Open Reputation Framework

Samuel M. Smith PhD
sam@worldtable.co

The World Table
758 E. Technology Ave
Building F, Suite 2101
Orem, Utah 84097 USA
worldtable.co

Draft Revision 1.2

2015/05/13

Abstract

Open Reputation is an open source software framework that supports the generation, evaluation, and dissemination of reputation in a trustworthy manner. While several approaches to reputation evaluation and scoring exist, they are all relatively narrow in scope. None of them provide a highly generalizable foundation for the collection, storage, distribution, association, ascription, authentication, authorization, encryption, aggregation, evaluation, and dissemination of reputational events across broad application domains.

In Open Reputation, a reputation is an indicator built up from the processing of information items called reputational events. This work presents a notional architecture for Open Reputation as well as a formal specification for some of the components such as the foundational data element for representing reputational activity, that is, a reputation event, or *repute* for short. Reputes provide the basis for higher level reputation computations.

In order to ensure that the resulting reputation is trustworthy the processing steps or transactions involved in generating each reputation must each be performed in a trustworthy manner as well. One way to to perform the transaction processing in a trustworthy manner is to use a transparent publicly verifiable methodology. One methodology for public verification of transactions is a shared public ledger where a distributed consensus algorithm is used to accept and validate entries in the public ledger. The key to this methodology is that the trust is diffused across all the members of the consensus pool so that the actions of some do not invalidate the group consensus. As a simplistic generalization, a shared public ledger is trustworthy if the majority of the members in the consensus pool are not colluding to defraud. Various consensus schemes have been developed to ensure this condition is met. These include proof-of-work, proof-of-stake, and Byzantine agreement. Distributed consensus algorithms can be used to provide a trustworthy methodology for other types of processing not just a public ledger. Open Reputation extends this concept to a public database.

In addition to trustworthiness, another requirement for Open Reputation is that the distributed consensus algorithm support a large number of low latency transactions that result in the generation of reputations for a large number of entities in a large number of contexts. Various distributed consensus algorithms have been developed with different performance

characteristics. This work will compare the characteristics of distributed consensus algorithms and describe a candidate set of algorithms that have the potential to meet the requirements of Open Reputation for further development as part of the Open Reputation framework.

Although the software framework is open source, the reputes themselves are not necessarily public. The timing and degree of disclosure should be controllable by the associated parties. Therefore this work also describes secure protocols for the transmission and storage of reputes as well as different degrees of privacy engendered by various combinations of anonymous/public identifiers and content encryption.

1 Reputation

1.1 Overview

The dictionary definition of *reputation* is a noun meaning, *the estimation in which a person or thing is held, especially by the community or the public generally*. The root of reputation comes from the Latin word *reputāre,* which is equivalent to *re* + *putāre*, that is, to re-think or re-consider.

A common usage of *reputation* is as a considered evaluation of how someone has behaved in the past that provides an estimation of how they might behave in the future. In other words, reputation is a measure of past behavior may be used to predict future behavior. The context of a reputation may also be important, that is, past behavior may be a better predictor of future behavior when applied to similar contexts.

One approach to the computational generation of a reputation is to aggregate relevant instances of behavior. These instances of behavior we call *reputational events* or *reputes* for short. In this formal approach, reputation is not directly ascribed to an individual but is built up indirectly from *reputes* associated with that individual.

Consider for example, two approaches to computationally generating a person's reputation for promptness. One approach is to have associates of that person provide promptness ratings. These direct ratings are then perhaps averaged to provide the reputation. In this first approach most of the work is performed in the minds of those associates who must internally evaluate and ascribe a promptness rating. The computation part is merely to calculate the average. Another approach is to have associates of that person provide attendance ratings based on the degree of lateness or earliness of that person's attendance. The reputation is then generated by computationally evaluating, aggregating, weighting, and averaging all the attendance ratings. The second approach is more general and indirect in that the ratings from the associates are applied to instances of behavior by a person that contribute to promptness, not to the promptness of the person directly. This allows for finer grained computational re-scoping, re-evaluating and re-weighting as more attendance ratings are collected. Indeed, this approach can be extended where the degree of lateness is derived from recorded behavior or some other lower level indicator. The motivation for a reputational event specification is to enable arbitrary levels of nesting, precision, and granularity in the data sources from which reputations are comprised..

Moreover, because reputation is derived from behavior, the act of rating could also be the basis for a reputational event ascribed to the *rater*. Thus the number and type of ratings created by a *rater*, might generate a reputation for the rater's fitness as a rater which may be fed back to change the relative weighting of that rater's contribution to the reputation of the *rated*.

Each repute may have associated with it several relevant information items or data elements. Consider for example the data elements of the following application of reputation:

A web commenting system that collects user provided comments and/or ratings about a blog post and about comments to that blog post. The blog post, the comments, and the ratings constitute the core elements of the associated reputes, that can be used to indirectly derive and ascribe reputations both to those being rated and to those making the ratings. The rated include the blog poster and anyone making a comment. The raters are those that rate the blog post or rate one of the comments.

Suppose a user rates the blog post. This rating is a repute that is associated with a behavior of the blog poster, that is, the blog post itself, but is also associated with a behavior of the user who rated the post, that is, the rating. Suppose also that the blog host sources this repute to a reputation curator that collects and aggregates the reputes associated with the blog poster. In order to ensure confidence in the provenance of the repute, suppose that the curator validates the reputes against forgery via a distributed public ledger (blockchain) managed by a consensus pool of validating hosts. The curator also saves reputes in a distributed storage system to enable auditing.

Thus the relevant data elements, associated with the repute, include not only the rating of the blog post but information about all the entities that contributed to the process of generating and validating the reputation. The reputation event could include the following information:

- *Context*, the associated context in which the rating was made such as a datetime, url, etc.
- *Blog Post*, the associated blog post and post identifier
- *Rating*, the rating
- *Blog Poster*, the user whose post is being rated.
- *Rater*, the user making the rating.
- *Source*, the blog host where the post and rating are made
- *Curator*, the reputation engine that stores and aggregates the repute
- *Validator*, the distributed consensus pool that is used to validate the origination and exchange of of the repute via a shared ledger
- *Storage*, the distributed storage system

1.2 Terminology

The following formal specification of *reputational events* (*reputes*) provides for all of the relevant information items associated together with definitions of the associated data element types. These include *repute, reputee, reputer, reputet, reputage, reputery, reputable entity, identity, nym, name, initiator, copartent, and arbiter*.

The terminology developed here provides a lexicon of defined terms that are designed to reduce ambiguity when discussing computational tasks associated with reputation. Some the terms presented here are new fabrications that are imbued with a specific meaning. The fabricated terms are designed to be evocative of their associated meanings while also avoiding the semantic baggage that comes from the reuse of pre-existing terms. The motivation is to provide a mental metaphor that is unique to this context.

A *reputation* is a measure of likely behavior based on past behavior.

A *reputable entity, RE, or reputee* for short is a potentially pseudonymous but uniquely identifiable entity. A *reputee* does not necessarily represent a distinct human individual, but

may also represent a group of individuals, an organization, or a non-human computational agent. The degree of disclosure or provable association of a *reputee* with a real entity may vary from full pseudonymity with no public disclosure or provable association to full public disclosure and provable association. Reputations are associated with *reputees*.

An *identity* is an data structure that uniquely represents, identifies, and describes a *reputee*. *Identities* are associated with one and only one *reputee*. *Identities* are serialized in JSON. *Identities* are composed of *identifiers* and *descriptors*. A *reputee* may be associated with multiple identities.

An *identifier* is symbol, typically a string of characters, that is uniquely associated with a *identity* representing a *reputee*. An *identity* may have multiple *identifiers*. These include at least one *primary identifier,* may include one or more *secondary identifiers*, and may or may not include *tertiary identifiers*.

A *descriptor* is a symbol, typically a string of characters, that provides additional relevant information about a *reputee*. Descriptors may or may not be unique to an *identity*. A *descriptor*, for example, may be a name, a picture, a video, a description, or a slogan.

A *cryptonym or nym* for short is a globally unique cryptographic key that is used to uniquely identify a *reputee*. It is a *primary identifier* in an *identity* representing a repu*tee*. The dictionary definition of cryptonym (krĭpʹ tə-nĭmʹ) is a noun, meaning, *A word or name that is used secretly to refer to another; a code name or code word*. A *reputee* may have multiple *nyms* associated with it but each *nym* is directly associated with one and only one *reputee*. The purpose of a Nym in Open Reputation is to enable the cryptographic authentication, signing, verification, encryption, and decryption of reputation related transactions.

An *alias or aka* is a human friendly globally unique string of text that may used to identify a *reputee* (*reputable entity*) in addition too, but not instead of, a *nym*. It is a *secondary identifier* in an *identity* representing a *reputee*. A *reputee's alias* must be a human readable globally unique *identifier* that is uniquely associated with the *reputee*. A *reputee* may have multiple *aliases* associated with it, but each *alias* is associated with one and only one *reputee*. The purposes of an *alias* in Open Reputation is to provide a human friendly way of interacting with *identities*. An alias provide a human friendly identifier than can be used to lookup the *nym(s)* associated with an *identity*.

A *URI* is a globally unique uniform resource identifier, it is one type of *tertiary identifier*. A *URI* may be used to generate an *alias* using a naming convention. Other types of *tertiary identifiers* include GPS locations, email addresses, street addresses, domain names, and phone numbers.

A *reputation event* or *repute for short* is the basic information construct of a *reputation* system. *Reputes* encapsulate information that is used to generate *reputation* scores or reputation profiles for a *reputee*. Notwithstanding the dictionary definition of *repute* which includes both noun and verb forms, that are respectively, *estimation in the view of others*, and *to consider or believe (a person or thing) to be as specified*, the definition used in this specification is that a *repute* (noun) is a *reputational event* and to *repute* (verb) is to generate a *reputational event*. A *repute* essentially makes a statement or claim about an activity that contributes to the *reputation* of a *reputee*.

A *repute (reputational event)* includes all the relevant information items needed to uniquely ascribe a distinct reputational activity to a given reputee (*reputable entity)*. Reputation scoring is primarily the aggregation and evaluation of *reputes* about a *reputee*. The *reputee* in this

context is the target of the *repute*. It is noteworthy that reputation scoring can be applied to all behavior associated with a given *reputee*. This includes *reputes* a *reputee* gives, not just *reputes* a *reputee receives*.

A *ruid* is short for repute unique identifier. A *ruid* provides a globally unique identifier for a single *repute*. A *ruid* is a pseudo-random number that may be encoded as a hex string.

A *reputer* is a *reputee* that is the originator or primary source of the information items in a *repute* about another *reputee*, that is, the *reputer* originates a *repute* that targets a *reputee*. The *repute* is about a distinct reputational activity of the *reputee*. The *reputer* and *reputee* for a given *repute* may be the same *reputable entity*. *Reputes* are reflexive, that is the act of originating a *repute* about another *reputee* also associates that *repute* with the *reputer*.

A *reputery* is an entity that formulates, records, stores, exchanges, aggregates, or otherwise manages *reputes*. A *reputery* must also be a *reputee*. *Reputeries* are intermediaries that either directly facilitate the targeted ascription of *reputes* from the origination by a *reputer* through to the eventual generation of the *reputation* of the targeted *reputee* or indirectly collect and distill information about another *reputee* without an originating *reputer*.

A *repute* may be direct, that is, a source *reputer* directly *reputes* a target *reputee*. Alternatively, a *repute* may be indirect, that is, derived or collected and distilled from some behavior or other information about a *reputee*. In this case, the entity that collects and distills the information to form the *repute* is considered a *reputery*.

A *reputation event transaction*, (*RET*), or *reputet* for short, is the primary mechanism for exchanging *reputes*. A *reputet* is a cryptographically signed and validated transaction between *reputeries*. A *reputet* includes additional information items needed to support the specific exchange transaction.

A *repute* may include multiple essential items of information. Some of these items may be needed to associate the *repute* with one or more *nyms*, those needed to associate a *repute* with a context, or those needed for provenance, classification, signing, validation, and chaining.

The *reputage* of a *repute* denotes additional ancillary information items associated with a *repute*. Some of these items may be less well defined, variable, or optional but provide useful ancillary details of the associated reputational activity. Once reason to separate the *reputage* from its associated *repute* is to speed up processing or reduce memory requirements since in many cases the information in the *reputage* may not be used directly or frequently in reputation evaluation algorithms.

The following diagrams show the relationships between *reputers*, *reputes*, *reputees* and *reputeries* for both direct and indirect reputes. In the following diagram a source *reputer* generates a *repute* that directly targets a *reputee*

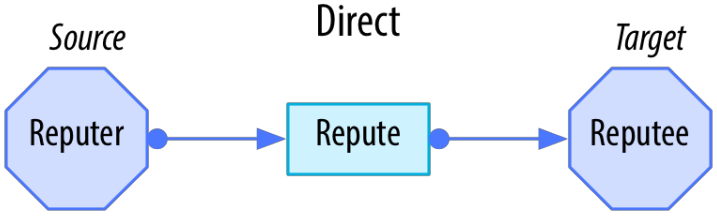

Fig. 1.2.1 Direct Repute

In Open Reputation the act of reputing also implies a repute on the reputer. By analogy, any reputes incident on a reputee effectively generate a reflected repute on the reputer.

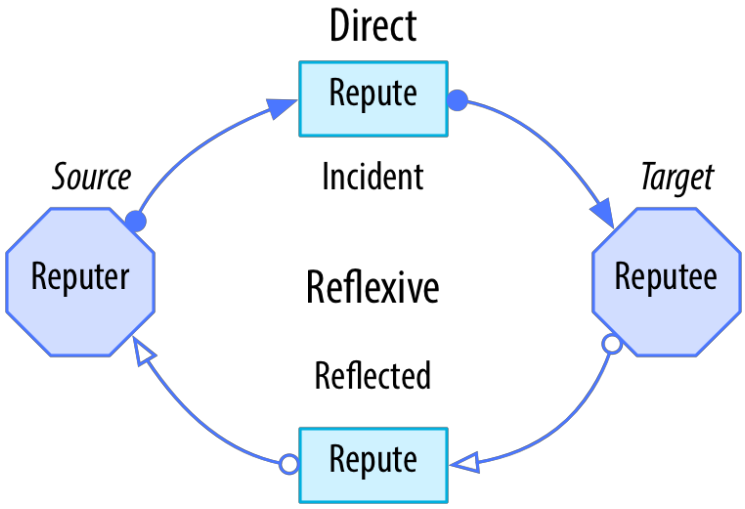

Fig. 1.2.2 Direct Incident and Reflected Reputes

When each repute includes references to both the reputer and reputee, only one repute is needed to convey the reflexive nature of a repute, as shown in the next diagram.

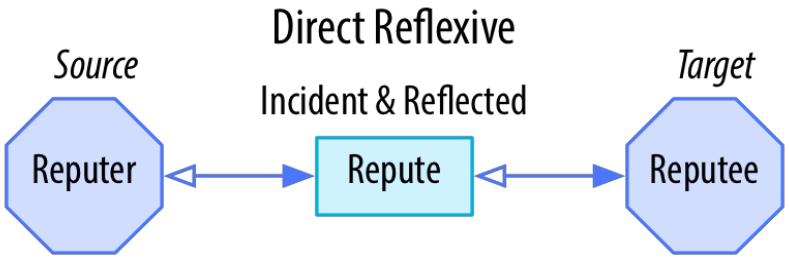

Fig. 1.2.3 Direct Reflexive Repute

In many cases multiple processing or transmission steps may be required before a repute is fully composed and ascribed to the target. The intermediaries are called reputeries. This relationship is shown in the next diagram.

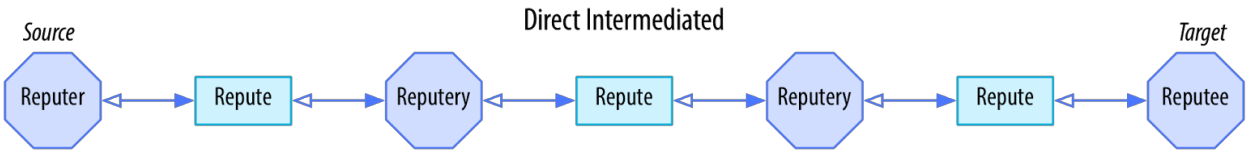

Fig. 1.2.4 Direct Intermediated Repute

The system needs to be able to manage the various ways that a reputer might attempt to game the system including indirect or direct reputes directed at itself. As shown in the following diagram, Open Reputation allows the modeling of self-reputes.

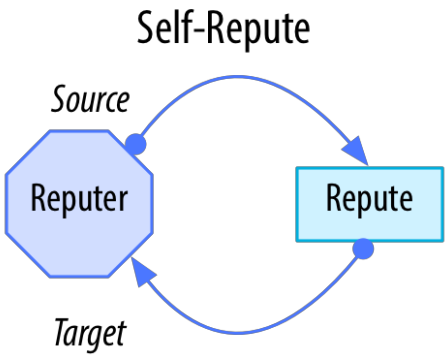

Fig. 1.2.5 Self Repute

When a repute is not sourced directly from a reputee but is derived from information items such as might occur with machine learning algorithm or some other derivation process the repute is said to be indirect. The entity performing the inference is a reputery. This is shown in the following diagram.

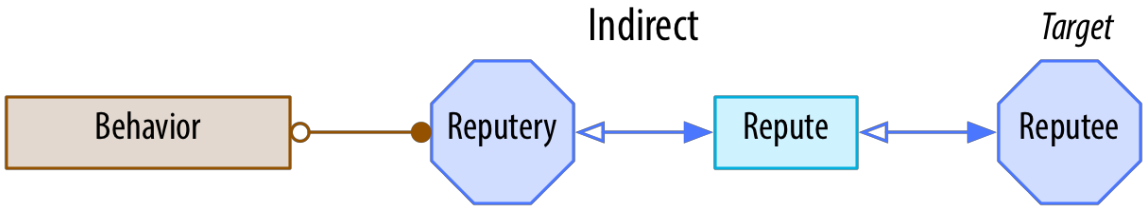

Fig. 1.2.5 Indirect Repute

An indirect repute might also have intermediaries as shown in the following diagram.

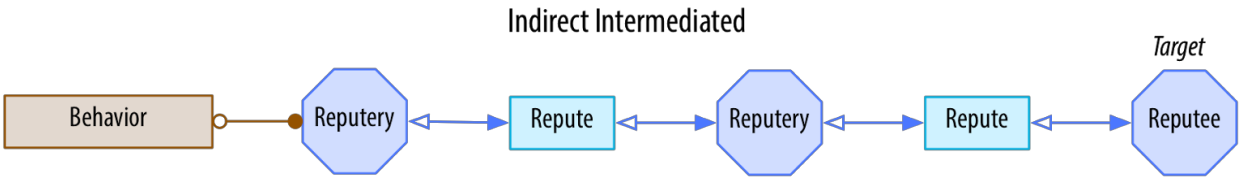

Fig. 1.2.7 Indirect Intermediated Repute

1.3 Identity-Cryptonym Registry

Because the transmission, storage, and validation of reputes relies heavily on cryptographic processing, *cryptonyms* (*nyms*) are the primary identifier included in reputes. Embedded storage of *nyms* in reputes facilitates the rapid application of the relevant cryptographic

processing without having to perform costly lookups. Human facing applications, however, may benefit from exposing the more human friendly *alias* as the identifier used to associate a repute with a reputee. Moreover, many information sources use other identifiers such as URIs that can be converted to aliases using a naming convention. Consequently there is a need for a mapping between the cryptonyms and aliases associated with an identity/reputee. This mapping is provided via a public registry of identities that provides both forward and reverse lookups of identities from cryptonyms and identities from aliases. The aliases in the registry are governed by a distributed consensus ledger (blockchain).

1.4 Identity Disclosure and Validation

As we see it, a reputation is a predictor of future behavior based on past behavior. One of the predictors of future behavior is the degree of public disclosure of an entity's real identity. Public disclosure exposes an entity to the consequences of its actions both good and bad. This risk/reward consequence of disclosure influences those actions and therefore provides predictive value. To be meaningful an identity disclosure must be validated. The nature of the validation is usually identifier dependent and may be application dependent. Because Open Reputation is a framework it is not opinionated about many application specific implementation details. This includes specific Identity validation mechanisms.

Open Reputation plans to support plugins or interfaces for identity validation services. A good example of an identity validation service is OneName. In OneName a user creates a unique named identity that is registered on the NameCoin blockchain. OneName then provides validation of identifiers for various siloed identities such as FaceBook, Twitter, and GitHub, as well as blockchain identities such as BitCoin. Open Reputation would then allow a user to claim an Open Reputation identity by validating against their OneName identity. Open Reputation will also include a native service for validating an Open Reputation identity..

For many forms of identifiers validation consists of authenticating user credentials. For example, most online social network use the OpenId Connect standard which is based on the OAuth2 standard, in these cases the identity can be validated using the OAuth2 protocol with the user supplying login credentials. Other approaches to validation include the user making a public post on a social media site that provides a token or one time unique content that the validation service can then read from the site. An email identifier can be validated by sending the user an email with a link that includes an embedded token. Clicking on the link posts to a web page thereby validating that the user logged into the web page is the same user that received the email. Other forms of identity validation such as government ids require trusted third parties or other mechanisms for validation. Several identity services provide validation of these type of credentials that could be integrated into Open Reputation.

Many so called reputation systems provide only identity validation, indeed familiarity with other reputation systems may lead one to conclude that reputation is primarily identity validation, but identity validation is merely one component of a reputation. Open Reputation views reputation as contextualized behavior based predictor of future behavior. Identity disclosure is one form of behavior. Other behaviors within a context may be more relevant and valuable predictors than merely disclosure. Indeed behavior based reputation is valid even if the identity is pseudonymous as long as the reputation is uniquely cryptographically associated with the identity. In other words, a behavior based reputation has predictive value without disclosure. This is relevant to class based reputation (see the section on group privacy below).

Consider for example the science of word printing where an author may be identified by patterns of common words in their writing despite the author's attempts at disguising their style of writing. Behavior based reputation provides a "behavior print" that qualifies an identity's reputation for a given context. In some use cases for reputation, identity disclosure is required, in others it is optional. In many cases behavior plus disclosure is a better predictor which should be reflected in a higher reputation score.

1.5 Identity Graph

In many cases, when generating a repute, the precise identity of the associated reputable entity (reputee) is not known at the moment. The identity may have to be associated later. For example, the only identifier that may be available is a tertiary identifier such as a URI. At some later time, it may be possible to associate the repute with the true reputee. But in order to bootstrap or initiate the repute processing, an identity must be assigned. In other words identity assignment needs to be flexible and dynamic. Open Reputation supports flexible and dynamic identity assignment using a directed identity graph wherein composite or grouped identities can be formed by linking or associating multiple identities into one. This allows both the association and dissociation of identities one from another. Consequently, a given repute can be assigned a dynamically generated identity that is later associated via a link in the identity graph with the identity of the actual reputee. If for some reason the association is later shown to be erroneous the identities can be easily dissociated by removing the link. Using a dynamic identity graph was chosen over the more arduous approach of merging and unmerging identities. The identity graph means that Open Reputation can always make associations between reputes and reputees as needed to facilitate immediate processing and then later reassociate as more information is provided about the targeted reputee. This approach also allows for more complex relationships between reputations as a composite identity for a group could be generated from the reputations of each individual identity in the group.

There are two primary ways reputes may be associated with a given identity/reputee, one is a direct and inseparable association where the reputes reference the cryptonym of the reputee explicitly. The other is an indirect and separable association where the reputes include the cryptonym of a different subordinate reputee that is linked to the given reputee via the identity graph. The linked reputes can be separated by changing the link. In some cases, group reputation may be reflexive, that is, the reputation of a member of the group may contribute to the reputation of the group and likewise the reputation of the group may contribute to the reputation of each of its members. Consequently, the identity graph may be traversed in either direction depending on the objectives of a given reputation algorithm. The nodes of the graph are identities. The edges are directed association links. The basic structure of the identity graph is shown in the following diagram (see Fig. 1.2.1). Major links encode associations from a member up to a grouped identity. Minor links encode associations from a grouped identity down to a member identity.

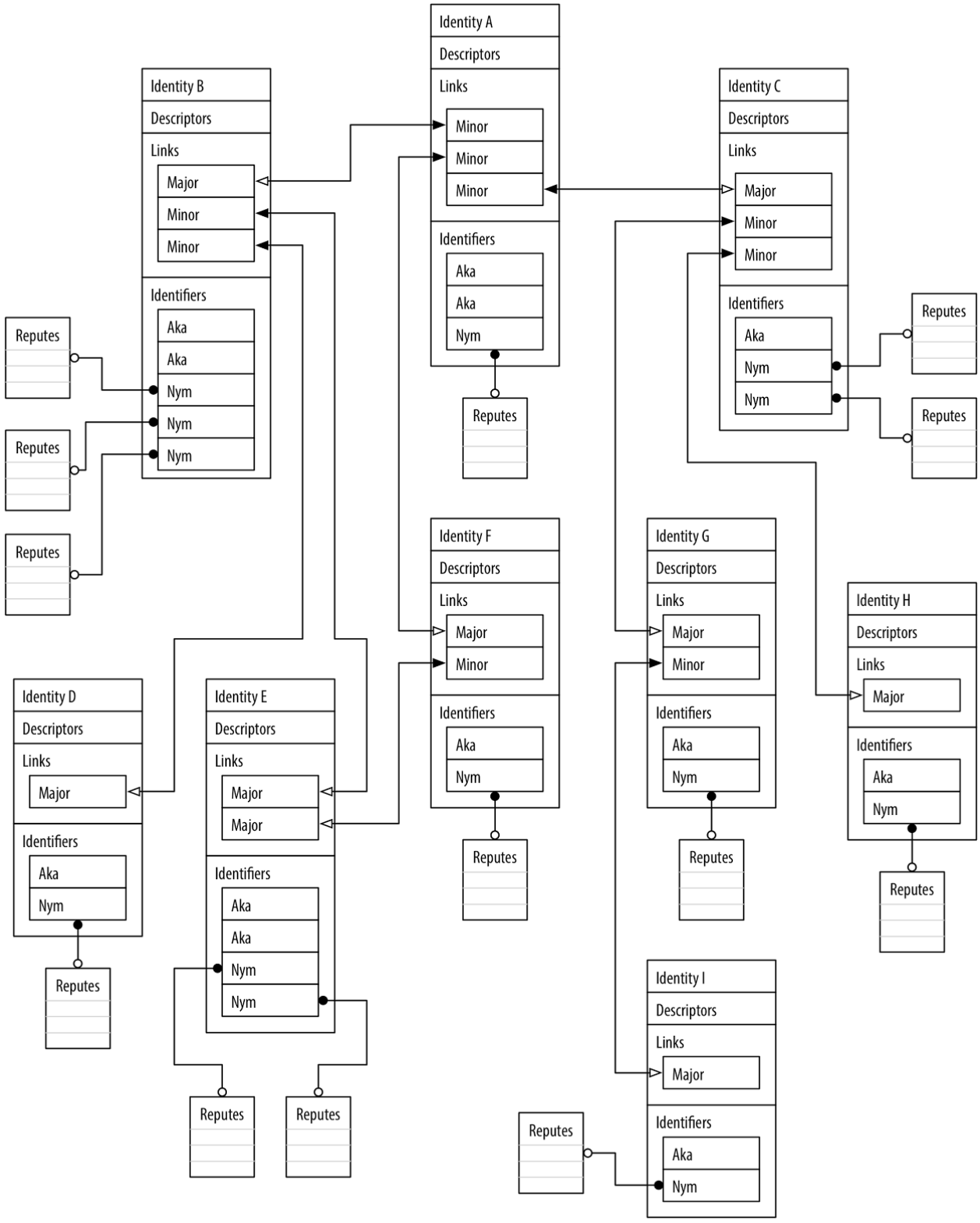

Figure 1.2.1 Identity Graph

The next diagram (see Fig. 1.2.2) shows a simplified version of the same identity graph.

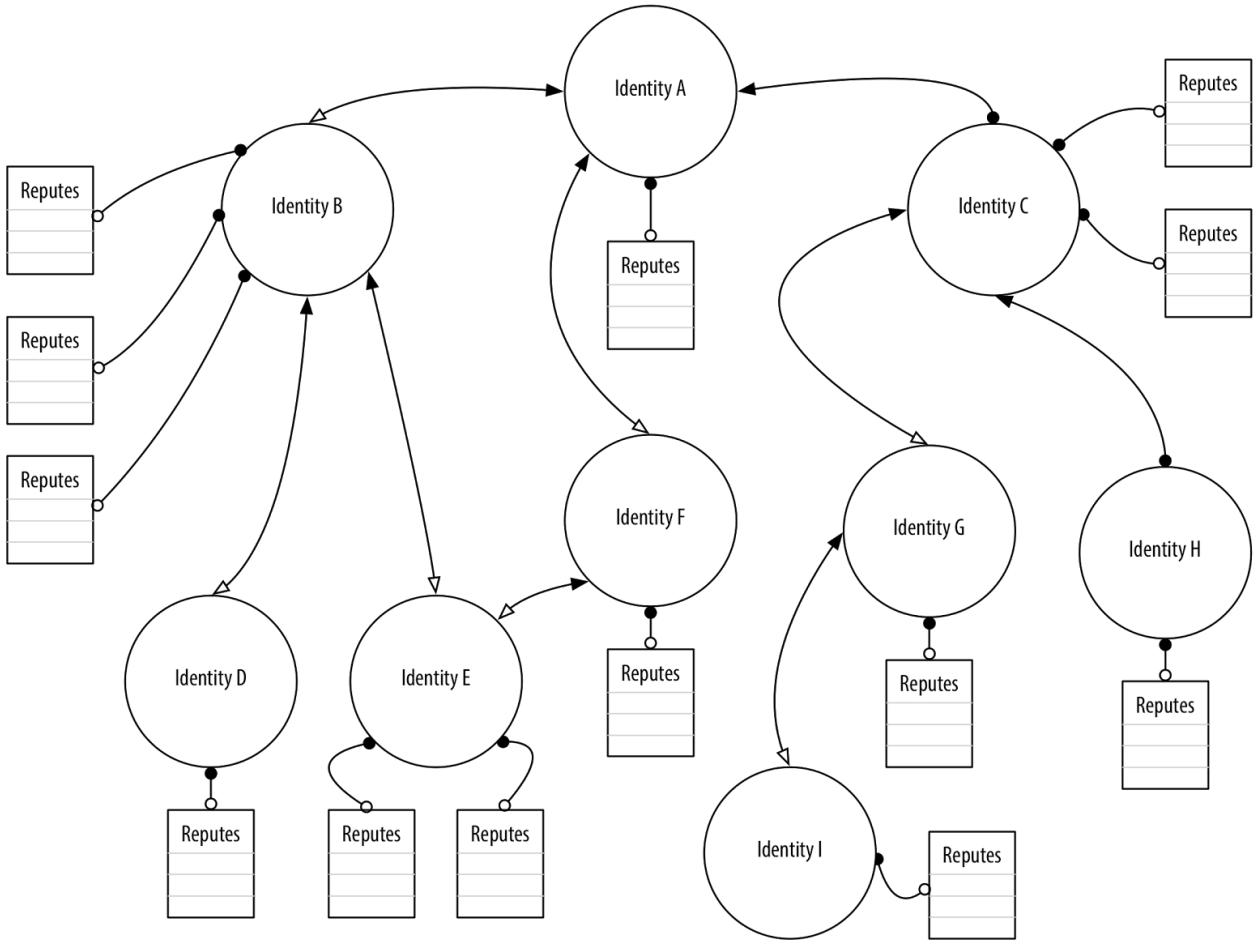

Figure 1.2.2 Simplified Identity Graph

The next diagram (see Fig. 1.2.3) shows upwards association, that is, when minor identity is a member of a larger grouped major identity, then in some contexts the minor identity's reputes could be indirectly associated with or contribute to the major identities reputation. An example, would be players on a sports team. The reputable abilities of the team members could be used to generate an overall ability reputation for the team. In this diagram, the blue shaded objects indicate the identities and their associated reputes that are both directly and indirectly associated with Identity B when moving upwardly in the graph.

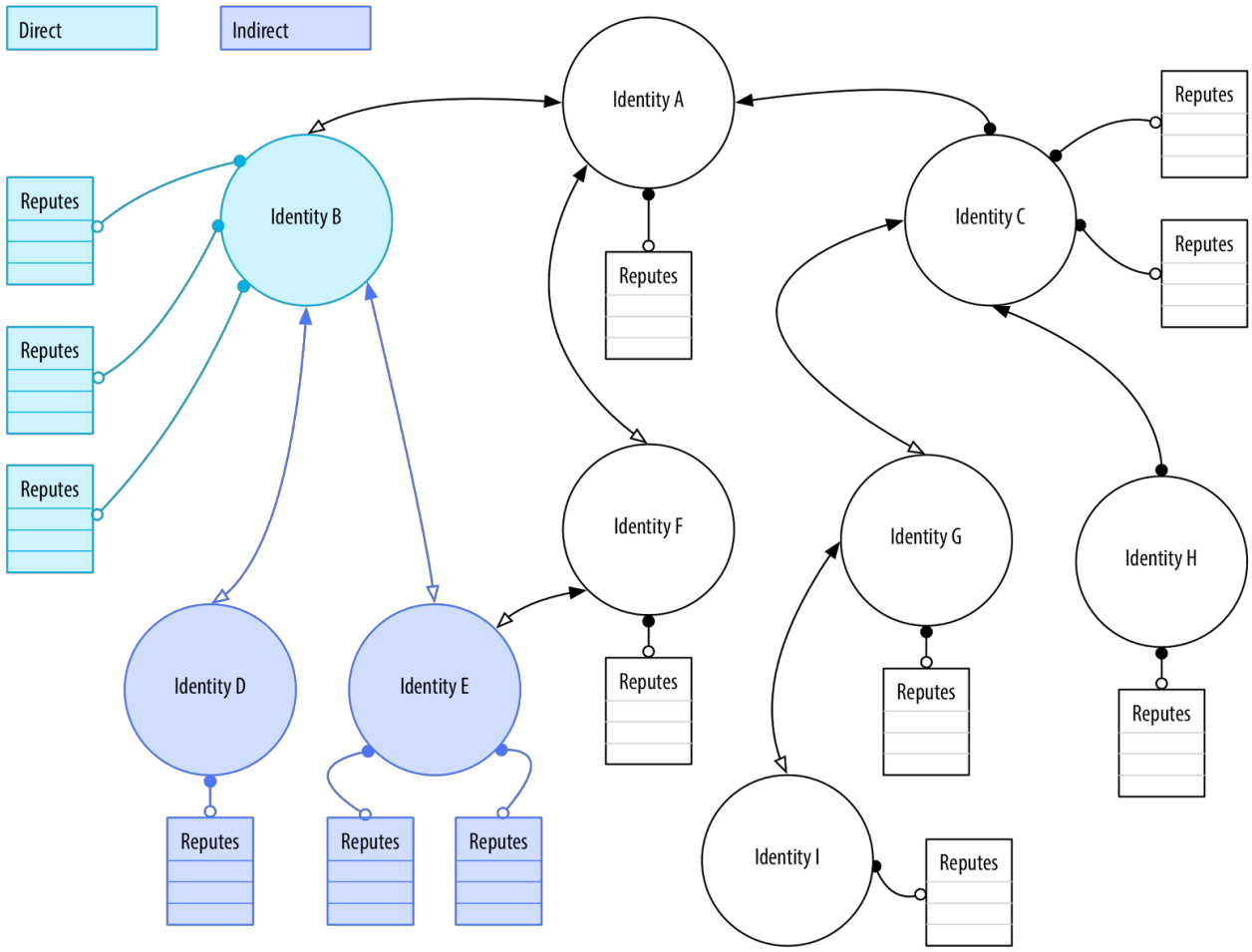

Figure 1.2.3 Reputes Upwardly Associated With Identity B

The next diagram (see Fig. 1.2.4) shows downwards association, that is, when a minor identity is associated with a major identity, then is some contexts the reputes associated with direct behavior of the major identity may reflect downward onto the reputation of the minor identity. For example, if a sports team wins a championship, the individual team members can be reputed at champion players. This diagram shows that the direct reputes are indirectly reflected down but not the indirect reputes. In other cases the indirect reputes might also be reflected down when it is justified by the nature of the associations. The objects shaded in pink show the indirect-direct reputes downwardly associated with Identity I.

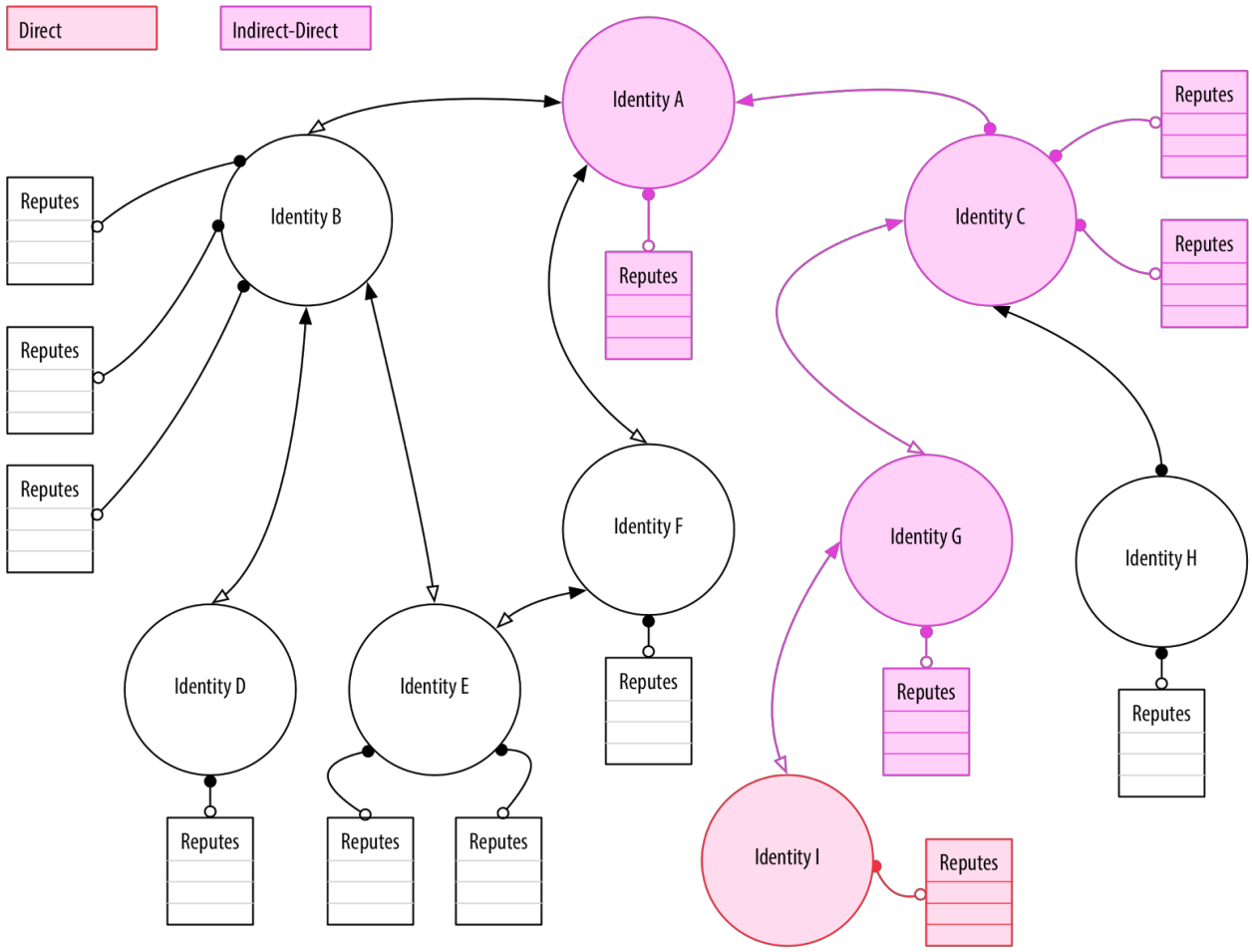

Figure 1.2.4 Reputes Downwardly Associated with Identity I

There may be other contexts where all the reputes both indirect and direct and incident and reflected may contribute, in this case the graph could be cyclic and the reputation algorithm would need to employ a weighting algorithm that allowed convergence of the reputation scoring.

1.6 Serialization Format

Enabling the transmission, exchange, and storage of reputes in a safe and secure manner is an important part of repute management. This requires various degrees of cryptographic authentication and encryption. Therefore the repute data format needs to be serializable. Also important is a high degree of flexibility and power of expressiveness so that reputes can be used in various contexts with differing data elements in a future proof way. The serialization data format should have broad cross-platform support. The format should have a human readable version that is not overly verbose and simple enough for convenient adoption. One form of smart contract is called a Riccardian contract. One attribute of a Riccardian contract is that it human readable . The requirement for a repute serialization format is a format that exhibits an advantageous combination of the following characteristics:

- Flexible/Nested/Hierarchical
- Ubiquitous

- Simple
- Human Readable
- Compact

The primary serialization format selected for reputes is JSON (JavaScript Object Notation). JSON exhibits all the desired characteristics above. There may be applications where data compression or some other attribute is more important, and in such cases an application specific format may be used, but facilities should be provided to convert the application specific format to/from JSON. For example msgpack is a binary serialization format that is flexible, nested, hierarchical, ubiquitous, compact, and simple but is not human readable.

As per web standards UTF-8 will be used as the unicode character encoding. One advantage of UTF-8 is that it is self synchronizing to ASCII safe single byte characters. This means that a parser may first partition UTF-8 encoded text using ascii character delimiters for boundaries and then decode the partitions. Also UTF-8 does not require a byte order mark.

It might also makes sense to prepend a header with lengths when multiple JSON blocks and signatures for a repute, reputage and/or reputet are included in a particular transmission stream or storage file.

In many cases the transmission or storage of a repute may have associated with it one or more cryptographic signatures for authentication. In these cases a signature may follow the JSON serialized data. This signature block is not technically JSON. In addition the transmission or storage of a repute may include its associated reputage and signature block. Multiple JSON entities and non JSON entities might therefore be included in the same transmission or file. Multiple JSON entities in the same serialized stream are not typically supported by many JSON parsers in default mode. Thus a particular transmission or storage medium might need to include a header that indicates the boundaries of the JSON entities to be sent to the JSON parser or the JSON parser would need to be run in a raw mode that allows for additional characters following a valid JSON entity.

1.7 Date-Time Stamps

Date-time stamps in reputes should use the ISO-8601 format. For example, 2007-04-05T14:30:20.123456Z provides a date-time to microsecond accuracy in ISO-8601 format.

1.8 Cryptography

This specification uses elliptic curve cryptography (ECC) almost exclusively. There are several reasons for this choice.

• Space efficient. ECC is the most space efficient. 32 byte ECC keys and 64 byte ECC signatures are still highly secure. Other protocols require keys and signatures that are several times larger for the same degree of security.

• Speed. ECC algorithms for authentication and encryption are some of the fastest for a given level of security.

• Open Source. The NaCL networking and cryptographic library is a widely used, well supported, and well-vetted open-source library for elliptic curve cryptography. The LibSodium implementation of NaCL has been ported to all the major operating systems and bindings to

LibSodium exist for all the major programming languages. A highly portable compact open source implementation of NaCL called TweetNaCL, is also available. For python, the open source LibNaCL is the preferred binding.

• Private. LibSodium does not have backdoors for government agencies.

1.8.1 Signatures

Each nym is the public side of an Ed25519 public/private key pair. Ed25519 is a form of the Edwards-curve Digital Signature Algorithm (EdDSA). Ed25519 keys are 32 bytes long in binary and 64 characters long in hexadecimal notation.

The Ed25519 keys are used to create Signatures. Signatures created using Ed25519 are 64 bytes long in binary and are 132 bytes in hex notation.

1.8.2 Encryption

Encryption is performed using the Curve25519 algorithm. Curve25519 uses public/private key pairs. Curve25519 keys are 32 bytes long in binary and 64 bytes long in hexadecimal notation. Curve25519 keys are not the same as Ed25519 keys but the two are birationally equivalent. This means that the same Ed25519 key pair could be used for encryption once it has been converted to a Curve25519 key pair.

In Curve25519, each side of a encrypt/decrypt transaction generates a shared key that is formed from the near side's private key and the far side's public key. The shared keys generated on each side are equivalent such that a message encrypted by one side can be decrypted by the other side. For example, suppose that the two sides are denoted A and B. If PA/pA is the Public/private key pair for side A and PB/pB is the Public/private key pair for side B, then the shared key SA formed from (PB, pA) is equivalent to shared key SB formed from (PA, pB), that is SA≈SB. Thus if side A encrypts with SA then side B can decrypt with SB and vice-versa.

2 Trustworthy Reputations

2.1 Overview

In Open Reputation, a reputation is an indicator derived from the processing of reputational events that provides a contextual evaluation or prediction about the future behavior of a reputable entity. Open reputation is based on the premise that the underlying processing used to generate a reputation should be open and transparent, not siloed in a closed system, that is, reputations should be trustworthy. One way to achieve trustworthiness is to enable verification by interested parties of the associated processing steps, or in other words, use a transparent publicly verifiable methodology.

A diffused trust process model achieves trustworthy verification by distributing the trust amongst multiple entities in a way that minimizes the likelihood of fraudulent collusion. One diffused trust process model consists of a distributed consensus algorithm where members of a voting pool govern the acceptance and validation of entries in a shared public ledger. The ledger entries are typically transactions. This diffused trust processing model is more commonly known as blockchain. Blockchain refers to how the ledger is constructed, as a chain of transactions blocks that are linked by cryptographic hashes. The key to this methodology is that the trust is diffused across all the members of the voting (consensus) pool so that the

actions of some do not invalidate the group consensus. As a simplistic generalization, a shared public ledger is trustworthy if the majority of the members in the pool are not colluding to defraud. Some call this approach a trustless model, this is technically incorrect, as trust is still placed in the consensus pool. This trust, however, is not concentrated in a single entity but is diffused across all the members of the consensus pool. The trust is that a majority of members do not misbehave. Various consensus schemes have been developed to ensure this condition is met. These include proof-of-work (POW), proof-of-stake (POS), and Byzantine agreement (BA).

Because almost any activity could be a source for a repute, the potential volume of reputational events and associated reputable entities may be very large. Moreover, reputation could be a qualifier on many types of transactions, many of which may need to be completed in seconds or even milliseconds. These potentialities require that the Open Reputation system be capable of supporting large numbers of low latency transactions. Therefore, in addition to trustworthiness, a requirement for Open Reputation is that the reputation processing support a large number of low latency transactions with the capability to support reputations for a large number of entities in a large number of contexts.

Unfortunately, many of the most popular distributed consensus algorithms have inadequately low transaction rates and inadequately high latency per transaction. A further complication is that many distributed consensus algorithms support at most a single ledger. This is problematic for Open Reputation where the number of reputees will be very large. A single ledger would form an unwieldy bottleneck for for transactions and storage. Moreover, most applications of blockchains support only a single ledger as a mechanism to prevent double spending of a cryptocurrency or some other unique zero sum asset. Because reputes do not need to be double-spend proof, there is not a limitation on the number of ledgers. Multiple ledgers, would enable better partitioning of computing resources. Indeed a more appropriate data structure for the distributed consensus pool to manage is a database not a ledger. A ledger is still needed to provide a serialized audit trail, but the ledger in this case merely needs to include hash proofs of the database transactions (reputes) not the transaction details.

Various distributed consensus algorithms have been developed with different performance characteristics. This section will compare the characteristics of distributed consensus algorithms and propose a candidate set of algorithms that have the potential to meet the requirements of Open Reputation for further development as part of the Open Reputation framework.

2.2 Distributed Consensus

This section briefly describes the three major approaches to distributed consensus, as follows: proof of work (POW), proof of stake (POS) and Byzantine agreement (BA). All three are solutions to what is commonly referred to as the Byzantine consensus decision problem where a valid consensus is arrived at despite some fraction of the participants (nodes) in the consensus decision behaving in a malicious and/or erroneous manner. In other words, the consensus decision making is subject to Byzantine faults. The word Byzantine is used by analogy to a group of generals engaged in war in the Byzantine era who need to come to consensus on whether or not to jointly attack a common enemy. Historically Byzantine generals were prone to traitorous acts such as one general pretending to agree to join an attack but back out thereby resulting in the defeat of the other generals.

A consensus algorithm that solves the Byzantine consensus problem in the presence of

malicious and/or erroneous behavior is called Byzantine fault tolerant (BFT). There are two features of any BFT consensus solution. The first one is ensuring liveliness, that is, the non-faulty nodes make a joint decision in finite time. The second one is ensuring safety, that is, the non-faulty nodes all make the same joint decision. All the solutions require that a majority or supermajority, of the nodes be non-faulty. The problem is usually couched as a problem of distributed replication of a sequence of states, that is, the non-faulty nodes will all replicate the same sequence of states (decisions). For example, the sequence of transactions entered into a shared public ledger or shared public database is the sequence of states.

2.2.1 Proof of Work (POW)

A conceptual simple solution to the Byzantine consensus problem is called proof of work or POW. This approach was popularized by Bitcoin. Several other crypto-currencies also use a POW algorithm. Each POW protocol has unique features. Only some of the core features of of POW are described here. More detail can be found in the references. A POW protocol relies on a consensus pool of members that manage a shared resource such as a public ledger. The pool has a leader who determines what transactions get written to the shared ledger. Each member of the consensus pool can compete to be the leader of the pool during a subsequent but finite time slot. The set of transactions that the leader adds to the ledger during that leader's time slot is called a block. The blocks are ordered. Each block includes a link to the previous block as well as a cryptographic hash of the previous block. The set of ordered linked blocks forms what is called a blockchain. The ledger is a block chain of transactions. Each member of the consensus pool has a copy of the ledger. A single computer might host copies of multiple ledgers.

A set of candidate members who wish to be the leader for the next block may compete by solving a computationally costly cryptographic problem. The solution is the proof of work (POW). The first candidate to solve the problem broadcasts its solution to the other members, both candidate and non-candidate, and assumes the role of leader by writing transactions to a new block. Verifying the solution is not computationally costly. So the non-faulty members can quickly verify that a given candidates POW is correct and concede to the new leader. The candidates then begin working on the next cryptographic problem. The cryptographic problems are designed such that the amount of time required on average to solve them is about the same. A winner becomes the leader until a new winner solves the next cryptographic problem.

Starting at any block, the contents of the previous block can be validated by recomputing and comparing the hash of the previous block with the hash stored in the current block. If the hashes match then the contents of the previous block have not been tampered with. This validation process can then be applied to the block before the previous block and so forth until the very first or genesis block is reached. To verify a transaction, a given node needs to validate that a majority of the copies of the block chain are identical.

Because of delays and faults it is possible that multiple candidates might believe they are the first to solve the cryptographic problem and are therefore entitled to write the next block of the ledger. When this happens the block chain is forked and different sets of members may choose one of the other forks of the ledger as the valid one. Likewise more blocks may be added to each of the forks. Eventually one of the chains will become longer than the other due to variations in the time its takes for subsequent competitions to complete on each block chain. Members determine which is the valid block chain by selecting the longest one. This means

that a new transaction cannot be validated until several blocks are added to the ledger after the block that contains the transaction. Trust is also required in the creation of the genesis block and diffused across the members of the consensus pool. As long as a majority of the candidates are non-faulty (for agreement on who gets to write the next block) and a majority of the members are non-faulty (for agreement on which copies of the ledger are valid), a valid ledger can be maintained.

The major advantage of POW is that the algorithm is conceptually simple and easy to implement. The primary disadvantages of POW is that the transaction rate is relatively low because only the leader can write new transactions during any time slot and the transaction latency is relatively high because it may take several time slots to eliminate forks. Take for example, Bitcoin, its current transaction rate is less than 10 transactions per second. Each time slot is 10 minutes long. Typically 6 time slots are needed to resolve any forks. Thus the average latency is an hour. It may be problematic to wait an hour for a transaction to complete. Imaging trying to log into an account but having to wait up to an hour for the login to be accepted.. Another disadvantage is that the block chain keeps increasing in size. Over time the size of the block chain would exceed the typical memory and storage capabilities of the average host. For example, Bitcoin's block chain is about 25 GB in size. Moreover, because POW uses lots of computational power, it is energy intensive. Only the most powerful candidates have any chance of winning. This means that the trust about which candidate is the winner gets diffused over fewer and fewer members making it more likely that a faulty set of candidates might become the majority.

2.2.2 Proof of Stake (POS)

Because POW blockchain consensus protocols can exhibit low transaction rates, high transaction latency, high energy cost, and a decrease in diffusiveness over time, an alternative solution to the Byzantine consensus problem has been developed called proof of stake (POS). Many different POS protocols have been developed with varying features. Only the core features of POS are described here. See the references for more details. A POS system also uses a form of POW to elect a leader who writes the next block of transactions. The main distinction is that in POS the amount of work a candidate has to perform is a function of the stake of that candidate. The higher the stake the less work. Thus POS could described as asymmetric or non-uniform POW. Examples of stake might be the time since that candidate was last a leader or might be an amount of currency placed in escrow.

Because the work is a function of the stake, it is possible to predict with some accuracy which candidate will become the next leader based on the size of that candidate's stake. The POW for members with high stake is trivial and can be solved in a short period of time. Any member can validate the stake of any other member as long as the majority of members are non-faulty. Based on this, any member can also quickly validate which candidate completed its stake dependent proof of work first. Thus POS uses POW but with each competitor having a different amount of work. This means that the work in POS for the eventual winner of each time slot is on average much less than that for uniform POW and can therefore be completed in much less time. POS in exchange for a small increase in complexity, achieves higher transaction rates with lower latencies than POW.

Another use for POS is to provide side chains to a POW system, where the POS blockchain is anchored to the POW blockchain. The side chain can have relatively high transaction rates and low latencies compared to the anchoring POW chain but may still leverage the anchoring

chain's security.

2.2.3 Byzantine Agreement (BA)

Byzantine agreement (BA) protocol solutions to the consensus problem have been available much longer than either POW or POS. The first practical BA protocol was released in 1999. It has been an active area of academic research since then. The drawback to BA protocols is that they are much more difficult to implement than either POW or POS. The main advantage of BA protocols is speed. Also BA protocols more easily support multiple simultaneous ledgers in cases where different asset classes need to be tracked. Thus BA protocols exhibit scalability advantages. Despite their complexity, as the demand for block chain solutions has risen, so has the interest in BA protocols for their superior performance. For example, the second largest crypto currency, Ripple, is a BA type protocol. Ripple can support over ten thousand transactions per second and latencies of one to three seconds.

It is beyond the scope of this paper to describe how a BA protocol works in any detail, but in simple terms a BA agreement protocol makes decisions by majority votes of the consensus pool members. These consensus pools may also have a leader who decides which transactions get written to the block chain. Leader election is performed by vote of the consensus pool. Multiple rounds of voting may be required. The process of performing votes that are Byzantine fault tolerant (BFA) can be complicated but not time consuming, and are primarily limited by network latencies, not arbitrarily complex POW tasks. In BA, validation of transactions in the block chain is also performed by voting. As long as a supermajority of the members are non-faulty a valid blockchain may be maintained. Management of new members joining a pool and auditing of member behavior are other facets of BA protocol behavior that ensure the supermajority are non-faulty.

2.3 Applications of Distributed Consensus

A distributed consensus algorithm can be used to manage many types of shared information not just block chains. Indeed the first applications of BA protocols were for managing database transactions. Other applications include state machines for complex processes or robotics.

2.4 Open Reputation Capable Repute Processing Protocols

It is expected that Open Reputation will need to support very high transaction rates and low latencies as a result of the large volumes of reputes that may be processed. Consequently, Byzantine agreement is the most promising family of consensus algorithms for Open Reputation.

Because reputes are specific to the associated reputees/identities, each reputee's reputes may be considered a different asset class. Therefore repute ledgers could be partitioned by reputee for auditing purposes. Support of reputation processing would best be provided by a database. A distributed hash table (DHT) database could be partitioned such that multiple consensus pools governed portions of the database. The database should be immutable so that any updates can be tracked with Merkle tree hashes anchored to a blockchain. Multiple ledgers and database partitions would allow better load balancing of the transactions amongst the computing resources.

Based on our analysis, the most promising BA protocols for Open Reputation are RBFT (Redundant Byzantine Fault Tolerant), SCP (Stellar Consensus Protocol), and Ripple. A version

of one of these protocols appropriately customized for reputation could serve as the basis for trustworthy repute transaction processing. The customizations needed to support Open Reputation are an active focus of research and development.

2.4 Chainmail

Distributed consensus algorithms enable transactions without the need for centralized trusted institutions and governments to monitor, moderate, and police those transactions. Distributed consensus algorithms replace centralized trust with decentralized or diffusive trust. The principal vulnerability, however, shared by all the distributed consensus algorithms is that the validity of the ledger (blockchain) will be compromised if a majority of the nodes hosting a distributed consensus ledger (blockchain) collude to defraud. Authenticity of a transaction in the shared public ledger is based on verifying with a majority of the nodes that each node's copy of the ledger identically contains the transaction in question. Should a majority of the nodes collude then they could forge a fraudulent ledger that would have to be accepted as the authentic ledger because a majority of nodes verify it as such. This vulnerability is a potentially catastrophic failure, because the complete history of the blockchain could be rewritten. Indeed, because of this vulnerability, a better analogy to shared public ledgers is not a block chain but a block knitting. In a knitting the loops in each row are held in place by linking with loops in a prior row but the whole knitting can be unraveled by pulling out the thread. To prevent unraveling a knitting must be bound or knotted. By analogy a majority collusion exploit is like unraveling a knitting. The unravel is a common mode of failure for all the transaction blocks in the ledger.

Some blockchain protocols add a checkpointing capability that makes older sections of the ledger immutable so that only a portion of the ledger would vulnerable to a majority collusion exploit. This is like adding a knot to the knitting. This may be a reasonable approach to limiting the amount of damage from a majority collusion exploit for a single ledger, especially where that ledger has a long history and a large number of nodes hosting it with a high degree of diffusive trust. But requiring that a large number of nodes be dedicated to hosting a single block chain limits the scalability of the system.

An alternative approach is to use one blockchain to anchor multiple-side chains. The genesis block of each side chain includes a reference to an anchor transaction in the anchoring chain. The anchoring chain supports the anchoring transactions and can run at both a lower transaction rate and a higher latency and also may have a higher degree of diffusive trust than the side chains. The anchor transactions, in a sense, act like knots or checkpoints for the side chains. Validating transactions in the side chain includes validating the anchor transaction in the anchor chain. If a majority exploit is successful on a side chain it can only unravel the side chain up to the anchoring transaction. As long as the nodes hosting the anchoring chain are diverse relative to the side chain, further unraveling would require a new majority exploit of the anchoring chain. This removes the common failure mode. The advantage of this approach is that the transaction load can be distributed across the side chains thereby increasing the overall transaction rate. The side chains benefit from the increased security of the anchor chain.

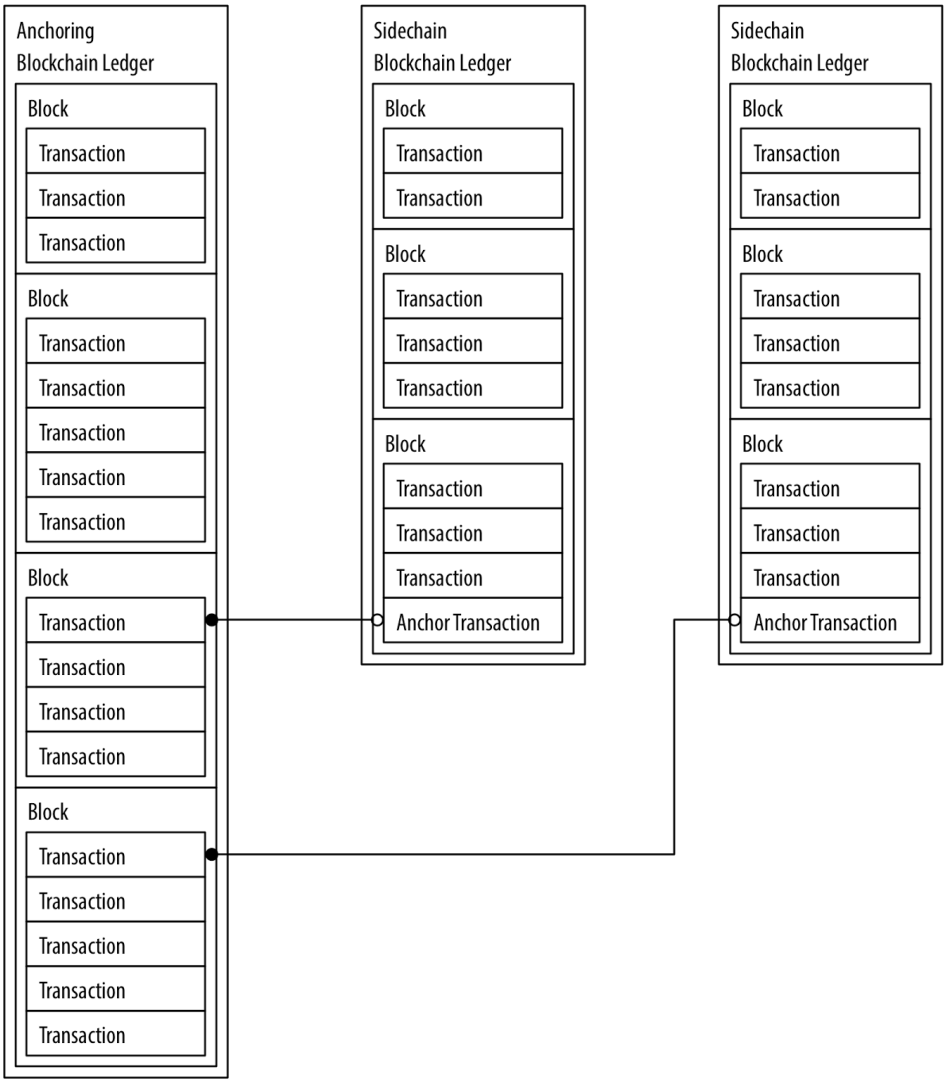

Fig. 2.4.1 Side Chain

Open Reputation extends this concept with a novel approach to balancing scalability and trustworthiness of shared public ledgers that is called chainmail or mesh-anchoring. In a mesh-anchored system each ledger (blockchain) is anchored to two or more other ledgers. The genesis block of each chain includes these multiple anchoring transactions. By analogy, instead of one knot to prevent unraveling, multiple knots are provided to prevent unraveling. Validating the transactions in a given chain includes validating the anchoring transactions in the anchoring chains. The anchors can also use an *m* of *n* scheme similar to multiple cryptographic signatures, where validating the transactions in a given chain requires validating only *m* of the *n* anchors.

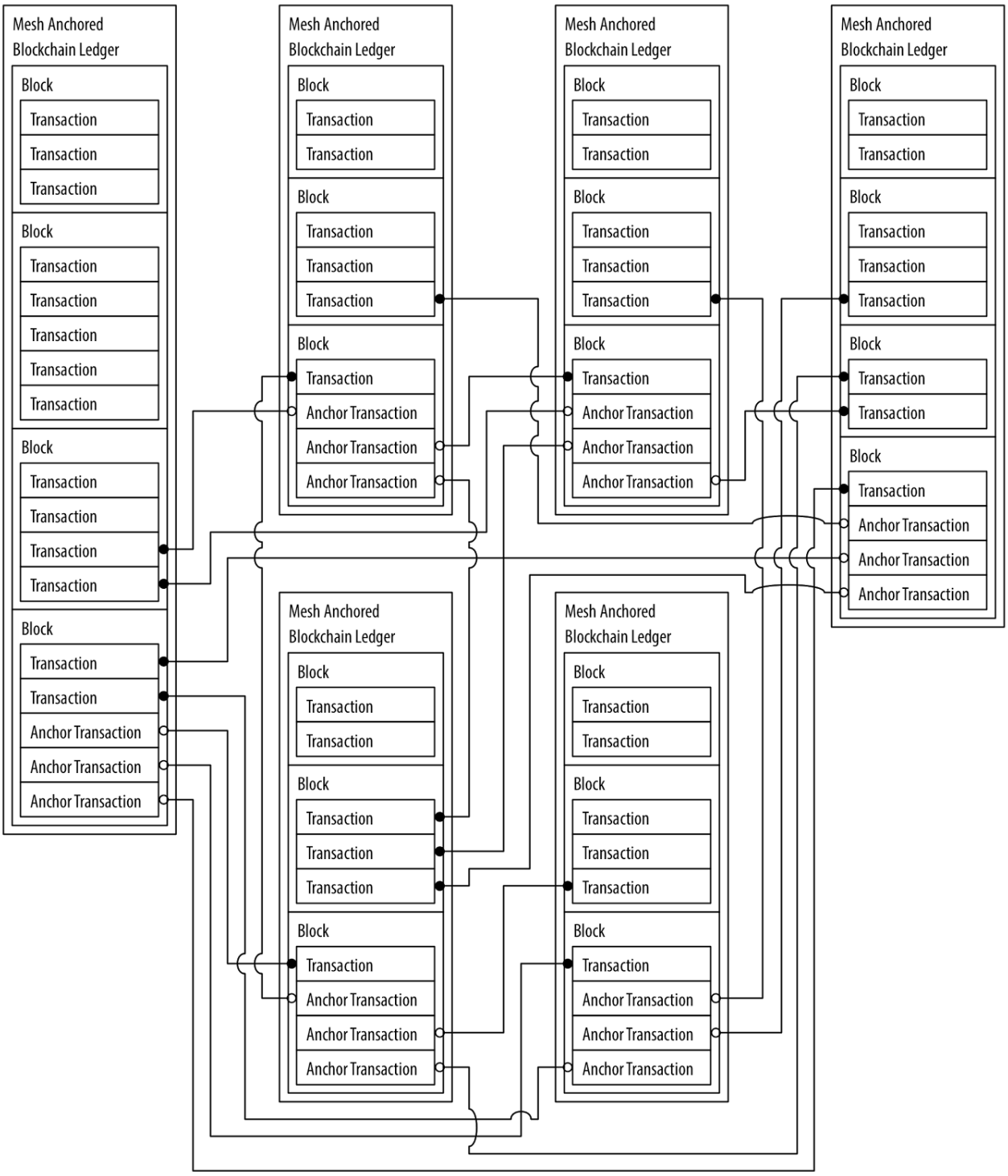

Fig. 2.4.2 Mesh Anchoring

A majority exploit of a given chain could unravel that chain up to its anchors. As long as the nodes hosting the different anchor chains are mutually diverse with one another and with the anchored chain, further unraveling would require new simultaneous majority exploits of all the anchor chains. This means that a very high degree of diffusive trust can be obtained even if any individual anchor chain has a relatively small degree of diffusive trust. Consequently, each chain, independently, can have fewer host nodes with lower cost, higher transactions rates,

and lower latency. This allows better partitioning of the set of host nodes across all the chains.

Furthermore, each chain can knot off a section by periodically re-anchoring itself to multiple other chains. If the re-anchors are diverse from prior anchors then even more simultaneous majority exploits would be required to fully unravel the chain. The re-anchors also limit the length of any section of a given chain that is vulnerable to a single majority exploit. The combination of mesh-anchors and re-anchors, therefore, creates a highly redundant system of diffusive trust. Not only does it exponentially increase the difficulty of a full unraveling of a given chain it also exponentially lowers the gain that can be obtained from any partial unraveling. Thus both vulnerability and susceptibility to exploit are simultaneously reduced. The following diagram shows a mesh anchored set of block chains with re-anchors.

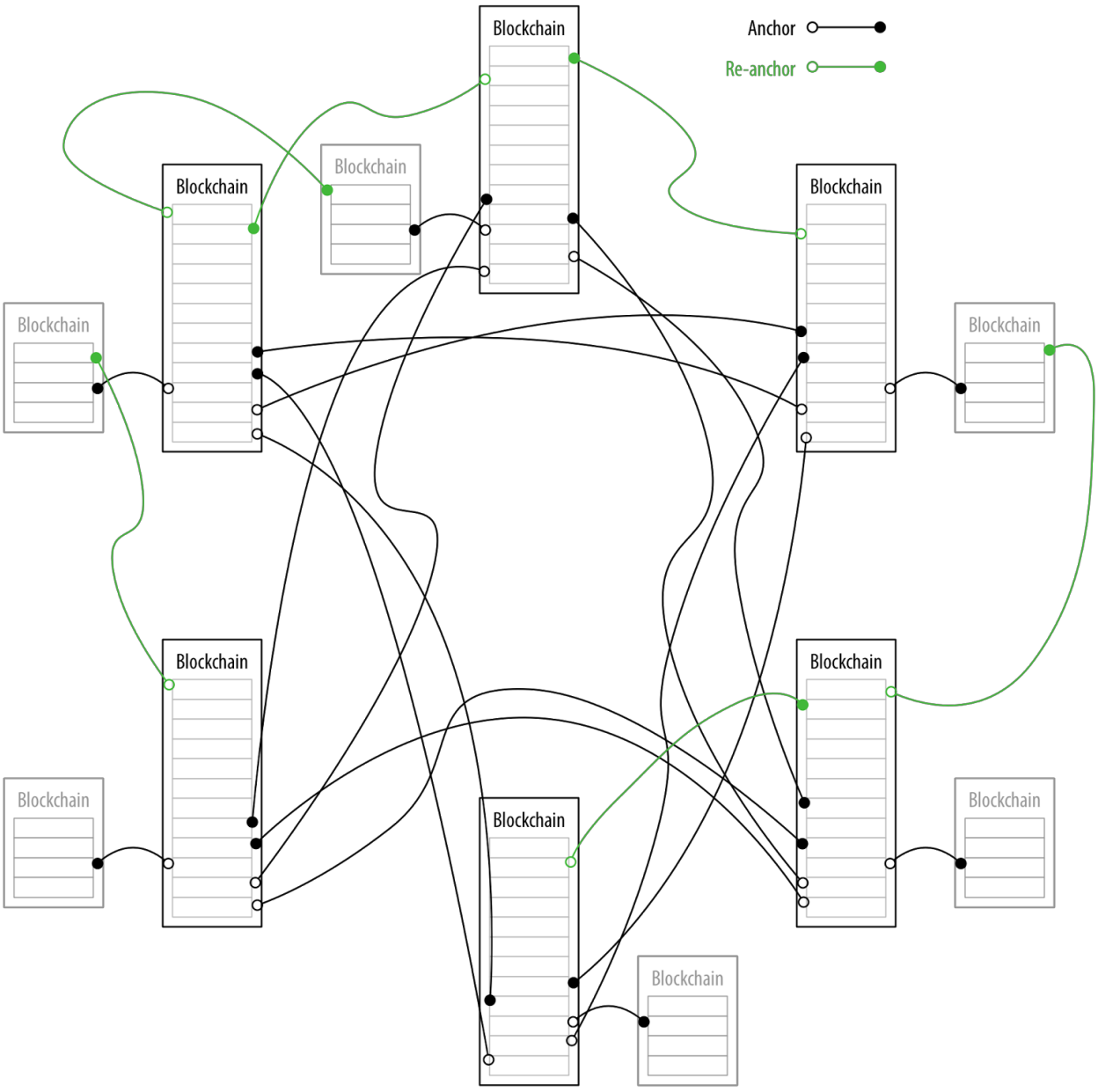

Fig. 2.4.3 Re-anchoring

The advantage of chainmail or mesh anchoring is that the tradeoffs between trustworthiness,

transaction rate, latency, and computation resources can now be much more flexibly fine tuned. This is a key enabling feature for a diffused trust system that must scale to high volumes of transactions with low latency.

2.5 Identity Ledger

Human machine interfaces to Open Reputation might benefit from unique human friendly identifiers for the reputable entities (reputees) associated with the reputes. One way to achieve this is with a distributed consensus ledger to ensure name uniqueness. NameCoin/OpenName is one example of a block chain protocol that provides unique human friendly identifiers. Because reputees may have to be generated on the fly as reputes are generated, the potentially high transactions rate and low latency requirements for repute generation likewise induce a relatively high transaction rate and low latency requirement for unique human friendly identifier generation. To the extent that an existing unique identifier blockchain is compatible with Open Reputation there is less need to create another one. The Open Reputation identity block chain could be a side chain to another like NameCoin. Another approach is for each reputery to reserve a name space and then create unique names by convention under the name space. This would allow batch creation of identifiers with a minimum of explicit transactions on the name block chain. Otherwise, the same block chain infrastructure that supports the repute database could be used to support a unique identifier blockchain. Mesh anchoring of block-chains does not require that the chains all host the same types of transactions. Thus an identity name chain could be anchored to repute block chains as well as NameCoin.

2.6 Self Certifying Identifiers

Another way of providing more human friendly identifiers is to use self-certifying identifiers. A self-certifying identifier encodes a hash of the location of the information associated with the identifier with a public key for the provider of the information at the given location. When another entity makes a request to the provider for the information using the self-certifying identifier, the provider signs the response with the associated private key. The requester can then verify the response with the public key provided in the identifier. This scheme enables self-contained certification and validation. Because the public/private key pair is unique, the self-certified identifier is also unique.

In the case of Open Reputation, the information is an identity. An alias for the identity could be formed from a human friendly name with the cryptonym. The cryptonym must be unique so the alias is also unique. Human interface software could hide the details of the validation from the user. This provides an intermediate approach. The self-certifying identifier has a human readable component and a hash component. Duplicates of the human readable will only validate to one identity. Thus even though a query using only the human readable portion of the identity may return multiple matching identities, only one would validate the hash portion.

3 Open Reputation Cred Coupons

Distributed consensus pools require host nodes. One way to pay for the costs of the host nodes is to embed a transaction fee in the distributed consensus protocol. The parties to any transaction pay the associated host nodes for that transaction. Typically, each distributed consensus protocol uses a cryptocurrency or crypto-coupon to make the transaction micropayments or redeemable credits. Likewise, Open Reputation proposes a custom

crypto-coupon in units of *Creds*. The purpose of *Creds* is to provide a convenient temporary medium of exchange for reimbursing or incentivizing transaction host nodes. A crypto-currency exchange allows the host nodes to exchange *Cred* coupons for more liquid cryptocurrencies. At some point *Creds* may have murrage capabilities within the Open Reputation ecosystem. *Creds* will either use a side chain like Factom to a cryptocurrency like BitCoin have a dedicated blockchain ledger that will be multiply anchored to other cryptocurrency blockchains such as BitCoin.

4 Protocols for the Secure Transmission of Reputes

4.1 Overview

One of the major weaknesses of many Internet protocols is that security in terms of authentication and encryption is not built in as a core capability but is an add-on, in many cases as an afterthought. This can make it difficult to ensure a desired degree of secure privacy when communicating. Many Internet standards use outdated security protocols that are vulnerable to exploits.

Open Reputation can include many heterogeneous distributed nodes connected in a reliable and scalable topology. This will include peer-to-peer communications channels. Moreover, Open Reputation communication protocols will need to scale up to support high volumes of reputes. Consequently, secure scalable protocols are needed for communication between the nodes.

Open Reputation includes secure transmission of data as a first order feature. This approach ensures a high degree of privacy when communicating, processing, and storing reputational events. The Open Reputation processing can include many heterogeneous distributed nodes connected in a reliable and scalable topology. This will include peer-to-peer communications channels. Moreover, Open Reputation communication protocols will need to scale up to support high volumes of reputes. Consequently, secure scalable protocols are needed for communication between the nodes.

The two protocols specified herein are RAET and HTTP+TLS. RAET (Reliable Asynchronous Event Transport) is a new modern scalable secure protocol. The disadvantage of RAET is that it is new and relatively unknown. On the other hand HTTP+TLS (Hyper Text Transfer Protocol + Transport Layer Security) is ubiquitous and very well known. The combination is also denoted HTTPS. The disadvantage of HTTPS is that is not as scalable and is vulnerable to exploit, primarily resulting from weaknesses in its centralized certificate authority model. Therefore, whenever practical RAET, is preferred; when not practical, HTTPS is acceptable. Primarily interfaces to external services and application will tend to be HTTPS, but internal communications between dedicated Open Reputation core components will be RAET. Over time the relative usage of RAET within Open Reputation is expected to increase.

One future solution to the centralized certificate vulnerability of HTTPS is DNSChain. With DNSChain, TLS certificates are entered into a blockchain that prevents man in the middle imposters of certificate authorities.

4.2 HTTP+TLS (HTTPS)

The ubiquity of HTTPS as a web standard means that it is often the only practical choice for interfacing various components of a distributed Internet based application. The TLS portion

provides an encrypted channel. Web browsers and most web services primarily use HTTPS. Older versions of TLS are called SSL (Secure Sockets Layer). All of the older versions of SSL and TLS have known exploits. This means that the latest version of TLS should be used whenever possible. Also whenever possible both client and server side certificate validation should be performed when forming an HTTPS channel in order to minimize vulnerabilities resulting from its trusted certificate mode. Typically many web applications perform server side validation only.

4.3 RAET

RAET is designed to provide secure reliable scalable asynchronous message/event transport over the Internet in a micro-threaded multi-process application framework that uses UDP/IP for interhost communication, the elliptic curve based NaCL/LibSodium for authentication and encryption, and the elliptic curve based CurveCP handshake for secure bootstrap. Elliptic curve cryptography provides increases in security with lower performance requirements relative to other approaches. The open source LibNaCL provides the API to CurveCP and NaCL/LibSodium.

Most if not all web services are based on HTTPS which is based on TCP/IP for transport. TCP/IP has much higher latency when compared to UDP and is therefore not well suited for the asynchronous nature of distributed event driven applications such as Open Reputation. This is primarily a result of the wire stream nature of TCP/IP connection setup, teardown, and failure detection. Fundamentally TCP/IP is optimized for sending large contiguous data streams but not many small asynchronous events or messages as is the case with reputes. While not a problem for small scale systems, the differences in the associated traffic characteristics can become problematic at scale. Layering HTTPS on top of TCP/IP adds even more latency and connection issues. Furthermore, TLS has problematic vulnerabilities due to its centralized trusted certificate model.

Because UDP/IP has lower latency and is connectionless, it scales better for asynchronous events like reputes. The drawback of bare UDP/IP is that it is not reliable. What is needed, therefore, is a tuned transport protocol that adds reliability to UDP/IP without sacrificing latency and scalability. A transactioned protocol is much more appropriate for providing reliable asynchronous event transport than a streaming protocol. RAET adds secure transactions to to UDP/IP.

LibSodium provides an open source elliptic curve cryptographic library with support for both authentication and encryption. The CurveCP protocol is based on LibSodium and provides a handshake protocol for bootstrapping secure network exchanges of information.

One nice feature of RAET is that it provides a convenient key exchange mechanism to facilitate the rapid deployment of large distributed applications in a protected environment where the exploit vectors are time and space constrained. The key exchange mechanism is optional as the keys can always be exchanged using an out of band mechanism.

RAET uses elliptic curve cryptography (ECC) exclusively. The reasons mirror those provided in section 1.5 for cryptonyms.

4.4 Other Protocols

Recently several other protocols have been developed for distributed Internet application communications. Many of these are still based on TCP/IP and suffer from many of the same

scalability problems relative to UDP/IP based protocols. One interesting protocol is TeleHash. It has optional support for UDP/IP. In our analysis, RAET is a more compactly designed protocol that provides the needed essential features. TeleHash is more ambitious. Also the initial developers of Open Reputation were more familiar with RAET, which facilitated adding features.

5 Archival Storage in the Fog or Cloud

The primary purpose of the repute database and ledgers is to provide trustworthy accounting of reputes used to generate reputations. The distributed consensus ledger provide a trusted point of reference for ascribing reputes and tracking them from source to target including any intermediaries, that is, they provide trustworthy provenance of the reputes. The repute database on the other hand is designed to support the memory and compute intensive query operations needed to calculate reputations.

Over time, however, the applicability of a given repute may decrease. Because reputes are records of past behavior that are aggregated to provide predictors of future behavior, as behavior changes over time old reputes become poor predictors of future behavior. Consequently as reputes become stale, they may no longer be need to be readily available for current reputation processing but only for archival purposes in the event of a dispute or merely for historical reasons. Moreover, associated with a repute may be a reputage, that is, ancillary information that provides additional information about the repute context that may also only be needed for historical reasons. Thus a long term term archival storage system would be useful for Open Reputation. Similarly, this mechanism should also be distributed and use diffusive trust. A distributed storage system that can be hosted in local devices such as Internet of Things (IOT) devices or spare capacity on neighborhood computers or servers is called fog storage. This is complementary to but not mutually exclusive with cloud storage which is storage on devices in servers farms such as AWS or GCE.

A diffusive trust storage model typically uses a cryptocurrency to make micro-payments to the storage nodes in much the same way that cryptocurrencies are used to make transaction payments for diffuse trust transaction processing. This approach is compatible with fog storage as it provides a way to incentivise the use of local computing resources.

The archival distributed diffuse trust storage system proposed for Open Reputation is StorJ. StorJ is compatible with both cloud and fog storage and relies on a cryptocurrency to reward the storage host nodes. StorJ is a relatively mature solution. StorJ provides not only encryption but data hiding of files. StorJ also provides storage verification support so that host nodes have to prove they actually have the files in storage. More details can be found on StorJ's web site.

6 Reputation Algorithms

Unlike other reputation systems, Open Reputation treats reputation as something to be built up from the aggregation of reputes. Reputes may include any relevant information items that are both directly and indirectly reputed to a given reputable entity or reputee. These include:

- disclosures of information that may identify a reputee and therefore expose that reputee to risk as a result of the disclosure.
- comments made by a reputee or any content generated by the reputee that could directly or indirectly contribute to that reputee's reputation.
- direct ratings by other entities of that reputee's behavior or content.

- counts such as likes or other indicators of behavior with respect to others.
- scores generated by machine learning algorithms that track the social interactions of a given reputee or inferred from a complex combination of behaviors.

Reputation processing may include some or all of the following: rating, ranking, weighting, aggregation from multiple sources, and reputes of multiple types. Reputation is highly contextual and different reputation algorithms may be more or less useful depending on the context.

Open Reputation supports a data-flow or flow-based architecture for processing reputes. This is a scalable generic approach that is compatible with the demands of distributed processing of disparate sources of information inherent in reputation algorithms. Currently Open Reputation is using the open source Ioflo flow-based programming framework. Although any sufficiently capable framework could be used. Open Reputation's primary goal is to provide trustworthy sources of repute data. Open Reputation is a framework to provides reputable reputes and enable trustworthy provenance of reputes. Open Reputation is agnostic about how to best process a given reputation algorithm. Any given application built on top of Open Reputation may use processing systems best suited to that application.

7 Privacy

7.1 Overview

Privacy and reputation in many ways are in direct conflict. Accountability for one's behavior implies some degree of publicity. Public disclosure is one way to build reputation because it puts the disclosing party at risk to the degree that consequences may arise from the association of their behavior with their identity. Thus one prerequisite to any set of parties engaging in an interaction or transaction may be that the parties mutually disclose their identities so that they may each be held accountable by other(s). This is not the same as saying that the parties necessarily need to disclose who they are to any uninvolved third party. Thus a degree of privacy is preserved if the disclosure is limited to the minimum required for the interaction or transaction to occur but no more. Many of the negative consequences of privacy violation come not from the use of the information by those parties directly involved in the interaction but from the use of that information by uninvolved third parties. Moreover, a third party could by tracking and aggregating the information disclosed in multiple interactions with multiple second parties over time infer additional private information not directly disclosed in any of the transactions. The balancing act between reputation and privacy is to control the degree of disclosure to the minimum that allows the interaction or transaction to proceed but no more.

Because public disclosure is a one way operation, that is, once something that was private is made public it may never be private again, Open Reputation must be architected such that reputes are private by default. The associated parties may make them public but they must opt into that publicity. The consequence of not making a repute public may be a lower reputation score. The degree of privacy is determined by a terms of use agreement and the degree of disclosure of the associated identities..

7.2 Three Degrees of Privacy

There are three basic privacy degrees in Open Reputation, that are full public, semi-private, and full private. Each repute has a unique identifier, that is, the ruid, that is always public. A

repute may also some other information that is always public such as a timestamp and classification information such as tags. A repute also has identifiers for the associated reputees including any reputeries, and a reputer if any. The reputee identifiers may associated with public identities or may be associated with anonymous or pseudonymous identities. Finally a repute has a details section that includes the actual repute content and context. The details section may be encrypted.

In a full public repute, the details section is unencrypted and the reputee identifiers are associated with public identities. In a semi-private repute the details section is encrypted but at least one of the reputee identifiers is public or the details section may be unencrypted but at least one of the reputee identifiers is anonymous. In a full private repute the details section is encrypted and all the reputee identifiers are anonymous.

Semi-private reputes allow interactions where a public reputation associated with a public identity can be used to initiate an otherwise private conversation. This is like a telephone conversation where the parties involved in the call, given by their phone numbers, are known but the actual spoken conversation over the phone is not.

The public ruid allows the data to be stored in the repute database and later retrieved. The ability to track behavior of an entity associated with a repute is influenced by the degree of anonymity of the associated identity given by the included identifier. For example, a commenting system could create reputes of the comments and rating made by the users. The comment system's identity could be fully public and therefore as the reputery creating the repute the identifier cryptonym of the comment system reputery would be public not anonymous. But based on the opt in level of the reputer and reputee targeted by the comment or rating, the corresponding identifiers could be anonymous or not and the actual comment could be encrypted or not.

Anonymous identifiers can be created by virtue of a key generation function that creates key pairs from a master key, where the key generation function is seeded with the ruid of the repute as well as a private master key. This way the associated anonymous key can be recreated by the owner of the master key but not traced to the owner of the master key. Various schemes exist for key generation functions that allow generating anonymous or blinded keys. Similar approaches can be used for generating encryption keys. It is noteworthy that encryption keys are not the same as signing keys. Most block chain system use signing keys to validate transactions but the transaction data is in the clear. An encryption key is used to hide the data.

Privacy of the details of a repute is obtained by encrypting the details. Only those parties that have access to the decryption key may view the information. In many cases the repute information is public to begin with and may not be encrypted at all. In either case, signatures allow verification that the underlying repute details have not been tampered with.

7.3 Group Privacy

Multiple entities may be associated with a given repute and therefore may need the ability to decrypt the details of the repute. One approach is to share the secret location of the decryption key with the members of the group. Another approach is to use a group encryption key. In either case a distributed autonomic service (DAS) can enable multiple entities the ability to decrypt that same repute. The keychain DAS will be responsible for implementing the key

distribution policy. The signatories of a repute will be the members of the associated group and have access to the associated key. The policies may include rules for inclusion and revocation of membership in the group as well as permission for disclosure to external parties. Groups may be nested. The keys themselves may be blinded such that the DAS does not have direct access to the bare keys. The keys may reside only in the private keychains of the group members. A group key can be created using a group shared Diffie Hellman ECC key. A group shared key may be created by successive creation of shared key pairs. For example A, B, and C can create a single shared key. First A creates shared key AB with B and then shared key ABC is created by sharing AB with C. The group key may not be created until the members of the group claim association with the repute. Group keys may be created for both encryption and signing.

One use of group encryption and signing keys is to enable group anonymity, that is, a group reputation may be used to facilitate an interaction between a member of a group and some external party without identifying the specific member of the group involved in the interaction. Group reputation effectively provides a fourth type of privacy, where the group is not anonymous but the specific member of the group associated with the repute is anonymous.

For example, suppose an entity wishes to accept comments only from other entities with a sufficiently high reputation, that is, the accepting party only needs to know that the comments come from an entity with a sufficiently high reputation but does not need to know a priori who is that entity. Consequently, entities could make comments that are reputed with the group identifier instead of the individual identifier. This provides a higher degree of privacy since the behavior, in this case the comment repute, cannot be associated with a specific entity. This approach diffuses the identity across the members of the group thereby reducing the information content usable by a third party and therefore effectively limiting the degree of privacy disclosure.

8 Open Reputation Architecture

The central component of Open Reputation is an immutable database managed by distributed consensus pools. The database is immutable so that all updates to the databased are serializable and traceable. Hash proofs of the database transactions will be serialized into a journal that is anchored to a blockchain. A computationally efficient approach is to create Merkle trees of the hashes of the transactions and then anchor the root of the Merkle tree to one or more blockchains. The database is a distributed hash table where each partition is managed by a consensus pool. The database is a hierarchical key value store that holds reputes, keys, and identities and and encrypted, blinded atlases of the same. Because reputes are data not crypto-currencies they do not need to be tracked in a ledger that is double spend proof. Instead the reputes use the more more efficient storage and retrieval capabilities of a database. The fog storage system provides secure file storage for reputage data as well as backup and archives of repute data. These components are shown in the following diagram.

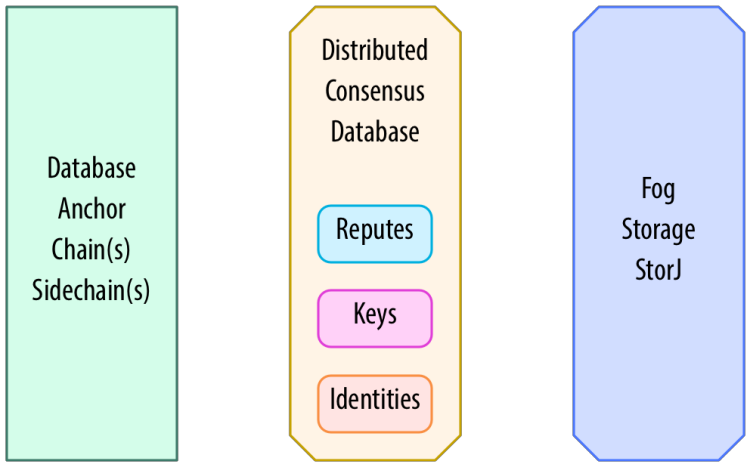

Fig. 8.1 Data Storage

Open Reputation includes several services for authentication, identity management, and key distribution. These are shown in the following diagram.

Fig. 8.2 Services

The following diagram shows the components of an example application that uses Open Reputation.

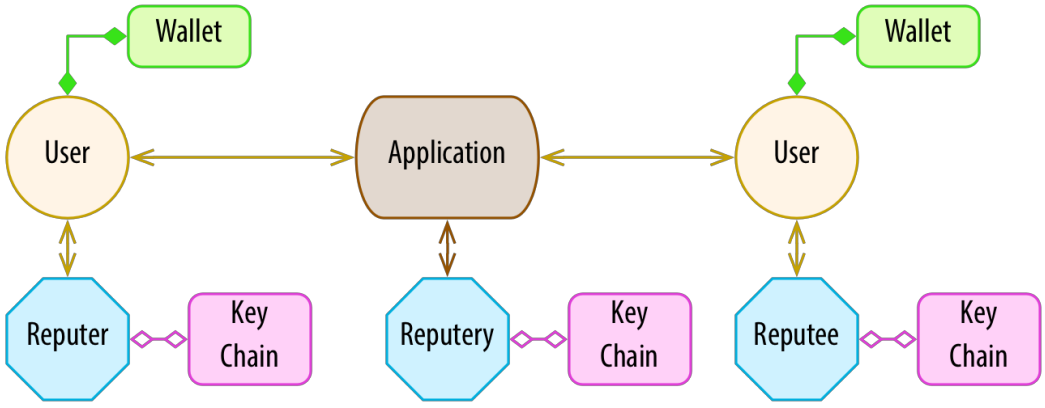

Fig. 8.3 Example Application Diagram

The following diagram provides a simplified view of a notional architecture of the major components of Open Reputation.

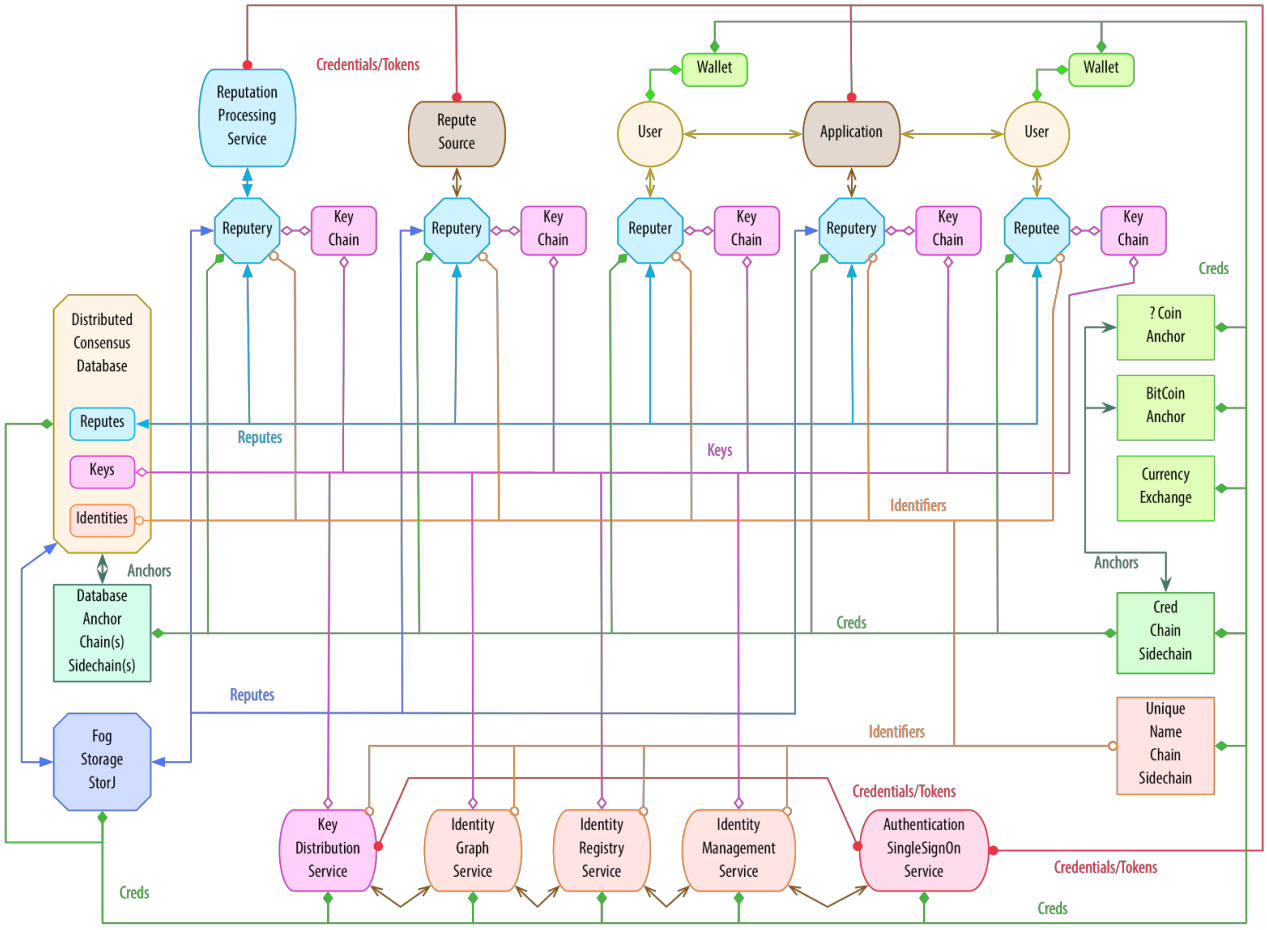

Figure 6.1 Notional Open Reputation Architecture

9 Notional Repute Compositions

The definition of different reputes is still at an early stage. Provided here are some notional compositions of reputes.

One example application of reputation is to a reputation score for a given *reputee*. This score might then be used to predict how the *reputee* may behave in a given context.

There may be multiple parties involved in this process. One party may be the source of the repute. Another party may be the collector of the repute. Yet another party may be a trusted arbiter that validates the exchange of the repute between the first and second parties. Yet another party is the *reputable entity* associated with the repute. The second party might collect multiple reputes about the fourth party reputable entity and from these generate a reputation score that is provided back to the first party or to other parties to be used to filter and modulate interactions between the other parties and the fourth party reputable entity.

9.1 Basic Repute

The basic repute includes the primary information items to provide provenance of the repute. This includes the the identification of the repute, a date time stamp in ISO 8601 format, repute details, reputee, reputer, associated reputeries which in this case include the source and

curator. There is one block of JSON data.

```
{   // repute
    "ruid": "bcd456",  // unique repute id
    "stamp": "2015-03-19T10:30:45Z:, // DateTime stamp repute
    "reputee":  // reputee identification
    {
        "nym": "4def6",  // nym
        "aka": "David my friend"  // alias
    },
    "reputer":  // reputer identification
    {
        "nym": "5efa7",  // nym
        "ada": "boring Mary"   // alias
    },
    "curator": // curator identification
    {
        "nym": "2bcd4",  // nym o
        "aka": "your aunt Alison"  // alias
    },
    "source":  // blog source id
    {
         "nym":"1abc3",  // nym
         "aka": "your cousin Alice"  // alias
    },
    "detail":  // repute details
    {
         "rating":
         {
             "likeable": 90,
             "honest": 80,
             "respectful" : 80,
             "fair": 75,
             "knowledgeable": 75,
         },
         "url": "http://myblog.com/article19/"  // url
    }
    "tags": {},  // classification tags
    "reputage": null
}
/r/n/r/n  // separator
"abcdef987654321"  // source signature
```

9.2 Basic Repute with Reputage

This shows a repute followed by an associated reputage. The repute includes a JSON block that has information about the associated reputage. There are two blocks of JSON data.

```
{   // repute
    "ruid": "bcd456",  // unique repute id
    "stamp": "2015-03-19T10:30:45Z:, // DateTime stamp repute
    "reputee":  // reputee identification
    {
        "nym": "4def6",  // nym
        "aka": "my friend David"  // alias
    },
    "reputer":  // reputer identification
    {
        "nym": "5efa7",  // nym
        "aka": "boring mary"   // alias
    },
    "curator": // curator identification same as copartent
    {
        "nym": "2bcd4",  // nym o
        "aka": "your aunt alison"  // alias
    },
    "source":  // blog source id same as initiator
    {
         "nym":"1abc3",  // nym
         "aka": "your cousin alice"  // name
    },
    "detail":  // repute details
    {
         "rating":
         {
             "likeable": 90,
             "honest": 80,
             "fair" : 80,
             "respectful": 75,
             "knowledgeable": 75
         },
         "url": "http://myblog.com/article19/"  // url
    }
    "tags": {},  // classification tags
    "reputage":
    {
         "puid": "abcefg",  // unique reputage id
         "hash": "def987",  // hash of reputage
         "url": ""  // remote storage location of reputage
    }
}
/r/n/r/n  // separator
"abcdef987654321"  // source signature
/r/n/r/n  // separator
{   // reputage
```

```
    "puid": "abcefg",  // unique reputage id
    "ruid": "bcd456",  // unique repute id of associated repute
    "stamp": "2015-03-19T10:30:45Z:, // DateTime stamp reputage
    "comment":
    {
        "cuid": "1234abc",  // comment unique identifier
        "url": "http://myblog.com/article19/comment19",  // url
        "contents": "You are so awesome.   "  // contents
    }
}
/r/n/r/n  // separator
"abcdef9871234567"  // source signature
```

9.3 Triple-Signed Reputation Transaction (Reputet)

A triple-signed reputation transaction is one way to conduct a reputation event transaction or reputet. One advantage of using triple-signed transactions is that it makes it extremely difficult for any single party to fraudulently engage in a transaction. Another advantage is that at any time in the future, the holder of a triple-signed transaction receipt can prove the validity of the transaction in the event of a dispute. A further advantage of triple-signed transactions is that the a balance agreement is included in the receipt. This enables the destruction of account history. In a reputation application, for example, some forms of reputes might be aggregated into a score that does not need to be reversible. Consequently only the score needs to be preserved not the individual reputes. In this case the "account" balance is just the aggregate of the reputes.

The principal participants in a triple-signed transaction are the *initiator* (party), the *copartent* (counter-party) and *arbiter* (notary). The arbiter must be a trusted third party that is trusted by both the initiator and copartent. The trust is that the arbiter is not colluding with either the initiator or copartent. One way to ensure that trust is to use a distributed consensus pool that is Byzantine Fault Tolerant (BFT) for the arbiter. The consensus pool may be selected such that the probability that a majority of the pool would collude to defraud one of the parties is extremely low. This is feature of many block chain-based distributed consensus algorithms.

Shown below is an example of a triple-signed transaction repute in JSON format. This does not show all the steps to the transaction but merely the end result. The cryptographic keys are 64 Hexadecimal character ECC keys, but in this example, are shortened for the sake of clarity. In this case the initiator is the source, the copartent in the curator, and the arbiter is a shared ledger consensus pool. There are three blocks of JSON data, these are, reputet receipt, repute, and reputage.

```
{   // reputet transaction receipt
    "tn": 15,  // monotonic transaction number
    "tuid": "abc123",  // unique transaction identifier
    "kind": "reputet", // transaction kind
    "initiator":  // initiator identifiers
    {
         "nym": "1abc3",  // nym
```

```
         "aka": "your uncle Bob"  // alias
    },
    "copartent":  // copartent identifiers
    {
        "nym": "2bcd4",  // nym
        "aka": "your aunt Alison",  // alias
    },
    "arbiter":  // arbiter identifiers
    {
        "nym": "3cde5",  // nym
        "aka": "your friend Carol"  // alias
    },
    "detail": {},  // non repute transaction detail
    "statement":  // receipt balance statement
    {
        "user":
        {
            "nym": "1abc3",  // nym
            "aka": "your uncle Bob"  // alias
        },
        "balances":{},  // asset balances
        "pendeds": [12, 13, 14, 16],  // pended trans numbers
        "openeds": []  // opened transaction numbers
    },
    "repute":
    {
         "ruid": "bcd456",  // unique repute id
         "hash": "def987",  // hash of repute
         "url": ""  // storage location of repute
    }
    "previous":  // previous transaction receipt in block chain
    {
        "tuid": "abced12343",  // transaction uid
        "hash": "eee123fabc"  // transaction hash
    }
}
/r/n/r/n  // separator
"abcdef987654321"  // initiator source signature
"123456789abcdef"  // arbiter validator signature
"fedcba456789012"  // copartent curator signature
/r/n/r/n  // separator
{ // repute
    "ruid": "bcd456",  // unique repute id
    "stamp": "2015-03-19T10:30:45Z:, // DateTime stamp repute
    "reputee":  // reputee identification
    {
        "nym": "4def6",  // nym
        "aka": "my friend David"  // alias
```

```
    },
    "reputer":  // reputer identification
    {
        "nym": "5efa7",  // nym
        "aka": "boring Mary"   // alias
    },
    "curator": // curator identification
    {
        "nym": "2bcd4",  // nym o
        "aka": "your aunt Alison"  // alias
    },
    "source":  // blog source id same as initiator
    {
         "nym":"1abc3",  // nym
         "aka": "your cousin Alice"  // alias
    },
    "validator":  // shared ledger validator
    {
         "nym":"4bca3",  // nym
         "aka": "mean Vlad"  // alias
    },
    "detail":  // repute details
    {
         "rating":
         {
             "likeable": 90,
             "honest": 80,
             "fair" : 80,
             "respectful": 75,
             "knowledgeable": 75
         },
         "url": "http://myblog.com/article19/"  // url
    }
    "tags": {},  // classification tags
    "reputage":
    {
         "puid": "abcefg",  // unique reputage id
         "hash": "def987",  // hash of reputage
         "url": ""  // remote storage location of reputage
    }
}
/r/n/r/n  // separator
"abcdef987654322"  // initiator source signature
"123456789abcdea"  // arbiter validator signature
"fedcba456789014"  // copartent curator signature
/r/n/r/n  // separator
{   // reputage
    "puid": "abcefg",  // unique reputage id
```

```
    "ruid": "bcd456",  // unique repute id of associated repute
    "comment":
    {
        "cuid": "1234abc",  // comment unique identifier
        "url": "http://myblog.com/article19/comment19",  // url
        "contents": "You are so awesome.   "  // contents
    }
}
/r/n/r/n  // separator
"abcdef987654322"  // initiator source signature
"123456789abcdea"  // arbiter validator signature
"fedcba456789014"  // copartent curator signature
```

10 References

Web Reputation Systems

Building Web Reputation Systems, Randy Farmer and Aaron Glass 2010

Block Chain

Blockchain: Blueprint for a New Economy, Melanie Swan, O'Reilly Media; 1 edition (February 8, 2015)

Cryptocurrency Market Size

http://coinmarketcap.com

Rank	Crypto Currency	Market Capitalization
1	Bitcoin	$ 3,454 million
2	Ripple	$ 294 million
3	Litecoin	$ 62 million

BitCoin Proof of Work (POW)

https://bitcoin.org/bitcoin.pdf

Proof of Stake (POS)

http://en.wikipedia.org/wiki/Proof-of-stake

Factom

http://factom.org

Factom WhitePaper

Byzantine Fault Tolerant Consensus Protocol

http://en.wikipedia.org/wiki/Byzantine_fault_tolerance

http://www.pmg.lcs.mit.edu/papers/osdi99.pdf

http://pmg.csail.mit.edu/~castro/osdi99_html/osdi99.html

http://usenix.org/events/nsdi09/tech/full_papers/clement/clement.pdf

Redundant Byzantine Fault Tolerant Protocol

http://tendermint.com/docs/tendermint.pdf

https://ripple.com/consensus-whitepaper/

https://www.stellar.org/papers/stellar-consensus-protocol.pdf

StorJ

http://storj.io

http://storj.io/storj.pdf

OneName and OpenName

https://onename.com
https://onename.org
http://namecoin.info
https://openname.org
https://openname.org/about
https://github.com/openname/specifications

DNSChain

https://github.com/okTurtles/dnschain
https://okturtles.com

JavaScript Object Notation JSON

http://json.org

http://www.ecma-international.org/publications/files/ECMA-ST/ECMA-404.pdf

https://tools.ietf.org/html/rfc7159

https://github.com/getify/JSON.minify

http://blog.getify.com/json-comments/

UTF-8 Unicode Format

http://en.wikipedia.org/wiki/UTF-8

ISO 8601 Date-Time Stamp Format

http://www.w3.org/TR/NOTE-datetime

http://www.iso.org/iso/home/standards/iso8601.htm

MsgPack Binary Serialization Format

http://msgpack.org

NaCl: Networking and Cryptography library

http://nacl.cr.yp.to/index.html

http://curvecp.org

http://doc.libsodium.org

https://github.com/jedisct1/libsodium

https://github.com/saltstack/libnacl

Self-certifying Identifiers

http://www.sigops.org/ew-history/1998/papers/mazieres.ps

http://pdos.csail.mit.edu/~kaminsky/sfs-http.ps

Triple-Signed Transactions

http://opentransactions.org/wiki/index.php/Main_Page

https://github.com/Open-Transactions/opentxs

http://opentransactions.org/wiki/index.php/OTX

http://opentransactions.org/wiki/index.php/About

http://opentransactions.org/wiki/index.php?title=Triple-Signed_Receipts

http://iang.org/papers/triple_entry.htmlhttp://iang.org/papers/triple_entry.html

http://iang.org/papers/ricardian_contract.html

https://github.com/OpenBazaar/OpenBazaar

https://gist.github.com/drwasho/a5380544c170bdbbbad8

RAET

https://github.com/RaetProtocol

Ioflo

http://ioflo.com

https://github.com/ioflo

HTTP+TLS

http://www.w3.org/Protocols/rfc2616/rfc2616.html

http://en.wikipedia.org/wiki/Transport_Layer_Security

http://papers.ssrn.com/sol3/papers.cfm?abstract_id=2031409

https://www.youtube.com/watch?v=Z7Wl2FW2TcA

https://s3.amazonaws.com/access.3cdn.net/e6130dacde804482e7_xvm6iigk1.pdf

TeleHash

https://github.com/telehash

https://github.com/openname/specifications/blob/master/profiles/profiles-v03.md