ads.cert Authenticated Connections Protocol Specification

January 2022

注意：

本ドキュメントは、IAB Tech Lab が公開しているドキュメントを簡易的に翻訳したものです。正確な情報は、下記オリジナルドキュメントを参照してください。

https://iabtechlab.com/wp-content/uploads/2021/09/3-ads-cert-authenticated-connections-pc.pdf

目次

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(IAB Tech Lab について)

IAB Technology Laboratory(Tech Lab)は、効果的で持続可能なグローバルデジタルメディアエコシステムの成長を促進するための標準、ソフトウェア、サービスを制作・提供する非営利の研究開発コンソーシアムです。デジタルパブリッシャーやアドテクノロジー企業、マーケター、広告代理店、その他インタラクティブマーケティング分野に関心のある企業で構成される IAB Tech Lab は、透明で安全かつ効果的なサプライチェーン、よりシンプルで一貫性のある測定、消費者にとってより良い広告体験を通じて、ブランドとメディアの成長を可能にすることを目指し、モバイルと TV/デジタルビデオチャネルの実現に焦点を当てています。IAB Tech Lab のポートフォリオには、消費者、パブリッシャー、広告主、サードパーティプラットフォームのデジタル体験を改善するために設計された DigiTrust リアルタイム標準化 ID サービスが含まれています。ボードメンバーには、AppNexus、ExtremeReach、Google、GroupM、Hearst Digital Media、Integral Ad Science、Index Exchange、LinkedIn、MediaMath、Microsoft、

Moat、Pandora、PubMatic、Quantcast、Telaria、The Trade Desk、ヤフージャパンが含まれます。2014 年に設立された IAB Tech Lab は、ニューヨークに本部を置き、サンフランシスコにオフィス、シアトルとロンドンに支部を置いています。

IAB Tech Lab の詳細はこちら: www.iabtechlab.com

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

これらの文書はすべて https://iabtechlab.com/ads-cert から入手できます。

Getting Started(はじめに)

ads.cert または Authenticated Connections プロトコルを初めて使用する場合は、ads.cert 入門からはじめることをお勧めします。これは、プログラマティック広告エコシステム参加者の大多数に推奨されるパスです。

Objective(目的)

本ドキュメントでは、ads.cert Authenticated Connections プロトコルで使用されるメッセージ形式について説明します。このドキュメントの目的は、IAB Tech Lab がホストする ads.cert オープンソースソフトウェアが実装するプロトコルの参考資料を提供することです。一般的に、コミュニティが管理するオープンソースアプリケーションを組織内で採用することを強く推奨します。このことを念頭に置いて、この仕様書とオープンソースコードを参考としてプロトコルを再実装することは技術的に可能です。そうすることを選択した場合、この文書は必要な詳細を提供します。

Background(背景)

公開されている eMarketer の推定によると、2021 年の世界のプログラマティック広告費は約 4,550 億

ドルと予測されています。私たちの業界では、個々の取引規模は非常に小さい傾向がありますが、総額は膨大です。この膨大な経済活動は、悪質業者が合法的なパブリッシャーから広告費を吸い上げる余地を生み出し、彼らの活動を検出することを難しくする(シグナルがノイズに紛れてしまう)。この経済活動のほんの一部でも自分たちの目的のために利用できれば、金銭的な報酬は非常に大きくなります。

IAB Tech Lab の Cryptographic Security Foundations Working Group の見解では、この経済活動を悪意ある行為者から保護するための既存のツールは、特にクライアントデバイスではなくサーバーから発信される取引や活動の場合には不十分です。長期的には、オンライン広告トランザクションを不正行為から保護するためには、多層的なアプローチが必要です。この多層的な戦略に必要なコンポーネントの 1 つは、サーバー間の通信を保護する能力です。
-- これが ads.cert Authenticated Connections の目的です。

例えば、ストリーミングビデオやオーディオコンテンツへの広告の挿入(Server-Side Ad Insertion)、burl を介したモバイルアプリのアクティビティに対する Server-Side の課金通知、ネイティブクリエイティブのマークアップを取得してウェブページに表示できる HTML に変換する、などです。将来的には、広告関連の作業が Client-Side で行われるのではなく、サーバーにオフロードされるため、Server-Side アクティビティの合法的なユースケースの数が増えることが予想されます。とはいえ、このプロトコルの主な動機となるユースケースの 1 つは、SSAI プラットフォームから発信されるビデオトランザクションを保護することです。

最近のセキュリティ研究では、当事者が SSAI プラットフォームになりすまそうとするスキームが浮き彫りになっています。このようなスキームは、実際の SSAI ビジネスにサービスを提供しているのと同じクラウドプラットフォームやホスティングプロバイダからトラフィックが発信されているように見えるため、特定が困難です。作業部会は、この特定の問題に対処することの重要性を認識しており、これが、他のユースケースよりも ads.cert 認証接続を優先する原動力となりました。特に、課金通知において ads.cert 認証コネクションを使用し、通知受信者が課金通知を送信すると主張する SSAI プラットフォームを識別できるようにすることを推奨します。その後、受信者は、SSAI プラットフォームを信頼するかどうかを決定できます。これにより、詐欺師がクラウド上でサーバーを立ち上げ、有効な SSAI プラットフォームであると主張し、不正な広告インプレッションを生成することを防げます。

Public key and shared secret generation(公開鍵と共有秘密の生成)

ads.cert プロトコルは、RFC 7748 の X25519 アルゴリズムを使用して、秘密鍵から公開鍵を生成しま

す。さらに、X25519 アルゴリズムは、あるパーティの秘密鍵と別のパーティの公開鍵を組み合わせて共有秘密鍵を生成します。同じ関係にある対応する鍵(相手の秘密鍵、こちらの公開鍵)を使って逆計算を行うと、同じ結果になります。

RFC より:

> X25519 関数は、楕円曲線ディフィー・ヘルマン(ECDH)プロトコルで以下のように使用できます:
>
> Alice は a[0]から a[31]までの 32 個のランダムバイトを生成し、K_A = X25519(a, 9)を Bob に送信します。
>
> Bob も同様に b[0]から b[31]の 32 個のランダムバイトを生成し、K_B = X25519(b, 9)を計算し、Alice に送信します。
>
> 生成された値と受信した入力を使って、Alice は X25519(a, K_B)を計算し、Bob は X25519(b, K_A)を計算します。
>
> これで両者は K = X25519(a, X25519(b, 9)) = X25519(b, X25519(a, 9))を共有秘密鍵として共有します。両者は、K の値に関する余分な情報を漏らすことなく、K がオールゼロの値であるかどうかをチェックし、そうであれば中止してもよい(MAY)。その後、Alice と Bob は K、K_A、K_B を含む鍵導出関数を使用して共通鍵を導出することができる。

X25519 の実装は、さまざまなソフトウェア言語で広く利用可能です。秘密鍵の生成には、安全な擬似乱数生成器を使用しなければなりません。これを怠ると、あなたの秘密鍵が攻撃者に漏洩し、攻撃者があなたのビジネスになりすますことを許してしまう可能性があります。

取引相手の DNS レコードから得た公開鍵を用いて、以下に説明するように共有秘密を計算します。

Message format specifications(メッセージフォーマット仕様)

Public key DNS record(公開鍵 DNS レコード)

Record name(レコード名)

公開鍵は、この目的のために設立された ads.cert Call Sign インターネットドメインにおいて、実施企業（example.com など）が登録する”Public Suffix＋1”(PS+1)ドメイン名(publicsuffix.org が公表する)の DNS レコード名” _delivery._adscert”内で公開されるものとします。この目的で有効なのは、ICANNが割り当てたサフィックスのみです。publicsuffix.org ファイルの”private”セクションの使用はサポートされていません。

Record format(レコードフォーマット)

レコードの値は以下のようになります:

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

DNS レコードは以下のフィールドで構成されます:

Field	Description
v	このレコードがads.cert配信キーを提供することを示すために、定数値”adcrtd”を設定します。このトークンは、DNSレコード値の先頭に表示されなければならない(MUST)。
k	将来的に別のスキームに移行する必要がある場合の互換性のために設計されています。現在のところ、これは常にX25519 Diffie-Hellman鍵交換アルゴリズムを表す"x25519 "に設定されます。
h	ハッシュアルゴリズム識別子を設定します。現在、これは常に"sha256 "に設定され、SHA-256セキュアハッシュアルゴリズムを表します。
p	値は32バイトの公開鍵を43バイトのbase64エンコード文字列で表現したもので、RFC 4648の”URL-safe”バリアントです。

フィールドはスペース 1 文字で区切られます。キーと値のペアは等号で区切られます。

すべてのキーと値は大文字と小文字を区別します。

Authority delegation record format(権限委譲レコードフォーマット)

Record name(レコード名)

権限委譲レコードは、DNS レコード名”_adscert”および上記と同じドメイン Public Suffix 規則内で公開されるものとします。

Record format(レコードフォーマット)

レコードの値は以下のようになります:

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

DNS レコードは以下のフィールドで構成されます:

Field	Description
v	このレコードがads.cert権限委譲レコードを提供することを示すために、定数値“adpf”を設定します。このトークンは、DNSレコード値の先頭に表示されなければなりません (MUST)。
a	このDNSレコードをパブリッシャーする運用ドメインに関連付けられた 権限ドメイン。このドメインは、前のセクションで説明した公開鍵のパブリッシャーに使用される ドメインと一致しなければなりません (“_delivery._adscert”サブドメイン部分を除く)。ドメインはASCII文字セットで提供され、解釈されなければなりません (MUST)。拡張文字セットのドメインは、punycode形式で提供できます。

デリミターは上記と同じ。
すべてのキーと値は大文字と小文字を区別します。ドメインは小文字で指定しなければなりません (MUST)。

Signature HTTP request header format(署名 HTTP リクエスト・ヘッダ形式)

HTTP リクエストには、1 つ以上の ads.cert Authenticated Connections 署名メッセージが含まれる場

合があります。各署名メッセージは、HTTP リクエストヘッダ内に表示されます:

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

ほとんどの実装では、署名メッセージを 1 つだけ送信する必要があります。現在のところ、2 つ目の署名メッセージの動作は定義されていません。

Signature message format(署名メッセージのフォーマット)

署名メッセージは、この順序で 3 つの部分から構成されます:

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

署名メッセージは次の例のようになります:

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

セミコロンとスペースはメッセージと署名を分離し、署名の計算では使用されません。どちらの値もアンパサンドで区切られた key=value のペアで構成される RFC 3986 クエリー文字列形式でエンコードされ、RFC 仕様に従って適切にエスケープされます。キーはメッセージ文字列の中でどのような順序で現れてもかまいません。

メッセージと署名のコンポーネント内で定義されたフィールド:

Field	Description
from	リクエストを送信する当事者のads.certコールサインドメイン。検証のためのHTTPリクエストを受信した当事者は、このドメインを使用して、対応する _delivery._adscert DNS TXTレコードを取得します。

from_key	送信側の公開鍵の最初の6文字。
invoking	呼び出されるURLホスト名のドメイン。
nonce	URLセーフなbase64エンコード形式で送信側からランダムに生成された番号。Nonceの長さは12文字。
timestamp	署名を生成した時刻: YYMMDDTHHMMSS。時間はUTCタイムゾーンで表されます。
to	署名を受け取る側のads.cert Call Signドメイン。
to_key	受信側の公開鍵の最初の6文字。
status	ここで定義されている”OK”(1)以外の署名プロトコルのステータス値を表す整数値。例えば、このフィールドは、署名者が特定の理由で署名を抑制していることを示すために使用されます。また、相手側の鍵を取得するDNSの問題を伝えることもできます。
sigb	リクエストのメッセージと本文の署名(最低12文字)。
sigu	リクエストのメッセージ、本文、URLに対する署名(最低12文字)。

署名者は、すべてのリクエストに署名メッセージを送信して、署名で検証できない情報であっても、自分の身元と意図を宣言すべきです（SHOULD）。この情報は、非参加の受信者が、認証を受けられる取引相手の範囲と機会を 理解するのに役立ちます。これらの署名なしメッセージには、最低限”from”および”status”フィールドを入力する必要がありますが、送信者の裁量で他の値を含めることもできます。

Computing signatures for signing and verification (署名と検証のための署名計算)

Calculating signatures(シグネチャーの計算)

リクエストボディとリクエスト URL に対してそれぞれ SHA256 ハッシュを計算します。

ボディ署名(sigb)は、メッセージの HMAC とボディの SHA256 ハッシュを結合したものです:

$$bodySignatureBytes = HMACSHA256(sharedSecret, message \,||\, bodyHash)$$

URL 署名（sigu）はまた、URL の SHA256 ハッシュをメッセージに連結します。

$$urlSignatureBytes = HMACSHA256(sharedSecret, message \,||\, bodyHash \,||\, urlHash)$$

前の HMAC 構造に付加することで、アルゴリズムは前のハッシュ計算を再利用できます:

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

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

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

結果の HMAC 値を URL セーフな base64 文字列としてエンコードし、少なくとも最初の 12 文字を取得するように切り捨てます。これは(6 ビット*12 文字=)72 ビットのセキュリティを提供し、 HMAC の切り捨てに関する NIST Special Publication 800-107 のガイドラインに準拠します。これらは sigb と sigu フィールドで提供される署名となる。12 文字より短い署名は無効な入力として扱われなければならない(MUST)。

Note:より長い署名を使用することによるセキュリティの向上との前方互換性を認めるために、ベリファイアは、256 ビットのハッシュスキームで使用可能な最大 43 文字までの 署名を受け入れ、検証できなければならない(MUST)。ベリファイアは、デフォルトの 12 文字より長い署名を要求してもよい[MAY]。署名者がより長い署名値を提出することが期待される状況を定義することは、 本仕様の範囲外です。

Calculating verifications(検証の計算)

スキーム(https)、ホスト名、パス、クエリー文字列など、クライアントがサーバーを起動する際に使用した URL を再構築します。パスとクエリー文字列は HTTP の GET もしくは POST スタンザに記述され、ホスト名は Host ヘッダから取得する必要があります。あるいは、別の仕組み(例えばカスタム HTTP リクエストヘッダ)を使って、サーバーに渡される TLS サーバー名識別(SNI)フィールドからホスト名を取得する必要があるかもしれません。

上記と同じアルゴリズムを使用して、メッセージ、ボディハッシュ、URL ハッシュの 署名を再計算します。リモートクライアントからの署名メッセージで提供された値と比較する。

Implementation Recommendations(実施に関する推奨事項)

このプロトコルのオープンソース実装を使用する手順は、この仕様書に付属するオープンソースソフトウェア実装者ガイドを参照してください。