A Data Model for Pre-Verified Data Quality Claims
Giridhar Manepalli, Suchitra Manepalli
gmanepalli, smanepalli @cnri.reston.va.us
Copyright © 2021. Corporation for National Research Initiatives

Abstract
Quality data is necessary for scientific research and for decision-making. However, data consumers do not usually possess the knowledge and the tools required to assess data quality. Current solutions rely on resourceful data producers asserting claims about quality of their data, but the main benefits may be lost if users cannot verify the accuracy of such claims, and have to rely only on their exogenous trust by signatories.

We propose a novel data model that includes provision for claims about data quality with proofs and also verification checks made by resourceful users, along with the source code used for that verification. The result is that users with limited resources only have the burden of ensuring the verification programs are correct and have the built-in knowledge to accept the merit of the claims. Pre-verified claims can be revoked, added, or pooled together making it possible for the burden of proof and the burden of verifying the proof to be amortized across time and distributed across multiple resourceful users.

1. Introduction and Related Work
For purposes of this paper,

- The term “data” shall mean information that is represented in digital form.
- We do not assume any particular definition for the term ‘quality’ in the context of data, but rather acknowledge that claims related to quality may be made about a variety of characteristics including data accuracy, completeness, latency, anonymity, and sourcing methods depending on the analytical and research needs at hand.

Data is increasingly used for decision-making in scientific research and for setting policies more generally. Given the zero marginal cost to replicate and disseminate information in digital form, data is replicated and re-used over and over again. Measuring data quality at the ends of dissemination trails remains the weakest part of the current practice: data-driven enterprises are placing implicit trust in repositories, in employees, and in researchers in lieu of explicitly evaluating the quality of data they use. This practice has the potential to cause serious harm, even inadvertently.

A representative example that demonstrates the prevalent nature of data distribution trails is the “Coronavirus in the U.S.: Latest Map and Case Count” page [1] put in place by The New York Times (NYT). The information and visualizations on the page are updated based on the dataset...
that NYT also makes available to the public. This dataset, which consists of counts of people who are infected and have died from the Coronavirus, is compiled from various state and local health agencies. The NYT described the methodology that they follow to merge, adjust, and transform the source data to arrive at the final dataset. While the methodology description is welcome, it is not an easy process to repeat for an independent reproduction, as can be evidenced from the cited methodology reference.

The trust that is placed in The New York Times may be justified by its legacy, but use of similar data transformation practices are plentiful and are carried out by different actors with different reputations without much discussion around those practices and certainly without offering verifiable claims.

A vast number of datasets are available for public use; but, in our research, we did not find any verifiable data quality claims to accompany those datasets beyond hashes and self-attested statements. A few communities and organizations stand behind the data quality claims, e.g., NIST Standard Reference Data [3], WMO Catalogue for Climate Data [4]. The combination of trust users may have in these organizations and their strong assertions about data quality these organizations make, may entice users to accept the data quality claims without verifying them. However, some data producers disseminate potentially valuable data, without any verifiable way to determine their quality. There are measures that may be taken to deal with this problem. Below, we highlight important aspects of the current state of practice for conveying quality of data to consumers.

There are several approaches to this issue in the marketplace, both open source [5] [6] [7] and proprietary [8], that enable “reproducible workflows” – a term that implies results from a (complex) computational experiment can be reproduced and matched against results from prior invocations of the experiment. These workflow systems allow researchers to inspect and validate the experiment and its resulting data, the code used in making claims, and perhaps even the math behind the resulting data. These systems, whenever used, help researchers to evaluate the quality and accuracy of data. Concepts such as seals of approval [9] are also implemented to characterize the trustworthiness of repositories from which data may be accessed.

These are valuable and important steps; however, if data is accessed from these systems, an artifact of dissemination trails, is that the data is just as meritless, quality and evidence-wise, as if the data had been produced in any other way.

Another related solution is proposed by Chiesa [10] to meet an objective of ensuring that messages circulating in a distributed system (i.e., a system based on coordinating components deployed in a network) comply with specified predicates. In spite of its origin in the distributed systems domain, their approach may be used to attach with data certain assurances about data, at least in some cases. However, users are required to integrate special programs at the
low-level within the original distributed system programs to be able to attach proofs of compliance. That can be cost-prohibitive and usually unattractive for most data producers; so a different solution was sought.

During the final stages of our current research, Barclay et al [11] argued for verifiable claims to be associated with data, instead of with the repositories that held the data. However, their effort too is about signed assertions rather than about making quality claims that can be independently verified as to their intrinsic merit by users of the data.

Finally, W3C has published a recommendation on Verifiable Credentials Data Model [12] that partially overlaps with our data modeling approach; however, their data model stops short of solving the problem and leaves the burden of proof verification to the consumers who usually lack the resources, knowledge, and tools to verify the claims.

Thus, there is a strong need to characterize data in such a way that allows users to infer the data quality with limited friction, and do so in a way that allows the transfer of such characterization to data products stemming from them.

2. Our Contribution

A key contribution of this research is a data model for generating a tamper-evident, revocable, and evolvable claims manifest that removes from consumers the burden of proof, the burden of proof verification, and the need for a system that validates trust in producers, but still empowers consumers to understand the intrinsic merit in the data quality claims.

Our data model has the following characteristics:

- Provides for portability and evolvability to include pre-verified claims about data that can travel with the data or separately from it.
- Provides actionable identifiers for source code of auditing verification programs.
- Allows data managers and downstream data users to add or revoke claims at any time in the future during the data lifecycle, and enables users to receive updates.
- Imposes no architectural limits or constraints on the number and nature of claims.
- Is well suited for building communities around claims registry, claim profiles, and verification programs and services.

The next section describes our technical approach and data model followed by sections that provide guidance on how to generate proofs and write verifiers for a few claims made about data.
3. Technical Approach and Data Model

Core Model

At the heart of our data model is a quadruplet: a claim, a proof, links to verifier software, and a signature that can be validated as illustrated in Figure 1. The uniqueness of this data model is the introduction of an software based verification program or service, called a verifier, and an attestation of a successful verification by that verifier making up the signature portion.

A claim, for example, can be made by a user that certain data has been generated no later than is given in a specific timestamp. A proof can be constructed to defend that claim, as described later, with the help of public timestamp servers. The verifier portion contains a reference to the source code of the verification program along with its public key that is used for verifying and attesting the claim. A reference to a service, i.e., the running code, that can re-verify the claim can additionally be included in the verifier portion. The signature portion contains the attestation from the verification program or service that it has successfully verified the claim.

![Figure 1: Components of a Pre-Verified Claim](Image)

Signatures are expected to be based on JSON Web Token (JWT) [13] notation, generated using asymmetric keys. A JWT is essentially an encoded version of the combination of some payload and signature of that payload, as made by some issuer. In this case, the payload for the JWT consists of the claim, the proof, the verifier, the data hash, and the issuer. The issuer of the signature (denoted using the property ‘iss’ in the payload) is connoted by the public key of the verification program or service. Public keys are expected to conform to JSON Web Key notation [14], but base64 encoded wherever relevant.

If the verification program determines that its attestation is only valid if the user trusts other public keys (related scenarios are discussed elsewhere in the paper), then that array of keys is to be included in the signature payload under the property called ‘dep’. The attestation may include JWT-spec defined iat, nbf, and exp properties in the payload. These properties convey when the signature is issued, when the signature validity period begins, and when it expires.
A “claims manifest” is a collection of pre-verified claims made against some data. It consists of four parts: an array of aforementioned quadruplets corresponding to various pre-verified claims, the hash of the data, a unique identifier, and an open-ended metadata portion to include arbitrary information as deemed necessary by the claimer, as illustrated in Figure 2. In this paper, ‘hash’ implies one or more cryptographic hashes produced using different hashing algorithms. Use of multiple hash algorithms is recommended, especially when computing the hash of data, to avoid the possibility of a second preimage attack.

Verification programs will potentially have bugs and their certification of the claim (and proof) may not be valid if those bugs have affected the claim verification procedures. Revocation of attestations is therefore important. Typically, this requires the verification parties to track certifications and periodically revoke mistakenly certified claims, all of which require a highly-available infrastructure. It is far simpler if the claims manifest itself provides a way for users to get an updated set of pre-verified claims bypassing the need for the verification parties to operate an infrastructure.

We introduce a new type of claim called ‘binding’ that offers a point of control for evolving and revoking claims in a manifest. A binding claim is made by the user generating the “claims manifest” and will contain identifiers that correspond to the latest version of the claims manifest. The proof is simply an attestation by the user that the binding claim contains the latest version of the claims manifest and that the reference applies to the data in question. It is the responsibility of the user to keep the online manifest up to date after revoking or adding claims as seemed relevant. The binding claim is not mandatory per this data model, although is strongly recommended.

Given the data distribution trails discussed in the introduction section, users may wish to add more claims about the data than what they found in the associated manifest. Users may also encounter multiple manifests of pre-verified claims pertaining to the same data from different channels. In such cases, users may produce new manifests by collating claims from prior
manifests for the data and allotting unique identifiers to the newer versions of the manifest. Previously allotted identifiers and references to associated claims from those manifest versions should be preserved in the metadata portion of the manifest.

While claims are verified and attested by verification programs, the consumer has the responsibility to inspect the source code of those programs, at least once, and make a list of programs (rather their public keys) that they can trust. The signatures of various claims in the manifest should be verified using those trusted keys. Such a verification program, which validates the manifest and is referred to as “manifest verification program”, could be integrated into consumer environments.

Given a set of trusted public keys and the hash of the data, the objective of the manifest verification program is to validate the signatures of the various claims in the manifest against the supplied trusted keys and ensure that the claims are indeed made against the data in question (based on the hash of the data in the manifest specified in the ‘sub’ property). That hash should also be matched against the hash the user supplied to this verification program to indicate that the user is dealing with the same data that the manifest verification program is validation against. The program should additionally verify that any dependent keys included in the ‘dep’ property of the signed payload of claims are also part of the trusted public keys. The program should retrieve the latest version of the manifest from the Internet using the reference in the binding claim prior to performing the above steps.

A prototype along with a demo and source code is made available here [15] in prototype form for users to validate the claims manifest.

![Figure 3: Roles of Various Parties associated with Data and Quality Claims](image-url)
Figure 3 illustrates the roles played by the various parties in the generation, evolution, and use of claims manifest for purposes of making data quality claims and subsequently evaluating them. Users who have the resources to generate proofs and write verification programs in order to support certain claims about data quality may do so independently from the users who are constrained in their capacity to produce such proofs and verifications. The data model proposed here enables the users with limited resources to still be able to leverage the work performed by resourceful users. And in fact the data model allows the work necessary to generate proofs and perform verifications to be amortized across time and various parties.

The argument here is two-fold: generating proofs and writing verification programs are difficult compared to auditing pre-written programs, and sometimes constructing proofs a posteriori is impossible or non-trivial requiring proofs and verification of proofs to co-occur with claim generation.

**Claims, Proofs, and Types**

The number and nature of claims and proofs are not constrained by the data model. However, to encourage reuse, the notion of types for claims, proofs, and verifiers are introduced.

Each claim has a type that defines the nature of the claim: that the claim is, say, about the data origination time or about data sourcing methods or about its identifier. Each proof has its own type that defines the nature of the proof: that the proof, say, provides evidence for a timestamp claim using public timestamp servers in a specific way. Effectively, because a given claim can be proved in multiple ways, the proof type communicates the specific approach taken to prove the claim to the verification program. Verifiers too have their own types to track the various pieces of information that could be packed for the benefit of the auditors. JSON serialization is adopted for claims manifest. A TypeScript interface for the manifest is illustrated in Appendix B.

There are a few different classes of claims users can make about data using the described data model. Below, three classes of claims are discussed to give a flavor for the various kinds of claims users could make about data, and provide examples that apply to each class:

- Explicit proof claims: Claims of this class require claimers to include explicit proofs that defend the claims. The verification programs, in this case, do not need external input to validate the claim and rely solely on the provided claim and proof (and the built-in knowledge). Examples include timestamp claims that convey data is available with the user by a certain time and non-repudiation claims that convey data is sourced from a certain entity.

- Reproducible claims: Claims of this class require the verification programs to have the built-in knowledge to process the data in question and/or some external data to validate the asserted claims. Examples are that a certain scientific dataset is consistent
with physical principles or that the data comports to differential privacy or that it is statistically similar to some reference data. Other examples are that the identifier of the data is such and such as allotted by a specific service or that its type is such and such as defined by a specific service or specification. Service endpoint information that can reproduce the data by repeating a workflow is another example that belongs to this class of claims.

- Attested claims: Claims of this class are signed assertions made by claimers where either there are no verification programs that can verify the claims or it is impractical to create verification programs for those claims. Claims of this class are not pre-verified, as such. A claims manifest therefore is a collection of pre-verified claims and attested claims. Examples of attested claims may include an assertion the data is derived from other data in some arbitrary way or that the data is ethically sourced. Binding claims too are attestations.

While it may be inferred from the above description that certain claims require specific kinds of proofs, that is not always the case. For instance, the timestamp claim can be made with an explicit proof or just be asserted without proof. In general, the claim types indicate what claims are being made and the proof types enable the claims to be defended. The verification program, however, should have the knowledge and the logic built-in to verify the combination of the corresponding claim and proof.

During the research project, CNRI created prototype software that includes verifiers for combinations of the following claim and proof types:

**Claim Types**

- Identifier: the identifier, such as handle, as allotted by a identifier service.
- Type: the type of the data, per some system or specification.
- Source: the source of the data; this may include a reference to a running service that can return the data.
- Workflow: the software method or methods used to produce the data; this may include a reference to a running service that can reproduce the data.
- Dependencies: the set of data from which the data of interest is derived.
- No Later Than: the timestamp indicating the latest time at which the data is known to come into existence.
- Binding: the latest version of the claims manifest for which this claim is being made.

**Proof Types**

- Reproducible: the claim can be reproduced if the verifier.
- Roughtime: the No Later Than timestamp is proven using Roughtime services (as discussed in Appendix A).
• Signature: a (JWT) signature on some information by a third-party. (The verifier needs to know how this proof can support the corresponding claim and also how to verify the signature independent from the data).
• Attest: the claim is merely asserted by the claimer.

Appendix A provides guidance on how to produce pre-verified claims, using above types, that conform to our data model with the help of a few examples.

4. Prototype

The utility of our approach will be in connection with the validation of the data model by various communities, and in its uptake. The power of the model is showcased by a prototype. A dataset produced using data from two US Government sources, specifically census.gov and usda.gov, can be accessed by following this link:
https://hdl.handle.net/20.5000.1080/a76b412c0a880dfb47b67651c55f1eb9?locatt=view:claimsUI.

This dataset is produced in stages as part of a workflow. Figure 4 shows the various pre-verified claims that dataset is associated with, such as its identifier, type, source, workflow used to produce it, the outcome from each of the workflow stages (i.e., dependencies), and the time they are produced. The claims visualization (a more legible version) can be accessed by following this link:
https://hdl.handle.net/20.5000.1080/a76b412c0a880dfb47b67651c55f1eb9?locatt=view:claimsUI.

The visualization includes links to the source code that corresponds to various verifiers. The online visualization also shows that the user reviewing the claims will need to trust the public keys associated with the verifier programs to accept the claims. Most of these claims are not just assertions, but verifiable (and indeed verified) statements about data.
Figure 4: Claim Manifest of a Dataset Produced for an Example Workflow

Handles

A large number of scientific resources are identified using persistent identifiers called handles, or digital object identifiers more generally. Some of these handles are branded as DOIs. At present, the handle infrastructure is primarily used to redirect browsers from handles to URLs that are used to access publications and datasets using web protocols. However, the handle system was originally designed to enable handle records to contain state information about digital objects that could include not only URLs but also other methods of access. We leverage that original design to store claim manifests in handle records. For example, the handle record associated with the identifier, 20.5000.1080/a76b412c0a880dfb47b67651c55f1eb9, of the aforementioned dataset contains the manifest in addition to links to various aspects of the dataset. The handle record screenshot is shown in Figure 5. The record can be accessed by following this link: https://hdl.handle.net/20.5000.1080/a76b412c0a880dfb47b67651c55f1eb9?noredirect.
The manifest is associated with a handle value type 20.Type/ClaimsManifest. This approach to managing claims manifest in the handle record makes it easy for data managers to update claims routinely and also removes the need for any extra machinery on the part of consumers to discover such claims: latest pre-verified claims are just one additional click away.

5. Burden of Proof and Verification of Proof

Our data model encourages claims to be made about data quality that can be defended in programmatic ways, enabling consumers with limited resources to ascertain the inherent merit of the claims. However, users making the claims must also provide claim-specific software that can verify those claims.

There are two distinct properties of our data model that allow such burden to be amortized across time and distributed among users: (1) there may be multiple claim manifests that can be associated with (the same) data and this is possible because the manifests need not be attached to data, and (2) the binding claim in a manifest provides a control point for a claimer to communicate an updated form of manifest with any consumer, with the help of an actionable identifier in the binding claim that can lead to the latest manifest.

The intrinsic advantage of those twin properties is that, as claim manifests emerge for a given unit of data at different points of the data dissemination trail, those manifests may be collated
and an updated version of the manifest may be made available via the appropriate identifier for the corresponding binding claims in these various versions.

How exactly will those versions be discovered is not prescribed in this paper. However, communities may be formed around any given data or sets of data to share the responsibility of generating proofs, writing verification programs, and issuing pre-verified claims against data. Those communities may leverage publicly hosted code repository systems such as GitLab and GitHub to report and maintain versions of manifests for a given data (based on its hash). Different groups may create and commit to different (code) branches of manifests. If the community has a moderator, then that party can bear the responsibility of merging the pre-verified claims into the canonical branch, and the binding claim may reference just the manifest from the canonical branch allowing consumers at different distribution trails to synchronize because of the periodic merging of manifests by the moderator.

6. Future

With the help of a prototype, we have demonstrated how a variety of claims can be evaluated for their intrinsic merit. We described a scheme that can be used by data producers and repositories to add claim manifests corresponding to existing data. The proposed model also provides users an option to define bespoke claims, proofs, and verifiers with the help of types, all without imposing any more burden on downstream users to evaluate and evolve them. We envision the possibility of profiles being created around the notion of types that pertain to quality-imputing claims, proofs, and verifiers. Registries of such types may emerge in different domains and industry verticals. In the various scientific disciplines, specifically, a shared code repository model as stated above or a registry of claim manifests may emerge that harvests various pre-verified claims enabling consumers in possession of data to discover associated claims based on the hash of the data. All these possibilities highlight a promising future for scientific researchers and social enterprises where the need for quality data is increasingly important.

7. Conclusion

Obfuscation of source code is no longer an accepted practice for achieving safety in the computer security domain; the use of open source programs is imperative when quality (of code), especially its conformance to specification and accuracy, is relevant. We adopted that approach to enable transparency in the data quality assessment enterprise. We defined a pre-verified claims model that relies on open source based programs and demonstrated how downstream users of data can ascertain the intrinsic merit of the claims at different waypoints in the data distribution trails, and also add or revoke claims along the way. The resource intensive tasks of generating proofs to demonstrate data quality and validating such proofs are left to the more resourceful players along the dissemination paths, thus easing the burden on users with limited resources to assess data quality.
8. Acknowledgements

We would like to thank the technical staff at CNRI for helping us conduct the relevant research and produce this paper. Ian Little (from CNRI) deserves a special thanks; he implemented some key portions of the project in software and, in the process, helped us identify and resolve issues in some of the design elements of the project.

This material is based upon work supported by the National Science Foundation under Award No: 1838981.

Appendix A: Example Pre-Verified Claims

Explicit Proof Claims

Timestamp Claim
To make a claim that the data is available with a user at a given timestamp, say, to state the data creation time, the user will have to somehow prove the data exists by that timestamp and that such a timestamp is not fictional. One solution to this is to rely on public timeservers that accept an arbitrary nonce at request time and respond back not only the current timestamp but also a cryptographic signature on the combination of the timestamp and the supplied nonce. If the nonce being sent with the request is the hash of the data, then the verification program can trust the data exists at the time of such request as long as the public timeserver is a reputed source of time. Roughtime Protocol [16] supports this way of producing signed timestamp responses.

If the timeserver is not a reputed server and therefore its timestamp cannot be trusted, multiple timeservers operated by different parties can be consulted to prove timestamp veracity. In fact, a chain of responses can be formed with the hash of the response from the prior request acting as the nonce supplied in the subsequent request, as described here [17]. The most recent timestamp from the majority of responses will form the claimed timestamp. The chain of signed responses will form the proof. The verification program, in this case, should validate the timestamp claim using the proof and produce a signature. Because the verifying user will have to trust timeservers in order to trust their proof, the verification program signature will reference the collection of public keys. The claims manifest verification program will ensure that these public keys are also in the user trusted key set before asserting the claim is valid.

Source Non-repudiation Claim
To make a claim that the data is provided by a given source, the user may reference the source by its identifier or domain name. If the source signed the hash of the data, then that signature should be placed in the proof portion. The verification program should validate the signature
and attests by signing the claim. In its signature, it references the public key of the source. The claims manifest verification program will ensure the source’s public key is in the keys that are supplied to the program.

If the source, however, has not provided a signature, but only made the data available via some security protocol such as TLS, then proving the source may require preserving all the bytes between the client and the source while retrieving the data. A verification program can then ascertain that the data was provided by the source from those bytes.

**Reproducible Claims**

*Conformance Claim*

To make a claim that the data conforms to a type or a specification (signifying either its format or its serialization or semantics or some combination of these), the user may include inline or by reference the specification information in its claim portion. The verification program should have the knowledge to extract such a specification and parse the data against the rules in that specification to determinate the data’s conformance. If the program succeeds in verifying the data to have conformed to the type, it may sign the claim. The proof portion will just state that the proof type is ‘reproducible’.

*Reproducible Workflow Claim*

To make a claim that the data is computed by certain code, the user in the claim may include inline or by reference the source code or the service endpoint that can reproduce the data by repeating a workflow. The verification program should have the knowledge to extract and execute the source code or to invoke the referenced service from its endpoint information to reproduce the data. The resulting data must be matched against the original data for the program to sign the claim. The proof portion will just state that the proof type is ‘reproducible’.

*Identifier Claim*

To make a claim that the data is associated with an identifier, as allotted by a certain service, the user may include the identifier along with any service endpoint information which recognizes the identifier in the claim portion. The proof portion will just state that the proof type is ‘reproducible’. The verification program should have the knowledge to interact with the identifier service to either retrieve the data and match against the original data (via hashes) or query to learn if the identifier is allotted for data of a certain hash. If the response is positive, the program may sign the claim.

Claims that verify the consistency of data with physical principles, or that the data statistically correlates with some reference data, or that the data is sufficiently anonymized and embedded with noise to exhibit differential privacy, or that data conforms to some know type or specification can be made similarly with the help of verification programs that have the knowledge to validate such claims.
Attested Claims

Attested claims are signed assertions of claims made by certain parties without proofs. Although attestation of claims is a prevalent practice, it is argued here that these claims should be used minimally so that users at the ends of the data dissemination trails can validate the intrinsic merit of the claims. A binding claim is an example of an attested claim (that is an assertion about the location of the latest manifest by the claimer). Other examples are that a user knows the data is synthesized from other data but the exact nature of derivation is unknown or that the data gathering process is known to follow a strict ethical protocol but that the protocol is hard to be (or cannot be) verified after the fact.

The proof type for these claims will be ‘attest’. No other information is necessary for the proof. The attester signs the claim as if it is the verifier, but no verifier is included as such. The program that verifies the claims manifest will validate the attested claims if the attester’s public key is supplied to the program as a key connoting that the user trusts the entity associated with that key.

Appendix B: Pre-Verified Claims Model in TypeScript

```typescript
interface ClaimsManifest {
  preVerifiedClaims: PreVerifiedClaim[];
  dataHash: DataHash;
  identifier: Reference;
  metadata: any;
}

interface PreVerifiedClaim {
  claim: Claim;
  proof: Proof;
  verifier: Verifier;
  signature: Signature;
}

interface DataHash {
  hash: string; // base64 of hashes below
  hashes: Hash[];
}

interface Hash {
  algorithm: string; // e.g., SHA1, SHA256, SHA512
  value: string;
  encoding?: string; // e.g., Base64
}
```
interface Claim {
    type: string; // e.g., identifier, type, no-later-than, source, workflow, dependencies, binding
    value: any;
    encoding?: string; // e.g., Base64
}

interface Proof {
    type: string; // e.g., attest, jwt, roughtime, reproducible
    value?: any;
    encoding?: string; // e.g., Base64
}

interface Verifier {
    type: string; // e.g., identifier_attest, type_reproducible, no-later-than_roughtime,
    source_jwt, workflow_reproducible, dependencies_attest, binding_attest
    value: object;
    encoding?: string; // e.g., Base64
}

type Signature = string; // JWT from ClaimSignaturePayload (see below) as signed using
verifier's private key

// Example values for claims
type ClaimValue = SourceReference | Reference | string | number | boolean |
SourceReference[] | Reference[] | string[] | number[] | boolean[];

// Example values for proofs
type ProofValue = RoughtimeProof | JWTProof | string | number | boolean | RoughtimeProof[] |
string[] | number[] | boolean[];

// Example value for verifier
type VerifierValue = SourceReference;

// Definitions referenced above - mainly suggestions

interface SourceReference {
    source: Reference;
    service?: Reference;
    publicKey?: Reference;
}
interface Reference {
    scheme: string; // e.g., hdl, url, domain, tcp
    identifier: string;
}

interface RoughtimeProof {
    hash: string; // either the data hash or the hash of the response from the prior Roughtime request if a series of requests are issued
    random: string; // a function of hash and random sent to the Roughtime server
    nonce: string; // a function of hash and random sent to the Roughtime server
    server: Reference; // endpoint
    timestamp: number; // epoch
    response: string; // signed response from Roughtime server.
    publicKey: JWKPublicKey | JWKPublicKeys;
}

interface JWTProof {
    payload: JWTProofPayload;
    signature: string; // jwt of the payload
}

interface JWTProofPayload {
    iss: string; // issuer's public key in JWKPublicKeys or JWKPublicKey, encoded in base64
    iat?: number; // epoch
    nbf?: number; // epoch
    exp?: number; // epoch
}

interface JWKPublicKey {
    kty: string;
    alg: string;
    crv?: string;
    x?: string;
    y?: string;
    use?: string;
    n?: string;
    e?: string;
    kid?: string;
}

interface JWKPublicKeys {
    keys: JWKPublicKey[];
}
interface ClaimSignaturePayload {
    claim: Claim;
    proof: Proof;
    verifier: Verifier;
    sub: string; // hash from dataHash
    iss: string; // verifier's public key in JWKPublicKeys or JWKPublicKey, encoded in base64
    dep?: string; // array of keys in JWKPublicKeys, encoded in base64
    iat?: number; // epoch
    nbf?: number; // epoch
    exp?: number; // epoch
}

References
[12] W3C Verifiable Credentials Data Model 1.0: https://www.w3.org/TR/vc-data-model/  
[14] JSON Web Key Proposed Standard: RFC 7517