iab. TECH LAB

ads.cert Authenticated Connections Protocol Specification

January 2022

Presented by the IAB Tech Lab Cryptographic Security Foundations working group

Please email support@iabtechlab.com with feedback or questions. This document is available online at https://iabtechlab.com/standards/ads-cert/

Program Leaders:
Curtis Light, Staff Software Engineer - Google
Rob Hazan, Senior Director, Product - Index Exchange

Other Significant Contributions from:
Ben Antier, CEO - Publica
Nabhan El-Rahman, CTO - Publica
Joshua Gross, Senior Engineering Lead - Index Exchange
Bret Ikehara, Staff Software Engineer, Publica
Johnny Li, Software Engineer, Index Exchange
Amit Shetty, Programmatic Products & Partnerships - IAB Tech Lab
Sam Mansour, Principal Product Manager - Moat
Miguel Morales, CTO & Co-Founder - Lucidity Tech
Colm Geraghty, Principal Architect - Verizon Media Group
Mani Gandham, Engineering - Index Exchange
James Wilhite, Director of Product management, Publica

IAB Tech Lab Lead:
Amit Shetty
VP, Programmatic Products & Partnerships - IAB Tech Lab

About IAB Tech Lab

The IAB Technology Laboratory (Tech Lab) is a non-profit research and development consortium that produces and provides standards, software, and services to drive growth of an effective and sustainable global digital media ecosystem. Comprised of digital publishers and ad technology firms as well as marketers, agencies, and other companies with interests in the interactive marketing arena, IAB Tech Lab aims to enable brand and media growth via a transparent, safe, effective supply chain, simpler and more consistent measurement, and better advertising experiences for consumers, with a focus on mobile and TV/digital video channel enablement. The IAB Tech Lab portfolio includes the DigiTrust real-time standardized identity service designed to improve the digital experience for consumers, publishers, advertisers, and third-party platforms. Board members include AppNexus, ExtremeReach, Google, GroupM, Hearst Digital Media, Integral Ad Science, Index Exchange, LinkedIn, MediaMath, Microsoft, Moat, Pandora, PubMatic, Quantcast, Telaria, The Trade Desk, and Yahoo! Japan. Established in 2014, the IAB Tech Lab is headquartered in New York City with an office in San Francisco and representation in Seattle and London.

Learn more about IAB Tech Lab here: www.iabtechlab.com

TABLE OF CONTENTS

Documents

- ads.cert Primer
- ads.cert Open Source Software Implementer's Guide
- ads.cert Authenticated Connections Protocol Specification (this doc)
- ads.cert Call Signs Protocol Specification
- ads.cert Open Source Software Design Doc

These docs are all available at https://iabtechlab.com/ads-cert

Getting Started

If you're new to ads.cert or the Authenticated Connections protocol, we recommend starting with the *ads.cert Primer*. That is the recommended path for the vast majority of programmatic ads ecosystem participants.

Objective

This document describes the message formats used by the ads.cert Authenticated Connections protocol. Its purpose is to provide reference material about the protocols implemented by the ads.cert Open Source Software hosted by IAB Tech Lab. Generally speaking, we **strongly recommend** adopting the community-maintained open source applications within your organization. With this in mind, it is technically possible to reimplement the protocols using this specification and open source code as a reference. Should you choose to do so, this document provides the necessary details.

Background

According to publicly-available eMarketer estimates, global programmatic ad spend in 2021 is forecasted at around $455 Billion USD. While individual transaction sizes in our industry tend to be very small, the total volume is massive. This tremendous amount of economic activity creates room for bad actors to siphon ad-spend away from legitimate publishers, and makes their activities more difficult to detect (signal is lost in the noise). If they can co-opt even a small fraction of this economic activity for their own purposes, the financial rewards are quite large.

It is the position of the IAB Tech Lab's Cryptographic Security Foundations Working Group that the existing tools for securing this economic activity against malicious actors are insufficient, particularly in the case of transactions or activities that originate from servers as opposed to client devices. In the long-run, securing online advertising transactions against fraud will require a multi-layered approach: the user, device, publisher, ad tech intermediaries, and buyer all need to be authenticated and validated to ensure the integrity of a transaction. One necessary component of this multi-layered strategy is the ability to secure communication between servers -- this is the purpose of ads.cert Authenticated Connections.

There are many valid use cases for advertising transactions or activities that originate from servers; for example: ads stitching into streaming video or audio content (Server-Side Ad Insertion), server-side billing notifications for mobile app activity via burl, fetching native creative markup to transform it into HTML that can be displayed on a webpage. In the future, we expect the number of legitimate use cases for server-side activity to grow, as more ads-related work is offloaded to servers, rather than happening on the client-side. That being said, one of the primary motivating use cases for this protocol is that of securing video transactions originating from SSAI platforms..

Recent security research has highlighted schemes where parties have attempted to impersonate SSAI platforms: these schemes are challenging to identify, since traffic appears to originate from the same cloud platforms and hosting providers that service real SSAI businesses. The working group recognized the importance of addressing this particular problem, and this was the driver for prioritizing ads.cert Authenticated Connections over other use cases. In particular, we recommend the use of ads.cert Authenticated Connections in billing notifications, letting these notification recipients identify the SSAI platform that is claiming to be sending the billing notification. The recipients can then decide whether or not they trust the SSAI platform. This prevents fraudsters from spinning up servers on the cloud, claiming to be a valid SSAI platform and generating fraudulent ad impressions.

Public key and shared secret generation

The ads.cert protocols use an RFC 7748 X25519 algorithm to generate a public key from a private key. In addition, the X25519 algorithm will combine one party's private key with another party's public key to generate a shared secret key. The reciprocal calculation using corresponding keys in the same relationship (their private key, our public key) will arrive at the same result.

From the RFC:

> The X25519 function can be used in an Elliptic Curve Diffie-Hellman (ECDH) protocol as follows:
>
> Alice generates 32 random bytes in a[0] to a[31] and transmits K_A = X25519(a, 9) to Bob, where 9 is the u-coordinate of the base point and is encoded as a byte with value 9, followed by 31 zero bytes.
>
> Bob similarly generates 32 random bytes in b[0] to b[31], computes K_B = X25519(b, 9), and transmits it to Alice.
>
> Using their generated values and the received input, Alice computes X25519(a, K_B) and Bob computes X25519(b, K_A).
>
> Both now share K = X25519(a, X25519(b, 9)) = X25519(b, X25519(a, 9)) as a shared secret. Both MAY check, without leaking extra information about the value of K, whether K is the all-zero value and abort if so (see below). Alice and Bob can then use a key-derivation function that includes K, K_A, and K_B to derive a symmetric key.

Implementations of X25519 are widely available in various software languages. You MUST use a secure pseudorandom number generator to generate private keys. Failure to do so could result in your private key becoming compromised by an attacker and allow them to impersonate themselves as your business to others.

Use the public key obtained from counterparty DNS records to calculate shared secrets as described below.

Message format specifications

Public key DNS record

Record name

Public keys shall be published at the ads.cert Call Sign Internet domain established for this purpose, within the DNS record name "`_delivery._adscert`" under the "public suffix + 1" (PS+1) domain name (as published by publicsuffix.org) registered by the implementing company (e.g. example.com). Only ICANN-assigned suffixes are valid for this purpose. Use of the "private" section of the publicsuffix.org file isn't supported.

Record format

The record value looks like the following:

```
v=adcrtd k=x25519 h=sha256 p=w8f3160kEklY-nKuxogvn5PsZQLfkWWE0gUq_4JfFm8
```

The DNS record consists of the following fields:

Field	Description
v	Set to the constant value "adcrtd" to indicate that this record provides an ads.cert delivery key. This token MUST appear at the start of the DNS record value.
k	Set to key algorithm identifier, designed for forward compatibility if we need to transition to another scheme in the future. Currently this will always be set to "x25519" representing the X25519 Diffie-Hellman key exchange algorithm.
h	Set to the hash algorithm identifier, again designed for forward compatibility. Currently this will always be set to "sha256" representing the SHA-256 secure hashing algorithm.
p	Values are 32 byte public keys represented as 43 byte base64 encoded strings, RFC 4648 "URL-safe" variant.

Fields are delimited by a single space character. Key/value pairs are separated by an equals character.

All keys and values are case sensitive.

Authority delegation record format

Record name

Authority delegation records shall be published within the DNS record name "`_adscert`" and same domain public suffix rules as above.

Record format

The record value looks like the following:

```
v=adpf a=exchange-holding-company.ga
```

The DNS record consists of the following fields:

Field	Description
v	Set to the constant value "adpf" to indicate that this record provides an ads.cert authority delegation record. This token MUST appear at the start of the DNS record value.
a	Authority domain associated with the operational domain publishing this DNS record. The domain must match the domain used to publish the public keys as described in the prior section (excluding the "_delivery._adscert" subdomain portion. Domains MUST be provided and interpreted in the ASCII character set. Extended character set domains may be provided in punycode format.

Delimiters are the same as above.

All keys and values are case sensitive. Domains MUST be provided in lowercase.

Signature HTTP request header format

An HTTP request may contain one or more ads.cert Authenticated Connections signature messages. Each signature message will appear within its own HTTP request header:

```
X-Ads-Cert-Auth: <<first signature message>>
X-Ads-Cert-Auth: <<second signature message>>
```

Most implementations will only require transmitting one signature message. No behavior for a second signature message is currently defined.

Signature message format

The signature message consists of three parts in this order:

- The message being signed
- The separator value (“; “)
- The signatures associated with message

The signature message looks like the following example[1]:

```
from=ssai-serving.tk&from_key=w8f316&invoking=ad-exchange.tk&nonce=u_sDzKMIp0eD&status=0&timestamp=210519T174337&to=exchange-holding-company.ga&to_key=bBvfZU&status=4; sigb=t1TupK6wn8pn&sigu=FQ5OQ6TmF2xU
```

A semicolon and space character separate the message from the signature and is not used within the signature calculation. Both values are encoded in RFC 3986 query string format consisting of key=value pairs separated by ampersand and appropriately escaped per RFC spec. Keys may appear in any order within the message string.

Fields defined within the message and signature component:

Field	Description
from	The ads.cert Call Sign domain of the party sending the request. Parties receiving HTTP requests to verify will use this domain to obtain the corresponding `_delivery._adscert` DNS TXT record.
from_key	The first 6 characters of the sending party’s public key
invoking	The domain for the URL hostname being invoked
nonce	Randomly generated number from the sending party in URL safe base64 encoded format. Nonce length is 12 characters.
timestamp	The time of generating signature in format: YYMMDDTHHMMSS. Times are represented in the UTC time zone.
to	The ads.cert Call Sign domain of the party receiving the signature
to_key	The first 6 characters of the receiving party’s public key

[1] Attribute naming not yet finalized

Field	Description
status	Integer value representing signing protocol status values other than “OK” (1) as defined here. For example, this field may be used to indicate that the signer is suppressing a signature for a specific reason. It can also communicate a DNS issue retrieving the counterparty’s keys.
sigb	The signature over the message and body of the request (minimum 12 characters)
sigu	The signature over the message, body, and URL of the request (minimum 12 characters)

Signers SHOULD transmit a signature message on all requests so that they declare their identity and intentions even if the information can’t be verified with a signature. This information helps nonparticipating recipients understand the coverage and opportunity of counterparties with whom they can receive authentication. These unsigned messages should populate the “from” and “status” fields at a minimum, but they may include other values at sender’s discretion.

Computing signatures for signing and verification

Calculating signatures

Calculate a SHA256 hash each over the request body and request URL.

The body signature (sigb) consists of an HMAC over the message concatenated with the body SHA256 hash:

$$bodySignatureBytes = HMAC_{SHA256}(sharedSecret, message \,||\, bodyHash)$$

The URL signature (sigu) also concatenates the URL SHA256 hash into the message.

$$urlSignatureBytes = HMAC_{SHA256}(sharedSecret, message \,||\, bodyHash \,||\, urlHash)$$

Appending to the prior HMAC construct lets the algorithm reuse the prior hash calculation:

```
h := hmac.New(sha256.New, sharedSecret)

h.Write([]byte(message))
h.Write(bodyHash)
bodyHMAC := h.Sum(nil)

h.Write(urlHash)
urlHMAC := h.Sum(nil)
```

Encode the resulting HMAC values as URL-safe base64 strings and truncate to obtain at least the first 12 characters. This provides (6-bit*12 char=) 72-bit security and conforms to NIST Special Publication 800-107 guidelines regarding HMAC truncation. These will be the signatures provided in the sigb and sigu fields. Signatures shorter than 12 characters MUST be treated as invalid input.

Note: to allow forward compatibility with increased security from using longer signatures, verifiers MUST be capable of accepting and validating signatures up to the maximum 43 character length available within a 256-bit hashing scheme. Verifiers MAY require longer signatures than the 12 character default. Defining the circumstances when a signer will be expected to submit a longer signature value is beyond the scope of this specification.

Calculating verifications

Reconstruct the URL used by the client to invoke the server, including the scheme (https), hostname, path, and query string. The path and query string will appear in the HTTP GET or POST stanza, while the hostname will need to be obtained from the Host header. Alternatively, it may be necessary to obtain the hostname from the TLS server name identification (SNI) field passed to the server using a separate mechanism (e.g. custom HTTP request header).

Use the same algorithm as above to recompute the signature over the message, body hash, and URL hash. Compare the values with those supplied in the signature message from the remote client.

Implementation Recommendations

Please refer to the *Open Source Software Implementer's Guide* that accompanies this spec document, which walks through using the open source implementation of this protocol.