Notes on Deploying STIR/SHAKEN Certificate Management

Chris Wendt & David Hancock

Contents

- Managing STI Certs in initial STI Deployments
- Using HTTP caching at STI-VS
- Background Information (HTTP caching details)

Managing STI Certs in initial STI Deployments

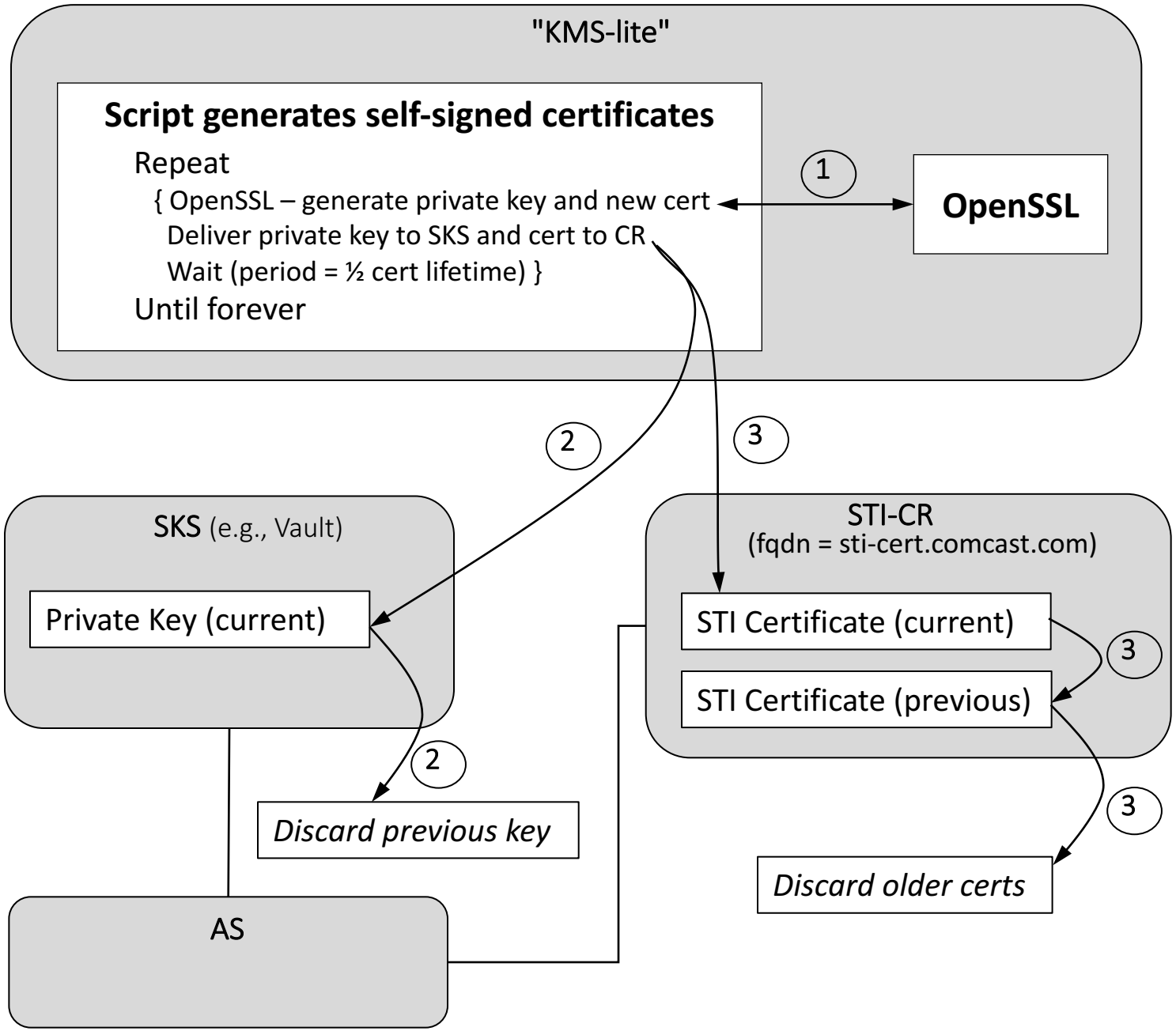
Proposal for Initial Phase Cert Management

Before STI-PA and STI-CAs are deployed and available

- Each Service Provider coins its own self-signed certs
  - STI authentication follows normal SHAKEN procedures, except the STI-AS uses self-signed cert, instead of an STI cert that chains to the STI-CA

- Since the self-signed cert can’t be validated by chaining it to a trusted root, the STI-VS maintains a white-list of valid STI-CR FQDNs
  - Long-term, this won’t scale to support a large number of SPs, but should be workable in the short-to-medium term

SHAKEN Governance & Cert Management – Initial Deployment



Overview

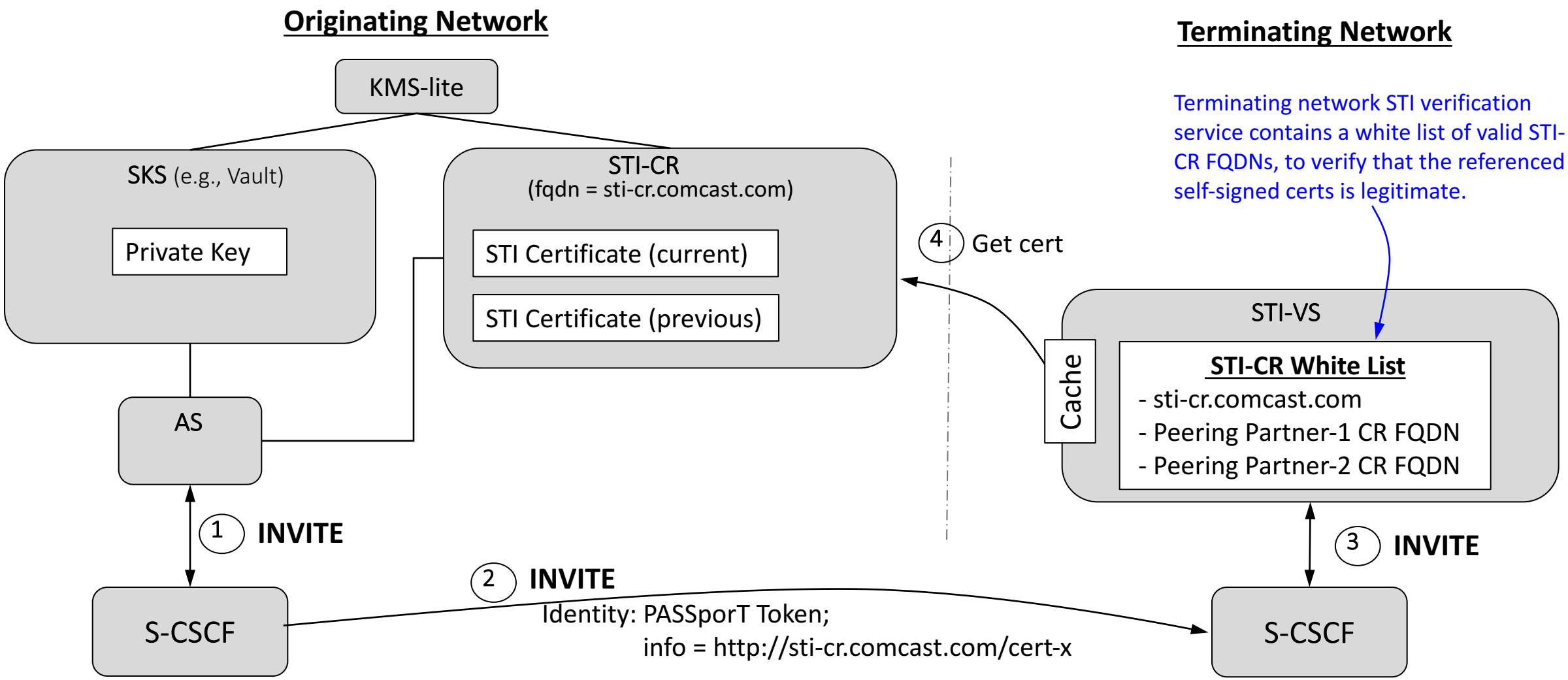
Since real-world deployments of the SHAKEN Governance and Cert Management framework are likely to be a year or two out, initial SHAKEN deployments will perform cert management with a "KMS-lite" that provides self-signed STI certs.

① The KMS script periodically generates an STI cert and private key, utilizing OpenSSL.

② KMS provides the private key to the SKS; i.e., the AS will use the private key of the most recently generated cert to authenticate new calls.

③ KMS provides the STI cert to the STI-CR. The STI-CR maintains a set of the most recently 'n' received certs ('n=2' in this example). The AS will refer to the most recently received cert in the Identity header of new calls. The older cert(s) are maintained in the CR to support verification of in-flight calls when a new STI cert is issued.

SHAKEN Call Establishment – Initial Deployment



Overview

On receiving (1), the STI AS performs STI authentication using the current STI certificate, and adds an Identity header containing the PASSporT token and cert URL. The INVITE is then sent to the terminating network in (2), where it is routed to the STI VS at (3). On receiving (3) INVITE, the STI VS verifies that the FQDN in the Identity info parameter is on the CR White List. If it is, the VS uses the info URL to fetch the STI cert via (4). In this initial phase before the full SHAKEN cert management framework is deployed, the cert validation checks will have to be paired down (no validation of cert chain to a trusted root, etc.). The VS then uses the public key in the STI cert to validate the PASSporT signature, per normal SHAKEN procedures.

Using HTTP caching to cache STI certs

Caching at the STI-VS

- Benefits of caching
  - Increases reliability (use locally cached info if server unavailable)
  - Lowers latency (no round-trip to Server if cached information is fresh)
  - Reduces load on HTTP Servers

- Types of information cached
  - STI SP certificates
  - List of valid STI-CAs obtained from the STI-PA
  - STI-CA root certificates
  - Certificate revocation lists

- Select cache lifetimes carefully
  - Optimize between better performance and faster discovery of compromised certs
  - For certificates, cache lifetime are partly tied to the selection of certificate lifetime

Proposal: use HTTP caching as defined in RFC 7234

Advantages over home-grown caching mechanisms

- Lowers development costs, speeds up deployments
  - Reuse an already well-defined mechanism – don’t reinvent the wheel
  - Hardened by wide deployment experience
  - Open-source implementations available
- Improves interworking and ensures deterministic behavior
  - Provides common & standard mechanism across STI Providers
- Supports features that will benefit caching
  - E.g., Server control of cache lifetime, efficient cache renewal

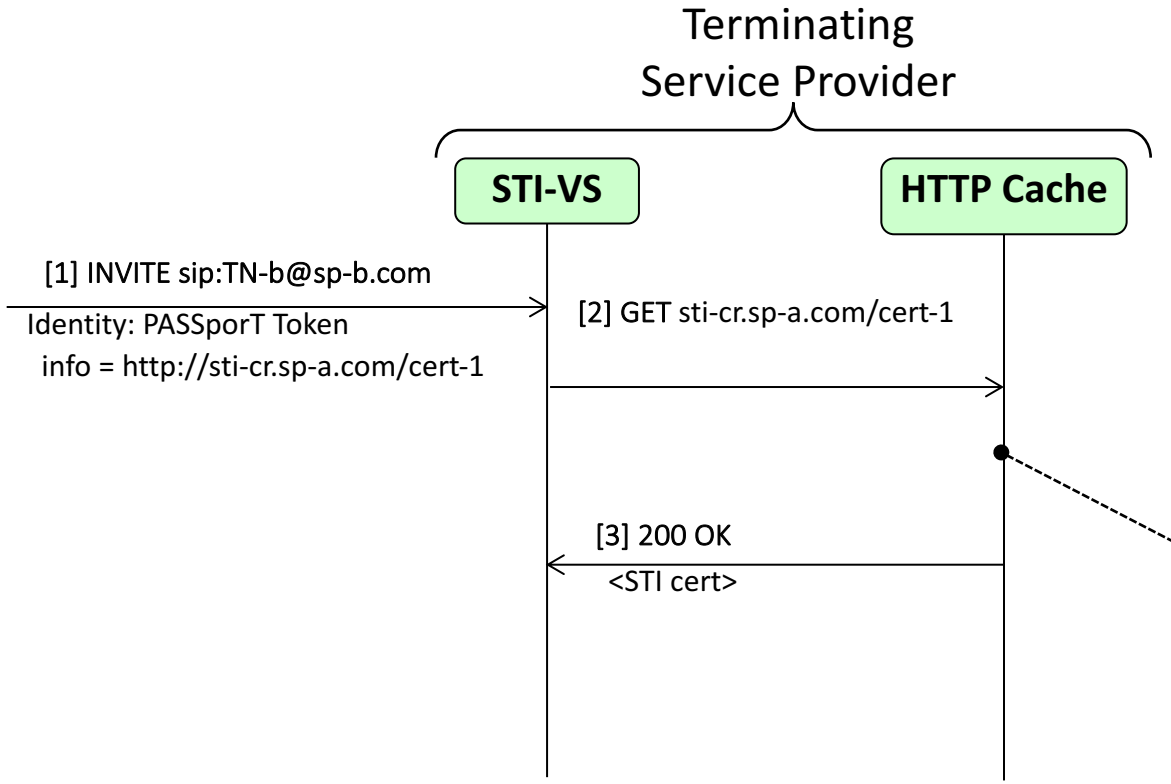
RFC 7234 HTTP caching feature highlights

- Server control of cache lifetimes
  - STI certificate owner can make intelligent cache lifetime decisions
  - Cache lifetime loosely coupled to cert lifetime (also under cert owner control)
  - Reputation/performance of certificate owner affected if cache lifetimes too long/short
- Efficient cache renewal
  - RFC 7234 defines a cache "validation" procedure for quickly checking if a stale cached response is still valid and can be made fresh again
  - Validation involves using the HTTP ETag and If-None-Matches headers to enable the STI-VS to quickly renew cache lifetimes without downloading certificate from Server
- Proxy caches
  - Proxy cache serves multiple clients
  - Helps STI-CR to scale when using short-term certs with a large number of verifiers
  - HTTP Age header enables Proxy and Client caches to manage lifetimes accurately

Applying HTTP Cache to SHAKEN STI Verification Service

During STI verification, the STI-VS simply sends the GET request for the STI cert to its local HTTP cache

- The cache returns a fresh response if it has one
- Otherwise, the cache obtains a fresh response from the STI-CR, and returns that




```
Case Cache-entry-status of {
No-cache-entry: {
        GET cert from STI-CR, cache response, and return response [3] to STI-VS
        }
Fresh-cache-entry: {
        Return cached response [3] to STI-VS
        }
Stale-cache-entry: {
        Perform cache validation procedure with STI-CR
        If cached response still valid, then return cached response [3] to STI-VS
        If cached response no longer valid, then {
                GET cert from STI-CR, cache response, and return [3] response to STI-VS
                }
        }
}
```

Background Information (HTTP caching details)

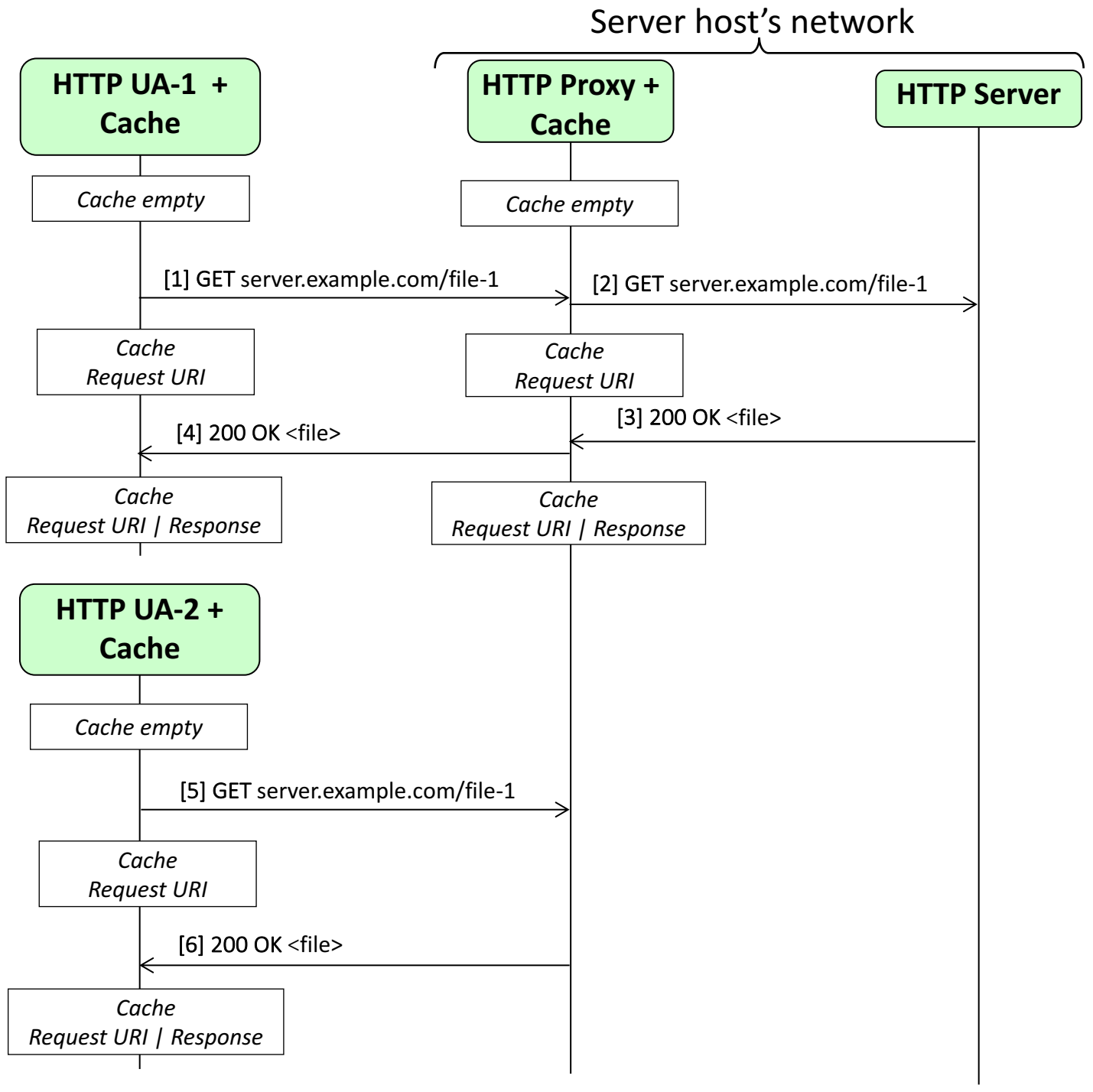
HTTP Caching Procedure Overview

HTTP caching is defined in RFC 7234. Both User Agent and Proxy may implement an HTTP Cache:

- UA cache is private (serves single UA)
- Proxy cache is public (serves multiple UAs)
- The Server's host provider can deploy Proxy cache to scale to a large number of UA requests

Message Sequence

1) UA-1 wants to obtain a file. Since its cache is empty, it sends request to Proxy. UA-1 caches GET request URI, which serves as key to this cache entry.
2) Since Proxy's cache is empty, it caches request URI and sends request to Server.
3) Server responds with file.
4) Proxy stores response in cache entry associated with request, and sends request to UA-1. UA-1 caches response as well. UA-1 will use the locally cached response for subsequent GET requests to the same request URI.
5) Some time later, UA-2 sends GET request for the same file.
6) Proxy notices that it has a cached entry for this request, and therefore sends cached response.



Controlling Cache Freshness

- RFC 7234 defines “freshness” as an attribute of a cache entry
  - response_is_fresh = (freshness_lifetime > current_age)

- How freshness is used…
  - When a cached response is "fresh", it can be used to satisfy subsequent requests without contacting the server.
  - When a cached response is discovered to be "stale", the cache must ask the server if the response has changed, and if it has, download a new response.

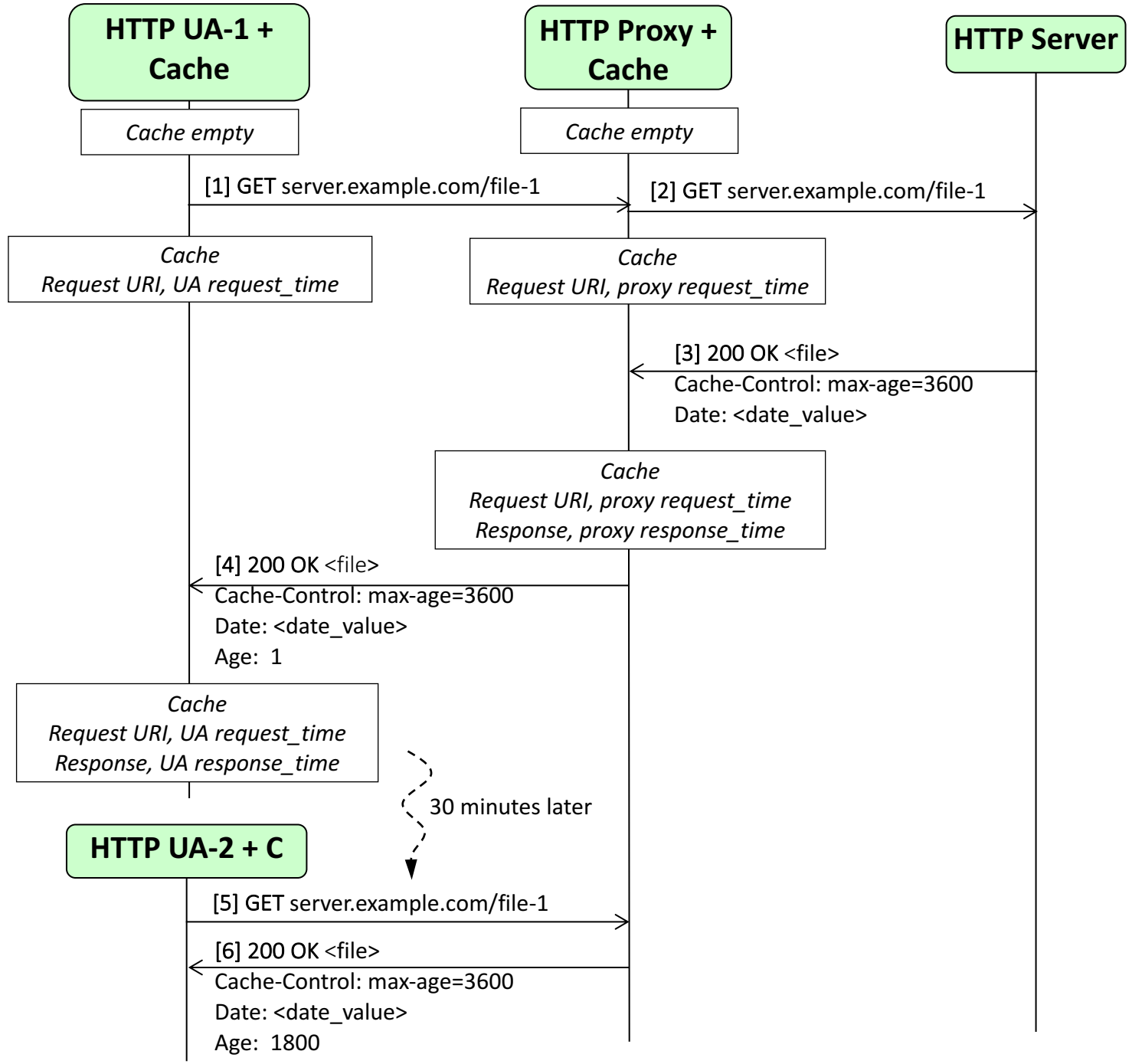
- The Server uses the HTTP Cache-Control header to set the lifetime of a cache entry
  For example, if a response contains…
  Cache-Control: max-age=3600
  … then the UA and Proxy cache entries for that response will become stale after 3600 seconds.

- Calculating *freshness_lifetime* and *current_age*
  - Freshness_lifetime equals the max-age value (exceptions, such as use of Expires header, described later)
  - current-age is the length of time since the response was generated by the Server

- HTTP Age header
  - When responding to a request, the cache adds an HTTP Age header to inform downstream caches of the current age of the response.
  - For example, a UA cache uses the received Age header value to adjust the remaining lifetime of a response to account for the fact that it has already spent some portion of its lifetime sitting in the cache of an an upstream proxy

HTTP Cache Freshness Example

1) UA cache sends HTTP GET request to Proxy. UA saves GET Request-URI in cache (used a key to this cache entry), plus the current timestamp of when the request was sent (used for freshness calculation).
2) Likewise, the Proxy saves the GET Request URI and current request timestamp in its cache, and forwards the request on to the Server.
3) The Server includes a Cache-Control max-age value in the response to specify the lifetime of downstream cached entries for this response. The response also contains a Date header which can be used as part of freshness calculation.
4) The Proxy caches the entire response, plus the current timestamp of when the response was received. The Proxy also adds an Age header specifying the current age of this response. Since in this case the response has been newly generated by the Server, the current age is pretty short; i.e., its the response propagation delay between Server and Proxy. The Proxy then sends the response on to the UA. The UA caches the received response and the current timestamp of when the response was received.
5) 30 minutes later, UA-2 sends an HTTP GET to the proxy with the same Request URI as in 1).
6) The Proxy detects that it has a fresh response for this request in its cache. The Proxy sends the cached response with an Age header value of 1800 indicating that the response has used 1800 seconds of its 3600 second lifetime.



Cache entry validation procedure

If the UA wants to use a cached response, but finds that the cache entry is stale, then it performs the "validation" procedure described here.

1) Thru 4) – same as previous slide, with the addition that the Server includes an ETag header in the response containing a token. Downstream caches can use this token to quickly validate whether or not they need to refresh this cache entry when it becomes stale.
4) 1 hour and 10 minutes later, UA-1 wants to use the cached response. The cache sees that the cache entry is stale. The cache sends a GET to the same Request URI, but this time containing a If-None-Matches header that contains the ETag token value.
5) The Proxy sends the request to the Server.
6) The Server uses the token to identify the response that was sent earlier. In this example it finds that the response has not changed, and so instead of returning the file in a 200OK response, it sends a 304 “Not Modified” response.
7) The Proxy sends the 304 response on to the UA. The UA and Proxy update their cached request and response timestamps, which transitions the cached entry back to “fresh”.