Workgroup: Network Working Group
Internet-Draft: draft-sandowicz-httpbis-httpa2
Published: 18 October 2022
Intended Status: Informational
Expires: 21 April 2023
Authors: S. Wang G. King N. Li N. Smith K. Sandowicz
Intel Intel Intel Intel Intel

The Hypertext Transfer Protocol Attestable (HTTPA) Version 2

Abstract

The Hypertext Transfer Protocol Attestable version 2 (HTTPA/2) is an HTTP extension. It is a transaction-based protocol agnostic to Transport Layer Security (TLS) in which the Trusted Execution Environment (TEE) is considered a new type of requested resource over the Internet. The original Hypertext Transfer Protocol Attestable (HTTPA) (referred to as HTTPA/1 in the rest of the document) includes remote attestation (RA) process onto the HTTPS protocol in the assumption of using Transport Layer Security (TLS) across the Internet. In contrast, the design of HTTPA/2 could establish a trusted (attested) and more secure communication without dependence on TLS.

The definition of Attestation for the purposes of this draft:

The process of vouching for the accuracy of TEE based services, configuration, and data where the TEE conveys Evidence about its environment, roots of trust and protected functions. The Evidence is a digital expression of TEE trustworthiness.

Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at https://datatracker.ietf.org/drafts/current/.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on 21 April 2023.

Copyright Notice

Table of Contents

1. Introduction

The HTTPA/1 [I-D-11] defines an HTTP extension to handle requests for remote attestation, secret provisioning, and private data transmission, so internet visitors can access a wide variety of

services running in Trusted Execution Environments (TEEs) to handle their requests with strong assurances.

The HTTPA/1 supports mutual attestation if both client and service endpoints run inside the TEE. Although HTTPA/1 helps build trust between L7 endpoints with data-level protection, HTTPA/1 needs TLS to defend against some specific attacks over the Internet, e.g., replay attacks and downgrade attacks, these attacks are not vulnerable in HTTP/2 due to specific improvements for them. Note that TLS cannot guarantee end-to-end security for the HTTPS message exchange [I-D-1] when the TEE-based services (TServices) are hosted behind a TLS termination gateway or inspection appliance (a.k.a. middle boxes). Although the TLS can provide Confidentiality, Integrity, and Authenticity (ConfIntAuth) to help ensure the security of message exchange for HTTPA/1 protocol, it is not a complete end-to-end solution for web services at L7. Both HTTPA/1 and TLS need to generate key material through key exchange and derivation processes. This requires additional round trips at L5 and increases network latency. Thus, there is room to optimize the network performance further and reduce the communication complexity by avoiding the repetition of key negotiation. Due to the limitation of TLS mentioned above, a version of HTTPA with message-level security protection is a natural candidate to address the issues mentioned above. This document proposes an upgrade protocol, HTTPA/2, which makes it possible to secure HTTPA transactions without dependence on TLS. The HTTPA/2 is designed to improve the processes of key exchange, RA, and secret provisioning. It also enables end-to-end secure and trustworthy request/response transactions at L7, which is cryptographically bound to an attestable service base that can be trusted by internet visitors regardless of the presence of untrusted TLS termination.

The protocol described in this document focuses on extending the functionality provided by the HTTPA/1 protocol message formats. This document alone is sufficient to understand the protocol, and the HTTPA/1 [I-D-11] could be used as supplemental material.

1.1. Conventions Used in This Document

The key words "**MUST**", "**MUST NOT**", "**REQUIRED**", "**SHALL**", "**SHALL NOT**", "**SHOULD**", "**SHOULD NOT**", "**RECOMMENDED**", "**MAY**", and "**OPTIONAL**" in this document are to be interpreted as described in [BCP 14], [RFC 2119], [KEYWORDS].

In addition to those listed above, this document uses the following terms:

Trusted Execution Environment (TEE):

TEE is an environment where genuine codes are executed on data with isolation and memory encryption inaccessible to anyone.

Rich Execution Environment (REE):

In contrast to TEE, codes are executed on data without isolation.

Client:

The endpoint initiating the HTTPA connection.

Server:

The endpoint did not initiate the HTTPA connection. In HTTPA/2, it refers to TEE-based Service (TService) running inside an enclave.

Client-side verifier(c-verifier):

Verifier from the client side.

Server-side verifier(s-verifier):

Verifier from the server side.

Attestation:

Attestation is the process of demonstrating that a software executable has been properly instantiated on a platform, thus establishing a level of confidence in the trustworthiness of a remote peer.

Attest Quote(AtQ):

AtQ is an opaque data structure signed by a Quoting Service (QService) with an attestation key (AK). It can be called a quote or attestation evidence, which is used to establish trustworthiness through identities.

Attest Base(AtB):

AtB is the totality of computing resources serving client request handling, including hardware, firmware, software, and access controls to work together to deliver trustworthy service quality with enforced security/privacy policy.

Attest header line(AHL)

It refers to the different types of header lines used during the handshake phase, including Attest Ticket, Attest Binder, etc.

Attest header Field(AHF)

Regarding the HTTP method, we propose a new HTTP method, called "ATTEST," to perform the transactions of AtHS and AtSP. The HTTP request using ATTEST method is called AtR. Regarding HTTP header fields, we propose to augment them with additional ones called Attest Header Fields (AHFs) prefixed with the string "Attest-." Without AHFs, it must be a UtR in terms of HTTPA/2.

Attest Ticket(AtT):

AtT is a type of attest header line(AHL) used to ensure the integrity and authenticity (IntAuth) of AHLs and freshness are protected cryptographically, except for the AtR of AtHS, the initiating request for the handshake.

Attest Binder (AtBr):

AtBr is a type of AHL used to ensure the binding between the HTTPA/2 request and the corresponding response.

Attest Request (AtR)

Regarding the HTTP method, we propose a new HTTP method, called "ATTEST," to perform the transactions of Attest Handshake(AtHS) and Attest Secret Provisioning (AtSP). The HTTP request using ATTEST method is called AtR.

Trusted Computing Base(TCB):

The minimal totality of hardware, software, or firmware must be trusted for security requirements.

Trusted Cargo(TrC):

TrC is a vehicle to carry confidential information which needs to be protected by authenticated encryption. It can appear in both HTTPA/2 request and response messages, except for the AtR of AtHS.

Trusted Transport Layer Security(TrTLS):

If users want to protect the entire HTTP message?every bit of the message, HTTPA/2 can leverage TLSto establish a secure connection at L5 between the client and its adjacent middle box, which we call TrTLS.

Preflight

This is the first phase of the HTTPA/2 transactions, and it is a lightweight HTTP OPTIONS request.

2. Protocol Overview and protocol flow

There are three types of requests defined by the HTTPA/2 protocol, including Un-trusted Request (UtR), Attest Request (AtR), and Trusted Request (TrR). UtR is used in HTTP(S) transactions; AtR is used in both transactions of Attest Handshake (AtHS) and Attest Secret Provisioning (AtSP); TrR is used in the trusted transaction. For convenience, we refer to the AtR and TrR as "HTTPA/2 request". Regarding the HTTP method, we propose a new HTTP method, called "ATTEST," to perform the transactions of AtHS and AtSP. The HTTP request using ATTEST method is called AtR. Regarding HTTP header fields, we propose to augment them with additional ones called Attest Header Fields (AHFs) prefixed with the string "Attest-." Without AHFs, it must be a UtR in terms of HTTPA/2.

The AHFs are dedicated to HTTPA traffic. For example, they can be used to authenticate the identity of HTTPA/2 transactions source, indicate which AtB to request, convey confidential metadata, provision secrets, present ticket, etc.

The last one is AHL, and it consists of AHF and its values in a standard form [RFC8941]. We use it to signify a single piece of annotated data associated with the current HTTPA/2 request.

2.1. Untrusted Request (UtR)

An untrusted request is for any transactions that are not sensitive. The UtR is simply an ordinary type of HTTP request, which does not use the ATTEST method nor contains any AHLs.

Before a UtR reaches a TService, the UtR can be easily eavesdropped on or tampered with along the communication path. Even protected by TLS, it can still be attacked when crossing any application gateway or L7 firewall since those intervening middle-boxes are untrusted and will terminate TLS connections hop by hop [I-D-1]. Therefore, there is no guarantee of ConfIntAuth. That's why the TService cannot treat the request as trustworthy, but it is still possible for TService to handle UtR if allowed by the service-side policy. Thus, we don't suggest TService to handle any one of them for the sake of security.

2.2. Attest Request (AtR)

The AtR is a HTTP request equipped with both ATTEST method and AHLs for AtHS and AtSP. If the corresponding TService did not accept any AtR, subsequent TrR will no longer be valid to this TService. The major difference between an AtR used in AtHS and AtSP respectively is as follows:

1)

The AtR used in AtHS is designed to request all necessary resources for handling both types of AtR used in AtSP and AtHS. For example, one of the most critical resources is AtB, which may be scheduled or allocated by a server-side resource arbiter. Typically (but not always), an upfront TService can directly designate itself as the AtB for this client.

2)

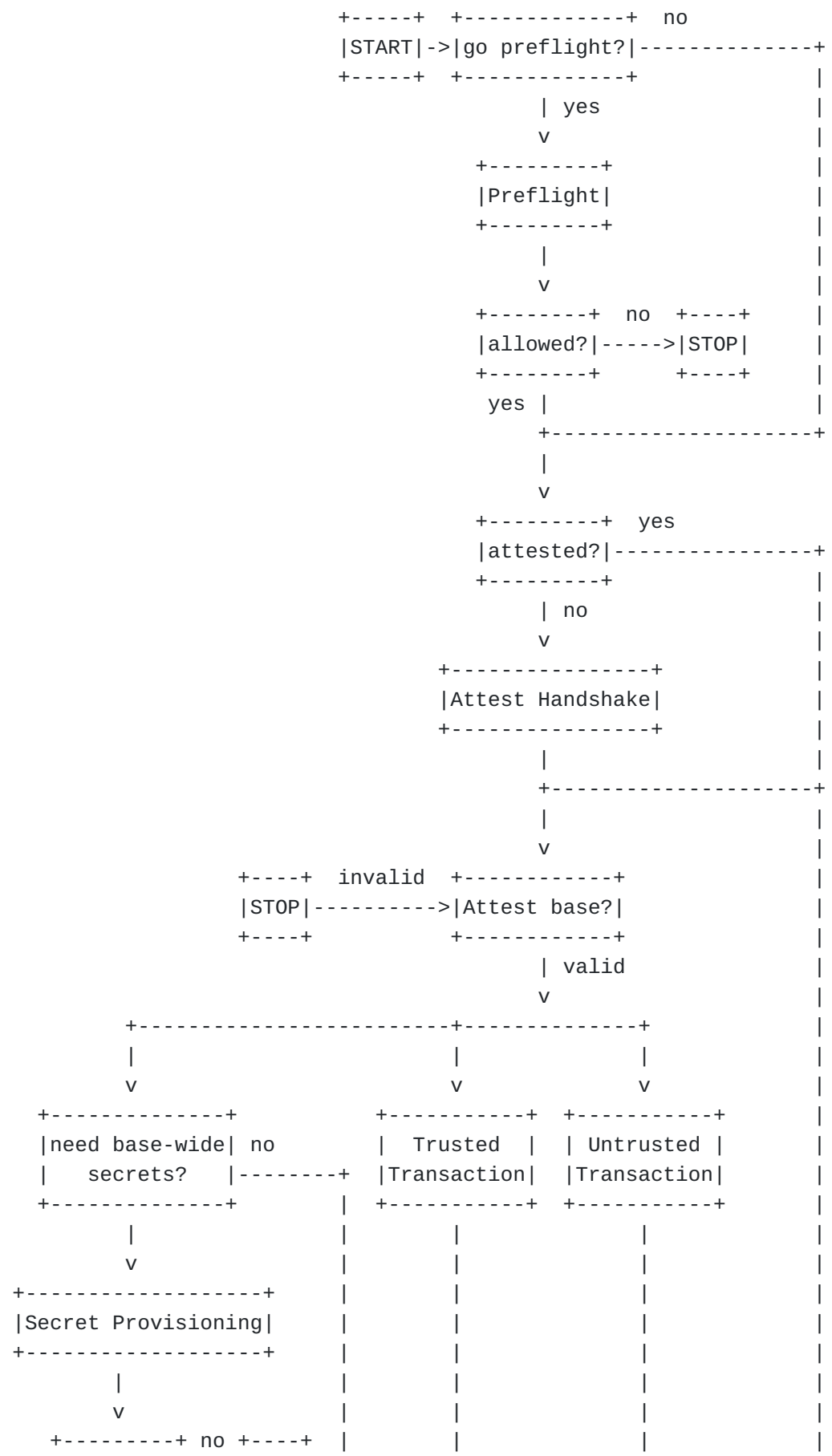
The AtR of AtSP is optional in HTTPA/2 traffic flow since, in some cases, the TService does not need any AtB-wide secrets provided by the client to work. In the typical case, TService needs secret provisioning to configure its working environment, such as connecting to databases, setup signing keys and certificates, etc. This AtR must be issued after all TEE resources have been allocated through the AtHS transaction described above. It's worth noting that this request is not required to be issued before any TrR.

2.3. Trusted Request (TrR)

The TrR can be issued right after a successful AtHS where an AtB is allocated. Although TrR does not use ATTEST method, it should contain AHLs to indicate that it is a TrR, not a UtR. In other words, the TrR is nothing but an ordinary HTTP request with some AHLs. Within those AHLs, one of them must be AtB ID to determine which AtB is targeted in addition to the specified URI. The TrR can be dispatched to the proper TService to handle this request.

2.4. Protocol flow

As shown in Figure 1, we illustrate those transactions from a client perspective, including preflight, AtHS, AtSP, and trusted requests in a workflow diagram. A detailed explanation of each phase is in the following chapters. In the design of HTTPA/2, only the phase of AtHS is required. This largely simplifies the interaction between the client and the TService and improves the overall service experience for both security and remote attestation.


START
go preflight?
no
yes
Preflight
allowed?
no
STOP
yes
attested?
yes
no
Attest Handshake
STOP
invalid
Attest base?
valid
need base-wide secrets?
no
Trusted Transaction
Untrusted Transaction
Secret Provisioning
no

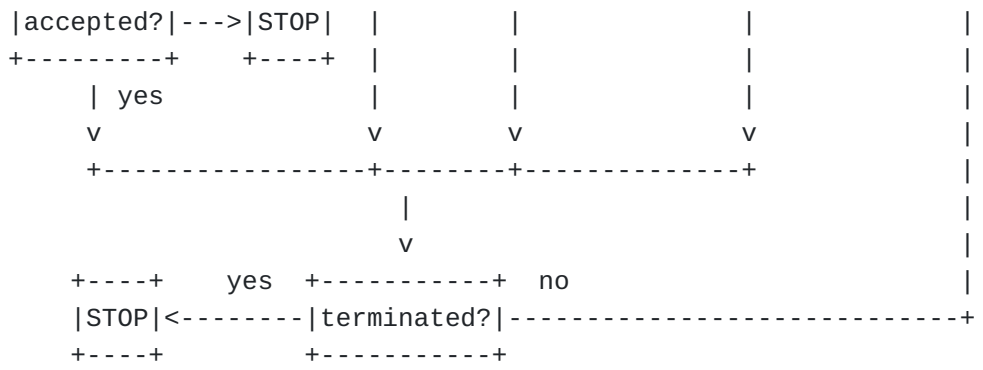
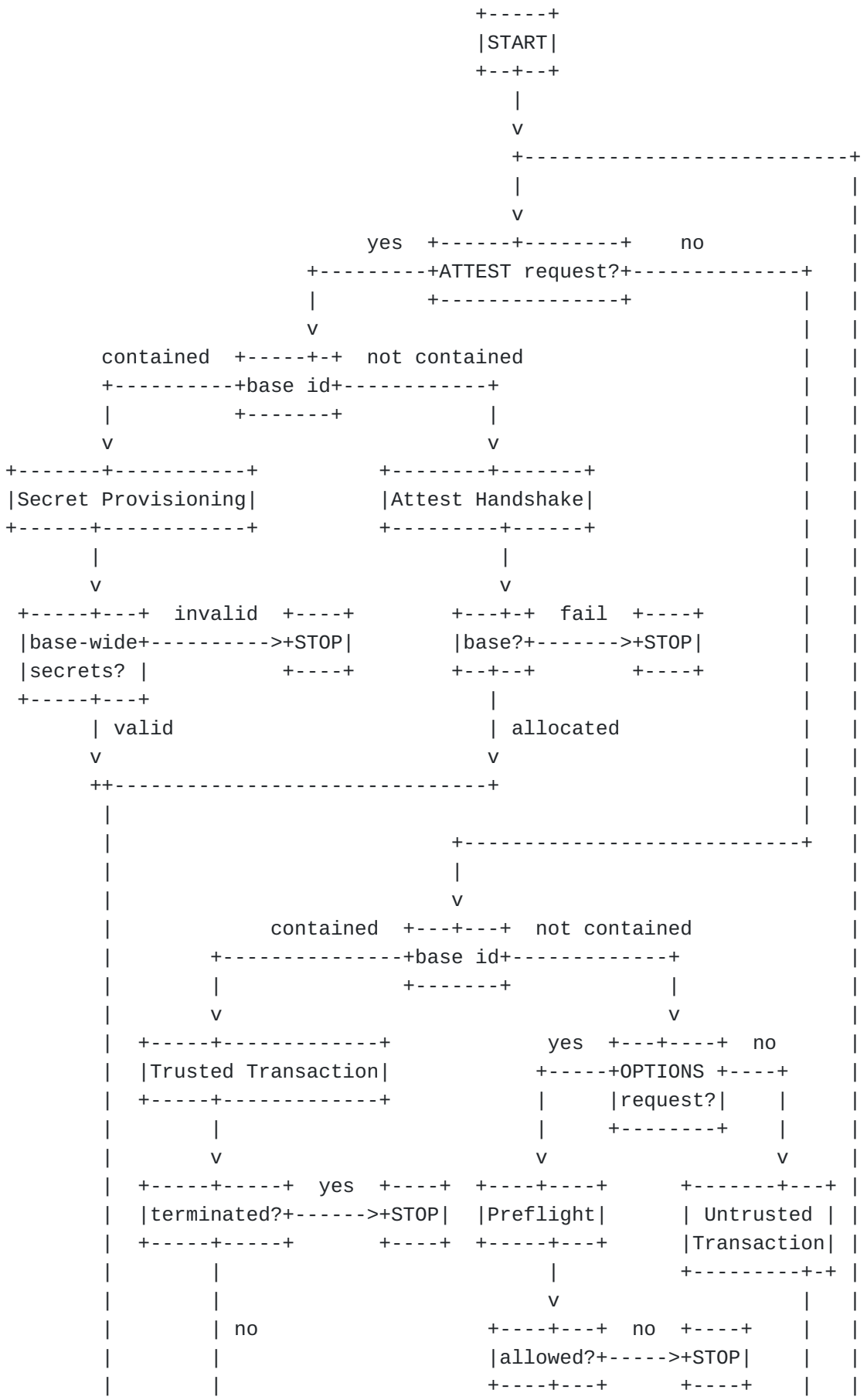

|accepted?|--->|STOP|
yes
yes
no
|terminated?|
|STOP|

Figure 1: HTTPA transaction workflow from the client view

The Figure 2 shows the workflow, which can help understand how those transactions are distinguished in TService.


START
ATTEST request?
yes
no
contained
base id
not contained
Secret Provisioning
Attest Handshake
base-wide secrets?
invalid
STOP
base?
fail
valid
allocated
contained
base id
not contained
Trusted Transaction
terminated?
yes
STOP
Preflight
OPTIONS request?
yes
no
Untrusted Transaction
no
allowed?
no
STOP

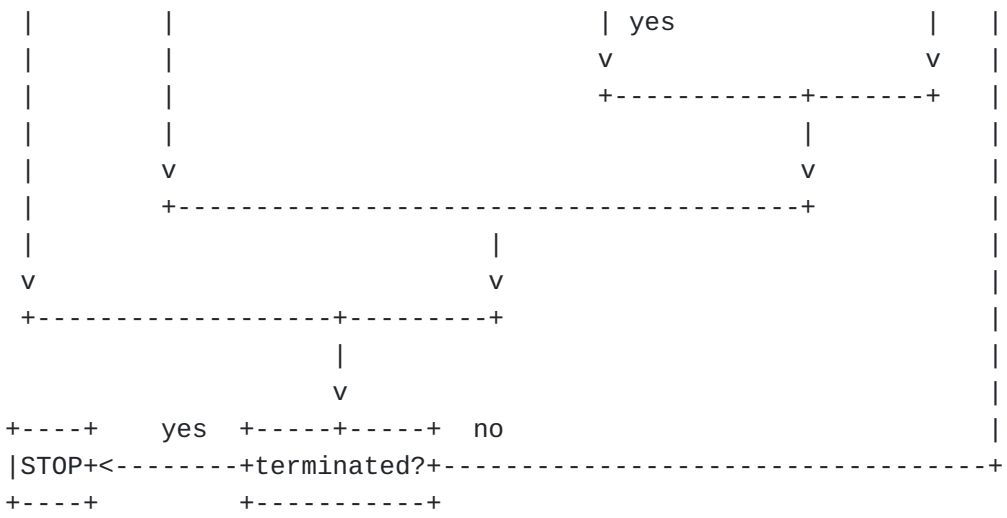

| yes
yes
no
STOP
terminated?

Figure 2: HTTPA transaction workflow from the TService view

3. Protocol Transactions

3.1. Preflight Check Phase

The preflight request uses OPTIONS request to give the web service a chance to see what the actual AtR looks like before it is made, so the service can decide whether it is acceptable. In addition, the client endpoint performs the preflight check as a security measure to ensure that the visited service can understand the ATTEST method, AHFs, and its implied security assurance.

To start HTTPA/2, a preflight request could be issued by a client as optional to check whether the Web service, specified by URI in the request line is TEE-aware and prepared for AtHS. If the client is a web browser, the preflight request can be automatically issued when the AtR qualifies as "to be preflighted." We need the preflight transaction because it is a lightweight HTTP OPTIONS [RFC7231] request, which will not consume a lot of computing resources to handle compared to the AtR. Caching the preflight result can prevent re-checking during a specified time window. In the case of out-of-sync, the TService will result in an invalid signal for HTTPA trusted requests.

Passing this check does not guarantee that this service can successfully handle the AtR. For example, the TService may run out of resources, or the client's cipher suites are not supported, and so on.

The client can also use the preflight to detect the capabilities of AtB, without implying any actual actions.

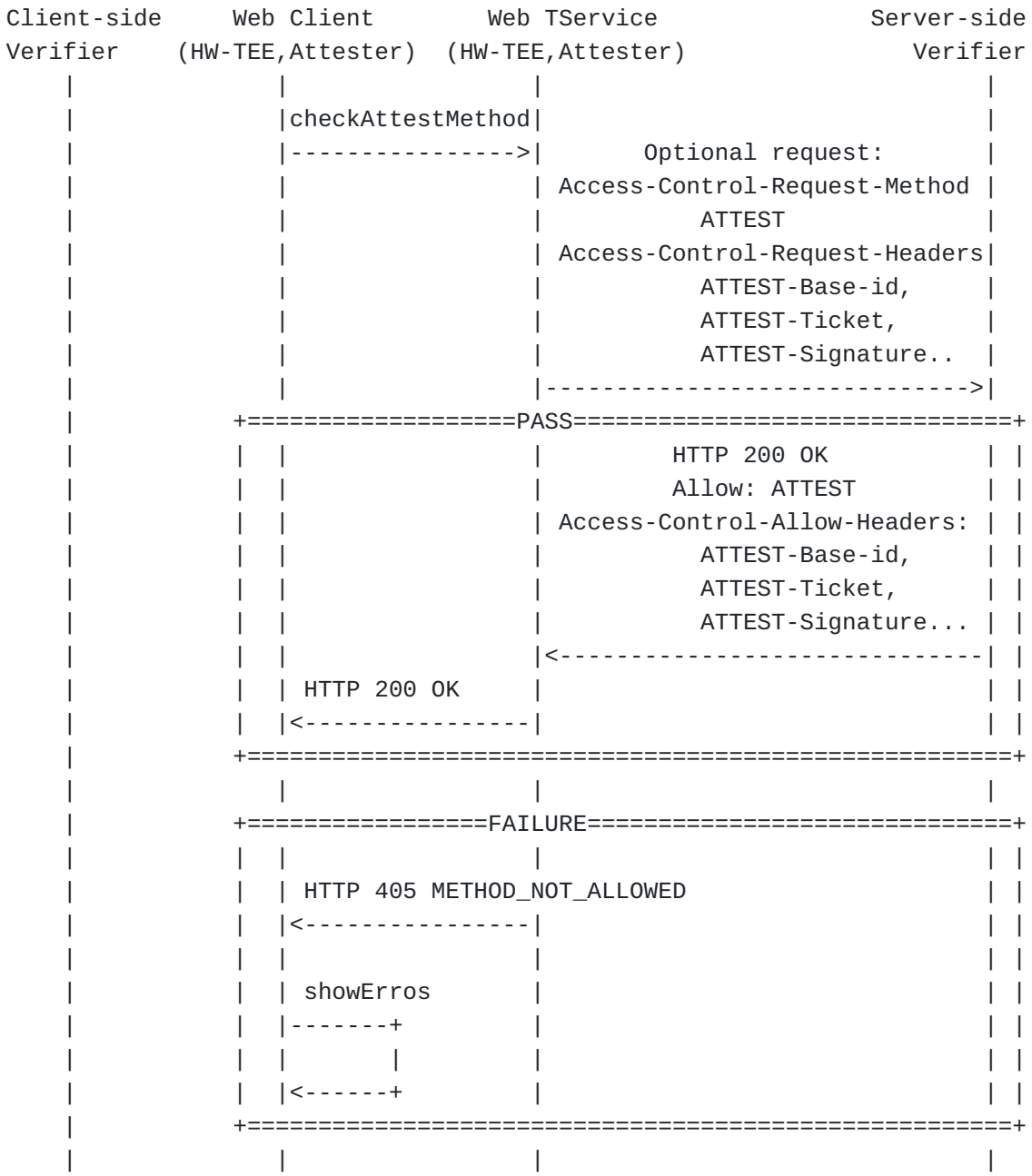


Figure 3: Message Flow example for Preflight transaction

As shown in Figure 3, an OPTIONS request should be honored by an HTTPA/2 compliant TService. The preflight transaction has standard HFs to specify the method and AHLs which will be sent out later to the same TService

if they are acceptable. Those HFs are described respectively as follows:

1. HFs in the request message

(a) Access-Control-Request-Method

This HF carries a list of methods indicating that ATTEST method will be used in the next request if the service can support it.

(b) Access-Control-Request-Headers

This HF carries a list of field names indicating that the AHFs will be included in the next request if the service can support them.

2. HFs in the response message

(a) Allow

This HF carries a list of supported methods by the visiting service. It must contain the ATTEST method for the client to proceed with AtR; otherwise, the AtR is not acceptable by this service and will be denied if received it.

(b) Access-Control-Allow-Headers

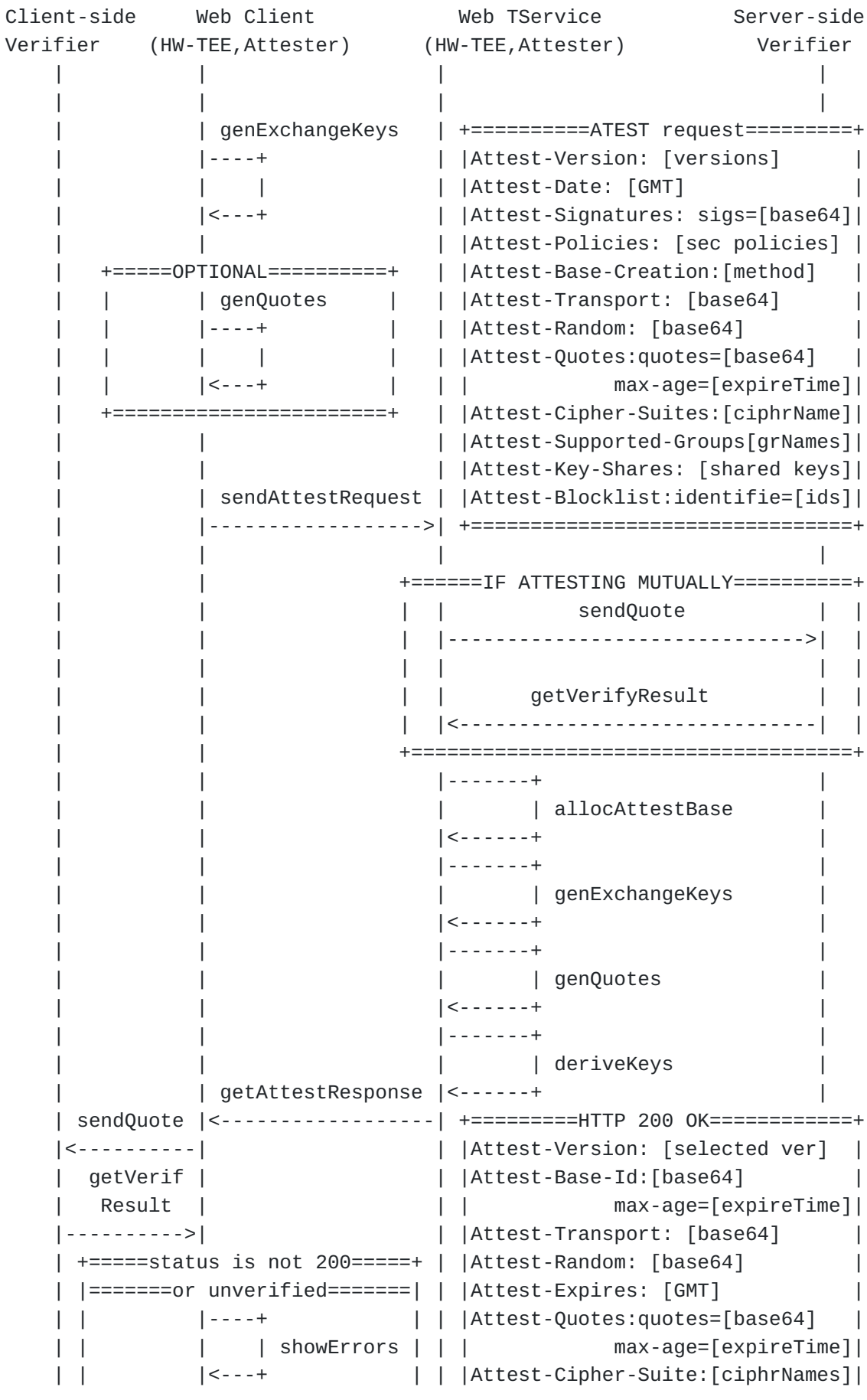
This HF carries a list of allowed AHFs. The client needs to check that all of the requested AHFs should be contained in this resulting field.

(c) Access-Control-Max-Age

This HF indicates how long the preflight check results can be cached.

3.2. Attest Handshake (AtHS) Phase

The AtHS phase contains a core transaction of HTTPA/2. In a single round trip time (one RTT), the AtR and its response accomplish three major tasks, including key exchange, AtB allocation, and AtQ exchange, as shown in Figure 4.


Client-side
Verifier
Web Client
(HW-TEE,Attester)
Web TService
(HW-TEE,Attester)
Server-side
Verifier
genExchangeKeys
OPTIONAL
genQuotes
sendAttestRequest
ATEST request
Attest-Version: [versions]
Attest-Date: [GMT]
Attest-Signatures: sigs=[base64]
Attest-Policies: [sec policies]
Attest-Base-Creation:[method]
Attest-Transport: [base64]
Attest-Random: [base64]
Attest-Quotes:quotes=[base64]
max-age=[expireTime]
Attest-Cipher-Suites:[ciphrName]
Attest-Supported-Groups[grNames]
Attest-Key-Shares: [shared keys]
Attest-Blocklist:identifie=[ids]
IF ATTESTING MUTUALLY
sendQuote
getVerifyResult
allocAttestBase
genExchangeKeys
genQuotes
deriveKeys
getAttestResponse
HTTP 200 OK
Attest-Version: [selected ver]
Attest-Base-Id:[base64]
max-age=[expireTime]
Attest-Transport: [base64]
Attest-Random: [base64]
Attest-Expires: [GMT]
Attest-Quotes:quotes=[base64]
max-age=[expireTime]
Attest-Cipher-Suite:[ciphrNames]
sendQuote
getVerif
Result
status is not 200
or unverified
showErrors

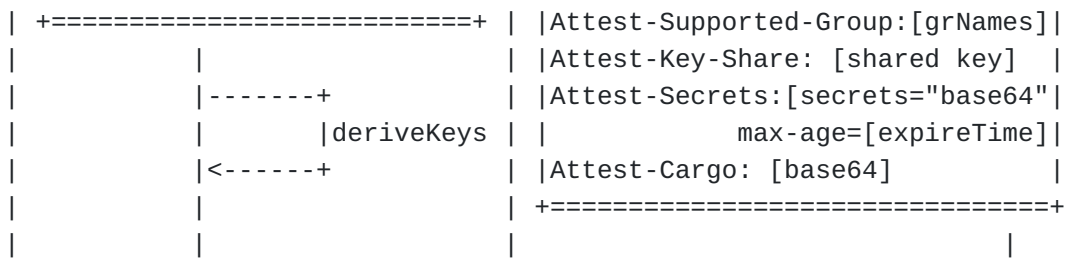

+============================+
|Attest-Supported-Group:[grNames]|
|Attest-Key-Share: [shared key] |
|Attest-Secrets:[secrets="base64"|
| max-age=[expireTime]|
|Attest-Cargo: [base64] |
+================================+
deriveKeys

Figure 4: Attest handshake (AtHS) transaction

1. Key Exchange

It is necessary to complete the key exchange process before any sensitive information can be transmitted between the client and TService. The exact steps within this will vary depending on the key exchange algorithm used and the cipher suites supported by both sides.

In HTTPA/2, the key exchange process follows TLS 1.3 [RFC8446] and recommends a set of key exchange methods to meet evolving needs for stronger security.

Insecure cipher suites have been excluded; all public-key-based key exchange mechanisms now provide Perfect Forward Secrecy (PFS), e.g., Ephemeral Elliptic Curve Diffie-Hellman (ECDHE). Note that it is mandatory that the fresh ephemeral keys are generated, used, and destroyed afterward [RFC8422] inside the TEE of TService. When the key exchange is completed, we recommend using the HMAC-based Extract- and-Expand Key Derivation Function (HKDF) [RFC5869] as an underlying primitive for key derivation. Also, note when a peer creates one or more (EC)DHE public keys, it must do so in a correct and standards-compliant manner. When a peer receives a set of (EC)DHE public keys, it must validate that the public key is in the specified group and has other required properties (e.g., it is not the group identity).

We describe the key negotiation between the client and the TService in terms of AHFs set in request and response, respectively, as follows:

(a) AHFs in request message (or AtR):

i. Attest-Cipher-Suites

It is a list of cipher suites that indicates the AEAD algorithm/HKDF supported by the client.

ii. Attest-Supported-Groups

A list of named groups [RFC7748] indicates the (EC)DHE groups supported by the client for key exchange, ordered from most preferred to least preferred.

iii. Attest-Key-Shares

Its value contains a list of the client's cryptographic parameters for possible supported groups indicated in the AHL of Attest-Supported-Groups for negotiation. We can refer to the corresponding

data structure described in TLS 1.3 [RFC8446]. It is a time-consuming operation to generate those parameters.

iv. Attest-Random

It is 32 bytes of a cryptographically random nonce, and the purpose of the random nonce is to bind the master secret and the keys to this particular handshake. This way mitigates the replay attack to the handshake as long as each peer properly generates this random nonce.

(b) AHFs in response message

i. Attest-Cipher-Suite

It indicates the selected cipher suites, i.e., a symmetric cipher/ HKDF hash pair for HTTPA/2 message protection.

ii. Attest-Supported-Group

It indicates the selected named group to exchange ECDHE key share generated by the TService.

iii. Attest-Key-Share

Its value contains the TService's cryptographic parameters accordingly.

iv. Attest-Random

It takes the same mechanism as the Attest-Random in the request. Instead, it is used by the client to derive the master secret and other key materials.

Note that anyone can observe this handshake process if the byte-to-byte encryption does not protect it at L5, but it is safe since the secrets of key exchange process will never be sent over the wire.

2. AtB Allocation

This task is responsible for resource allocation. The upfront TService needs to prepare essential resources before assigning a unique AtB identifier to the AtB, which the client uses to ask TService to process its sensitive data on this AtB, AHF is part of AHL

(a) AHFs in request message (or AtR):

i. Attest-Policies

It can contain various security policies, which this AtB of TService can selectively support. There are two aspects to consider as follows:

Service instances attestation

direct: all instances should be verified by the client.

Indirect: only the contact instance(a proxy instance could be used for attesting other instances) should be verified by the client remotely.

Un-trusted requests

allowUntrustedReq: it allows UtR to be handled by the TService on this AtB (disabled by default).

ii. Attest-Base-Creation

It specifies a method used for the creation of AtB. There might be several options available to select:

new

It means that the AtB should be newly created for the client to use. If the contact TService is new, it can be assigned to this client immediately.

reuse

This option allows reusable AtB to be used by this client, but the AtB should ensure that all traces associated with the previous client are erased.

So far, there is no such TEE, which can achieve this security feature strictly, and we cannot fully rely on software to emulate it. As a result, the client should evaluate the risks before specifying this option.

shared

A shareable AtB can be allocated to this client. The client doesn't care whether it is a clean base or not,so use it with caution.

iii. Attest-Blocklist

It indicates a list of blocked identities and another type of identifier which allows TService to filter out unqualified AtB beforehand. This feature is used to optimize the performance of AtB allocation, as it is quite expensive and inefficient to rely only on

the client to collect a set of TService instances by allow list using the trial and error method.

(b) AHFs in response message:

i. Attest-Base-ID

This identifier signifies the allocated AtB, which has been tied to this particular client who sent the AtHS request. It should be used in subsequent HTTPA/2 requests to ensure those requests can be efficiently dispatched into TServices. Given that the HTTPA/2 request dispatcher may not be trustworthy and be unable to check its integrity. As a result, it cannot guarantee that those requests could be delivered to their matched AtBs. To remedy this problem, the dispatcher should be capable of identifying as invalid AtB ID as possible, and the receiving TService should validate it right after the integrity check (see REF _Ref107244144 \h * MERGEFORMAT Figure 5 and REF _Ref107244151 \h * MERGEFORMAT Figure 6).

Note that the max-age directive set here indicates how long this AtB could be kept alive at the server-side, it follows GMT, and the unit is second.

3. AtQ Exchange

In HTTPA/2, a successful RA increases the client's confidence by assuring the targeting services are running inside a trustworthy AtB. The client can also determine the level of trust in the security assurances provided by TServices through AtB.

The RA is mainly aimed at provision secrets to a TEE. In this solution, we leverage this mechanism to set it as the root trust of the HTTPA/2 transactions instead of certificate-based trust, e.g., TLS. To facilitate it, we integrate the RA with the key exchange mechanism above to perform a handshake, which passes the assurance to derive ephemeral key materials. Those keys can be, in turn, used to protect secrets and sensitive data designated by the client or TService on either direction.

During the RA process, the AtQ plays a key role in attesting TService. It provides evidence (quote) to prove the authenticity of the relevant TService and provides assurance that the TService is a trustworthy client can just rely on it to decide whether the TService is a trustworthy peer or not.

To appraise AtQ, we need a trusted authority to be the verifier to perform the process of AtQ verification and report issues on this AtQ, e.g., TCB issues. The verification result produced by the verifier should be further assessed by the client according to its pre-configured policy rules and applied security contexts. Notably,

the TService should ensure the integrity and authenticity of all AHLs of AtR, and its response through a piece of user-defined information, called Quote User Defined Data (QUDD) of AtQ, the QUDD can provide extra identities specific to a TService. Therefore, the AtQ can, in turn, help protect the integrity of Attest Header Lines (AHLs). The following AHFs should be supported by HTTPA/2 protocol for RA request message is optional.

(a) AHFs in request message (or AtR):

i. Attest-Quotes

It can only appear in mHTTPA/2 mode to indicate a set of AtQs generated from the TClients for targeting TService to verify. These quotes should be used to ensure IntAuth of the AHLs of this AtR through their QUDD.

Note that the max-age directive indicates when these quotes are outdated, and its cached verification results should be cleared up from AtB to avoid broken assurance. In addition, all client-side quotes must be verified by the server-side verifier and validated by TServices before an AtB ID can be issued.

(b) AHFs in response message

i. Attest-Quotes

An AtB must present its AtQs to the client for client-side verification. The IntAuth of both AHLs of the AtR and its response should be ensured by its QUDDs to protect the transaction completely.

The client must verify the AtQ to authenticate its identity of remote AtB. The client should not trust anything received from TService before AtQs are successfully verified and evaluated by the trust authority. Whether the integrity of AHLs is held should be determined by client-side security policies. Note that the TService quotes can be selectively encrypted in its parts through TrC to hide their identity information.

There are several remaining AHFs, which are important to this transaction as they provide other necessary information and useful security properties:

(a) AHFs in the request message

i. Attest-Versions

The client presents an ordered list of supported versions of HTTPA to negotiate with its targeting TService.

ii. Attest-Date

It is the Coordinated Universal Time (UTC) when the client initiates a AtHS.

iii. Attest-Signatures

It contains a set of signatures, which are used to ensure IntAuth of AHLs in this AtR through client-side signing key?

iv. Attest-Transport

With this, the TService can enforce a trustworthy and secure connection at L5, which is a bit similar to what HTTP Strict Transport Security (HSTS) does.

(b) AHFs in response message

i. Attest-Version

It shows the client which version of HTTPA is selected by TService to support, server has to apply in this selection process

ii. Attest-Transport

Similarly, the TService returns its HELLO message to the client for a secure transport layer handshake.

iii. Attest-Expires

It indicates when the allocated AtB will expire, and when its related resources will be released. It provides another layer of security to help reduce the possibility of this AtB being successfully attacked.

iv. Attest-Secrets

It is an ordered list of AtB-wide secrets, which TService provisions if the client expects them. This way can save a round trip time of AtSP in case the TService won't demand secrets from the client immediately.

v.Attest-Cargo

It is used to carry any sensitive information which 14is meaningful to TService. Note that "Attest-Cargo" is an AHF while TrC is the corresponding content which plays an important role in sensitive data encryption and authentication.

Apart from those tasks above, this AtR can act as a GET request, but it cannot be trusted due to incomplete key exchange at this moment, which means it cannot contain any sensitive data, but its response can be trusted as the key exchange process completed at TService-side, and before it gets returned. Therefore, the TService-side sensitive data can be safely transmitted back to the client through the TrC.

3.3. Attest Secret Provisioning (AtSP) Phase

The main purpose of AtSP is to securely deliver secrets to a trustworthy AtB, which a server-side verifier has verified. The AtR of AtSP is intended to be used for this purpose. To be precise, it is for AtB-wide and client-wide secret provisioning. On the contrary, the request-wide or response-wide secrets should be carried by the TrCs of HTTPA/2 transactions. In addition, the failure of AtSP will cause AtB termination immediately.

As shown in Figure 5, the AtSP transaction can be used to provision secrets in two directions since the AtB and its key materials have already been derived through AtHS on both sides; thus, the AHLs can be better protected during this phase. Moreover, AtR of AtSP can be issued by the client any number of times at any time after AtHS.

Figure 5: Attest secret provisioning (AtSP) transaction

These AHLs are described in the following:

1. AHFs in request message (or AtR)

(a) Attest-Base-ID

This identifier is used to specify which AtB is targeted to handle this AtR of AtSP. With this ID, the TService can validate it against its serving list to ensure this request is correctly handled. However, the TService should quietly ignore it if the ID is not valid for its residing AtB as the receiving TService should not expose any information for an adversary to exploit.

(b) Attest-Ticket

AtT is a type of AHL used to ensure the integrity and authenticity (IntAuth) of AHLs and freshness by applying AAD to each HTTPA/2 request, except for the AtR of AtHS, which is the initiating request for the handshake. The value of this field must be unique to previous values to prevent replay attacks. Also, it ensures the IntAuth of the AHLs in this request.

(c) Attest-Secrets

It contains an ordered list of secrets, which is wrapped up using AE as a standard way for strong protection. Moreover, each secret should be able to be referred to by the client later using the index. For example, specifying a provisioned secret that is used to decrypt embedded sensitive data. Again, the receiving AtB should be terminated if any of these provisioned secrets cannot be validated or accepted by the AtB.

(d) Attest-Cargo

This field is optional, it can be used to carry any sensitive information which is meaningful to TService. Note that this paper is not intended to define the structure of its content, which could be addressed in another one.

2. AHFs in response message:

(a) Attest-Binder

It is used to make sure the response request is binding and uniquely identifies this transaction.

(b) Attest-Secrets

In this HF, these contained wrapped secrets will be provisioned back to the client. As noted earlier, this can be merged into the response AHLs in AtR of AtHS.

(c) Attest-Cargo

Similarly, it can be used to carry sensitive information/data back to the client.

3.4. Trusted Phase Communication

When AtB is allocated for the client, it can subsequently issue to do the real work. Basically, the TrR is an ordinary HTTP request with some extra AHLs, which are described in detail as follows:

1. AHFs in request message:

(a) Attest-Base-ID

It specifies which AtB to handle this request and should be validated by targeting TService before processing this request.

(b) Attest-Ticket

This field has been explained above, which is intended to authenticate this request and prevent other AHLs from being tampered with or being replayed.

(c) Attest-Cargo

As noted earlier, this field is optional, and the client can use it to transfer arbitrary sensitive information to TService.

(d) Attest-Base-Termination

We can include this AHF if it is the last TrR towards the AtB. It is recommended way to terminate a AtB actively. If the server never receives this header field in a request, attest base will eventually expire, which is specified in the Attest-Expires.

The termination method can be one of the following options:

cleanup

This means that other clients can reuse the terminated AtB.

destroy

Specify this method if the AtB should not be reused or shared by any other clients.

keep

This allows AtB to be shared with other clients. Be careful that this method is less safe as the residual data could be exploited and leaked to the next client if any.

2. AHFs in response message:

(a) Attest-Binder

As explained earlier, the HTTPA/2 uses it to ensure the IntAuth of both request and response.

(b) Attest-Cargo

As noted earlier, the TService can leverage this mechanism to transfer arbitrary sensitive information back to its client.

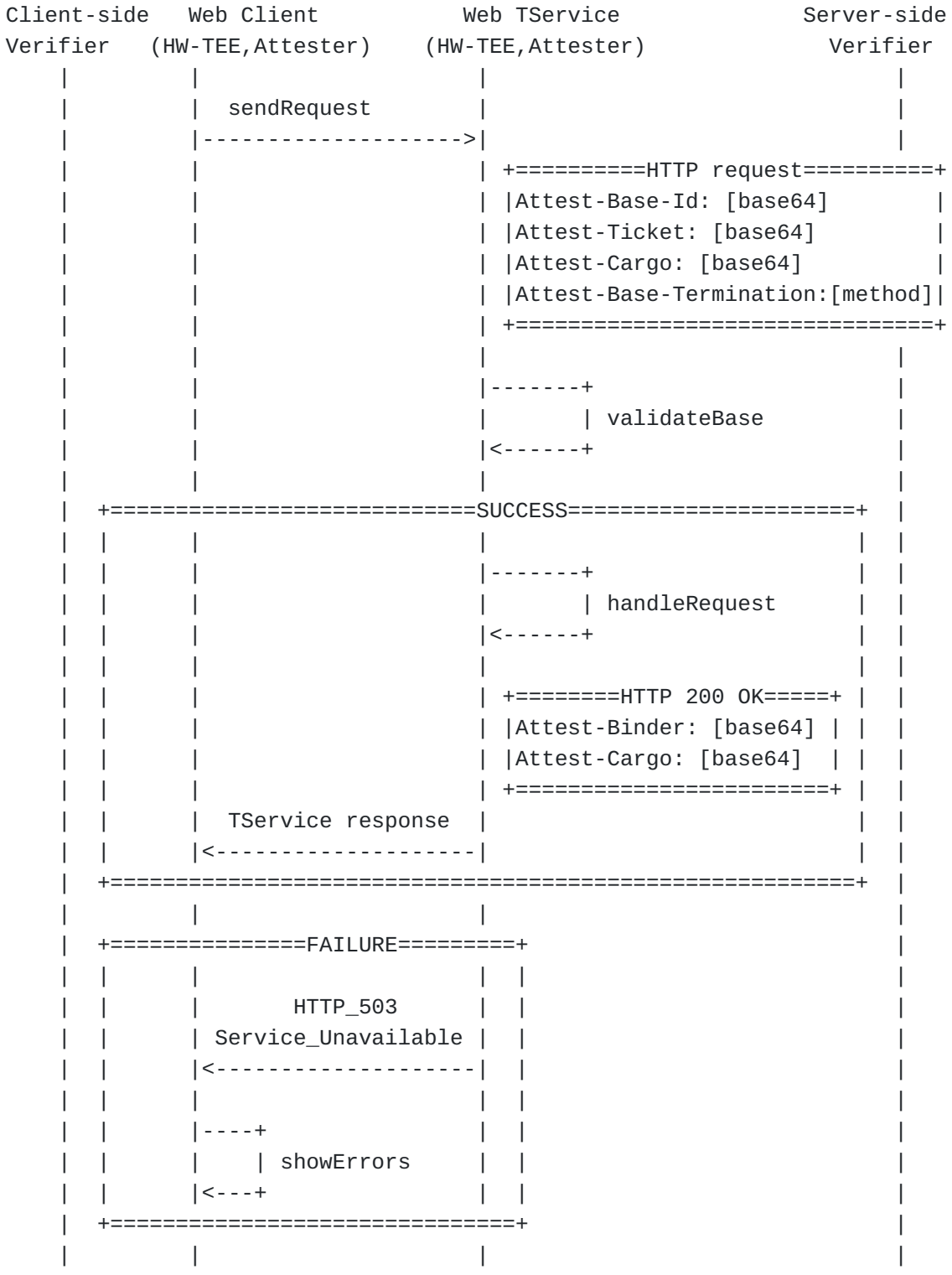


Figure 6: Trusted transaction

3.5. Changes in Mutual HTTPA/2(mHTTPA)

With mutual HTTPA being used, the client must be running on a TEE as TEE-based client (TClient), which can generate a client-side AtQ. The AtQ can be used to ensure the IntAuth of AtR by means of

including the digest of AHLs into its QUDD, and the server-side should have a proper trusted attestation authority to verify it.

This is the recommended approach to build mutual trust between TClient and TService, but the client-side usually lacks TEE feature support.

4. Security Considerations

4.1. Layer 7 End-to-End Protection

In cloud computing scenarios, intermediary nodes, such as L7 load balancer or reverse proxy, are commonly used to improve the network performance to deliver the best web experience. The secure communication based on TLS only protects transmitted data hop-by-hop at layer 5 (L5). The intermediary nodes

may need TLS termination to inspect HTTP messages in plain text for better network performance. Consequently, the intermediary nodes can read and modify any HTTP information at L7. It is the gap between L5 and L7 that may cause an underlying vulnerability. Therefore, the trust model, including intermediary nodes, which are over L5, is problematic because intermediary nodes are not necessarily trustworthy. Intermediary nodes may leak the privacy and manipulate the header lines of HTTP messages. Even in the case where intermediaries are fully trusted, an attacker may exploit the vulnerability of the hop-by-hop architecture which may lead to data breaches. HTTPA/2 helps protect AHLs and the sensitive information of HTTP messages end-to-end at L7. As long as the protection does not encrypt the necessary information against proxy operations, HTTPA/2 can help provide guarantees that the protected message can survive across middleboxes to reach the endpoint. Thus, the parts of HTTPA information without protection may be exploited to spoof or manipulate. If we want to protect every bit of HTTPA message hop-by-hop, TLS is highly recommended in combination with HTTPA/2

for use.

In the implementation, the TServices can make a privacy policy to determine to what degree the HTTPA message is protected to the L7 endpoint without TLS for better network performance. If the message is highly sensitive, entirely

TLS can come to help in addition, but only up to the security of the L5 hop point.

4.2. Replay Protection

A replay attack should be considered in terms of design and implementation. To mitigate replay attack, most AEAD algorithms

require a unique nonce for each message. In AtR, random numbers are used. In TrR, a sequential nonce is used on either endpoint accordingly. Assuming strictly increasing number in sequence, the replay attack can be easily detected if any received number is duplicated or no larger than the previously received number. For reliable transport, the policy can be made to accept only TrR with a nonce that is equal to the previous number plus one.

4.3. Downgrade Protection

The cryptographic parameters of configuration should be the same for both parties as if there is no presence of an attacker between them. We should always negotiate the preferred common parameters with the peer. If the negotiated parameters of configuration are different for both parties, it could make peers use a weaker cryptographic mode than the one they should use, thus leading to potentially a downgrade attack. In HTTPA/2, TService uses AtQ to authenticate its identity and the integrity of the AtHS to the client. In mutual HTTPA/2, the client uses AtQ carried by AtR for proving its own authenticity and the message integrity. Thus, the communication traffic of the handshake across intermediaries cannot be compromised by attackers.

4.4. Privacy Considerations

Privacy threats are considerably reduced by means of HTTPA/2 across intermediary nodes. End-to-end access restriction of integrity and encryption on the HTTPA/2 AHLs and payloads, which are not used to block proxy operations, aids in mitigating attacks to the communication between the client and the TService. On the other hand, the unprotected part of HTTP headers and payloads, which is also intended to be, may reveal information related to the sensitive and protected parts. Then private dat may be exposed. For example, the HTTP message fields visible to on-path entities are only used for the purpose of transporting the message to the end- point, whereas the AHLs and its binding pay- loads are encrypted or signed. It is possible for attackers to exploit the visible parts of HTTP messages to infer the encrypted information if the privacy preserving policy is not well set up. Unprotected error messages can reveal information of the security state in the communication between the endpoints. Unprotected signaling messages can reveal information of the reliable transport. The length of HTTPA/2 message fields can re- veal information about the message. TService may use a padding scheme to protect against traffic analysis. After all, HTTPA/2 provides a new dimension for applications to further protect privacy.

4.5. Roots of Trust (RoT)

Many security mechanisms are currently rooted in software; however, we have to trust underlying components, including software, firmware, and hardware. A vulnerability of the components could be easily exploited to compromise the security mechanisms when the RoT is broken. One way to reduce that risk of vulnerability is to choose a highly reliable RoT. RoT consists of trusted hardware, firmware, and software components that perform specific, critical security functions. RoT is supposed to be trusted and more secure, so it is usually used to pro- vide strong assurances for the desired security properties. In HTTPA/2, the inherent RoT is the AtB or TEEs, which provide a firm foundation to build security and trust. With AtB being used in HTTPA/2, we believe that the risks of security and privacy can be greatly reduced.

5. Acknowledgements

We would like to acknowledge the support from the HTTPA workgroup members, including our partners and reviewers. We thank them for their valuable feedback and suggestions.

6. IANA Considerations

This document has no IANA actions.

7. References

7.1. Normative References

[SEMANTICS] Fielding, R. T., Nottingham, M., and J. Reschke, "HTTP Semantics", Work in Progress, Internet-Draft, draft-ietf-httpbis-semantics-19, 12 September 2021, <https://datatracker.ietf.org/doc/html/draft-ietf-httpbis-semantics-19>.

7.2. Informative References

[I-D-1] Backman, A., Richer, J., and M. Sporny, "HTTP Message Signatures draft-ietfhttpbis-message-signatures-08", Work in Progress, Internet-Draft, IETF, 2019, <https://www.ietf.org/archive/id/draft-ietf-httpbis-message-signatures-08.html>.

[I-D-10] Anati, Ittai., Gueron, Shay., Johnson, Simon P., and Vincent R. Scarlata, "Innovative Technology for CPU Based Attestation and Sealing", <https://www.intel.com/content/

dam/develop/external/us/en/documents/hasp-2013-innovative-technology-for-attestation-and-sealing.pdf>.

[I-D-11] King, Gordon. and Hans. Wang, "HTTPA: HTTPS Attestable Protocol", DOI 10.48550/ARXIV.2110.07954, 2021, <https://arxiv.org/abs/2110.07954>.

[I-D-13] Krawczyk, Hugo., "SIGMA: The 'SIGn-andMAc' Approach to Authenticated DiffieHellman and Its Use in the IKE Protocols. In: Advances in Cryptology - CRYPTO 2003. Ed. by Dan Boneh. Berlin, Heidelberg: Springer Berlin Heidelberg pp. 400-425", ISBN 978-3-540-45146-4, 2003.

[I-D-16] Ménétrey et al., Jämes., "An Exploratory Study of Attestation Mechanisms for Trusted Execution Environments", DOI 10.48550/ARXIV.2204.06790, 2022, <https://arxiv.org/abs/2204.06790>.

[I-D-19] "Remote Attestation", <https://www.intel.com/content/www/us/en/developer/tools/software-guard-extensions/attestation-services.html>.

[I-D-2] Banks, Alexander Sprogø., Kisiel, Marek., and Philip. Korsholm, "Remote Attestation: A Literature Review. In: CoRRabs/2105.02466 (2021). arXiv: 2105.02466.", 2021, <https://arxiv.org/abs/2105.02466>.

[I-D-21] Regenscheid, Andrew., "Roots of Trust (RoT)", August 2016, <https://csrc.nist.gov/Projects/Hardware-Roots-of-Trust>.

[I-D-3] Bhargavan et al, Karthikeyan., "Downgrade resilience in key-exchange protocols. In: 2016 IEEE Symposium on Security and Privacy (SP)", IEEE pp. 506-525, 2016.

[I-D-4] Birkholz et al, H., "Remote Attestation Procedures Architecture draftietf-rats-architecture-12", Work in Progress, Internet-Draft, IETF, 2021, <https://www.ietf.org/archive/id/draft-ietf-rats-architecture-12.html>.

[I-D-5] Marinescu, Dan C., "Execution Environments", <https://www.sciencedirect.com/topics/computer-science/execution-environments>.

[I-D-8] Helble et al., Sarah C., "Flexible Mechanisms for Remote Attestation”. In: ACM Trans. Priv. Secur. 24.4", ISSN 2471-2566, DOI 10.1145/3470535, September 2021, <https://doi.org/10.1145/3470535>.

[RFC5116] McGrew, David., "An Interface and Algorithms for Authenticated Encryption", RFC 5116, DOI 10.17487/RFC5116, January 2008, <https://www.rfc-editor.org/info/rfc5116>.

[RFC5869] Krawczyk, Dr. Hugo. and Pasi. Eronen, "HMAC-based Extract-and-Expand Key Derivation Function (HKDF)", RFC 5869, DOI 10.17487/RFC5869, May 2010, <https://www.rfc-editor.org/info/rfc5869>.

[RFC6797] Hodges, Jeff., Jackson, Collin., and Adam. Barth, "HTTP Strict Transport Security (HSTS)", RFC 6797, DOI 10.17487/RFC6797, November 2012, <https://www.rfc-editor.org/info/rfc6797>.

[RFC7230] Fielding, Roy T. and Julian. Reschke, "Hypertext Transfer Protocol (HTTP/1.1): Message Syntax and Routing", RFC 7230, DOI 10.17487/RFC7230, June 2014, <https://www.rfc-editor.org/info/rfc7230>.

[RFC7231] Fielding, Roy T. and Julian. Reschke, "Hypertext Transfer Protocol (HTTP/1.1): Semantics and Content", RFC 7231, DOI 10.17487/RFC7231, June 2014, <https://www.rfc-editor.org/info/rfc7231>.

[RFC7748] Langley, Adam., Hamburg, Mike., and Sean. Turner, "Elliptic Curves for Security", RFC 7748, DOI 10.17487/RFC7748, January 2016, <https://www.rfc-editor.org/info/rfc7748>.

[RFC8422] Nir, Yoav., Josefsson, Simon., and Manuel. Pégourié-Gonnard, "Elliptic Curve Cryptography (ECC) Cipher Suites for Transport Layer Security (TLS) Versions 1.2 and Earlier", RFC 8422, DOI 10.17487/RFC8422, August 2018, <https://www.rfc-editor.org/info/rfc8422>.

[RFC8446] Rescorla, Eric., "The Transport Layer Security (TLS) Protocol Version 1.3", RFC 8446, DOI 10.17487/RFC8446, August 2018, <https://www.rfc-editor.org/info/rfc8446>.

[RFC8613] Selander et al., Göran., "Object Security for Constrained RESTful Environments (OSCORE)", RFC 8813, DOI 10.17487/RFC8613, July 2019, <https://www.rfceditor.org/info/rfc8613>.

[RFC8941] Nottingham, Mark. and Poul-Henning. Kamp, "Structured Field Values for HTTP", RFC 8941, DOI 10.17487/RFC8941, February 2021, <https://www.rfc-editor.org/info/rfc8941>.

Authors' Addresses

Shih-han Wang
Intel Corporation
US
Email: hans.wang@intel.com

Gordon King
Intel Corporation
US
Email: gordon.king@intel.com

Nick Li
Intel Corporation
US
Email: nick.li@intel.com

Ned Smith
Intel Corporation
US
Email: ned.smith@intel.com

Krzysztof Sandowicz
Intel Corporation
PL
Email: krzysztof.sandowicz@intel.com Phone: +48587661619