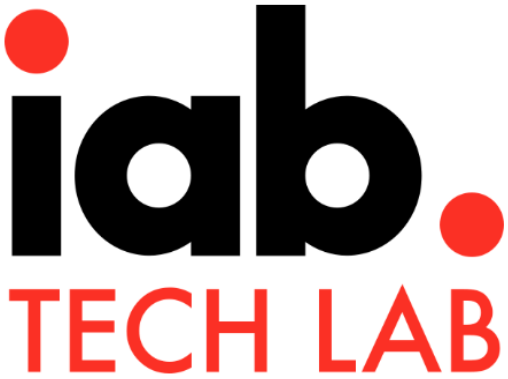

OpenRTB 3.0 Authentication: Signed Bid Requests

Proposed Update in the OpenRTB 3.0 Framework

DRAFT FOR PUBLIC COMMENT
September 2017

Executive Summary

“OpenRTB 3.0 Authentication: Signed Bid Requests” is part of the OpenRTB 3.0 Framework release.

Comments and feedback should be submitted using this OpenRTB 3.0 Framework Public Comment Submission Form. Public comments are open until December 15, 2017.

A major component of OpenRTB 3.0 is the shift to an authenticated supply chain. This move to standardize cryptographically signed bid requests is the next step needed in OpenRTB to ensure security and trust in the supply path hops in real time bidding.

At a high level, the principle of signed bid requests is that Publishers and Exchanges should sign messages within the supply chain of the real-time bidding transaction. This provides a traceable path to verify critical data of the inventory such as domain, publisher id, and a timestamp of the bid request.

Publishers will benefit from this anti-fraud measure in knowing that their inventory is securely passed for sale.

Advertisers and buyers will benefit from this by reviewing the supply authentication to gain confidence in where the inventory is coming from in the real-time bidding transaction.

Signed bid requests complement the recent ads.txt protocol. Ads.txt (Authorized Digital Sellers) and the data within it should be used to validate authorized sellers their platforms for a source of inventory. Publisher signing of bid requests allow a buyer to validate the bid request and know that it’s trusted from the publisher and key elements of the bid request are unmodified. Together, these technologies are a powerful combination in fighting fraud to allow buyers to check for authenticity and authorization of the sales channel.

After the public comment period, the OpenRTB working group will incorporate feedback into a final version of this proposal, which will be incorporated into the OpenRTB 3.0 specification.

About IAB Technology Laboratory
The IAB Technology Laboratory is an independent, international, nonprofit research and development consortium charged with producing and helping companies implement global industry technical standards. Comprised of digital publishers and ad technology firms, as well as marketers, agencies, and other companies with interests in the interactive marketing arena, the IAB Tech Lab's goal is to reduce friction associated with the digital advertising and marketing supply chain, while contributing to the safe and secure growth of the industry. The organization's governing member companies include AppNexus, Extreme Reach, Google, GroupM, Hearst Magazines Digital Media, Integral Ad Science, LinkedIn, Moat, Pandora, PubMatic, Sonobi, Tremor Video, and Yahoo! JAPAN. Established in 2014, the IAB Tech Lab is headquartered in New York City with an office in San Francisco.

Original Author
Neal Richter, Rakuten Marketing

Major Contributors
Curt Larson, Sharethrough; Sam Tingleff, Rubicon Project

IAB Tech Lab Contact
Jennifer Derke, Director of Product, Programmatic & Data, IAB Tech Lab
openrtb@iabtechlab.com

IAB Tech Lab OpenRTB Working Group Members
https://iabtechlab.com/working-groups/openrtb-working-group/

TABLE OF CONTENTS

1 ABSTRACT 4

2 INTRODUCTION 4
- 2.1 APPROACH 4
- 2.2 LANGUAGE AND TERMINOLOGY 5

3 MESSAGE SIGNING BUSINESS LOGIC 5
- 3.1 RULES OF SIGNATURES 5
- 3.2 SIGNATURE BLOCK 6
  - 3.2.1 SIGNATURE FIELDS 6
- 3.3 MESSAGE SERIALIZATION 7

4 KEYS AND SIGNATURES 7
- 4.1 KEY GENERATION 7
- 4.2 MESSAGE SIGNATURE BASICS 8
- 4.3 PUBLIC KEY DISTRIBUTION 9
- 4.4 PUBLIC KEY FILE 9
- 4.5 PRIVATE KEY FILE 10
- 4.6 CERTIFICATE EXPIRATION 10

5 SIGNATURE AUTHORITIES 10

6 IMPLEMENTER'S NOTES 11
- 6.1 VERSION 11
- 6.2 INTEROPERABILITY 11
- 6.3 SECURITY 11

DRAFTING COMMENTS AND OPEN ITEMS 11

8 ACKNOWLEDGEMENTS 12

9 REFERENCES 12

1 ABSTRACT

Here we are proposing a standard that enables reducing the feasibility of counterfeit and misrepresented inventory in the open digital advertising ecosystem via an optional cryptographic signature that can be used to authenticate the source of the bid request. Buyers are free to validate the signatures as often as they wish in order to reduce their risk of buying inauthentic inventory.

2 INTRODUCTION

For brevity, we'll assume readers are already familiar with the problem of fraud in ad tech and its vast scale [1][2][3]. Fraud can come in various forms; here we are concentrating on the form wherein ad inventory is being offered to buyers with a false label during the real-time bidding process. Typically, the domain of the webpage (or the bundleID of the mobile app) has been falsified to look like a site that is more valuable than the actual impression available. A variety of other fields are subject to falsification such as the IP address (to make the inventory appear to be in a more desirable geography), the TID (to make it appear to be a unique impression), the device ID (to correspond to IDs known to have dense/high bids in the past), etc.

2.1 APPROACH

The approach taken here is inspired by the "Domain Keys Identified Mail" or DKIM standard [17]. That standard defines a mechanism where the sending domain of an email system can cryptographically sign an outbound email message with a private key. Final recipient and intermediary servers can use the domain's public key to then validate the message. This allows several features and effects:

- No attempt is made to include encryption of the message as part of the mechanism, instead a digital signature of a cleartext message is used;
- The message can be proven (within the bounds of the signature's security) to be from the declared sending domain by the recipient or intermediary servers;
- Any alterations of the signed elements can be detected by the recipient or intermediary servers;
- There is no dependency on public and private key pairs being issued by well-known, trusted certificate authorities;
- There is no dependency on the deployment of any new protocols or services for public key distribution or revocation;

2.2 LANGUAGE AND TERMINOLOGY

Requirements Notation:

The keywords "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in [RFC2119].

"Authentic Inventory" denotes that the inventory as declared in the OpenRTB or other RTB protocol's bid request can be validated as coming from that source.

A "Request Originator" is the root originating system of the ad inventory.

An "Intermediary" is a system that is involved as a third party between a seller end-point and one or more buyer end-points during the bidding process of the transaction.

A "Signer" is the agent system that signs a message.

A "Request Buyer" is any system that may make an offer in response to the bid request and may optionally authenticate it.

3 MESSAGE SIGNING BUSINESS LOGIC

This section discusses the message creation and signing.

This spec proposes that the signature is based upon a subset of data points in the bid request that enable the buyer to authenticate the request.

3.1 RULES OF SIGNATURES

There are a few rules for using digital signatures for this purpose. Below we use the term message to mean an OpenRTB bid request.

1) The originating publisher of a message must be the entity that signs the message or delegates this authority to a singular primary system.
2) The message needs to have something unique (a random message identifier) to the message to prevent reuse of the message and signature by a downstream system (TID is used for this purpose).
3) Any downstream consumer of the message that wants to authenticate it needs access to the full message, the signature and the public key of the source of the message.
4) No downstream system can make changes in the signed fields

3.2 SIGNATURE BLOCK

We are beginning with a very simple solution for 3.0 to address the key problem in a simple way. We will allow a single signature done at the publisher or publisher ad server/ssp level. A small set of key fields will be signed. While there are certain use cases for additional or more comprehensive signatures, we believe this balances the security versus complexity concerns in an appropriate way for an initial release. The design also allows for the addition of new signatures and types of signatures in the future (for example, you could add an array of intermediary signatures).

Specifically, we will add a single new field called ps, for publisher signature, to the Ad Specification object in the AdCOM spec on the request side to contain the signature string that results from signing the fields listed below. Each field value should be concatenated, with a semicolon delimiter, to produce the signature. An example is shown below.

If a value is not available, null, zero length, or inapplicable for any of these fields an empty string is used.

3.2.1 SIGNATURE FIELDS

Spec	**Object**	**Field**	**Example Value**
Transaction	Source	tid	ABC7E92FGD6A
AdCOM	Publisher	ID	12345678
AdCOM	Site	domain	newsite.com
AdCOM	App	bundle[1]	
AdCOM	Device	ip	192.168.1.1
AdCOM	Device	ipv6	
AdCOM	Device	ifa	
To-be-added creative format information			
To-be-added universal id			

[1] Note: presently in-app signatures are not fully supported due to the lack of a reliable method to retrieve the public key since there is no known domain. The issue is similar for ads.txt, and a solution is under discussion in that context. App bundle and ifa are included in this spec with the goal that mobile apps can be supported by version 1 signatures.

3.3 MESSAGE SERIALIZATION

Each message to be signed must be canonically serialized. Each message is a non-empty set of elements, separated by a delimiter. Valid delimiters are either the semicolon ';' or the ampersand '&'.

<FIELD1> <SEPARATOR> <FIELD2> <SEPARATOR> . . . <FIELDN>

For the sample values in the table, the input string into the signature would be as follows:
ABC7E92FGD6A;12345678;newsite.com;;192.168.1.1;;
That string would be used to create the signature string which would be inserted into ps (in the case of generating the request) or checked against the value that is received in ps (in the case of validating the request).

4 KEYS AND SIGNATURES

This spec proposes a new mechanism within OpenRTB to enable an optional digital signature on bid requests. Buyers can authenticate the requests in real-time or offline.

Every publisher (or publisher's technology provider) that will cause ad requests to go out to exchanges should create and sign a simple message that will be re-transmitted with the bid request by the Exchange/SSP.

We propose that the ECDSA Algorithm [1] be used with public and private keys. The ECDSA creates a shorter signature overhead compared to original DSA Algorithm [2].

4.1 KEY GENERATION

OpenSSL's command line security software can be used to generate the ECDSA public and private keys in a standard format known to compatible security packages. An example key generation process is below. Here we have used the "secp112r2" elliptic curve algorithm.

```
jupiter: bob$ openssl ecparam -name secp112r2 -genkey -out secret_key.pem
jupiter: bob$ cat secret_key.pem
-----BEGIN EC PARAMETERS-----
BgUrgQQABw==
-----END EC PARAMETERS-----
-----BEGIN EC PRIVATE KEY-----
MD4CAQEEDjF4pXve+u2AKI9pH/OwoAcGBSuBBAAHoSADHgAEWQvY9hOZoLmetUIA
NEuG3O9h+exmTzUW2Z+6OA==
-----END EC PRIVATE KEY-----
jupiter: bob$ openssl ec -in secret_key.pem -pubout -out public_key.pem
read EC key
```

```
writing EC key
jupiter: bob$ cat public_key.pem
-----BEGIN PUBLIC KEY-----
MDIwEAYHKoZIzj0CAQYFK4EEAAcDHgAEWQvY9hOZoLmetUIANEuG3O9h+exmTzUW
2Z+6OA==
-----END PUBLIC KEY-----
```

4.2 MESSAGE SIGNATURE BASICS

Signing a message is straightforward with many OpenSSL compatible tools.

As an example, we show the process and result of signing an entire OpenRTB bid request JSON example at [15] that is 1174 bytes in size and a second example [16] that is 2815 bytes in size. Note that the private/secret key is used to sign the request.

Each OpenRTB example resulted in a signature of 49 base64 encoded bytes in size. For the "secp112r2" EC algorithm this is a fixed overhead of 49 base64 encoded bytes on the OpenRTB request, which is typically more than 1,000 bytes in practice.

```
jupiter: bob$ openssl dgst -ecdsa-with-SHA1 -sign secret_key.pem -out openrtb_request.sig
openrtb_request.json
jupiter: bob$ openssl dgst -ecdsa-with-SHA1 -sign secret_key.pem -out openrtb_request2.sig
openrtb_request2.json
jupiter: bob$ base64 openrtb_request.sig > openrtb_request.sigb64
jupiter: bob$ base64 openrtb_request2.sig > openrtb_request2.sigb64
jupiter: bob$ cat openrtb_request.sigb64
MCACDgocFT8gYoqedLvzdrB0Ag4G2TyLPNSSvECsFz4cAA==
jupiter: bob$ cat openrtb_request2.sigb64
MCACDgvmFAHxNY1fGbU3d51vAg4joHZLOrV8GL/DLAoKBw==
jupiter: bob$ wc -c openrtb_request*
    1174 openrtb_request.json
      34 openrtb_request.sig
      49 openrtb_request.sigb64
    2815 openrtb_request2.json
      34 openrtb_request2.sig
      49 openrtb_request2.sigb64
```

Validating a signature is also straightforward with OpenSSL compatible tools. Note that the Public key is used for validating the signature against the message.

```
jupiter: bob$ openssl dgst -ecdsa-with-SHA1 -verify public_key.pem -signature
openrtb_request.sig openrtb_request.json
Verified OK
jupiter: bob$ openssl dgst -ecdsa-with-SHA1 -verify public_key.pem -signature
```

```
openrtb_request2.sig openrtb_request2.json
Verified OK
```

4.3 PUBLIC KEY DISTRIBUTION

The Public Key file must be accessible via HTTP and/or HTTPS from the website that the instructions are to be applied to under a standard relative path on the server host: "/ads.cert" and and HTTP request header containing "Content-Type: text/plain".

For the purposes of this document the “root domain” is defined as the “public suffix” plus one string in the name. Crawlers should incorporate Public Suffix list [16] to derive the root domain.

For convenience we will refer to this resource as the "/ads.cert file", though the resource need in fact not originate from a file-system.

If the server response indicates an HTTP/HTTPS redirect (301, 302, 307 status codes), the advertising system should follow the redirect and consume the data as authoritative for the source of the redirect, if and only if the redirect is within scope of the original root domain as defined above. Multiple redirects are valid as long as each redirect location remains within the original root domain. For example, an HTTP to HTTPS redirect within the same root domain is valid. Any redirect off the website should be ignored.

If the server response indicates Success (HTTP 2xx Status Code,) the advertising system must read the content, parse it, and use that key to validate the digital signatures. Validation of the content returned should be done and only valid PEM format certificates should be stored and used.

If the server response indicates the resource is absent (HTTP 404) or forbidden (HTTP 403) or any other HTTP error the advertising system should interpret the response that no public key is available for use.

It is advised to use the HTTP header "Content-Type: text/plain; charset=utf-8" to signal UTF8 support.

4.4 PUBLIC KEY FILE

The public key is encoded as a formatted plain text object, described here. Note that the public key must be secured yet accessible to be read by outside systems to validate the messages against the signatures.

TBD research on the typical format of an SSL certificate file.

TBD on how to provide a history of prior keys when they change.

4.5 PRIVATE KEY FILE

The private key is encoded as a formatted plain text object, described here. Note that the private key must be secure and inaccessible to outside systems yet accessible to the systems that generate and sign messages.

TBD research on the typical format of an SSL certificate file.

4.6 CERTIFICATE EXPIRATION

Consuming systems of /ads.cert should cache the files, but if they do they must periodically verify the cached copy is fresh before using its contents.

Standard HTTP cache-control mechanisms can be used by both origin server and robots to influence the caching of the /ads.cert file. Specifically, consumers and replicators should take note of HTTP Expires header set by the origin server.

If no cache-control directives are present consuming systems should default to an expiration date of 1 year.

5 SIGNATURE AUTHORITIES

There are two model methods of signing messages. The first is that the publisher uses a self-hosted software system to create the message and signature, then providing both to the approved advertising systems listed in the ads.txt file.

The second model is a primary delegate system. In this model, the publisher designates a single primary advertising system to create and sign the messages. This system then provides the message and signature to all advertising systems listed in the ads.txt file.

In both cases the buyer system is able to validate the signatures via ads.cert as well as validate the IDs and domains against the ads.txt.

6 IMPLEMENTER'S NOTES

6.1 VERSION

This is a pre-version 1.0 draft specification for public review. Once the draft specification achieves version 1.0 status, every attempt will be made to make future versions backward compatible if possible.

6.2 INTEROPERABILITY

Implementors should pay particular attention to the robustness in parsing of the /ads.cert file. It is expected that the /ads.cert files are created with automated systems such as OpenSSL.

6.3 SECURITY

The /ads.cert declarations are retrieved and applied in separate, possibly unauthenticated HTTP transactions, and it is possible that one server can impersonate another or otherwise intercept a request for /ads.cert, and provide a consuming system with false information.

If this is a worry then the website owner should redirect unsecure http requests to https requests for the /ads.cert file.

7 DRAFTING COMMENTS

Status of this document:
This proposal document has a few open items, noted throughout each section as TBD, to be decided. With further public review and industry input, the working group will finalize specification details and then integrate this proposal into OpenRTB 3.0.
"OpenRTB 3.0 Authentication: Signed Bid Requests" is part of the OpenRTB 3.0 Framework release.

Comments and feedback should be submitted using this OpenRTB 3.0 Framework Public Comment Submission Form. Public comments are open until December 15, 2017.

Other Notes:
Jan Winkler requests that we audit the proposed ECDSA algorithm for implementations in common languages. Popular languages tend to have useable OpenSSL bindings.

Allen Dove requests that we insert language about failed signings needing retry with latest certificate as they may in fact be valid.

8 ACKNOWLEDGEMENTS

The authors would like to thank the original authors of the DKIM Standard [17] as well as elements of Blockchain [18] for providing inspiration. We would also like to thank numerous people within the IAB, TAG, and multiple companies for their comments on the early drafts.

9 REFERENCES

1. https://en.wikipedia.org/wiki/Elliptic_Curve_Digital_Signature_Algorithm
2. https://en.wikipedia.org/wiki/Digital_Signature_Algorithm
3. https://techcrunch.com/2016/01/06/the-8-2-billion-adtech-fraud-problem-that-everyone-is-ignoring/
4. http://adage.com/article/digital/ana-report-7-2-billion-lost-ad-fraud-2015/302201/
5. http://boingboing.net/2016/12/21/methbot-a-3m-5mday-video-a.html
6. https://www.emarketer.com/Article/Ad-Industrys-Focus-on-Fraud-Has-Intensified/1014430
7. https://en.wikipedia.org/wiki/Digital_Signature_Algorithm
8. https://en.wikipedia.org/wiki/Elliptic_Curve_Digital_Signature_Algorithm
9. http://www.robotstxt.org/norobots-rfc.txt
10. https://en.wikipedia.org/wiki/Robots_exclusion_standard
11. https://www.iab.com/guidelines/real-time-bidding-rtb-project/
12. https://developers.google.com/ad-exchange/rtb/downloads
13. https://www.iab.com/guidelines/iab-opendirect-specification/
14. https://tools.ietf.org/html/rfc1123
15. https://github.com/openrtb/examples/blob/master/brandscreen/example-request-pc-single.json
16. https://github.com/openrtb/examples/blob/master/spotxchange/example-video-request-single_impr.md
17. http://ietf.org/rfc/rfc4871.txt
18. https://en.wikipedia.org/wiki/Blockchain