Scammers and Spammers and Bots, Oh My

Utilizing Authenticity, Integrity, and Non-Repudiation to Address the Inefficiencies of Ad Tech

MadHive ● May, 2019

1.0 Introduction

1.1 Abstract

The goal of this document is to describe a generic system that may be utilized to add strong guarantees to the advertising supply chain. The authors of this paper realize that not all systems will be implemented at once. Due to this concern, the system has been broken down into discrete components that address a specific concern. In isolation, some of these systems are still capable of solving difficult problems facing the Ad-Tech industry today. In combination, these systems provide a robust, cryptographically secure mechanism to trace assets across the advertising supply chain. In practice, an implementation of this specification would enable cryptographic validation of information across the supply chain, providing immutable proof between buyer and seller.

1.2 Organization of Information

This paper has been organized into three primary sections. Section one (Introduction) describes the intent of this paper and defines the organization of the paper as a whole. Section two (Data Distribution) describes the construction of a general purpose datastore with strong cryptographic guarantees and write level access control based on blockchain technology. Section three (Data Integrity) describes mechanisms to introduce strongly authenticity, non-repudiation, and integrity claims to the advertising supply chain. The remaining two sections are the Conclusion and Works Cited.

2.0 Data Distribution

2.1 Secure Data Distribution

In order to secure the advertising supply chain, a generalized mechanism of authenticated data exchange is needed, even if the distributing party does not itself host a publicly accessible domain from which this data may be served. In order to solve this issue, a blockchain-based record store has been constructed.

This record store has been built as a smart contract in a Permissioned Ethereum-based blockchain implementation. The Ethereum blockchain has been selected for the purpose of building a proof of concept system. This selection was based on the ability to rapidly iterate designs utilizing the semi Turing Complete programming language Solidity. Once a final specification has been built, it may be advantageous to implement a less general purpose mechanism, but until such time as an industry agreement may be reached, the adaptable nature of Ethereum like chains is advantageous.

The exact motivation for this system is to allow for secure self management of cryptographic identity and verifiable information in a public repository, while also allowing all information to be stored under a time to live (TTL). The inclusion of a TTL is proposed to allow for the programmatic expiration of records.

It is of significant importance that this system allows for real-time revocation and re-issuance of cryptographic identities by the owner of those identities, without the need for a blacklist. This is accomplished through the distributed state machine of the blockchain. In the event that a record exists in

the current state of the blockchain data store, and that record is not expired, the record should be taken as valid. In the event that a record is not present, at the expected location in the blockchain datastore, this should be taken as an invalid reference. Such an invalid reference may indicate a forgery or a revocation, depending on the circumstance. In the event that a record is expired, the return data for a request for this record should return the same data as a missing record. These requirements have been achieved in the current implementation.

In order to register a cryptographic identity in this system, the owner of a record store may write, at a designated record name, the public key of an asymmetric keypair. In effect this allows the records store to act as a Public Key Infrastructure (PKI). Due to reasons that will be covered later in this document (see section 3.4, Signature Generation Algorithm), it is recommended that these cryptographic identities be generated using the Libsodium library, against the Ed25519 elliptic curve.

In order for a PKI to be trusted, there must be a root of trust in the system. The proposed root of trust for this system is AdLedger, the non-profit consortium charged with implementing global technical standards and solutions for the digital media and blockchain industries.

The operation of this system is as follows. When an entity requests to be included in the record store system, a certifying entity must validate that the applying entity is who they claim to be. For the sake of clarity, let's say that the entity that wishes to be certified is a fictitious publisher called The New York Gazette or NYG, and the certifying entity is AdLedger. Once validated, NYG securely generates a private key. The public key associated with this private key is shared with AdLedger. This is NYG's new blockchain-based account.

In response to receipt of this public key, AdLedger creates a publicly readable datastore that may only be modified by NYG by utilizing the private key associated with their blockchain account. This datastore is a dedicated smart contract for NYG that enforces access control policies on requests for changes of state.

The datastore smart contract may be resolved by the hash (Sha-256) of the name of the entity, in this case NYG, through a registry contract that is controlled by AdLedger. The resolution query, against the registry smart contract, will return the address of the dedicated record store contract that is controlled by NYG, to the requesting party. Once this address is obtained, queries for records that are pertinent to NYG may be obtained through requesting the record of concern by name from the record store contract located at the resolved contract address. In order to resolve a specific record in the record store, the name of the record may be hashed (Sha-256) and passed to a lookup function. This lookup function will resolve the sub-record for the entity of concern. This construction of the registry contract with a per-entity dedicated record store that contains hash based value resolution is analogous to a 2D hashmap.

Once registered, The New York Gazette may then utilize it's blockchain based account to write data into the record store smart contract for public distribution. This record store may only be modified by the owning entity, and all modifications are cryptographically signed. Thus, it is provable that only the owning party did modify the records in this record store.

This record store will also enforce TTL on records such that a seller may issue a buyer a right to sell for some interval of time, where the record will be treated as invalid after the date of expiration.

2.2 Realtime Authenticity and Integrity

In order to prove authenticity, integrity, and non-repudiation in the advertising supply chain, message data may be cryptographically signed. The signature appended to these messages should be derived from a key that is registered in the secure datastore that has been previously specified. These signatures will provide non-repudiation, integrity, and authenticity guarantees for those elements of a message that are signed. These signatures may be validated by ensuring that the signature is valid, and

that the claimed origin of the information does in fact have a record, in the blockchain datastore that claims possession of the associated public key.

An entity is able to issue a new key simply by transacting with the smart contract system. This action can be done at any time by the owner of the account. An entity is able to revoke a cryptographic key through either expiration, or through active deletion and/or modification of the associated record.

2.3 General Purpose Entity Claims

In addition to the ability to register and revoke cryptographic identities in real time, blockchains also enable the distribution of arbitrary cryptographically verifiable information in real time. This may take the form of authorized sellers, authorized buyers, or any other information that need be distributed between parties in the supply chain.

3.0 Data Integrity

3.1 Verifiable Supply Chain

Once a mechanism for the distribution of cryptographic identity with respect to asymmetric keypairs has been established, data transmitted between parties in the advertising supply chain may be cryptographically verified. This verification may be performed by validating a cryptographic signature appended to the OpenRTB messages that are being exchanged. In order to be backwards compatible with existing protocols, it is proposed these cryptographic signatures be appended through the utilization of the existing *ext* fields present in the OpenRTB specification.

3.2 Signature Protocol Primitives

Due to the need to compactly represent a cryptographic claim in both strongly typed structures, such as protobuf, and generic structures, such as JSON, it is the opinion of the authors of this paper that signatures should be compactly encodable as UTF-8 strings.

Rather than specify a new encoding mechanism for these objects, it is proposed that an existing standard be utilized. The motivation for this decision is twofold. First, the selection of an existing protocol will minimize the amount of code that must be written to facilitate adoption. Second, the selection of an existing protocol will minimize the introduction of security flaws due to the public review that has already been conducted on the previously publicly adopted protocol.

Specifically, it is proposed that the selected protocol be constructed from the Javascript Object Signing and Encryption Documents (JOSE) [1]. These standards have been written for the specific purpose of securing machine to machine and human to machine interactions in a compact, url safe manner. Further, these standards have been written to facilitate asymmetric and symmetric cryptographic operations that include cryptographic signatures. Finally, these specifications have been designed specifically to work with JSON objects, which means they lend themselves well to OpenRTB messages with minimal modification.

These standards have seen wide adoption across many industries today. For example, the Oath2.0 specification directly cites the proposed standards for the construction of bearer tokens [2].

3.3 Signature Object Serialization

In order to create url safe, compact, encodings of signature objects, the JWS Compact Serialization format is hereby proposed. This format allows for the compact encoding of nested JSON data structures as a dot separated, base64 encoded, url safe, UTF-8 string.

3.4 Signature Generation Algorithm

At this time, Ed25519 has not seen widespread adoption as a signing algorithm for JWS. As a result, some modifications may be needed in the utilization of open source dependencies. The selection of this algorithm is for two specific reasons. First, this algorithm is dramatically more performant than other asymmetric signing algorithms. Second, if the

LibSodium implementation is utilized, the security of this protocol is far superior to other systems due to recent Zero-Day exploits discovered in microprocessor hardware [3]. The security benefits are derived from the non-conditional execution path during signature generation in the LibSodium libraries. This prevents these libraries from being susceptible to the probabilistic branch execution exploits found in recent months [4]. For additional information on these attack vectors, the reader should look to the aforementioned reference to the exploits and the reference for the LibSodium libraries themselves.

3.5 Proposed Signature Object

The authors of this paper realize that the ability of a single entity to fully anticipate all desired characteristics of a protocol is an unrealistic expectation. Due to this pragmatic viewpoint, the signature objects will be generally described in such a manner that the industry may adopt conventions that best utilize these primitives for a specific interaction. In order to allow such generalization, it is proposed that the signature object should not contain any information directly. Rather, the signature object should contain a query path, or collection of query paths that describes a collection of objects from the associated OpenRTB object. These query path objects may take several forms, based on existing industry standards. Examples of these standards may be found in jmesPath, JSONPath and protobuf3 [5][6][7]. The general form of these query path descriptors is dot separated notation for key value lookup paths in a nested data structure. More advanced systems allow for glob style matching across subkeys, but this functionality should be used with caution due to the potentially inadvertent inclusion of unnecessary data. The operation of a signature construction and verification is as follows.

A signature may be generated by encoding a collection of paths that describe a projection of an OpenRTB object and the public key used to sign the object into the signature object itself. This projection describes all data that has been signed. In the projection map, the values from the openRTB object will be stored at the path that describes how to access the projected element. Once this projection map has been constructed, the existing signature and serialization specifications of JOSE may be utilized for robust signature generation.

The token that is built contains three major parts: (1) the header, (2) the claims, and (3) the signature. The content of these sections if described as follows:

1. The header specifies the algorithm that is used for signing, and the encoding that is used for the object.
2. The claims specify the following information:
    a. The issuer of the token.
    b. The time at which the token was created.
    c. An expiration that after which, the token should be treated as invalid.
    d. The path that describes a projection of the object that should be signed.
    e. The hash of the projection described by the path.
    f. The public key that will verify this token’s signature.
    g. A reference to how this key may be verified as belonging to the issuer of the token.
    h. A randomly generated identifier for the token that will later allow for uniqueness and fraudulent message checking.

The motivation for this specification is simple. By encoding only the information that describes how to access all elements that have been signed in a deterministic manner, any subset of information may be signed in such a way that any other entity may validate the signature trivially. Second, by including the public key in the object, verification may be performed in parallel with the lookup of the key to ensure it is an authorized cryptographic identity for the claimed signing entity. Lastly, by layering signatures more complex linking of records may be performed.

The following demonstrates the actual encoding of a JWT token in the cryptoRTB protocol. The following is an actual signed message taken from the data stream of an operational cryptoRTB system:

eyJ0eXAiOiJKV1QiLCJhbGciOiJFZERTQSJ9.

eyJpc

...

U2ZGJlZiJ9.

ldC38FotC0xU220GrrnwtXGYyjF7rnJ78tPZx_d40U

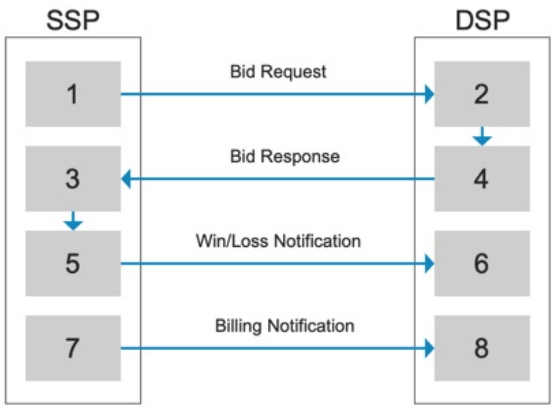
UTJ4_wP75hCmYaVzKudk3dQpgH5OnHWCR2xg

QlzipDDQ

The token has three primary sections. The header, in green, the claims (truncated) in blue, and the signature, in red. This the is base64 encoded token format.

3.6 Example Workflow

At this time, the proposed workflows have only been specified for the VAST specification. The proposed workflows are the minimum viable flow. Ideal implementations will be further described, but the authors do not expect full implementation of these workflows until such time as the minimal workflows have been adopted. Further iterations will include more use case specific examples in addition to VAST and more detailed recommendations.

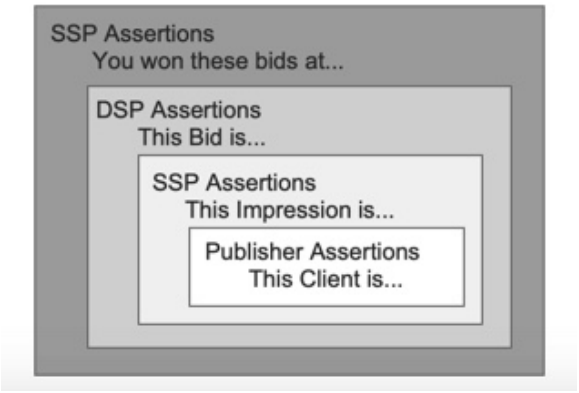


Simplified Diagram of openRTB Interactions

The minimum interaction flow may be specified as follows for a VAST interaction:

1. A publisher has a presentation opportunity.
2. The publisher either directly contacts a SSP through backend systems, or the SSP is informed of an opportunity through tags, pixels, or logic from an ad SDK that is in the content of the publisher.
3. The SSP constructs a bid offer based on the information provided from the publisher.
4. The SSP signs this bid request prior to distribution for bids.
5. The DSP(s) receive this request and countersign the response object. This signature should include, as a portion of its data fields, the bid request signature from the SSP. Further the DSP notification URLS from the VAST document should include a base64 encoded signature object as a portion of the url.
6. When the media is observed and the notification tags fire, the SSP and DSP observe the url and may correlate this event with the previously generated signature objects from the bid request and bid response messages.

In this way, the full record of an exchange is cryptographically verifiable between the interacting parties and no party may deny having said something that was transmitted.



CryptoRTB Cryptographic Linkable Assertions

The closure of the verification channel is performed through the client side interactions. Upon completion of these events, the SSP has proof of purchase, the DSP has proof of offer, the DSP has proof of impression, and the advertising agency may audit the full flow of data. Further, the publisher may see that only an authorized buyer was utilized, if such a restriction was specified.

During the initial stages of this implementation, the cryptographic identities of the participating parties may be directly distributed, but in order to simplify the problems around identity and information distribution, the proposed blockchain system may be utilized.

Although this system offers strong guarantees to the participants, there is room for improvement. Specifically, the ability of the SSP and DSP to resolve disputes with respect to observed impressions is not fully secure. This defect is due to the lack of client side participation in the validation circuit. Although the client can never be fully trusted, the client may perform actions that assist in validation. An example of an improvement would be the ability of a publisher to sign impression opportunities prior to being sent to an SSP. In the event that a publisher is not technologically capable of this action, a third party may perform this action on behalf of the publisher. If the proliferation of information is of concern, the data itself may be signed under encryption such that the delegated signatory is incapable of reading the information. This may be achieved through a concept known as a “sealed box” within the LibSodium encryption libraries. It is beyond the scope of this document to fully explain these mechanisms, but further information will be provided in future iterations. It is of note that the real time revocation abilities inherent to the blockchain based PKI previously described would allow the delegation of an entity in such a manner that their privileges may be rapidly revoked under detection of abuse. This also means that a publisher would never need to share a private key with an outside entity. It is the opinion of the authors of this paper that a private key should NEVER be shared under any circumstance.

In addition to the ability of the client to be the originating point of cryptographic traceability, it would also be advantageous for the client to participate in the closing event in order for the authenticity and integrity guarantees to be full circle. Through the same mechanisms described for the purpose of allowing a delegated signing entity, that acts on behalf of the publisher, the client may validate that an action did occur. This validation may be constructed by the SSP by encrypting a piece of information in such a way that only the client side may decrypt this information. Simultaneously the DSP would also encrypt a piece of information such that only the client could decrypt this information. Specifically, this would be the encryption of a signature object such that the callback url would contain the decrypted signature object encoded as a base64 url safe string. These urls should point to the resources of the opposing party. For example, the SSP generated url should point to the DSP and the DSP generated url should point to the SSP. In this manner the DSP and SSP may have independent proof that the client observed the interaction if either party may present the evidence contained in the encrypted data. This functionality could be embedded in the ad SDK of the client. As a stand in for this

functionality, an external party may also perform this function until such time as the client side ad SDK may be updated.

3.7 Operational Impact vs Ads.Cert

In order to create non-repudiation and authenticity guarantees in the ad tech supply chain, the current proposed solution of ads.cert suggests the utilization of the cryptographic curve named secp256r1. This is a poor choice for the engineering requirements of the advertising industry. On consumer grade hardware, Secp256k1 has max performance of 4,000 signatures per second. Verifications are far worse with a value around 1,300 verifications per second [6].

Comparatively, the Ed25519 curve, on modern consumer grade hardware, has the ability to sign over 100,000 times per second with an equivalent security to the secp256r1 curve. Additionally, Ed25519 algorithms are capable of verifying over 70,000 signatures per second. [4]

From the numbers previously provided, it is evident that the operational expense of secp256r1 will be at least 25 times more expensive than Ed25519.

4.0 Conclusion

This document is not a full specification. Several details will need to be further refined. The motivation is to provide a set of solutions that may address known shortcomings of the ad tech supply chain. The final iteration of this document should include formal verification of the security models and granular descriptors on all messages. This should be considered a living document until such time as otherwise stated.

5.0 Works Cited

[1] Internet Assigned Numbers Authority. [Online]. Available: https://www.iana.org/assignments/jose/jose.txt.

[2] IETF. [Online]. Available: https://www.ietf.org/rfc/rfc6750.txt

[3] "Intel Side Channel Vulnerability MDS," *Intel*. [Online]. Available: https://www.intel.com/content/www/us/en/architecture-and-technology/mds.html.

[4] "Digital Signatures," *Digital Signatures - PyNaCl 1.3.0 documentation*. [Online]. Available: https://pynacl.readthedocs.io/en/stable/signing/#ed25519.

[5] "JMESPath," *JMESPath*. [Online]. Available: http://jmespath.org/.

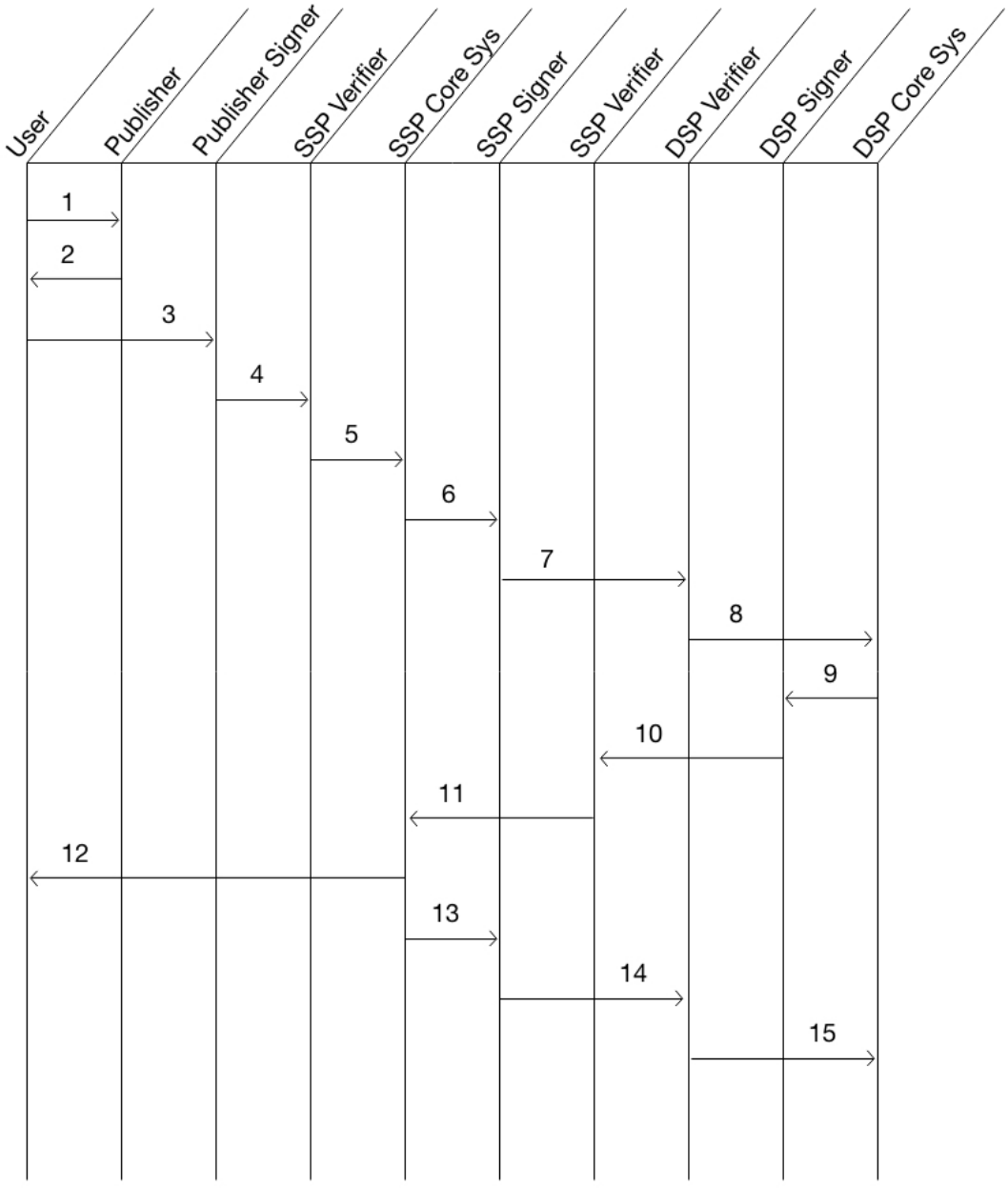
[6] "JSONPath Online Evaluator," *JSONPath Online Evaluator*. [Online]. Available: https://jsonpath.com/.

[7] "Language Guide (proto3) | Protocol Buffers | Google Developers," *Google*. [Online]. Available: https://developers.google.com/protocol-buffers/docs/proto3.

[8] Ali, Al Imem. "Comparison and Evaluation of Digital Signature Schemes Employed in NDN Network." *International Journal of Embedded Systems and Applications*, vol. 5, no. 2, 2015, pp. 15–29., doi:10.5121/ijesa.2015.5202.

6.0 Addendum

6.1 Protocol Timing Diagram



1.

6.2 Explanation of Diagram

2. Client (user) loads content from publisher.
3. Publisher sends back content with ad tags.
4. Ad tags fire and are sent to Publisher signing server.
5. Publisher signing server ensures data is correct from ad tag event and signs this data sending to SSP verification server.
6. SSP verification server receives signed messages from Publisher signing server. Verification server checks all cryptography and forwards to core logic of SSP.
7. SSP core logic constructs bid request and transmists to SSP signing server.
8. SSP signing server signs outbound bid requests and proxies call into DSP verifier.
9. DSP verifier receives signed bid request from SSP signing server. DSP verifier server checks all cryptography and forwards message into DSP core logic.
10. DSP core logic constructs a bid response and sends back to the SSP via the DSP signing server.
11. The DSP signing server signs the bid response and forwards to the SSP verifier server.
12. The SSP verifier server checks the cryptography of the bid response and forwards into SSP core logic.
13. SSP core logic selects winning bid and forwards content back to client.
14. SSP notifies the DSP, out of band, via the signing server, the win notification.
15. The DSP verifies the signature on the win notification.
16. DSP updates local records to reflect the win of an auction. This data will later be used to construct billing information.