iab. TECH LAB

# ads.cert Open Source Software

Implementer's Guide

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](http://www.iabtechlab.com)

## TABLE OF CONTENTS

## Documentation links

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

## Introduction

The ads.cert protocols add security to various aspects of programmatic advertising.

Authenticated Connections adds origin authentication and tamper resistance to HTTP requests made server-to-server, such as bid request, creative fetches, and impression pings. In particular, parties receiving these requests now have a verifiable way to know who is sending them even when they don't have a direct relationship with the other party.

Authenticated Delivery (in development, when introduced) adds authentication and tamper resistance to bid requests and bid parameters as they traverse the supply path. While our initial release doesn't focus on this protocol, the infrastructure we're adding will enable support for it in a future version.

These protocols address real-world programmatic ads security issues faced by the industry. For example, recent security research has highlighted schemes where parties have attempted to impersonate server-side ad insertion (SSAI) platforms: challenging to identify, since traffic appears to originate from the same cloud platforms and hosting providers that service real SSAI businesses.

This document provides guidance for parties seeking to adopt the Authenticated Connections protocol to secure server-to-server communication, using the *ads.cert* open source software library. This is the recommended approach for adopters, rather than creating bespoke implementations of the various sub-components (key generation, signing, verification, etc).

This implementer's guide walks you through the details of ads.cert that you'll need to know to support it within your organization. Many technical details will be handled for you by open source code that's hosted by IAB Tech Lab and contributed by the community, so this guide focuses on the steps to integrate it with your software. It also explains how you'll publish keys in a way that lets organizations authenticate each other.

## How ads.cert Authenticated Connections works

In ads.cert Authenticated Connections, you'll be adding a standardized HTTP request header containing a signature that secures:

- URL being invoked
- Body of POST requests
- Timestamp, origin, and destination values

Together, these help show that the request came from the originator it claims and hasn't been tampered with.

The request header looks like the following[1]:

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

| Field | Description |
|---:|---|
| from | The ads.cert Call Sign domain of the party sending the request |
| 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 base64 encoded. |
| timestamp | The time of generating signature in format: YYMMDDTHHMMSS. This should be in UTC. |
| 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 |
| sigb | The signature over the message and body of the request |
| sigu | The signature over the message, body, and URL of the request |

One very important feature of this signing scheme to highlight is that signatures are between two parties: the signer and the identified verifier. **Anyone not a party to this signature cannot verify it.** This characteristic is by design: for implementers, it prevents third parties from attempting to interpret information from signatures. The symmetric nature of the signatures also

[1] Attribute naming not yet finalized

means that they cannot be used for non-repudiation of events: one business cannot record signed messages and provide them as “proof” of activity. Most important is privacy: it prevents the protocol from generating a public key verifiable cryptographic record of consumer activity on the Internet.

For this reason, both the signer and verifier must publish public keys that mutually authenticate each other. From those values, a cryptographic algorithm called “X25519” calculates a “Diffie-Hellman” shared secret between the parties, used in signing and verifying messages. Our open source software implementation handles these details for you, so you need not worry about the details other than this non-intuitive requirement for both parties to provide keys.

The signer and verifier both publish their respective public keys in DNS so that the counterparty may find them. This is a departure from other IAB Tech Lab specifications that publish files at well-known URLs on web sites, and it should be easier to reliably implement. DNS serves as a common location for publishing cryptographic keys in other security protocols such as that used by email servers (a strong influence on our own product).

Let's say a signer (using the ads.cert Call Sign ssai-serving.tk) wants to invoke this URL (`https://ads.ad-exchange.tk/impression?auction=6d8a826b02a2715e44`) hosted under the domain ad-exchange.tk, with the latter business operated by that identified with ads.cert Call Sign exchange-holding-company.ga. The latter party will verify the signature.

| | |
|---|---|
| The SSAI platform: | ssai-serving.tk |
| needs to invoke: | `https://ads.ad-exchange.tk/impression?auction=6d8a826b02a2715e44` |
| hosted as: | ad-exchange.tk |
| which is controlled by: | exchange-holding-company.ga |

Both the signer and verifier start by publishing their respective public keys in DNS on their ads.cert Call Sign domain.

```
$ host -t TXT _delivery._adscert.ssai-serving.tk
descriptive text "v=adcrtd k=x25519 h=sha256 p=w8f3160kEklY-
nKuxogvn5PsZQLfkWWE0gUq_4JfFm8"

$ host -t TXT _delivery._adscert.exchange-holding-company.ga
descriptive text "v=adcrtd k=x25519 h=sha256 p=bBvfZUTPDGIFiOq-
WivBoOEYWM5mA1kaEfpDaoYtfHg"
```

These well-known DNS subdomains let the counterparties find each other once they're aware of the need to exchange credentials.

One additional link needs to be made for the signer to know whose key will be used as the verifier. The latter also publishes an additional record on an ad-exchange.tk subdomain that indicates who is the ads.cert Call Sign domain authority responsible for verifying requests. The signer uses this record to understand that relationship.

```
$ host -t TXT _adscert.ad-exchange.tk
descriptive text "v=adpf a=exchange-holding-company.ga"
```

With these three DNS records in place, the signer and verifier can act. On preparing to send an HTTP request, the signer uses the language-specific ads.cert API to generate a signature and adds that signature into the HTTP request headers.

```
signature, _ := signer.SignAuthenticatedConnection(
      adscert.AuthenticatedConnectionSignatureParams{
            DestinationURL: destinationURL,
            RequestBody:    []byte{}})
req.Header["X-Ads-Cert-Auth"] = signature.SignatureMessages
```

The verifier receiving the request reconstructs the URL that was invoked by the signer. It passes that URL, the request body, and the signature header to the ads.cert verification API to check for a valid signature. The verifier may then act on that verification outcome accordingly, although the safest initial policy may be to just log the outcome.

```
signatureHeaders := req.Header["X-Ads-Cert-Auth"]
reconstructedURL := …
body, _ := ioutil.ReadAll(req.Body)
verification, _ := signer.VerifyAuthenticatedConnection(
      adscert.AuthenticatedConnectionSignatureParams{
            DestinationURL: reconstructedURL,
            RequestBody:    body,
            SignatureMessageToVerify: signatureHeaders})
```

The ads.cert OSS implementation encapsulates the details behind these operations using a simple API.

Review the examples found at https://github.com/IABTechLab/adscert to see API used in working sample code.

## The ads.cert open source software

There are three integration models that we support (or plan to support) using our open source library.

- In-process integration of our Golang open source library; ephemeral DNS
- RPC integration using an open source signatory server we provide; ephemeral DNS
- Centralized crawling infrastructure with DNS persistence, suitable for larger enterprises

Clients must choose the appropriate solution for their online HTTP client environment, as the signature must be calculated at some point where the full request body and URL being invoked are known.

Servers may integrate these options in either an online or offline[2] (logs processing) environment, depending on whether the server implementer wants to obtain real-time signature validation feedback.

Starting out, an implementer natively using Go may choose to implement in-process integration, as this simplest solution only requires importing a Go module and using it in your code. The application will rely on DNS lookups from the hosting OS to obtain necessary info. Clients written in other languages will need to deploy the supplied RPC server (a wrapper around the native Go implementation) to invoke the signing/verification logic. This RPC server can be hosted over a network, or the server can be deployed as a sidecar/subprocess to the main application.

[2] Offline signature verification requires logging (1) the signature message, (2) a hash of the HTTP request body, and (3) a hash of the invoked URL. Support for this verification mode has not yet been added to the API.

# Implementation process

The remainder of this document walks you through setting this up for your organization, including:

- Integrating the ads.cert open source API within your software
- Selecting the domain used to identify your organization to others
- Generating private keys for initial setup and rotation
- Publishing your public keys in DNS
- (For verifiers) Publishing delegation DNS records

## Integrate the ads.cert OSS API

First, initialize the integrator API based on the desired configuration at an appropriate point in your application startup.

```
import (
	"flag"
	"github.com/IABTechLab/adscert/pkg/adscert"
)

func main() {
	flag.Parse()
	...
	config := adscert.ConfigureIntegratorFromFlags()
	signer := adscert.NewAuthenticatedConnectionsSigner(config)
	...
}
```

Some organizations prefer using command line flags to pass in runtime parameters, while others prefer configuration files in YAML, etc.  Rather than dictate a specific configuration technique, the ads.cert implementation will provide initialization parameters through a generic data structure, and different pluggable strategies may be used to build it via flags, files, databases, etc.

This technique also lets integrators configure the signer in a predictable way for use within testing environments, as the keys, wall clock, and pseudorandom number generator can be seeded with predictable values while still allowing for broader end-to-end code execution.

Once initialized, the ads.cert signer may be used to sign and verify signatures using the SignAuthenticatedConnection and VerifyAuthenticatedConnection functions as shown in the prior section.

## Select your identification domain

You'll need to determine the canonical domain name representing how you want your business to be identified within the ads ecosystem, as this will be the domain under which you'll publish public keys. This requires a bit more thought than it sounds, but the criteria should be straightforward.

Your identification domain needs to *securely* distribute public keys. DNS makes this challenging, as DNS doesn't contain cryptographic security protocols on its own. While somewhat challenging to accomplish, there are situations where an attacker could falsify DNS records and attribute an illegitimate key to your organization.

We structured ads.cert to permit publishing your keys under a dedicated domain that can activate the DNSSEC protocol, adding an additional layer of security that minimizes this attack risk. DNSSEC signs DNS records with an encryption key, and most hosting provider DNS resolvers should verify that signature upon lookup to check for tampering.

Adopting DNSSEC is optional. We recommend signers adopt a DNSSEC-capable identification domain from the outset to better protect from DNS spoofing attacks that could impersonate your organization to others, but it isn't mandatory. You have the option to enable DNSSEC at a later time. We present these details so that you can evaluate the impersonation risks that your organization may face and plan for current or potential future use of this feature.

Before adopting DNSSEC on a domain or DNS zone, evaluate the activities associated with that domain. Most large consumer-facing Internet platforms DO NOT implement DNSSEC for their consumer-facing traffic since the certificate authority ecosystem provides a suitable alternative for browsers to authenticate servers. A domain you currently use for advertising delivery, for instance, might not be a good candidate for activating DNSSEC, as you could experience a small fractional loss of traffic. DNSSEC misconfigurations have also been the cause of various DNS outages, so isolating this to a key distribution domain reduces this risk. We suggest that you allocate a separate domain for this purpose, such as a corporate vanity domain. Verify that your desired top-level domain, domain registrar, and authoritative DNS server support DNSSEC before trying to set this up. Various online tools such as VeriSign's DNSSEC analyzer may be helpful for checking your domain's DNSSEC status.

## Generate your keyring file

The ads.cert OSS implementation helps you manage your private keys in a safe and secure way. It uses a keyring JSON file containing encrypted private keys to maintain your key configuration. Following a "configuration as code" DevOps methodology, you can safely store this keyring file within your source control system alongside other configuration details. The strong private key encryption protects these values in a way that even full public disclosure of your keyring file should be safe. A "key encryption key" (KEK), maintained separately, lets the application decrypt these keys for use at runtime.

We **strongly recommend** that you use a software version control system (Git, Subversion, etc) to track changes to your key configuration file, although you can just keep a filesystem copy (with adequate backups) at your own risk. The VCS will allow you to reconstruct changes applied to your keyring over time.

This scheme provides the most security when operating in an environment where a key management system (KMS) is available, as this doesn't disclose the KEK to anyone, and the KMS only provides services that will let authorized roles encrypt/decrypt using the key identified by URI. We **strongly recommend** that you use a KMS. Security can be further enhanced by limiting which roles have access to the key decryption operations. By only granting decrypt privileges to the roles that run applications, you can retain tight control over potential attack vectors that could compromise security of your ads.cert key material, as changes to permissions or deploying unauthorized code within the environment can generate auditable events (if configured).

If you do not have access to a KMS or choose to use a more simple solution, you may instead use a KEK provided by the local environment (e.g. filesystem). We do not encourage using this option.

Your keyring file should either be deployed alongside your application in a configuration sidecar (e.g. mounted to your containerized application) or built into your application deployment. If you choose the latter option, keep in mind that key rotation opportunities will be coupled to your software release schedule and could contain rollback risks. Follow the same process you would use for deploying other such application startup configuration data of this nature. We **strongly recommend** incorporating your keyring distribution process into your deployment automation.

In addition to holding encrypted private keys, your ads.cert keyring contains various metadata about the keys which will help you manage versioning and key rotation. All key versions follow this lifecycle:

- New - key generated and submitted to your source control system but not yet fully deployed to your production environment. Not all application instances in your fleet would be able to use the key yet.
- Published - key fully rolled out and stable within your application fleet. The public key can be safely published in DNS, and counterparties may start using the key.
- Primary - key available in DNS for sufficient time so that counterparties would have had opportunity to crawl it. Private key now used for signing operations, and counterparties are assumed to be able to verify using the public key.
- Secondary - a former primary key that has been replaced by a newer primary key. The public key remains published in DNS and usable for signature verification, but the key is being wound down for signing.
- Archived - a former secondary key that is now unpublished from DNS. Counterparties may have outdated info about your keys, so this state will let your system continue to verify signatures expecting the obsolete version.
- Removed - a former archived key that has been removed from the keyring and no longer available for signing/verification.

Key rotation need not be a frequent event, but this arrangement lets you perform it in a controlled fashion where you can gradually ramp up use of a new key for signing over a longer (e.g. multiple day) period.  Unlike symmetric keys used for encryption (e.g. the AES algorithms), the public keys and derived shared secrets do not suffer from cryptographic key “wear-out” that occurs after heavy use.  The main benefit is that rotation reduces risks that a compromised key could be used for illegitimate purposes for an extended period, although this only holds true if the attacker isn't exploiting a persistent vulnerability that would provide access to new private key versions.

Because the keyring captures this lifecycle information, our tools support deterministic DNS record generation based on the keyring file state.  The tool will output the precise DNS record value to copy/paste into your DNS authoritative servers.  Currently this is a manual process, although we are investigating automation options to configure automatically pushing DNS updates and/or enabling automated monitoring of DNS records being out-of-sync from your keyring.

To avoid duplicating material in these draft documents, please see this section of the ads.cert Open Source Implementation Design Doc for a walkthrough of the key generation tooling.

## Publish your public key in DNS

The key generation tool assembles the public key message you'll publish. Follow the in-app instructions to create your new public key record or update an existing one.

You'll publish this record within a `_delivery._adscert` subdomain[3] of your organization identity domain. Follow the instructions in the keyring tool for the customized details regarding what to publish.

Select a time-to-live (TTL) for the DNS record that will minimize unnecessary DNS fetches but still let you rotate keys without introducing extensive delays. A 300 second (five minute) TTL could be a reasonable starting point for initial testing, and this value might later be increased to longer (e.g. 3600 seconds) after confirming that the configuration works as desired. Your keys should not change frequently, so a longer TTL should be sufficient.

The ads.cert Open Source Implementation automatically fetches DNS records. While you should be aware of the underlying protocols, it is not necessary to write your own code to consume these records.

## Publish delegation DNS records

If you will be receiving HTTP requests with signatures you want to verify, you will need to publish a DNS record that reflects the domain which serves as the signing authority for the domain receiving the request.

Earlier in this document, we provided an example URL (`https://ads.ad-exchange.tk/...`) that will receive ad requests. To associate this domain with a signing authority domain, the implementer publishes a record under the well-known subdomain name `_adscert.ad-exchange.tk` that points to `exchange-holding-company.ga` as the proper signing authority:

```
$ host -t TXT _adscert.ad-exchange.tk
descriptive text "v=adpf a=exchange-holding-company.ga"
```

Multiple "operational" domains of this nature can point to the same authority domain.

[3] Naming not yet finalized

## Testing your integration

### Testing signers

Use the supplied integration testing server to receive sample requests from your QA or live advertising environment and exercise the DNS records you publish.  To facilitate testing, the `ads.ad-exchange.tk` hostname resolves to 127.0.0.1, so invoking URLs using it will invoke the test server running locally on your workstation.

```
$ host ads.ad-exchange.tk
ads.ad-exchange.tk has address 127.0.0.1
```

This also results in the software automatically discovering the keys published on the `exchange-holding-company.ga` domain.

### Testing verifiers

Use the supplied integration testing client to generate sample requests that will exercise the DNS records you publish and submit HTTP requests to the URL you specify.

## Monitoring your deployment

The ads.cert API provides feedback useful for monitoring in a few forms:

- The response message produced by the API provides structured feedback about the operation's outcome
- The implementer's stub generates monitoring metrics that can be collected by the organization's monitoring systems
- The core signing and DNS crawl components collect exportable monitoring metrics

Choose the technique that works best for your organization.  Review the design document for more information on available metrics.