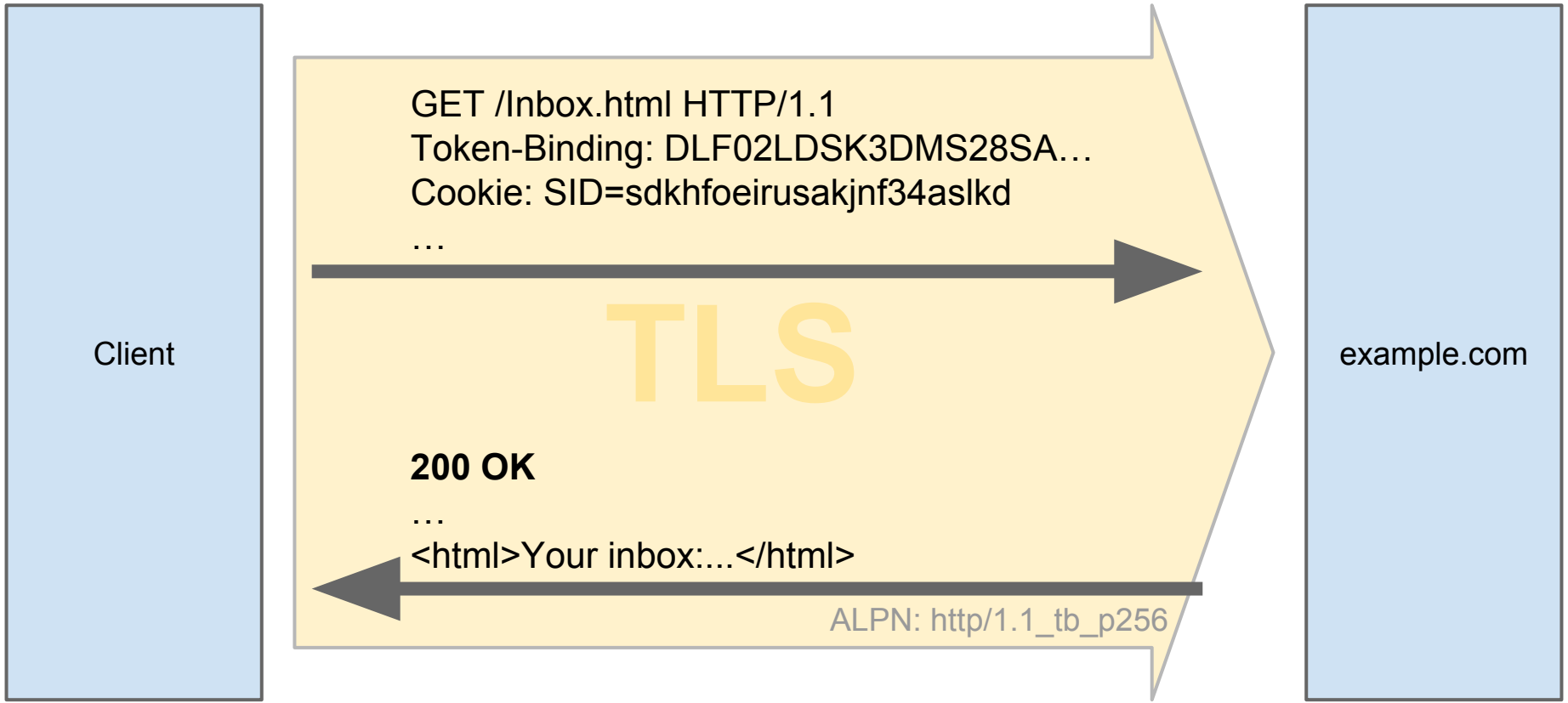
First-Party Binding



First-Party Binding



First-Party Binding



Why Federation Support?

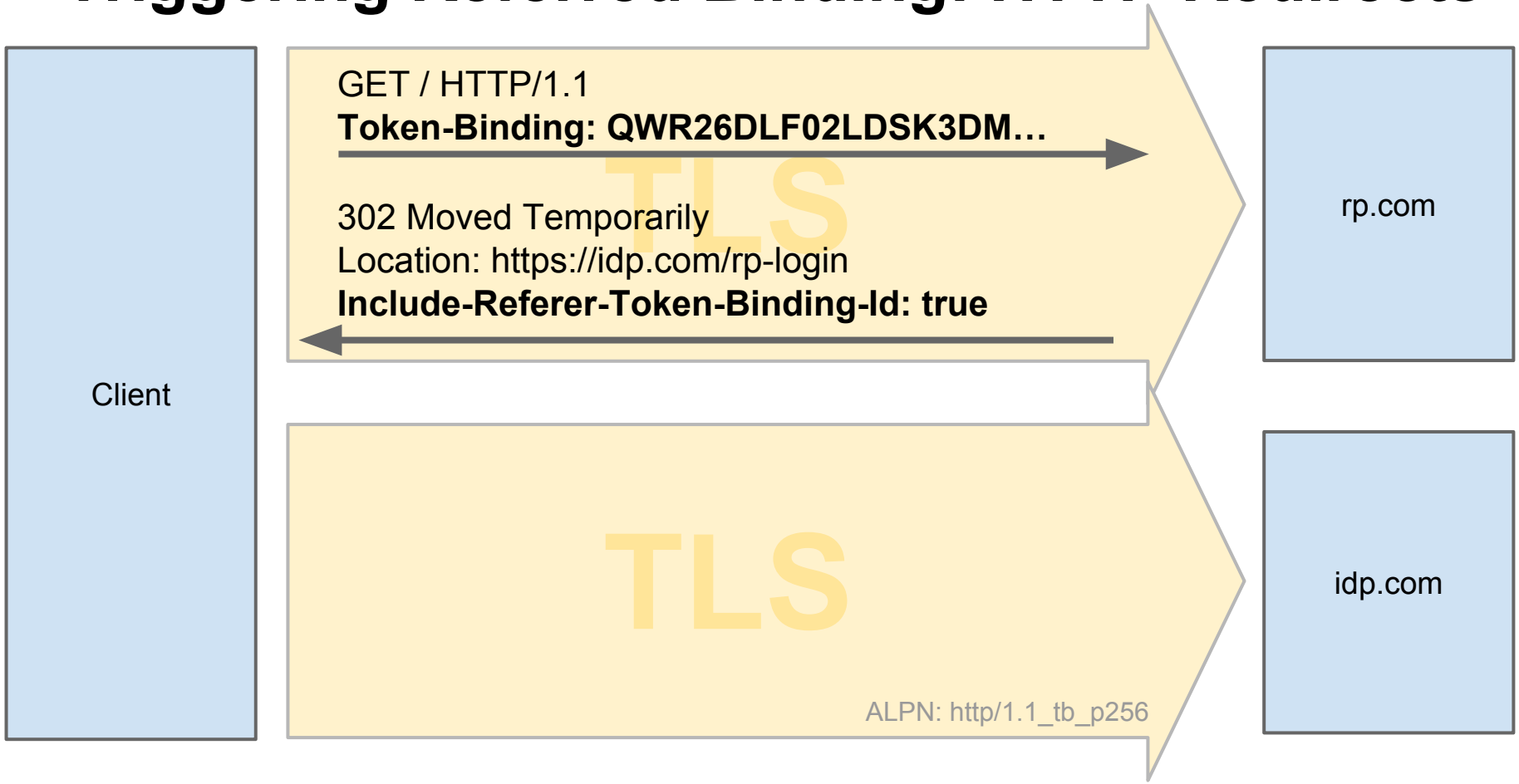
- In Federated Sign-On Protocols (e.g., OpenId Connect)
  - Client gets identity token from Identity Provider and
  - Presents it to Relying Party

- Client uses different Token Binding Keys for IdP & RP
  - RP is typically in different domain than IdP

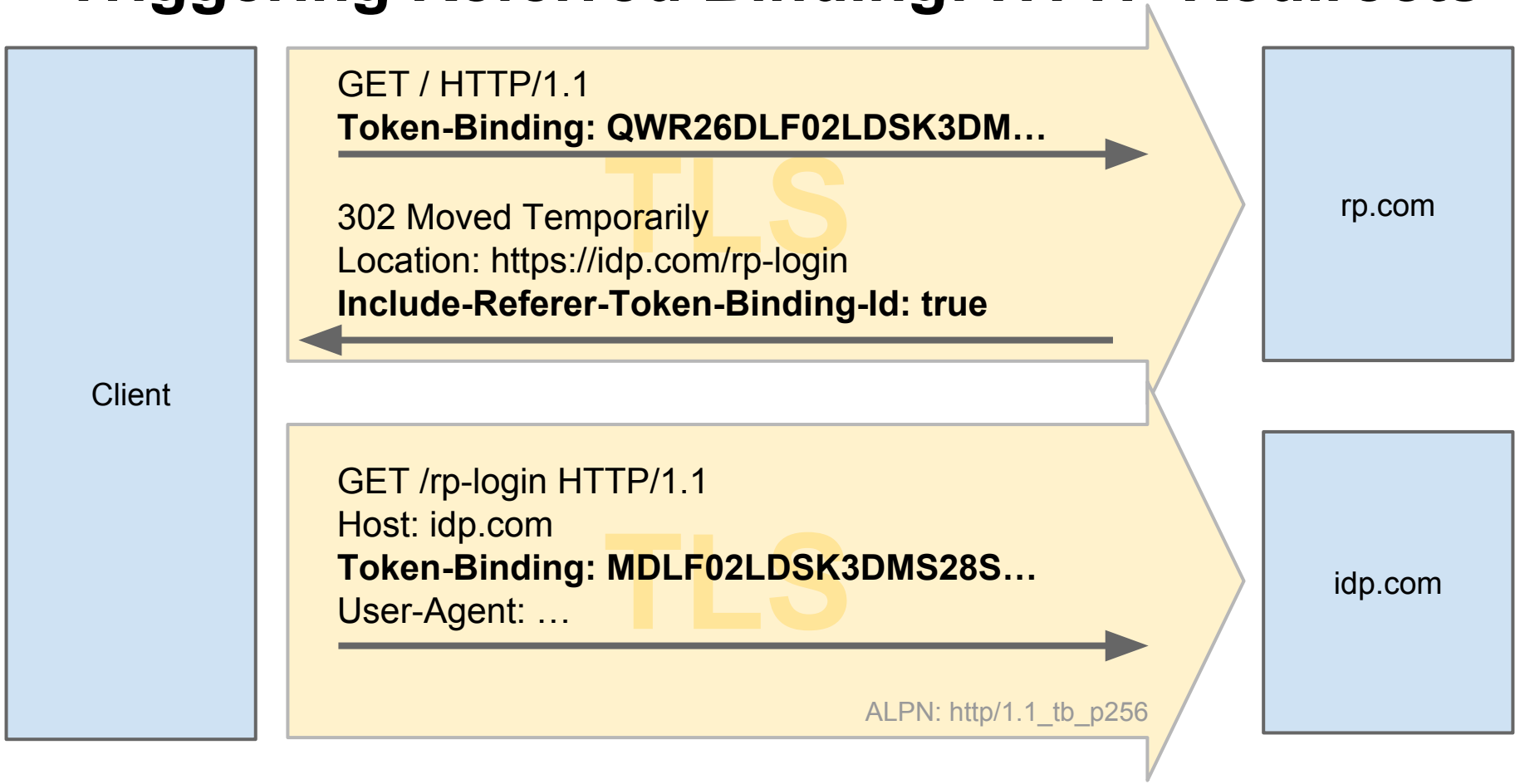
- Identity token from IdP is bound to client key for IdP
- Token will not pass validation by RP

- Need to enable IdP to issue token bound to RP

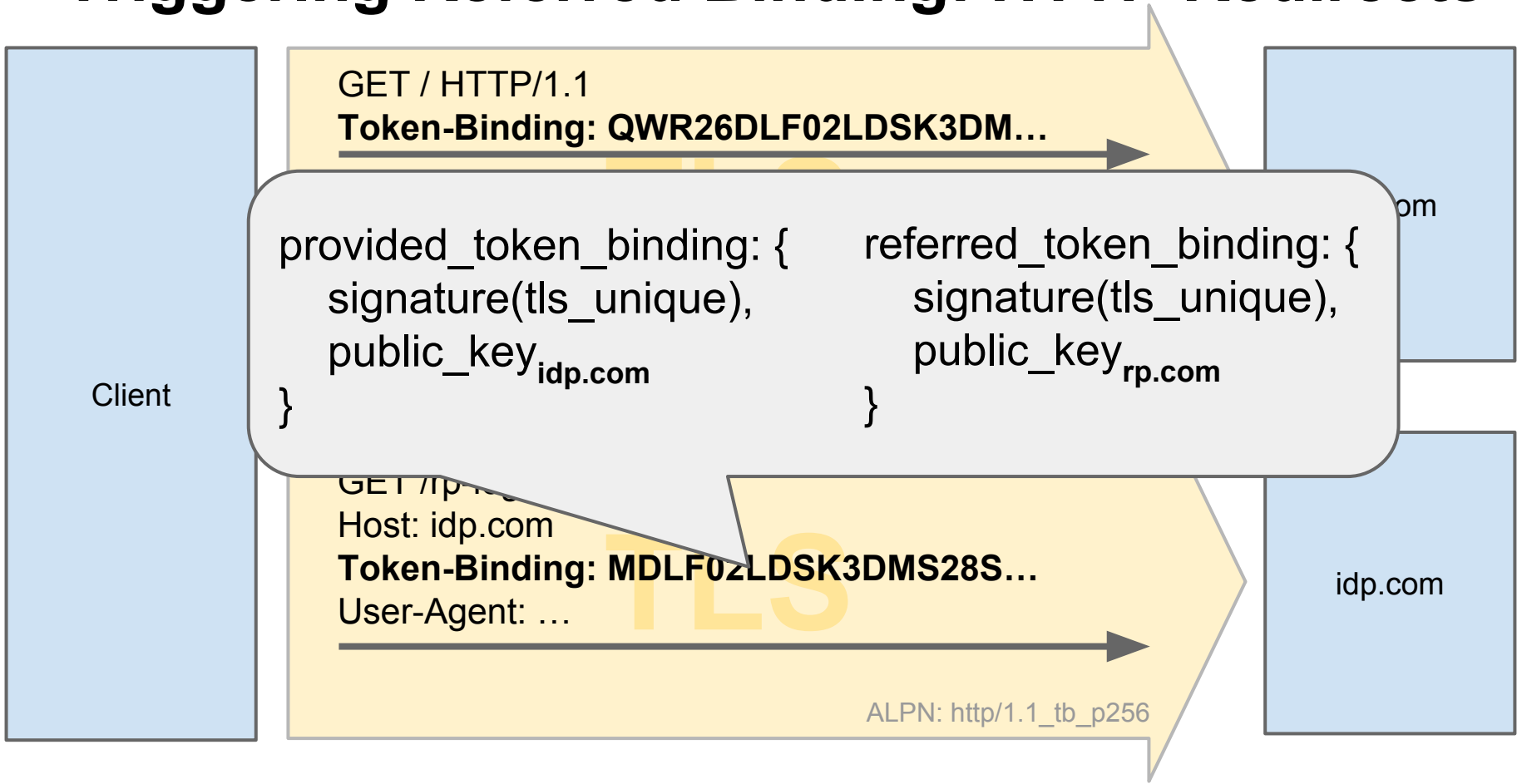
Triggering Referred Binding: HTTP Redirects



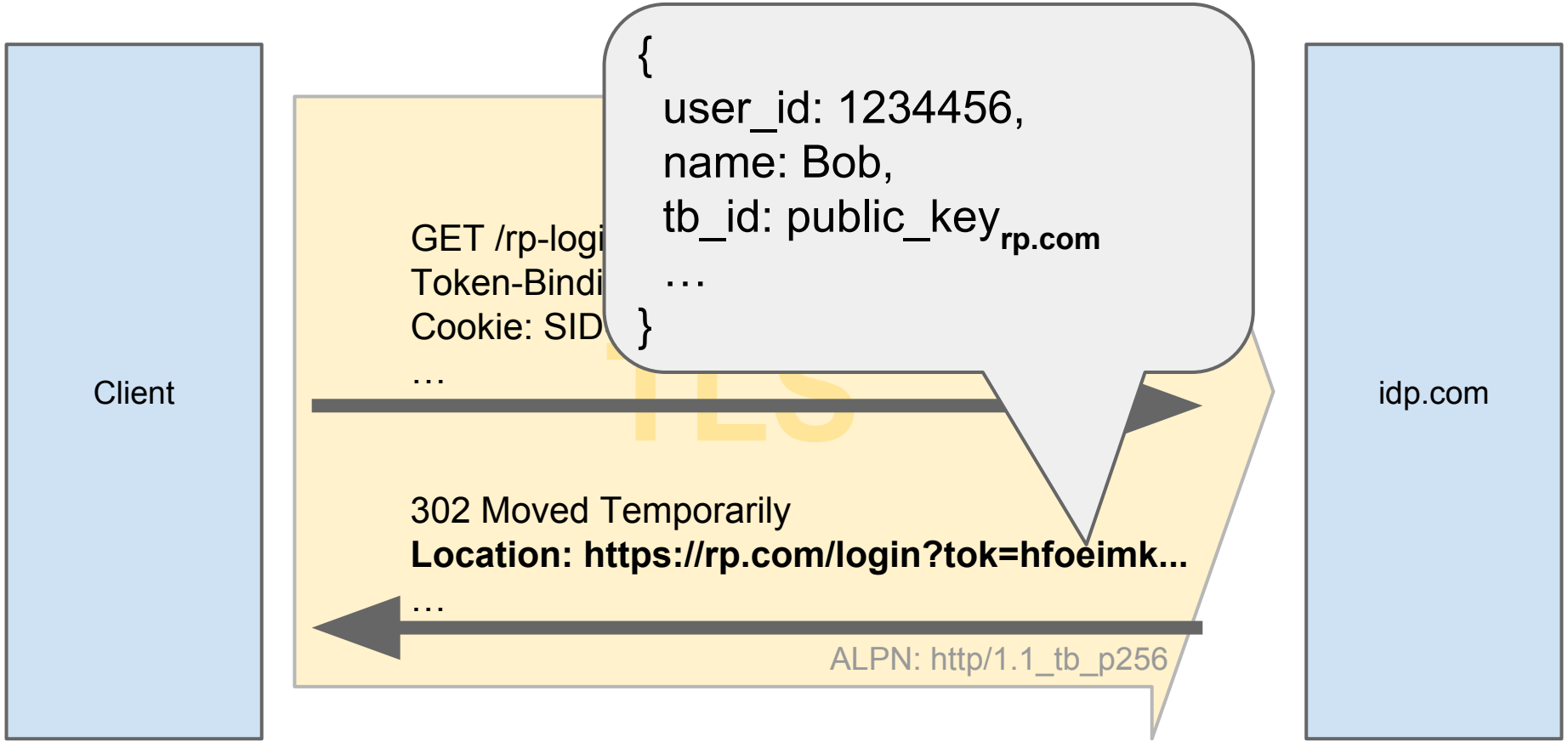
Triggering Referred Binding: HTTP Redirects



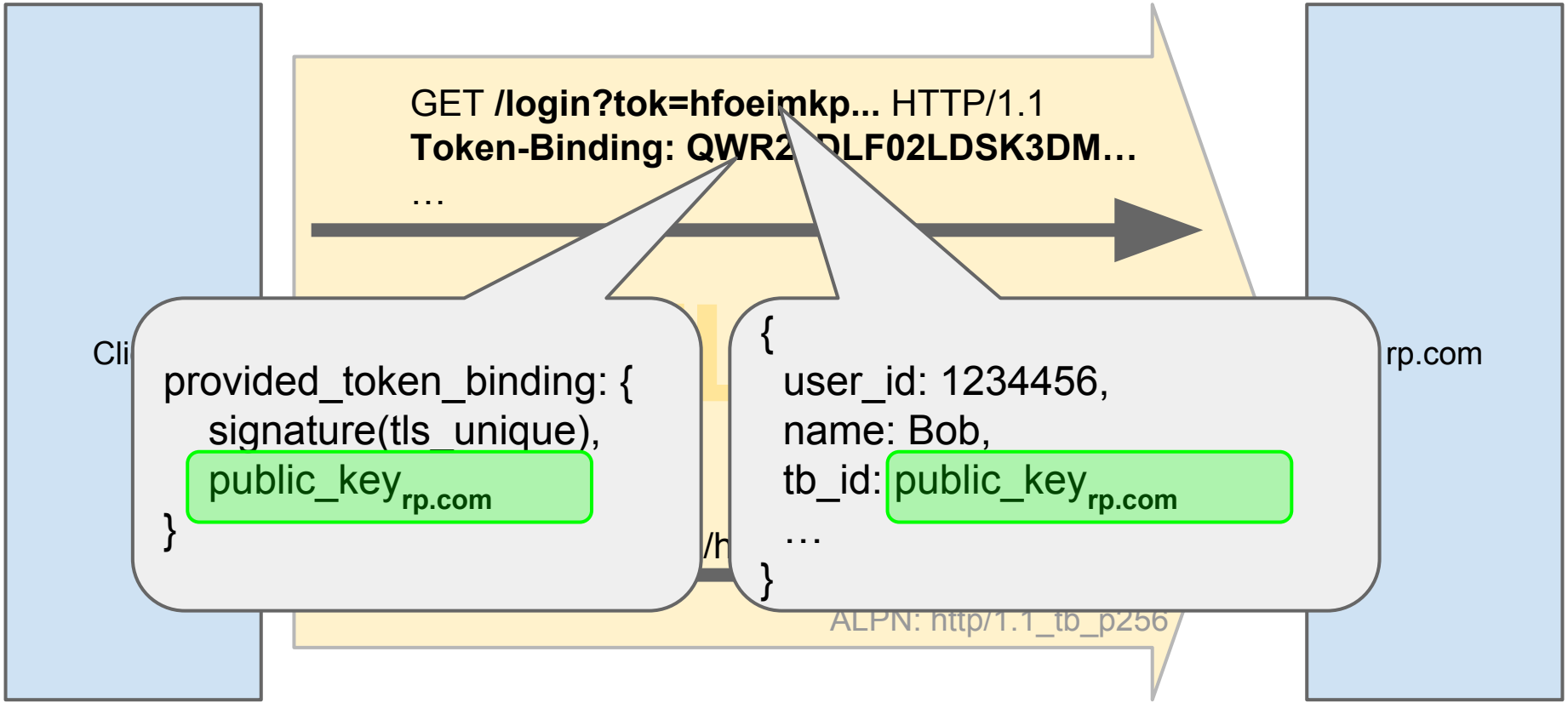
Triggering Referred Binding: HTTP Redirects



Federated Binding



Federated Binding



Triggering Referred Binding: XHR

- RP uses new property of XmlHttpRequest to make client ask IdP for RP-bound token

```
var xhr = new XmlHttpRequest();
xhr.withCredentials = true; // send cookies
xhr.withRefererTokenBindingId = true;
xhr.open(method, url, async);
```

- If property is true, Client includes `referred_token_binding` in `Token-Binding` header to IdP

Privacy Considerations

Clients must

- Use different Token Binding Ids for different eTLDs
  - Protects against cross-domain linking of user identities
- Let users manage TB Ids like cookies
  - They are persistent identifiers that are automatically presented by clients to identify themselves to servers
  - TB Ids must obey all user-specified cookie control settings
  - TB Ids can be cleared by user at any time
- Never transmit Token Binding Ids in cleartext

Security Considerations

Clients must

- Negotiate Enhanced Master Secret Extension
  - Protocol uses `tls_unique` to create Token Binding Ids
  - EMS Extension protects against Triple Handshake Attack which allows propagation of `tls_unique` value into other TLS connections & thus could allow bound tokens to be used over such connections
- Protect Token Binding private keys
  - Bound tokens cannot be used by clients without right private keys
  - Ideally, clients *should* use hardware security modules to prevent extraction of private keys

Links And Contact Information

- The Token Binding Protocol Version 1.0: http://tools.ietf.org/html/draft-popov-token-binding-00

- Token Binding over HTTP:http://tools.ietf.org/html/draft-balfanz-https-token-binding-00

- Dirk Balfanz balfanz@google.com
- Vinod Anupam vanupam@google.com
- Andrei Popov andreipo@microsoft.com