Anonymous Identities for Permissioned Blockchains

(DRAFT v05 – 1/20/2016)

Thomas Hardjono
MIT Internet Trust Consortium
Massachusetts Institute of Technology
Cambridge, MA 02139, USA
Email: hardjono@mit.edu

Ned Smith
Intel Corporation
MS:JF1-255, 2111 NE 25th Ave
Hillsboro, OR 97124
Email: ned.smith@intel.com

Alex (Sandy) Pentland
MIT Media Lab
Massachusetts Institute of Technology
Cambridge, MA 02139, USA
Email: sandy@media.mit.edu

***Abstract*—In this paper we address the issue of retaining user anonymity within a *permissioned* blockchain. We present the *ChainAnchor* architecture that adds an identity and privacy-preserving layer above the blockchain, either the private blockchain or the public Blockchain in Bitcoin. ChainAnchor adds an anonymous identity verification step such that anyone can read and verify transactions from the blockchain but only *anonymous verified identities* can have transactions processed. We refer to such blockchains as *semi-permissioned* blockchains. ChainAnchor builds upon and makes use of the zero knowledge proof mechanisms of the EPID scheme, which has the advantage of an optional cryptographic binding to a TPM tamper-resistant hardware. The use of tamper-resistant hardware provides a significant increase in security, not only for identity-related information but also for the protection of keys used by Bitcoin wallet applications.**

Index terms: Cryptography, Identity Management, Anonymity, Digital Currency.

I. Introduction: Identities and Anonymity in Permissioned Blockchains

The rise to prominence of the Bitcoin decentralized digital currency system [1] has introduced new interest in blockchains as a infrastructure mechanism for maintaining a shared ledger. The success of the Blockchain distributed ledger within Bitcoin as a *permissionless* and public blockchain system has created interest in the possibility of *permissioned* and *private* blockchains. Furthermore, the decentralized processing of transactions in Bitcoin has raised interest in the possibility of a "decentralized digital identity" system for public and private blockchains.

In considering permissioned private blockchains, there is a risk that users of the system may loose the degree of anonymity which they enjoy within Bitcoin as a permissionless system. In the Bitcoin system a user obtains anonymity because he or she generates the public-key pair used to transact in Bitcoin. Only the user knows his/her private-key. When designing permissioned blockchains, there is the temptation to simply link the user's Internet identity (from outside the blockchain) to the user's public-key for the purposes of enforcing access control over the private blockchain. However, this act of linking may result in the disclosure of the true identity of the user holding the public-key. In turn, this may limit the social acceptability of permissioned blockchains and limit their adoption to only private organizations or closed consortiums.

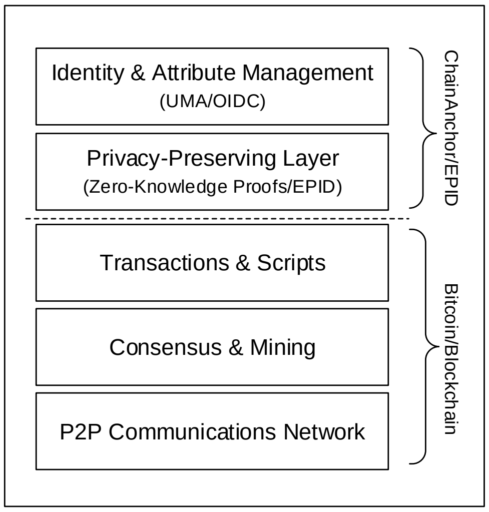


Fig. 1. ChainAnchor Layers

In this paper we address the issue of retaining user anonymity within permissioned blockchains. We present the *ChainAnchor*[1] architecture that adds an identity and privacy-preserving layer above blockchains (including the Blockchain in the current Bitcoin system).

ChainAnchor adds an *anonymous identity verification* step using the EPID scheme [2] that allows anyone to read and verify transactions from the blockchain but allows only anonymously verified identities to have their transactions processed. We refer to such semi-private blockchains as *semi-permissioned blockchains* to distinguish it from "hybrid blockchains" – which refers to a business model [3].

Semi-permissioned blockchains are useful in a number of deployment scenarios, such as within a consortium of competing members who need to share a common ledger but who

[1]The name "Chain Anchor" is derived from the analogous concept of trust anchors in the TAMP protocol (RFC5934).

need to retain anonymity when transacting to the ledger in order to maintain their competitive edge.

In developing the ChainAnchor architecture we seek to fulfill the following objectives:

- *Anonymity of users*: Users achieve the same degree of anonymity as is currently achieved in the Bitcoin system.
- *Anonymous permission verifiability*: Verification of a User or Miner leaves them in the same degree of anonymity as in the Bitcoin system.
- *Permissions Enforcement*: Only anonymous Users who have obtained permission will have their transactions processed. Similarly, only Miners who have obtained permission will have their work remunerated.
- *Revocation of transaction keys*: A verified anonymous User whose private-key has been lost or stolen can anonymously request the priovate-key be placed on a "revoked-keys" list.
- *Functional independence*: The permissions mechanism must be independent from the blockchain (including the current Blockchain) and does not alter its operation.
- *No change to current Bitcoin keys*: The current Bitcoin keys and addresses remain unchanged.

A note regarding terminology: in this paper we use the current public permissionless Blockchain (cap "B") in the Bitcoin system as the focus of our discussions regarding ChainAnchor. This is primarily because the Blockchain is an operational distributed ledger within an operational peer-to-peer (P2P) network of several thousand nodes. It has been well studied and readers are assumed to be familiar with it. As such, designing ChainAnchor for the public Blockchain will allow it to be deployable also in private blockchain systems – which may have a different transaction/block format and may have different consensus algorithms. We use the term *transaction keys* generically to refer to public-key pair that the user self-generates and owns in Bitcoin.

The rest of the paper is organized as follows. In Section II we provide some background to the EPID and DAA schemes that underlie ChainAnchor. Readers familiar with EPID and DAA can skip this section.

In Section III where present the proposed ChainAnchor architecture. Section IV briefly explains two possible deployment modes of ChainAnchor (a fully permissioned deployment, or a mixed permissioned/permissionless deployment). We also briefly discuss some possible remuneration models for Miner who participate in ChainAnchor.

We follow with a discussion on integration of ChainAnchor with Identity Providers and identity management protocols in in Section VI, and close the paper with some proposed future work.

The current paper seeks to be readable to a broad audience and to focus on deployment aspects in the context of services. As such it does not cover in-depth the cryptography behind EPID and DAA, which has already been well treated elsewhere. Here we focus instead on the functions and protocols above the EPID layer.

Readers seeking more details on the EPID scheme will find a short summary in the Appendix which follows the notational convention of [2]. The current paper focuses on an RSA-based EPID scheme based on the Camenisch-Lysyanskaya signature scheme [4] and the DAA scheme of Brickell, Camenisch and Chen [5]. Readers are directed to the authoritative papers of [5] and [2] for an in-depth discussion. An EPID scheme using bilinear pairings can be found in [6]. It is based on the Boneh, Boyen and Schacham group signature scheme [7] and the Boneh-Schacham group signature scheme [8].

II. Background: EPID and DAA

We propose to build the ChainAnchor system for permissioned blockchains based on the *Enhanced Privacy ID* (EPID) scheme [2], which is an extension of *Direct Anonymous Attestation* protocol (DAA) [5] for user privacy in the TPMv1.2 hardware [9].

A. DAA and the Trusted Platform Module

The DAA protocol was developed initially to solve a requirement for privacy within the *Trusted Platform Module* (TPM) hardware chip [9]. The TPM is the security hardware that was developed by the PC industry starting from 1999 within the Trusted Computing Group (TCG) consortium for the purpose of providing low-cost tamper resistant cryptographic hardware for the worldwide PC market. To date, several hundred million TPMs (version 1.2) have been manufactured and have shipped within PC computers worldwide.

The design of the TPM followed three (3) important principles of trustworthy computing [10]:

- *Unambiguous identification*: A given TPM instance must be unambiguously identifiable.
- *Operates unhindered*: A given TPM instance must be able to operate unhindered.
- *Truthful attestations*: A given TPM must be able to correctly report its internal status truthfully.

Although the TPMv1.2 hardware possesses a number of advanced security and privacy-enhancing features, currently the TPM is most commonly used to store cryptographic keys for files/folder encryption (e.g. Microsoft's BitLocker [11]) or for keys to access self-encrypting disk drives [12]. Thus it is not an exaggeration to state that much of the TPMv1.2 advanced functions today remain underutilized. Part of the reason is the current lack of supporting infrastructure for deploying these advanced features (see [13]–[15]).

The DAA protocol was a feature built into the TPM version 1.2 as a privacy mechanism to prevent the tracking of TPM hardwares. Each TPM is unambiguously identified by a public key pair (referred to as the *Endorsement Key* (EK) pair). The

EK private key resides inside the TPM hardware and cannot be read-out of the TPM. The EK public key is placed within an *EK-Certificate* structure as a way to convey the manufacturing provenance of the TPM hardware. However, since the EK-Certificate (containing EK public key) is accessible outside the TPM, there was concern about the possibility of tracking TPMs based on the EK public key.

In order to counter this potential privacy weakness, the DAA scheme was added in TPMv1.2 by the Trusted Computing Group to prevent an external entity from tracking a TPM. At the same time the DAA scheme allowed any external party to gain assurance about the provenance of a given TPM hardware – that the TPM is a genuine hardware produced by a legitimate manufacturer that conforms to the TCG specifications.

In the DAA scheme, an entity called the *Issuer* would create a group public key that is shared across many TPMs. Each TPM, however, would obtain a unique membership private key from the Issuer. The notion of a "group" here refers to a group of legitimate TPM chips, manufactured by a known manufacturer compliant to the TPM specifications.

To "authenticate" as a group member – namely to prove that a TPM is legitimate – the TPM generates a signature using its membership private key such that the signature can be verified by a *Verifier* entity using the group public key. Essentially, the DAA scheme allows a verifier to know that a TPM was produced by a manufacturer, but not learn about the identity of the TPM (i.e. which TPM created the DAA signature).

B. Why EPID and DAA: Motivations

There are a number of reasons why we believe EPID (and the DAA on which it is built) offers an attractive direction for anonymous verifiable identities for permissioned blockchains and other Internet related applications:

- *Substantial deployment base*: The DAA protocol is a core part of the TPMv1.2 standard specification, and supported by the majority of industry PC OEMs. Today several hundred million TPMs (Version 1.2) are already in the field within PC computers and other devices.
- *Standards Status*: The EPID scheme reached ISO International Standard status in 2013 (see [16] and [17]).
- *Option to bind to tamper-resistant hardware*: The EPID protocol can be deployed without TPMv1.2 hardware, with the option to add and enable a tamper-resistant TPM at a later stage. This option may be attractive to Identity Providers who may wish to deploy ChainAnchor in a phased approach. The TPM hardware can internally generate a Bitcoin public key pair, and sign the Bitcoin transactions using these on-board keys. If the User loses his/her device with the TPM, the Bitcoin currency will be irretrievable, but will be secure from theft (i.e. currency destroyed).
- *Backup of hardware-based keys*: The hardware also offers a Backup-and-Migration protocol (see [18] and [19]) that allows sealing of keys (including user's keys) for off-device secure backups. As such, it provides a strong mechanism for users to "backup their currency" (i.e. Bitcoin private keys). The TPMv1.2 backup protocol will only restore the sealed keys to the same TPM hardware. The TPMv1.2 migration protocol will only move/transfer the sealed keys to a new TPM hardware that has undergone a take-ownership by the same user/owner.

EPID is not the only anonymous identity protocol available today. The work of Brickell et al. [5] introduced the first RSA-based DAA protocol in 2004. A related anonymity protocol called *Idemix* [20] employs the same RSA-based anonymous credential scheme as the DAA protocol. However, Idemix cannot be used with the TPMv1.2 hardware (or the new TPMv2.0 hardware).

Another related protocol called *U-Prove* [21] can be integrated into the TPM2.0 hardware (see [22]). However, the U-Prove protocol has the drawback that it is not multi-show unlinkable [23], which means that a U-Prove token may only be used once in order to remain unlinkable.

III. ChainAnchor Design

Our proposed ChainAnchor system makes use of the zero-knowledge proof protocol of EPID to allow a User to prove to a *Permissions Verifier* entity that the User is a member of the *Permissioned Group* and therefore has the privilege to have their transactions processed and added to the blockchain. The permissioned-group is created by a *Permissions Issuer* entity at the request of a creator (organization or person).

The User has to cryptographically "bind" his or her transaction public-key to the zero-knowledge proof sent to Permissions Verifier, resulting in that transaction public-key being recognized as an "identity" that had obtain permission to transact on the blockchain. The Permissions Verifier adds the approved transaction public-key to a *Verified Identities Database* operated by the the Permissions Verifier. Depending on the deployment mode, this database may instead be operated by the Permissions Issuer. Similarly, a Miner (mining node) who wishes to participate in ChainAnchor must perform the same zero-knowledge proof process.

Depending on the mode of operation (see Section IV) the Verified Identities Database can be private or be publicly readable. The method to read from the database is out of scope here, though there is considerable deployment experience for such databases in the form of CRL-databases in X509-based PKI [24] using protocols such as OCSP [25].

Note that the User can bind as many self-asserted transaction public-keys to the zero-knowledge proof sent to Permissions Verifier as needed (either through submitting in batches, or submitting individual public-key one at a time). This is consistent to the design and operations of Bitcoin today where

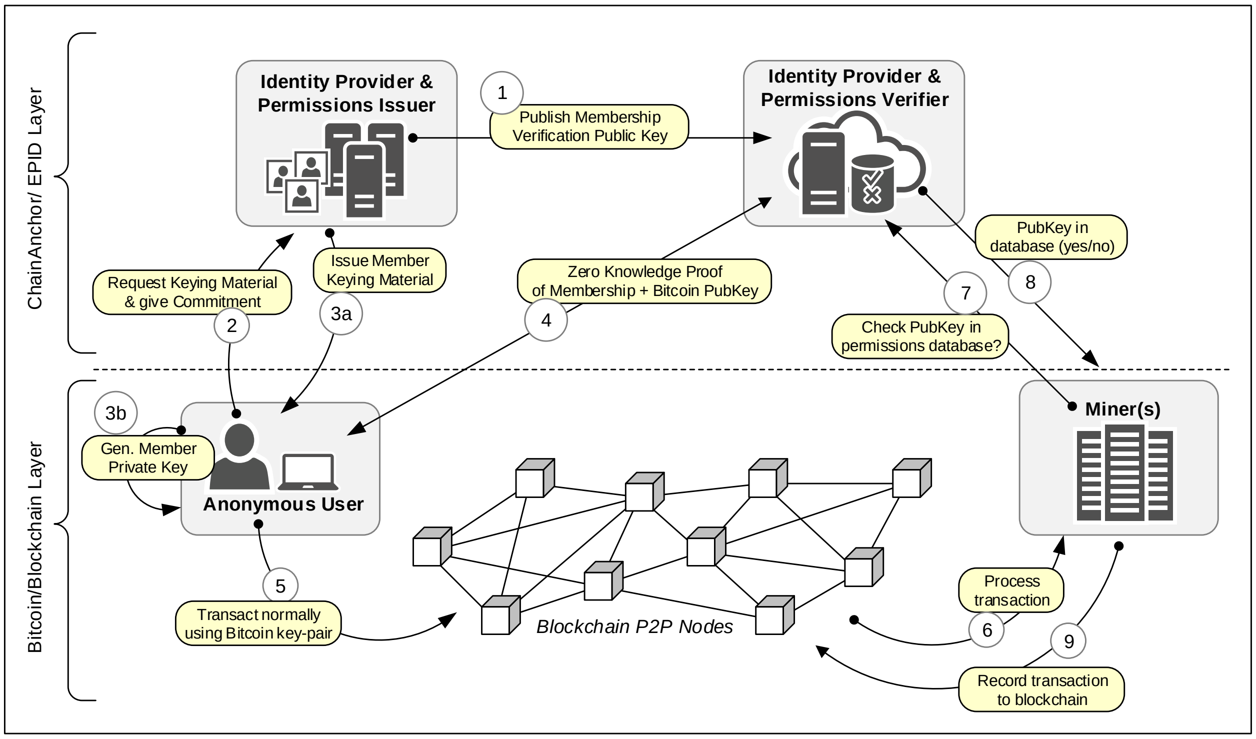

ChainAnchor/ EPID Layer
Bitcoin/Blockchain Layer
Identity Provider & Permissions Issuer
Identity Provider & Permissions Verifier
1
Publish Membership Verification Public Key
Request Keying Material & give Commitment
2
Issue Member Keying Material
3a
Zero Knowledge Proof of Membership + Bitcoin PubKey
4
PubKey in database (yes/no)
8
7
Check PubKey in permissions database?
3b
Gen. Member Private Key
Anonymous User
Miner(s)
5
Transact normally using Bitcoin key-pair
Blockchain P2P Nodes
Process transaction
6
9
Record transaction to blockchain

Fig. 2. Overview of ChainAnchor interactions

a User can use any number of self-created public key pairs in Bitcoin.

ChainAnchor adds an additional simple step to the processing (mining) of a transaction performed by a Miner. Prior to mining a transaction signed by a given transaction public-key, the mining node needs to look-up the Verified Identities Database to check that the transaction public-key is listed in that database. That is, it needs to check that User's public key is "permissioned" to transact on the blockchain.

Similar to EPID and related schemes, the Permissions Verifier is assumed not to be in collusion with the Permissions Issuer, and both are expected to be a separate entities (physically, operationally and legally).

In the following we describe the entities in the ChainAnchor system and the steps of ChainAnchor.

A. Entities in the System

The set of entities in ChainAnchor does not depart significantly from the those in EPID and DAA. However, in order for EPID to be deployable with the current Bitcoin system – which is currently a system running independently from any existing identity infrastructures – in ChainAnchor we propose to converge the roles in EPID with those found in identity management protocols today as deployed by Identity Provider (IdP) services. Section VI will discuss integration with Identity Providers.

In ChainAnchor, we define transactions and blocks of transactions as follows:

- *Permisisoned-Transaction*:
  We define a permissioned-transaction to be one in which the originator and recipient of the transaction are members of the same permissioned-group.

- *Permisisoned-Block*:
  We define a *permissioned-block* to be a block of permissioned-transactions. That is, for each transaction in the block the originator and recipient of the transaction are members of the same permissioned-group.

Figure 2 summarizes the entities and the interactions among the entities:

- *Identity Provider and Permissions Issuer* (IdP-PI):
  ChainAnchor merges the Permissions Issuer function with the Identity Provider function to reflect the need, among others, for *addressability* of the anonymous User who holds the self-asserted transaction public-key (i.e. Bitcoin public-keys).

  That is, the Identity Provider function will need the ability to communicate out-of-band with the anonymous User outside the blockchain system in order to engage the User in the ChainAnchor-related protocols (e.g. notifying a User of a suspected compromise of their private keys, group discovery, etc). The IdP-PI also acts as the revocation manager for EPID-related revocation variants.

The IdP-PI creates a *Permissioned Group* that implements the permissioned blockchain on behalf of an owner. For a given permissioned-group, there is one (and only one) IdP-PI.

- *Permissions Verifier* (IdP-PV):
  The Permissions Verifier is the entity that performs the anonymous identity verification of a given a User by running the EPID zero knowledge proof protocol with that User.
  In ChainAnchor the Permissions Verifier function is also operated by an Identity Provider that must be distinct from the Permissions Issuer. This ensures both IdP-PI and IdP-PV entities remain honest. Indeed, this is a core design aspect of the DAA feature in TPMv1.2.
  The Permissions Verifier maintains *Verified Identities Database* containing all the Bitcoin public keys belonging to the anonymous Users who have successfully run the the zero-knowledge proof protocol against the Permissions Verifier. The database only contains the Bitcoin public keys and the time-stamp of the successful zero-knowledge proof protocol completion. No other identifying information is stored by the Permissions Verifier.
  The IdP-PV together with the IdP-PI realize the permissioned-group that implements the permissioned blockchain. For a given permissioned-group, there can be multiple independent IdP-PV entities (although only one IdP-PI).

- *Miner*:
  The Miner in ChainAnchor is entity that processes (mines) a permissioned-block and records it on the blockchain.
  In implementation, the ChainAnchor Miner is the same as the miner in Bitcoin with the extra step performed by the ChainAnchor Miner. When composing transactions into a block (i.e. the candidate block), for each transaction the Miner must check that the public-keys of the transaction are in the Verified Identities Database at the IDP-PV. That is, it must verify that every transaction going into the candidate block is a permissioned-transaction.
  This simple look-up step will prevent the Miner from wasting CPU cycles on transactions that are not part of the permissioned-group.
  Similar to the User, a Miner wishing to participate in a given permissioned-group must perform the anonymous identity verification to the Permissions Verifier (IdP-PV) and have its transaction (Bitcoin) public-key added by the IdP-PV to the Verified Identities Database of the permissioned-group.
  This is to ensure that the Miner can have query-access to the database of the group, and that the Miner can claim the reward for mining permissioned-blocks. Section IV discusses the modes of operation of ChainAnchor and the possible remuneration models.

- *User*:
  The User in ChainAnchor is the same as the originator/recipient of a transaction in Bitcoin. The User can have any number of self-generated transaction public-key pairs. However, in order to participate in ChainAnchor permissioned-group the User must perform the anonymous identity verification to the Permissions Verifier (IdP-PV) and have its transaction (Bitcoin) public-key added by the IdP-PV to the Verified Identities Database of the permissioned-group.
  By being a member of a permissioned-group, a User has query-access to the database of that group. This allows the User as the originator of a transaction to verify that a recipient is also a member of the same permissioned-group prior to sending the transaction.

- *Creator/Owner*:
  Although not shown explicitly in Figure 2, a permissioned blockchain must be created and owned by an organization or individual. We refer to this entity or person as the owner of a permissioned-group (mechanism) that implements the permissioned blockchain. The IdP-PI and IdP-PV are service providers that realize and deploy the permissioned-group on behalf of the owner of the permissioned blockchain.

B. Keys in the System

The ChainAnchor system uses a number of cryptographic keys – beyond the User's transaction public-key pair. These keys are summarized as follows:

- *Membership Issuing Private Key*:
  This key is generated by the IdP-PI for each permissioned-group that the IdP-PI establishes. This key is unique for each permissioned-group. This key is used by the IdP-PI in enrolling or adding new Users to the permissioned-group. For each permissioned-group impelemented by the IdP-PI, the IdP-PI entity must generate a unique Membership Issuing Private Key,

- *Membership Verification Public Key*:
  This key is generated by the IdP-PI and is delivered over a secure channel to the Permissions Verifier entity (IdP-PV). This key is unique for each permissioned-group. The key allows the IdP-PV to validate the membership of a User (in the corresponding permissioned-group) through the zero knowledge proof protocol that the IdP-PV executes with the User.

- *User's Transaction Public-Key Pair*:
  This is the transaction public-key pair (i.e. Bitcoin

public-key pair) that the User uses to transact on the permissioned blockchain.

- *Personal Identity Public-Key Pair & Certificate*:
  These are traditional public-key pairs and X509 certificates that identify the entity that posses them. Being service providers, the IdP-PI and IdP-PV are assumed to have X509 certificates. Users may not have a certificate but are assumed to have personal public-key pair. These keys are denotes as follows (in the notation of [2]):
  - *IdP-PI public-key pair*: We denote the public key pair of the IdP-PI as (K_{PI}, K_{PI}^{-1}) with the public key being K_{PI}.
  - *IdP-PV public-key pair*: Similarly, we denote the public key pair of the IdP-PV as (K_{PV}, K_{PV}^{-1}) with the public key being K_{PV}.
  - *User's transaction public-key pair*: We denote the User's transaction (Bitcoin) public-key pair as $(K_{bitcoin}, K_{bitcoin}^{-1})$, with the public key being $(K_{bitcoin})$.
  - *User's personal public-key pair*: We denote the personal public-key pair of the User as $(K_{user}, K_{user}^{-1})$ with the public key being K_{user}. The User *must never* associate his/her personal public-key pair with his/her transaction (Bitcoin) public-key pair

C. ChainAnchor Protocol Steps

In the following, we describe the steps of the ChainAnchor design (see Figure 2).

[Step 0] *IdP-PI Establishes Permissioned Group*:
This step is not shown in Figure 2. Depending on the business model of the IdP-PI and IdP-PV, the IdP-PI can establish permissioned-group as fee-paying service to customers (e.g. Enterprises).

As part of the creation of a permissioned-group, the IdP-PI generates a number parameters that are unique to the permissioned-group and are used to create two important keys related to the function of the IdP-PI as the Permissions Issuer:

- *Membership Verification Public Key*: K_{MVPK}
  The IdP-PI creates this key to be used later by the Permissions Verifier entity (IdP-PV) when engaging the User in the zero-knowledge proofs protocol. (See Equation 2 in Appendix A).
- *Membership Issuing Private Key*: K_{MIPK}
  The IdP-PI creates this key in order to issue unique keys to Users in the system that allows the User later to prove membership to the Permissions Verifier entity (IdP-PV). (See Equation 3 in Appendix A). This issuing private key is kept secret by the IdP-PI.

[Step 1] ***IdP-PI Publishes Verification Public Key***:
In this step, the IdP-PI makes known the Membership Verification Public Key (K_{MVPK}) to the Permissions Verifier (IdP-PV). We assume a secure channel with mutual authentication is used between the IdP-PI and IdP-PV entities.

[Step 2] ***User Requests Membership***
An anonymous User obtains permission to transact on the permissioned blockchain by requesting membership to the permissioned-group that implements the permissioned blockchain. The User sends the request to the IdP-PI that manages the permissioned-group of interest. The User must perform a number of steps to become a member:

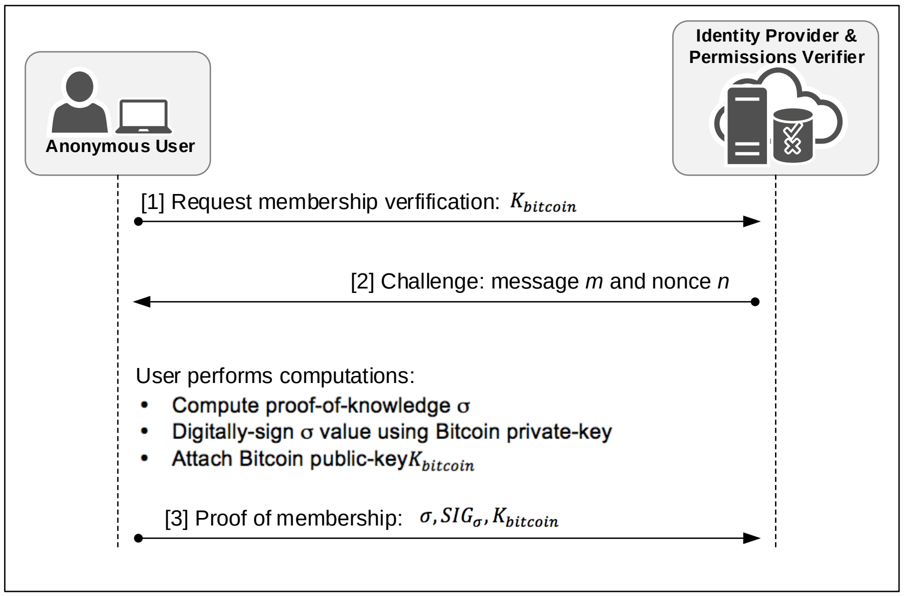
- *User obtains the Membership Verification Public Key*: The User must obtain K_{MVPK} from the IdP-IP using a secure channel, with mutual authentication (e.g. using their respective public keys K_{PI} and K_{user}). The Membership Verification Public Key is shown in Equation 2 in Appendix A.
- *User validates the Membership Verification Public Key*: Prior to using some of the parameters in the key the User must verify that these parameters are formed correctly.
- *User generates commitment parameters*: The User uses some of the parameters in the Membership Verification Public Key to create his/her own *commitment* parameters that "blinds" the User's own secret keying material. (See Equations 4 and 5 in Appendix A).
- *User sends commitment parameters to the IdP-PI*: The User sends the commitment parameters to the IdP-PI, who in-turn must verify that these parameters are formed correctly.

In requesting membership to the IdP-PI as the service provider, the User may need to reveal his/her actual identity in order to be admitted into the permissioned blockchain. This process is external to the blockchain. Hence the User's personal public-key pair used to secure the download of K_{MVPK} and to upload the blinded commitment parameters

Although the IdP-PI may learn the User's true identity (e.g. for business purposes), a User must never reveal their personal public-key pair or their identity to the IdP-PV entity.

In creating secure channels with either the IdP-PI or IdP-PV the User must never use his/her transaction (Bitcoin) public-key pair, as that would destroy the anonymity of the transaction public-key pair.

[Step 3A] ***IdP-PI delivers Group-Member Keying Parameters***
In this step, the IdP-PI generates a number of group-member keying parameters that are specific to the requesting



Anonymous User
Identity Provider & Permissions Verifier
[1] Request membership verfification: $K_{bitcoin}$
[2] Challenge: message *m* and nonce *n*
User performs computations:
• Compute proof-of-knowledge σ
• Digitally-sign σ value using Bitcoin private-key
• Attach Bitcoin public-key$K_{bitcoin}$
[3] Proof of membership: $\sigma, SIG_{\sigma}, K_{bitcoin}$

Fig. 3. User Anonymously Proves Membership to IdP-PV

User, based on the commitment parameters that the User had submitted in the previous step.

[Step 3B] ***User Generates User-Member Private Key***

Upon receiving the user-specific group-member keying parameters, the User uses these parameters to generate his/her own *User-Member Private Key*, denoted as K_{UMPK}. (See Equation 6 in Appendix A).

It is worthwhile to note at his point that there is a *Many-to-1* asymmetric relationship between multiple User-Member Private Keys and the single Membership Verification Public Key (K_{MVPK}) that was delivered from the IdP-PI to the IdP-PV entity in Step 1. That one verification public key K_{MVPK} allows the IdP-PV verify all permissioned-group members (i.e. Users) who wield their own respective User-Member Private Key K_{UMPK}

More specifically, if two Users U_1 and U_2 independently presents a message with a signature-of-knowledge (see Equation 8) created using keys $K_{UMPK_{u1}}$ and $K_{UMPK_{u2}}$ respectively, then the Permissions Verifier IdP-PV can verify both signature using the one verification public key K_{MVPK} but it will not be able to distinguish between Users U_1 and U_2. Indeed, it is this very feature found in the DAA scheme [5] that motivated the adoption of the DAA scheme for privacy-enabling the TPM hardware.

As part of this step, the User has to choose a *base* parameter, which can be a random base or a named-base (See Equations 4 and 5 the Appendix). In choosing between the random-based or named-based approaches, there is essentially a trade-off between full anonymity (privacy) and convenience. The named-based approach may be useful if several IdP-PV entities exist and the User seeks to use the IdP-PV that he or she trusts. The signatures from the User states (identifies) the identity of the IdP-PV entity that the User seeks to use. However, in this case the IdP-PV entities may build-up a correspondence list between the EPID key used in the signature and the transaction (Bitcoin) public-key, thereby somewhat reducing the anonymity of the User in exchange for improved value added services provided by the IdP-PV.

[Step 4] ***User Anonymously Proves Membership to IdP-PV***

The anonymous membership verification protocol consists of a number of sub-steps following the challenge-response model. The User sends a request to the Permissions Verifier (IdP-PV), and in-turn the IdP-PV challenges the User with some parameters that the User must respond to.

In engaging the IdP-PV entity, the User *must never* use his/her own personal public-key pair, as this would disclose the User's true identity.

In requesting the IdP-PV for an anonymous membership verification, a secure channel with only one-way authentication is required. That is, only the IdP-PV needs to prove its true identity as a service provider. This is because the User must obtain assurance that it is engaging the correct IdP-PV, and not a bogus server masquerading as the IdP-PV. As such, the secure channel between the User and the IdP-PV must use the server-side certificate of the IdP-PV. (This is already common everyday practice today by many service providers).

The sub-steps of the anonymous membership verification protocol are as follows (Figure 3):

- The User sends a request to the IdP-PV for an anonymous membership verification. Although unnecessary, depending on the implementation the

User may include a copy of his/her transaction public key $K_{bitcoin}$.

- The Permissions Verifier IdP-PV responds by returning a challenge message m and a random nonce n_{pv} to the User. Note that if the User is a bogus entity or person, they would not have engaged the IdP-PI in Step 2 and Step 3 with a unique (User-chosen secret) commitment parameter. As such, a bogus user will not be able to continue undetected by the IdP-PV beyond the next step.

- Upon receiving the challenge message m and the random nonce n_{pv} from the verifier IdP-PV, the User must compute a "signature of knowledge" of the commitment parameter that the User supplied to the IdP-PI in Step 2. The signature-of-knowledge is denoted as σ. (See Equation 8 in Appendix A).

  As input into the signature-of-knowledge computation, the User inputs:
  - The User's Membership Verification Public Key (K_{MVPK}) which the User obtained from the Permissions Issuer IdP-PI in Step 2. (See Equation 2 in Appendix A).
  - The User's own User-Member Private Key K_{UMPK} which the User computed in Step 3. (See Equation 6 in Appendix A).
  - The challenge m and the nonce n_{pv} obtained from the Permissions Verifier IdP-PV.

- As part of the ChainAnchor protocol, the User must sign the value σ using the User's transaction private key $K^{-1}_{bitcoin}$. The signature is denoted as SIG_{σ}. The User can use the signature algorithm that is already built-in and deployed in the Bitcoin system (e.g. ECDSA using secp256k1 curve). This provides a very simple cryptographic binding between the User's transaction key-pair and the signature-of-knowledge proof σ.

- The User sends the following three values to the IdP-PV:

$$(\sigma, SIG_{\sigma}, K_{bitcoin}) \tag{1}$$

- The IdP-PV validates signature-of-knowledge σ, and returns an acknowledgement of a successful verification process to the User. The IdP-PV then adds the User's transaction public key $K_{bitcoin}$ to the Verified Identities Database.

  (At this point the IdP-PV has the option to generate a new "anonymous Internet identity" for the User and returning it to the User as part of the acknowledgment. This identity provides addressability of the anonymous User outside the blockchain system. This will be discussed further in Section V).

Equation 1 represents the cryptographic binding between the proof of membership values and the User's transaction public key pair. Depending on the implementation of the permissioned-group, the User could also send a batch of transaction public keys $(K_{bitcoin_1}, K_{bitcoin_2}, \ldots, K_{bitcoin_j})$ in Equation 1 to the IdP-PV. However, this batch processing approach may allow the IdP-PV to later track and correlate transactions using these keys.

A further improvement can be made by the User including the *basename* value in the signature sent to the Permissions Verifier. This allows the User to choose the Permissions Verifier that he or she trusts, several of which may exists at any one time. This approach is taken the *DAA-SIGMA* key exchange protocol [26], which embeds DAA within the key agreement flows.

[Step 5] ***User Transacts on Permissioned Blockchain***
In this phase the User transacts on the permissioned blockchain in the usual manner (as in Bitcoin), using the transaction private-key $K^{-1}_{bitcoin}$ to sign transactions.

[Step 6] ***Miner Processes Transaction***
Following the normal Bitcoin transaction processing in the peer-to-peer network, a Miner fetches a transaction (i.e. from the pool of unprocessed transactions) and prepares to process that transaction.

[Step 7] ***Miner Validates User's Public Key***
Prior to processing a transaction for inclusion into a block, a Miner participating in the ChainAnchor permissioned-group must check that the public-key found in the transaction has been approved to participate in the permissioned-group. That is, the Miner must first look-up the Verified Identities Databases at the IdP-PV to ensure the public key is in the database.

Miners who are not participating in the ChainAnchor permissioned-group will be oblivious to this validation step and will process the transactions in the usual Bitcoin way.

[Step 8] ***Miner Records Transaction***
If the public key $K_{bitcoin}$ used in the User's transaction exists in the Verified Identities Database, the ChainAnchor Miner proceeds with processing the transaction (i.e. add to block, perform proof of work, etc). Otherwise the ChainAnchor Miner ignores the transaction.

D. Revocation of Lost or Stolen Keys

One of the major weaknesses – or strengths, depending on one's point of view – of Bitcoin is its lack of a key management infrastructure that supports end-users revoking keys which they suspect have been compromised or stolen. When a Bitcoin private key is lost or stolen, there is the danger that any Bitcoin currency associated with that lost key will be stolen (i.e. currency transferred in a transaction to another Bitcoin public key or address). Furthermore, the decentralized and permissionless design of Bitcoin ensures that there is no centralized authority or control. Consequently, there is no entity in Bitcoin to whom a user can request or effect the revocation of keys (i.e. keys that the User self-generated). This weakness (strength) is a direct corollary of the anonymity of the self-asserted (self-generated) key pairs in Bitcoin.

In a permissioned-group in ChainAnchor there is opportunity for revocation services to be provided by the IdP-PI or the IdP-PV entities depending on the type revocation required. Currently we propose ChainAnchor to support two types of revocations:

- *Simple revocation (IdP-PV)*:
  In the simple revocation, we assume that the User still has a copy of their transaction public-key pair, but suspects that the private key has been compromised (i.e. copied). Here the User wishes to prevent further use of that transaction public-key pair in the permissioned blockchain.

  The User sends a revocation-request message to the Permissions Verifier (IdP-PV), which is signed using the still-extant private key. Here the IdP-PV essentially plays the role of a "Revocation Authority" (much in the manner of X509 Certificate Authorities operate a Certificate Revocation List (CRL) service [27]).

  Note that the IdP-PV will only respond to requests from existing anonymous verified members (i.e. Users whose transaction public-keys are already in the Verified Identities Database).

- *EPID-based revocation (IdP-PI)*:
  The second type of revocation makes use of the underlying EPID revocation mechanisms, where the Permissions Issuer (IdP-PI) entity becomes the revocation authority for the Permissions Group (since it was the IdP-PI who generated and "published" the Membership Verification Public Key in Step 1 and who provided the User with a unique Group-Member Keying parameters in Step 3).

  EPID supports three variants of revocations: (a) revocation based on the User-Member Private Key, (b) revocation based on the signature-of-knowledge that the User computed in Step 4 above, and (c) revocations done proactively by the Permissions Issuer who may suspect that a User has lost their keying material which the User had obtained from the Permissions Issuer (in Step 3A).

The reader is directed to Appendix A and to [2] for more details on the EPID-based revocation variants.

E. Discussion

It is important to pause here to review what has been achieved in binding the the transaction (Bitcoin) public-key pair with the zero knowledge proof of membership:

- *User remain anonymous to IdP-PV*: It is important to note that when the User requests the Permissions Verifier IdP-PV for an anonymous membership verification the User signs the request using one of the user's transaction public-key pairs (and not a personal public-key pair that can identify the User). Following Bitcoin, it was the User who self-generated the transaction public-key pair (keeping the private-key secret). As such, the User remains anonymous to the Permissions Verifier.

- *User remain anonymous to IdP-PI*: Although the User may have used his/her own personal public-key pair in requesting membership to the IdP-PI (in Step 1) and therefore made known their real-world identity to the IdP-PI, the IdP-PI has no knowledge of which transaction (Bitcoin) public-key pairs are owned by the User. Subsequent to Step 3A the User becomes anonymous even to the IdP-PI because the User injects a secret parameter when generating the User's *User-Member Private Key* K_{UMPK} (see Equations 4, 5 and 6 in the Appendix).

- *Simple binding to transaction key pair*: We have used a digital signature as a simple binding mechanism between the transaction public-key $K_{bitcoin}$ and the proof of membership (the value σ in Step 4). This is primarily due the availability of the digital signature function within the Bitcoin-core open source code. We note that other more complex cryptographic bindings can be also be applied, including the injection of the User's transaction key $K_{bitcoin}$ as input into deriving the User's secret parameters (i.e. parameter f in Equations 4, 5 and 6 in the Appendix).

IV. Modes of Deployment and Incentives

In this section we discuss two modes of deployment for ChainAnchor. Although other modes can be used, we limit discussion to only these two modes.

A. Deployment Modes

- *Homogenous Permisioned Nodes*:
  In this deployment mode (Figure 4), all of the nodes participating in the peer-to-peer network are ChainAnchor nodes. That is, they all implement and enforce the ChainAnchor anonymous identity verification steps. All validated transactions in every block are permissioned transactions.

  This mode of deployment may be attractive to a closed private blockchain deployment that is separate from the permissionless blockchain in Bitcoin, and may use its own non-standard transaction and/or block payload format.

  In this mode every node on the network must perform independent validation of each new transaction block, including verifying that each transaction is performed by a permissioned User (i.e. the transaction public

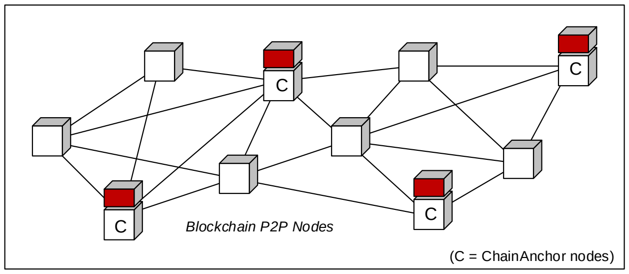


Fig. 4. Mixed Permisioned-Permissionless Nodes

key is listed in the Verified Identities Database). Non-conforming transactions are simply dropped.

- *Mixed Permisioned-Permissionless Nodes*:
  In this mode of deployment, ChainAnchor is deployed as an *overlay* above the current permissionless Blockchain. Thus the peer-to-peer network is a heterogenous mix of Bitcoin nodes and ChainAnchor nodes. The Bitcoin nodes will be oblivious to permissioned-transactions.
  Here a Miner has the capability to act as either a Bitcoin miner or a ChainAnchor miner. That is, a ChainAnchor mining node operates as a "dual-headed" server, where it can process both ordinary Bitcoin transactions and ChainAnchor permissioned-transactions.
  However, the difference lies in the homogeneity of the blocks of transactions they process and the remuneration obtained from processing permissioned-transactions. In order to be remunerated, when a Miner chooses to work as a ChainAnchor mining node, it must create a block consisting only of permissioned-transactions.
  In this mode of deployment, both the Permissions Issuer entity (IdP-PI) and the Permissions Verifier entity (IdP-PV) are assumed to have bitcoin-addresses that are known to members of the corresponding permissioned-group.

B. The Mixed Permisioned/Permissionless Mode

As mentioned before, we define a *permissioned-block* to be a block of transactions where for each transaction in the block the originator and recipient of the transaction are members of the same permissioned-group.

Depending on the degree of strickness of implementation, the mixed permisioned/permissionless network in ChainAnchor offers a number of interesting features:

- The originator of a permissioned-transaction can verify if the recipient is a member of the same permissioned-group.
- The recipient of a permissioned-transaction can verify if the originator is a member of the same permissioned-group.
- A ChainAnchor mining node participating in a given permissioned-group can verify if an unconfirmed transaction belongs to the permissioned-group. That is, both the originator and recipient are members of the same permissioned-group.
- The Permissions Issuer entity (IdP-PI) and Permissions Verifier entity (IdP-PV) can independently verify whether a block contains only of permissioned-transactions, or if a block contains a mix of ordinary transactions and permissioned-transactions.

In this mixed mode, when a Miner chooses to operate as a ChainAnchor mining node, it must create blocks consisting only of permissioned-transactions. The goal here is not to create a separate chain, but rather use the current permissionless Blockchain to carry permissioned-transactions relating to Users in ChainAnchor.

Mining and consensus over a block of permissioned-transactions is achieved the same way as in Bitcoin transactions. In Bitcoin miners receive two types of rewards for mining, namely new coins (created with each new block of transaction) and transaction-fees (from the User/originator) related to each transaction included in the block.

In ChainAnchor a successful miner receives a further additional reward for completing a block consisting only of permissioned-transactions. Being a participant in a ChainAnchor permissioned-group, the mining node can look-up the Verified Identities Database to check if both the originator and recipient are members of the same permissioned-group. As such, the mining node can be selective in choosing transactions to process.

Note that to a plain Bitcoin mining node (i.e. not participating in a ChainAnchor permissioned-group) a block of permissioned-transactions looks no different than ordinary Bitcoin transactions. A plain Bitcoin node may be oblivious to the fact that a transaction originated from (and destined to) Users who are participating in a permissioned-group. The plain mining node will not know to look-up the Verified Identities Database at the IdP-PV.

Although beyond the scope of the current work, a ChainAnchor miner that created the block may embed information about the ChainAnchor permissioned-group inside the block header (e.g. permissioned group-ID) but such information will be ignored by non-ChainAnchor nodes.

When ChainAnchor is deployed in a mixed permissioned/permissionless mode, both the Permissions Issuer entity (IdP-PI) and Permissions Verifier entity (IdP-PV) must independently verify that a block contains only of permissioned-transactions. Furthermore, both the IdP-PI and IdP-PV must verify that the successful miner is also a member of the permissioned-group. Note that both the IdP-PI and IdP-PV have access to the Verified Identities Database.

A dishonest Permissions Verifier entity (IdP-PV) – who denies payment to a successful participating miner – will be found-out by the IdP-PI and other mining nodes who participate in the same permissioned-group. This is because the IdP-PI can see (on the Blockchain) if the IdP-PV had made payment to the successful miner (i.e. payment to the miner's bitcoin-address). It is in the interest of the IdP-PI – who is a service provider operating the permissioned-group on behalf of the group creator – to ensure that the IdP-PV remain honest.

C. Remuneration Models: Fixed Fees & Pro-Rata

In the ChainAnchor mixed permisioned/permissionless mode a successful miner receives a further additional payment (beyond the new coins and transaction-fees in Bitcoin) for completing a block consisting only of permissioned-transactions. This additional fee is paid by the Permissions Verifier entity (IdP-PV) to the successful miner using a separate Bitcoin transaction. Since the originator and recipient of this payment are members of the same permissioned-group, the occurrence of this payment can be verified by other members of the permissioned-group (and indeed by anyone) by looking at the from the underlying Blockchain (ie. search through confirmed transactions).

Nodes who are not members of the permissioned-group will only see a currency transfer from an originator (IdP-PV) to a recipient (the successful miner). However, members of the permissioned-group can independently verify that a bitcoin-address K_{miner} has successfully mined N_{PG} number permissioned-blocks and has received the same N_{PG} number of payments from the bitcoin-address K_{PV}. Here K_{miner} is the miner's bitcoin-address and K_{PV} is the bitcoin-address of the IdP-PV.

The reward fee for completing a ChainAnchor permissioned-block is advertised by the Permissions Issuer entity (IdP-PI). The mechanism to advertise this fee is beyond the scope of the current work, but it should contain at least the following information:

- *Group ID*: This is the identity (e.g. GUID) of the permissioned-group. This value may be meaningful only to members of the permissioned-group.
- *Identity and address of the IdP-PI and IdP-PV*: This is the identity (e.g. X.509 certificate) and the bitcoin-address of both entities.

Although outside the scope of the current work, a less strict approach can be used for the block of transactions where a block is permitted to carry both ordinary Bitcoin transaction and permissioned-transactions. In this case, the reward for the a successful miner may be computed as a "pro rata" proportional to the number of permissioned-transactions in the block.

Finally, the mixed permisioned/permissionless mode of deployment maybe attractive to organizations who seek to run their own permissioned-group but who do not wish (or cannot afford) to deploy their own private blockchain consisting of a private peer-to-peer network of nodes.

In this mode of deployment, the private organization (as the owner of the ChainAnchor permissioned-group) can set their own reward structure for participants in the permissioned-group. If the reward for mining a permissioned-block is considerably higher than the reward of mining an ordinary Bitcoin block, a miner may opt to solely process permissioned-transactions. A miner may in fact participate in multiple permissioned-groups simultaneously, thereby increasing the overall income from mining these various permissioned-blocks of transactions.

V. Addressable Anonymous Internet Identities

The Bitcoin system currently does not provide "identities" in the sense of Internet identities that possesses the feature of *addressability* [28]. In Bitcoin an *address* is derived from a User's public key by simply passing the key through a SHA256 hash, and then passing the result through a RIPEMD160 hash. The result is a 160-bit value (i.e. 20 bytes) that is referred to as an "address" in Bitcoin. Sending currency to a User means sending a transaction instructing a transfer of the currency to that address. As such, the notion of a bitcoin-address only has semantic meaning within the Bitcoin system. That is, this 160-bit identifier is "addressable" (routable) only within the Bitcoin system.

Currently there are some new services within the Bitcoin ecosystem that offers linking between a user's Bitcoin address and an Internet identity (e.g. the user's Gmail address or Twitter handle). However, today most free email services require the user to submit additional personally-identifying information under the guise of "account recovery" information (e.g. a mobile phone number). As such, these services effectively remove the anonymity of the Bitcoin public key.

In order for ChainAnchor to support human-friendly identities, it must provide a mapping between the human-identifier at the digital identities layer and the bitcoin-addresses (or public keys) at the Blockchain layer (see Figure 5).

A. Degrees of Relationships in ChainAnchor

Figure 5 illustrates the degrees of relationships among identities in the ChainAnchor architecture:

(a) *User's known Internet identity and the IdP-PI*: In ChainAnchor when a human user seeks to participate within a permissioned group, he or she must be approved buy the group-owner (creator). This implies that some form of *personally-identifying information* (PII) must be disclosed from the user to the group-owner (and possibly the IdP-PI entity that is hosting the permissioned group for the owner).

As such in Figure 5(a) we assume that the Permissions Issuer (IdP-PI) entity has some PII about the user Alice (e.g. her mobile phone number) and can identify the human person (whose identity is `alice@idp-issuer.com` in the Figure). Indeed, today it is common practice for providers of free email services to request the user's phone

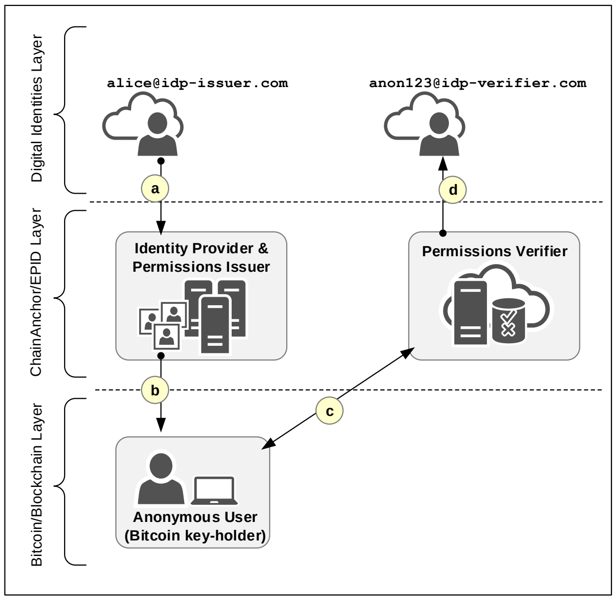


Fig. 5. Degrees of Identity Relationships in ChainAchor

number (and possibly location) at the time of account creation.

Thus, in Step-2 of Section III-C, The IdP-PI knows the true identity of the human User (Alice) – namely when the User obtains the Membership Verification Public Key K_{MVPK} from the IdP-IP, and when the User submits the *commitment* value that "blinds" the User's own secret keying material (Also see Equations 4 and 5 in Appendix A).

(b) *IdP-PI and the User's Bitcoin public key*: The anonymity of the User is realized (Figure 5(b)) when in Step-3B of Section III-C the User generates on his/her own *User-Member Private Key*, denoted as K_{UMPK}. (Also see Equation 6 in Appendix A).

The cryptographic *blinding* function that the User employed in Step 2 of the protocol prevents the Permissions Issuer (IdP-PI) entity from knowing that the key K_{UMPK} was generated by the person (Alice) whose Internet identity is `alice@idp-issuer.com` in Figure 5.

(c) *User's Bitcoin public key and the IdP-PV*: The Permissions Verifier (IdP-PV) entity has no way of correlating between the User's bitcoin key pair (which is self-generated by the User) and the Internet identity employed by the User (Figure 5(a)) to engage the Permissions Issuer entity.

When the User anonymously proves his or her membership to IdP-PV (in Step-4 of Section III-C, Equation 1), no personally-identifying information is given from the User to the IdP-PV. In the entire ChainAnchor protocol, the Internet identity of the User is never disclosed to the IdP-PV.

Thus the relationship in Figure 5(c) is truly an anonymous one. The IdP-PV only knows that the key-holder of the Bitcoin public key has successfully prove that he or she has been given permission to join the permissioned-group being operated by the IdP-PI as a service provider.

(d) *Derived anonymous Internet identity for the key-holder*: The cryptographically anonymous relationship between the Bitcoin key-holder (i.e. our User) and the IdP-PV in Figure 5(c) lends to the possible creation by the IdP-PV of a new anonymous Internet identity for the User. We treat this in the next section.

B. Derived Anonymous Internet Identities

As mentioned previously, the cryptographically anonymous relationship between the User and the IdP-PV in Figure 5(c) lends the IdP-PV to create an anonymous Internet identity for the User, as shown in Figure 5(d).

As a service to the anonymous User (Alice), the IdP-PV could create a random (but humanly readable) Internet identity for the User (e.g. `anon123@idp-verifier.com`) and maintain the association between the user's Bitcoin public key and this new derived identity. Thus, the user Alice now has a new anonymous Internet identity that she can use on the Internet – which is linked to her transaction public key (Bitcoin address) by the IdP-PV.

What makes this approach different from current systems that link a Bitcoin public key directly to an Internet identity (carrying PII) is the following:

- *Trust derived from the IdP-PI permissions*: In creating the new anonymous Internet identity, the Permissions Verifier is deriving trust from the Permissions Issuer entity who had admitted the User (Alice) to the permissioned-group (but without knowing Alice's Bitcoin public keys). Thus the IdP-PV has sufficient assurance that the User is a known user to the IdP-PI, and that he or she possesses some Internet identity that was either issued by the IdP-PI or was issued by another entity who was federated to the IdP-PI.

- *No personally identifying information is disclosed*: Since the IdP-PV is deriving trust from the IdP-PI who admitted the User into the permissioned-group, the IdP-PV does not need any PII from the User in order to create the new anonymous Internet identity for the User.

- *Anonymously addressable*: The Internet identity issued by the IdP-PV (e.g. `anon123@idp-verifier.com`) in Figure 5(d) is addressable and routable. It can be used like any other Internet identity, with the IdP-PV entity stepping-in as its validation-point when its legitimacy as an identity is challenged.

  That is, when a Relying Party (such as an online merchant) queries the status of the identity

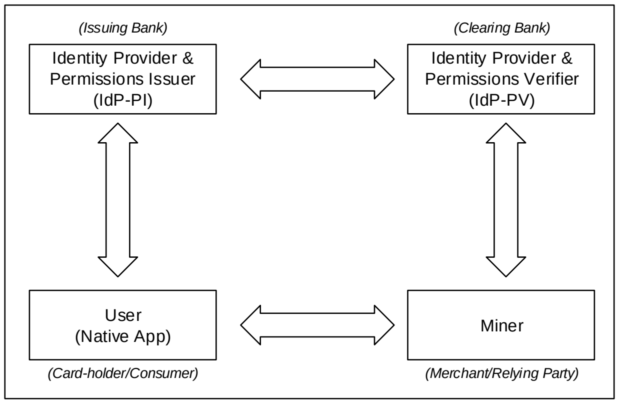


Fig. 6. ChainAnchor within the 4-Corners Model

`anon123@idp-verifier.com`, the RP can obtain a signed assertion (e.g. SAML2.0 assertion, or OIDC Claim) from the IdP-PV as the identity-issuer that attests to the truthfulness of this anonymous address. In this role, the IdP-PV is acting in its full capacity as an Identity Provider operating under some legal trust framework.

VI. Integration with Identity Management Protocols

In ChainAnchor we propose to integrate the EPID-related functions of the Permissions Issuer and Permissions Verifier with identity management functions and protocols found in Identity Provider (IdP) services today.

We use the term "identity management" in the sense of IdP as defined in *SAML2.0* [29] systems, and within the more recent *OpenID-Connect* [30] system and *User Managed Access* (UMA) system [31]. The later two systems belong to the same family due to their common underlying token-based authorization mechanism called OAuth2.0 [32].

To provide historical context, Figure 6 illustrates the 4-corners model of authentication and authorization, mapping ChainAnchor entities into that model. The 4-corners model emerged out of the credit-card industry in the 1970s and 1980s and provided a scalable processing model, where a card from an issuing bank could be processed by any merchant and any clearing bank. The model played an influential role in the PKI space in the mid-1990s and in the development of core concepts within identity management, notably SAML2.0 [29].

A. Integration with IdP Systems and Services

Integration with the current identity management protocols and systems brings a number of advantages:

- *Mature technology based on standards*: Identity management functionality today is a mature technology that is supported in both Enterprise systems and in consumer-facing services.

  This lends ChainAnchor to be embodied within an IdP server product for an intra-Enterprise scenario, or as a service offered by an IdP service provider.

- *Support for basic models of delegation*: Identity management technology today capture and technologically represent both session-based in-person authorizations (e.g. web browser single-sign-on (Web-SSO)) [33] that require the user to be present throughout the session, and also "delegated authorizations" from the user to an application (native application or web-based) that do not require the user to be present at all times [32].

  This lends ChainAnchor to be deployable in both scenarios. For example, ChainAnchor could be integrated into a software wallet application on the User's PC computer or mobile device. Alternatively, ChainAchor could be integrated into a hosted wallet web-application (operated by a third party) where the User would "delegate" the web-application to transact (on the Blockchain) on behalf of the User without needing the User to be present at all times.

- *Existence of Legal Trust Frameworks*: Federated identity management is a reality today within market verticals and related industries. These are typically governed by a Legal Trust Framework (LTF) agreements that clearly delineate the right and obligations of entities participating in the federation ecosystem. Examples of LTF agreements are FICAM [34] and SAFE-BioPharma [35]. Note the X509 Certificate Practices Statement (CPS) agreements are also a form of a legal trust framework.

  This is relevant for ChainAnchor because we believe that a scalable deployment of anonymous verifiable identities will require extensions to the current LTF that delineates the roles and legal obligations of each entity in the system. Extending existing LTF agreements to include EPID-functions represents a more achievable goal that creating a new legal trust framework solely for EPID.

- *Existence of CAs for hardware-certificates*: Although Certificate Authorities (CAs) have "dropped out of favor" in the public mind in recent years, CAs and X509 certificates remain a crucial infrastructure component for the Internet today and into the future. This infrastructure will be needed should ChainAnchor be deployed with the TPMv1.2 hardware.

B. Goals in Integration

In integrating ChainAnchor to identity management entities, we seek to fulfill the following objectives:

- *Extend IdP functions to support blockchain identities.* Provide new protocols and layer-bindings to allow

Identity Providers to manage blockchain identities, including Bitcoin addresses.

- *Support Management of Permissioned Groups by IdPs*. Allow Identity Providers to support the function of creating and managing *Permissioned Groups* (e.g. as a service), independent of the underlying blockchain technology (i.e. current Bitcoin/Blockchain or future blockchain ledgers).

  This allows Enterprises and other organizations to outsource part or all of their private-blockchain management to Identity Providers that implement ChainAnchor.

- *Support federation between IdP-PI and IdP-PV entities*. Extend the the current identity-federation protocols to carry EPID-related parameters and configuration metadata between IdP-PI and IdP-PV entities.

- *Support attribute assignments*. Support the binding between anonymous identities in ChainAchor with attributes issued by external *Attribute Authorities* (or *Attribute Providers*).

C. UMA Model for Identity & Resource Protection

In integrating the Permissions Issuer and Permissons verifier with identity management functions of the IdP, we are proposing to follow the *User Managed Access* v1.0 (UMA) standard [31] for distributed authorization. This is because the UMA standard is a broader super-set of the OIDC1.0 [30] and OAuth2.0 [32] standards. The term "resource" in UMA includes objects (e.g. files, URLs, configurations, etc) as well as RESTful API Endpoints.

Figure 7 shows ChainAnchor entities within the context of the UMA distributed authorization model:

- *Resource Owner creates permissions group* (Figure 7(a)): The group-creator requests IdP-PI to create a permissioned-group, and the Creator selects one or more Permissions verifier entities IdP-PV. In UMA terminology, the group creator/owner is referred to as *Resource Owner* (RO).

  The RO selects the protection levels required (e.g. authentication methods), as well as the access-policies to be applied to the resource. The resource in this case includes the Group-Member Keying Parameters (that the User must obtain from IdP-PI in Step 3A of Section III-C) and the *API End-Point* at the IdP-PV to which the User executes the anonymous membership proof protocol.

- *Client requests access to Group-Member Keying Parameters* (Figure 7(b)): Here the User running the Native Client requests access to a resource (namely the the Group-Member Keying Parameters) that is sitting at the Resource Server (RS#1) and is protected by Authorization Server (AS#1) owned by the IdP-PI. The Client must first obtain authorization from AS#1, which may also include an authentication step.

  Depending on the business model of the IdP-PI entity, the Resource Server (RS#1) could be owned and operated by the IdP-PI or it could owned by the Group Owner (as the RO). This may be a subtle difference, but this allows the IdP-PI some flexibility in offering its services. For example, a corporate customer might want to operate its own Resource Server RS#1 in-house (containing the sensitive Group-Member Keying Parameters belonging to the corporation), but may wish to outsource the authorization tasks to a commercial IdP-PI service running AS#1.

- *Client requests access to Membership Verification Endpoint at IdP-PV* (Figure 7(c)): Here is the User running the Native Client seeks to anonymously prove its membership of the permissions group to the IdP-PV. As such, the Client is requesting access to the *Membership Verification Endpoint* at Resource Server RS#2 at the IdP-PV. Since the endpoint itself is a protected resource, the Client needs to obtain authorization from Authorization Server AS#2 that is protecting RS#2.

  Note that the Verified Identities Database is another protected resource at Resource Server RS#2 at the IdP-PV. As such, a Miner in ChainAchor who wishes to verify that a Bitcoin public key is in the Verified Identities Database would need to obtain access authorization from the Authorization Server AS#2 belonging to the IdP-PV.

Depending on the configuration of the Authorization Server AS#2 and the Resource Server RS#2 at the IdP-PV, the Membership Verification Endpoint could in fact be located at the AS#2. In other words, the Resource Server function at the IdP-PV entity could be co-located at (or collapsed into) the Authorization Server AS#2.

D. User choosing Permissions Verifier entity

In ChainAnchor as in EPID and DAA, it is important that the Permissions Verifier IdP-PV entity is distinct from the Permissions Issuer IdP-PI, both in terms of legal services as well as implementation. This is to avoid collusion on the part of the IdP-PI and the IdP-PV, who together may wish to sell "bogus" ChainAchor services at the expense of the Group Creator/Owner and the Miners who participate in the permissioned-group.

As such, in ChainAnchor a User is given the power to choose the Permissions Verifier that the User trusts (e.g. by reputation or by independence). How the User selects the IdP-PV service provider is a business model that is beyond the scope of the current work.

Once the user finds a suitable Permissions Verifier IdP-PV, the User must get the IdP-PI to "publish" (i.e. give) a copy

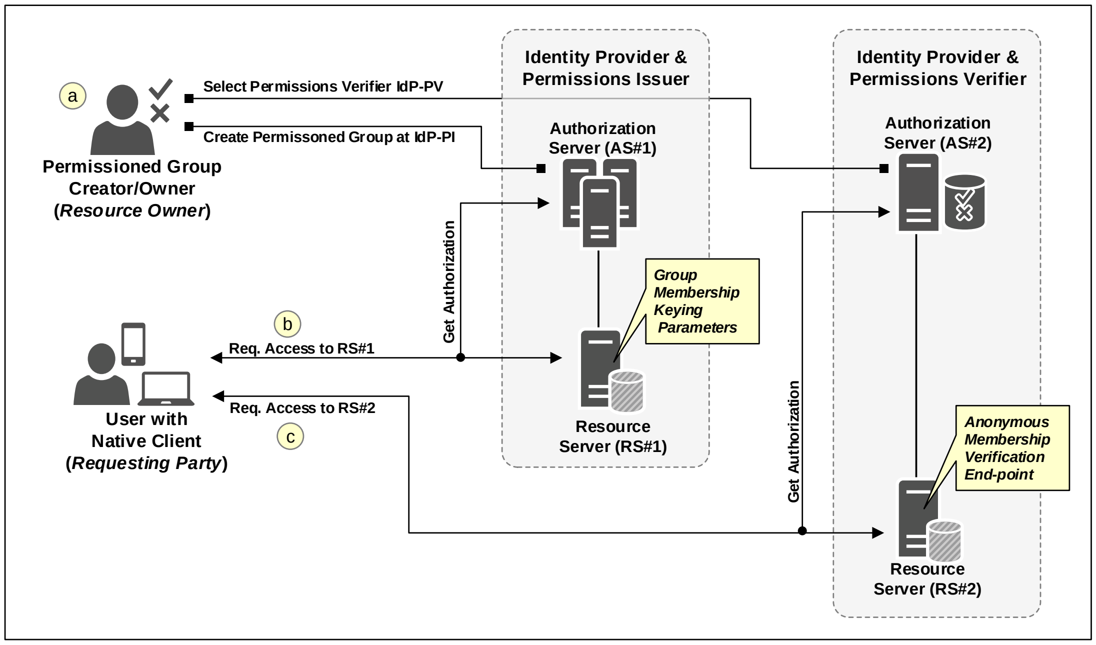

Identity Provider & Permissions Issuer
Identity Provider & Permissions Verifier
a
Select Permissions Verifier IdP-PV
Create Permissioned Group at IdP-PI
Permissioned Group Creator/Owner (Resource Owner)
Authorization Server (AS#1)
Authorization Server (AS#2)
Get Authorization
Group Membership Keying Parameters
b
Req. Access to RS#1
Req. Access to RS#2
c
User with Native Client (Requesting Party)
Resource Server (RS#1)
Get Authorization
Anonymous Membership Verification End-point
Resource Server (RS#2)

Fig. 7. Mapping IdP-PI and IdP-PV to the *User Managed Access* (UMA) Model

of the Membership Verification Public Key (K_{MVPK}) to the IdP-PV. This is Step 1 of ChainAnchor in Section III-C.

In ChainAnchor we propose to extend the OIDC [30] exchange (authorization-code grant sequence) to implement this phase. See Figure 8.

- *User authentication using Internet identity*:
  In messages (1) to (2) of Figure 8 the User chooses the Permissions-Group (denoted as `group_id`) that he or she wishes to join. The User requests (or triggers) the Permissions Verifier to obtain the Membership Verification Public Key pertaining to the desired group.
  Note that here the User uses his or her Internet identity (e.g. `alice@freemail.com`) to connect to the User's account at the IdP-PV entity. The User does not release any information about their Bitcoin keys.

- *User authorize IdP-PI to deliver public key K_{MVPK}*:
  In messages (3) to (5) of Figure 8 the Permissions Issuer requests the User to authenticate himself or herself to the Permissions Issuer and at the same time "authorize" the Permissions Issuer to release a copy of the Membership Verification Public Key to the Permissions Verifier.
  The "authorization" construct used here is the mechanism used by OAuth2.0 to permit one entity to obtain access to a resource (which in this case is the relevant key K_{MVPK}). In OAuth2.0 the process takes two steps, where an authorization `code` is issued to a recipient (here the IdP-PV), who then swaps it for a token proper.

- *IdP-PV swaps code for tokens*:
  Following standard OIDC flows, in messages (6) to (7) of Figure 8 the Permissions Verifier (acting as the Client to the Permissions Issuer) obtains two tokens, namely the `access_token` and the `id_token`. These two tokens are part of the OIDCv1.0 standard.
  In ChainAnchor we extend the protocol by adding a third token (`epid_token`) for the purpose of delivering the Membership Verification Public Key K_{MVPK}.

- *IdP-PV uses EPID-token to get public key K_{MVPK}*:
  ChainAnchor extends OIDC by introducing two additional messages (8) and (9) and a new end-point (called the `EPID-Info` end-point).
  In messages (8) and (9) the Permissions Verifier uses its `epid_token` to obtain access to the desired resource (namely the key K_{MVPK}) by presenting the token to the relevant `EPID-Info` end-point at the Permissions Issuer.

At the end of the above exchange, the Permissions Verifier is in possession of the Membership Verification Public Key K_{MVPK} for the relevant permissioned-group.

E. User proving anonymous membership

There are a number of ways to implement the anonymous membership verification protocol (Step 4 of Section III-C). Here we describe two promising avenues for implementing this step:

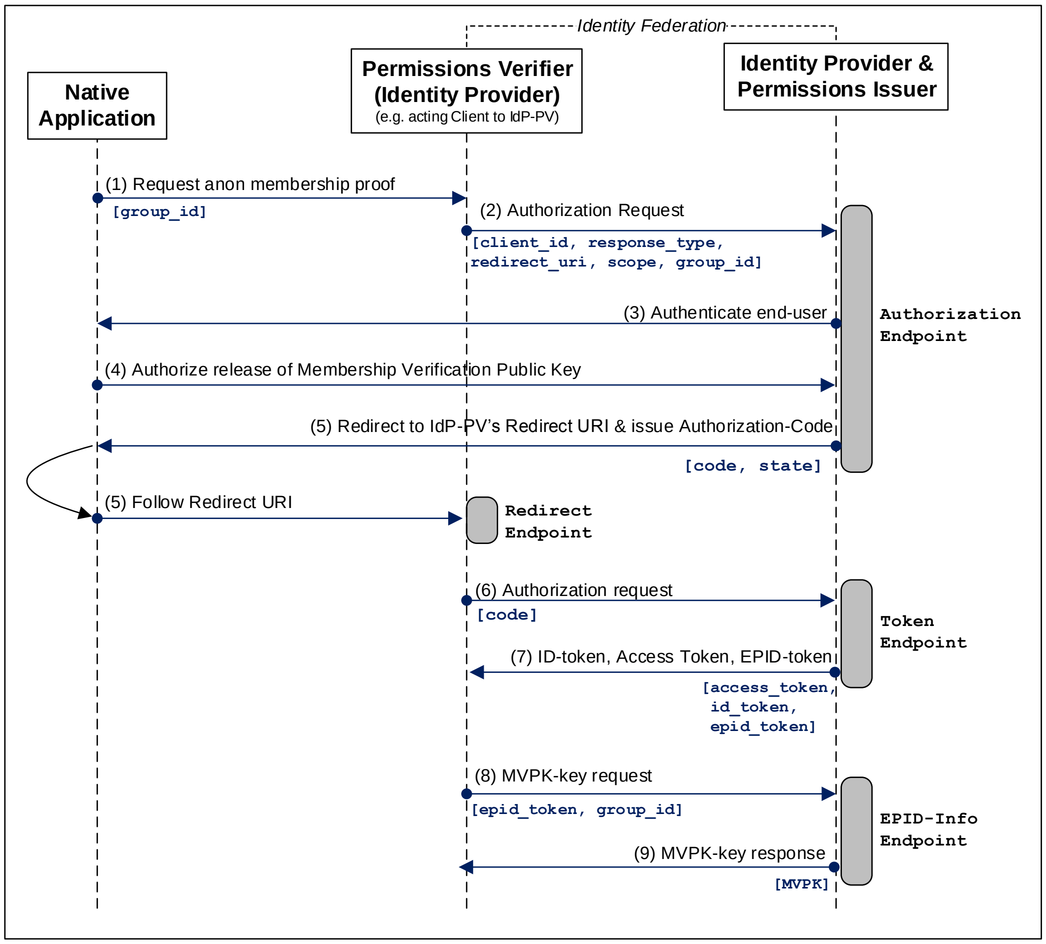

Identity Federation
Native Application
Permissions Verifier (Identity Provider) (e.g. acting Client to IdP-PV)
Identity Provider & Permissions Issuer
(1) Request anon membership proof
[group_id]
(2) Authorization Request
[client_id, response_type, redirect_uri, scope, group_id]
Authorization Endpoint
(3) Authenticate end-user
(4) Authorize release of Membership Verification Public Key
(5) Redirect to IdP-PV's Redirect URI & issue Authorization-Code
[code, state]
(5) Follow Redirect URI
Redirect Endpoint
(6) Authorization request
[code]
Token Endpoint
(7) ID-token, Access Token, EPID-token
[access_token, id_token, epid_token]
(8) MVPK-key request
[epid_token, group_id]
EPID-Info Endpoint
(9) MVPK-key response
[MVPK]

Fig. 8. Using OIDC/OAuth2.0 to publish Membership Verification Public Key

- *REST-based approach*: In this implementation approach the Client (operated by the User) goes through the steps of Figure 3 using a REST-based request-response message format (i.e. set of `GET`, `PUT`, `POST`, and `DELETE` messages).

  Here the Constrained Application Protocol (CoAP) [36] may provide a suitable transport implementation for these REST messages. Firstly, CoAP supports different types of payloads, including JSON [37] payloads which is the format used in the JSON Web Tokens (JWT) [38] deployed in OIDC and UMA. Furthermore standard methods exists for JWT tokens signing [39] and encryption [40].

  Secondly, CoAP supports constrained devices which the User may be using (e.g. mobile phones, specialized wallet hardwares, etc). As such, choosing to implement the ChainAnchor anonymous membership verification protocol in a CoAP-friendly manner may have benefits in terms of expanded use-cases and broader types of devices.

- *SSL cipher-suite approach*: In this implementation approach the anonymous membership verification exchanges are implemented as part of the SSL handshake between the Client and the IdP-PV. This is the approach taken in the SIGMA protocol [26] and its variant SIGMA-CE implementation for constrained environments.

  In this approach, a new SSL/TLS cipher suite must be defined and must be implemented by both the Client and the Authorization Server (or Resource Server) that provides the Membership Verification Endpoint.

VII. Conclusions & Further Work

In this paper we have proposed the ChainAnchor system that allows a User (i.e. holder of a transaction key-pair) to prove anonymously that the User is a member of a *permissioned* blockchain, and therefore have his or her transactions be processed and be added to the permissioned blockchain. ChainAnchor makes use of the zero-knowledge-proof protocol of the underlying EPID scheme that is part of ChainAnchor.

The User has to cryptographically “bind” the User’s transaction public-key (i.e. Bitcoin public-key) to the zero-knowledge proof sent by the User to a Permissions Verifier entity. Users who successfully prove their membership of the permissioned blockchain to the Permissions Verifier will have their transaction public key added to the Verified Identities Database operated by the the Permissions Verifier. The database contains the transaction public-keys of Users who have successfully prove their membership using the zero-knowledge proof protocol, together with the time-stamp of the proof. No other identifying information is stored by the Permissions Verifier. Similarly, a Miner (mining node) who wishes to participate in ChainAnchor must perform the same zero-knowledge proof process and have its public key be added to the database.

We suggest two modes of deployment, with differing possible incentive models. In the first deployment mode – namely the *homogenous permisioned nodes* – all of the nodes participating in the peer-to-peer network are ChainAnchor nodes. This is equivalent to a fully private blockchain, where a private organization operates all nodes of a private peer-to-peer network.

In the second deployment mode – called the *mixed permisioned-permissionless nodes* – ChainAnchor is deployed as an overlay above the current permissionless public Blockchain. Thus the peer-to-peer network is a heterogenous mix of Bitcoin nodes and ChainAnchor nodes, with the plain Bitcoin mining nodes being oblivious to permissioned-transactions that are part of a permissioned-group.

When a Miner chooses to work as a ChainAnchor miner, it must create blocks of transactions that are purely permissioned-transactions. The goal here is not to create a separate chain, but rather use the current permissionless Blockchain to carry permissioned-transactions originating from Users in ChainAnchor. ChainAnchor miners are rewarded higher for performing proof-of-work on a permissioned-block, with the source of remuneration coming from the owner of the permissioned group. The node performs the same proof-of-work as in the current Bitcoin, and the same consensus mechanism is used.

This mode of deployment has the advantage that it allows an organization to overlay their own permissioned group atop the public permissionless Blockchain, while maintaining the anonymity of its group-members. This obviates the need for the organization to operate their own private peer-to-peer network of nodes, which maybe costly and have a lower level of resilience against collusions. This mode also has the advantage that it permits a private organization to recruit miners who are willing to process permissioned-blocks on an exclusive basis, and remunerate these miners at a higher rate.

Finally, we have proposed to integrate the ChainAnchor zero-knowledge-proof protocol with the identity management protocols found in Identity Provider (IdP) services today (notably OpenID-Connect and UMA). There are multiple advantages to integrating with standard identity management protocols. These include a high degree of maturity of the protocols, the availability of mechanisms for delegation of authorizations, and the existence of legal trust frameworks that can be extended to support the roles and responsibilities of entities within a ChainAnchor deployment.

The current design of ChainAnchor fulfills the objectives set at the start of the current paper. These objectives include retaining the same degree of anonymity as is currently provided for in Bitcoin, providing anonymous permission verifiability, and achieving functional independence from the current Blockchain in the Bitcoin system.

Looking ahead, there are number of features or aspects that we plan to address:

- *Support for Anonymous Attribute-Groups in IdP*: ChainAnchor allows a user to prove that he or she is a member of an *attribute-group* – which is simply a permissioned group whose members are users who posses a given attribute (also called *assertions* in SAML2.0 or *claims* in OpenID-Connect).

  The User must obtain evidence of an attribute (e.g. “Residence of Massachusetts”, ”Age over 18”, etc) from external sources or attribute authorities (e.g. bank, transportation authority, school, etc). The User then presents these assertions to the Permissions Issuer (IdP-PI) entity when the User seeks to obtain the group-member keying material from the IdP-PI. The User can proves to the IdP-PV that he or she is a member of the “Age over 18” group.

  This approach – though much less sophisticated than the approach in [23] – provides a practical on-ramp for Identity Providers who wish to support anonymous attribute-groups without departing from the ChainAnchor/EPID scheme.

- *RESTful design for zero-knowledge proof protocol*: We plan to address the issue of implementing the EPID zero-knowledge proof protocol within a RESTful exchange. This would allow ChainAnchor to be deployed using different transport protocols (e.g. HTTP, CoAP, et).

- *Support for Anti-Money Laundering* (AML): ChainAnchor in the *mixed permisioned-permissionless* mode can be used for AML purposes. This would need proactive voluntary disclosure from the User since it is the User who self-generated his/her Bitcoin public key pair. Here ChainAnchor would be used to establish a permissioned-group for “verified” (AML-friendly) and “unverified” transactions.

  This feature may be attractive to concerned citizens who may wish to see Bitcoin grow over time but who may wish to reduce the amount of laundered currency passing through the Bitcoin network. Using ChainAnchor such users can remain anonymous but have the option to disclose their identity (i.e. disclose the link between derived anonymous Internet identity and Bitcoin keys) when legally challenged regarding a suspicious transaction. Such legal challenges should come from the appropriate authorities (e.g. US Treasury Department).

Note that when a user discloses one of their ChainAnchor public key pairs it does not affect the user's remaining public key pairs that are also part of the same permissioned group.

Appendix A
Summary of EPID

The EPID Scheme consists of a number of protocols or phases leading to a user proving his/her membership in a given group. In the following we summarize the RSA-based EPID scheme as defined in [2].

A. Issuer Setup

In order to create a group membership verification instance, the Issuer must choose a *Group Public Key) and compute a corresponding* Group-Issuing Private Key).

For the Group-Issuing Private Key the Issuer chooses an RSA modulus $N = p_N q_N$ where $p_N = 2p\prime_N + 1$ and $q_N = 2q\prime_N + 1$ and where p_N, p_N, $p\prime_N$ and $q\prime_N$ are all prime.

The Group Public Key for the particular group instance will be:

$$(N, g\prime, g, h, R, S, Z, p, q, u) \tag{2}$$

The Group Issuing Private Key (corresponding to the Group Public Key) is denoted as:

$$(p\prime_N, q\prime_N) \tag{3}$$

which the Issuer keeps secret).

In order to communicate securely with a User, the Issuer is assumed to possess the usual long-term public key pair denoted as $(K_I, {K_I}^{-1})$, where K_I is publicly know in the ecosystem.

Any User who has a copy of the Group Public Key $(N, g\prime, g, h, R, S, Z, p, q, u)$ can verify this public key by checking the following:

- Verify the proof that $g, h \in \langle g\prime \rangle$ and $R, S, Z \in \langle h \rangle$.
- Check whether p and q are primes, and check that $q \mid (p-1)$, $q \not| \frac{(p-1)}{q}$ and $u^q \equiv 1 \pmod p$
- Check whether all group public key parameters have the required length.

B. Join Protocol: User and Issuer

In the join protocol, a given User seeks to send to the Issuer the pair (K, U) which are computed as follows.

- The User chooses a secret f and seeks to convey to the Issuer a *commitment* to f in the form of the value U.
- The value U is computed as

$$U = R^f S^{v\prime} \tag{4}$$

  where $v\prime$ is chosen randomly by the User for the purpose of *blinding* the chosen f.
- Next the User computes

$$K = {B_I}^f \pmod p \tag{5}$$

  where B_I is derived from the *basename* of the Issuer (denoted as bsn_I).

The goal here is for the User to send (K, U) to the Issuer and to convince the Issuer that the values K and U are formed correctly.

In the above Equation 5, a User chooses a base value B and then uses it to compute K. The purpose of the (B, K) pair is for a revocation check. We refer to B the *base* and K as the *pseudonym*. To sign an EPID-signature, the User needs to both prove that it has a valid membership credential and also prove that it had constructed the (B, K) pair correctly, all in zero-knowledge.

In EPID and DAA, there are two (2) options to compute the base B:

- *Random base*: Here B is chosen randomly each time by the User. A different base used every time the EPID-signature is performed. Under the decisional Diffie-Hellman assumption, no Verifier entity will be able to link two EPID-signatures using the (B, K) pairs in the signatures.

- *Named base*: Here B is derived from the Verifier's basename. That is, a deterministic function of the name of the verifier is used as a base. For example, B could be a hash of the Verifier's basename. In this named-base option, the value K becomes a "pseudonym" of the User with regard to the Verifier's basename. The User will always use the same K in the EPID-signature to the Verifier.

C. Issuer generates User's Membership Private Key

In response, the Issuer performs the following steps:

- The Issuer chooses a random integer $v\prime\prime$ and a random prime e.
- The Issuer computes A such that

$$A^e U S^{v''} \equiv Z \pmod p$$

- The Issuer sends the User the values $(A, e, v\prime\prime)$.

Note that the CL-signature [4] on the value f is $(A, e, v := v\prime + v\prime\prime)$. As such, the User then sets his/her Membership Private Key as:

$$(A, e, f, v) \tag{6}$$

where $v := v\prime + v\prime\prime$. Recall that f is the secret chosen by the User at the start of the Join protocol.

D. User proving valid membership

When a User seeks to prove that he or she is a group member, the User interacts with the Verifier entity. This is performed using the Camenisch-Lysyanskaya (CL) signature [4] on some value f.

This can be done using a zero-knowledge proof of knowledge of the values f, A, e, and v such that

$$A^e R^f S^v \equiv Z \pmod{N} \tag{7}$$

The User also needs to perform the following:

- The User computes $K = B^f \pmod{p}$ where B is a random base (chosen by the User).
- The User reveals B and K to the Verifier.
- The User proves to the Verifier that the value $\log_B K$ is the same as in his/her private key (see Equation 5).

In proving membership to the Verifier, the User as the prover needs to send the Verifier the value

$$\sigma = (\sigma_1, \sigma_2, \sigma_3) \tag{8}$$

where each of the values are as follows:

- σ_1: The value σ_1 is a "signature of knowledge" regarding the User's commitment to the User's private key and that K was computed using the User's secret value f.
- σ_2: The value σ_2 is a "signature of knowledge" that the User's private key has not been revoked by the Verifier (i.e. not present in the signature revocation list sig-RL (see section below on Revocations)).
- σ_3: The value σ_3 is a "signature of knowledge" that the User's private key has not been revoked by the Issuer (i.e. not present in the issuer revocation list Issuer-RL (see section below on Revocations)).

E. Revocations

The EPID scheme supports three (3) revocation schemes:

- *Private-key based revocation (priv-RL)*:
  The first is based on a revocation-list (RL) of the private key belonging to the User. If a User's private key (A, e, f, v) (see Equation 6) is compromised, the User's f is then placed on the revocation list. As such, this revocation scheme is referred to as the *priv-RL* revocation scheme.
- *Signature based revocation (sig-RL)*:
  If a Verifier receives a signature from a User and determines that the the User was compromised, the Verifier places the (B, K) values of the signature on the signature-based revocation list (where $\log_B K$ is the secret of the compromised User).

  When a User seeks to prove that he or she is not on the sig-RL revocation list, the User (with private key (A, e, f, v)) needs not only to show that $A^e R^f S^v \equiv Z \pmod{N}$ (see Equation 7), but also to prove that his/her current value f (part of his/her private key) is not in the sig-RL revocation list.

  That is, his/her value f must be shown to be different from $\log_{\hat{B}} \hat{K}$, for every $(\hat{B}, \hat{K})$ pair listed in the sig-RL revocation list.
- *Issuer based revocation (Issuer-RL)*:
  The Issuer-based revocation addresses the case where the Issuer takes the proactive step of removing (i.e. revoking) a User form a given group. The Issuer might do so, for example, when it sees that a User has left the group (e.g. no activity detected).

  To revoke a User, the Issuer places the K value that the User submitted to the Issuer (see Equation 5 in the Join protocol) on the Issuer-RL revocation list. Note that $\log_{B_I} K$ is the secret of the revoked User

  When a User seeks to prove membership, he/she must prove that their secret f is not on the Issuer-RL revocation list. That is, the User must prove that their f is different from $\log_{B_I} \hat{K}$ for each $\hat{K}$ present in the Issuer-RL revocation list.

References

[1] S. Nakamoto, "Bitcoin: A Peer-to-Peer Electronic Cash System." [Online]. Available: https://bitcoin.org/bitcoin.pdf

[2] E. Brickell and J. Li, "Enhanced Privacy ID: a Direct Anonymous Attestation Scheme with Enhanced Revocation Capabilities," *IEEE Transactions on Dependable and Secure Computing*, vol. 9, no. 3, pp. 345–360, 2012.

[3] BitFury Group, "Public versus Private Blockchains – Part 1: Permissioned Blockchains," BitFury Group, White Paper – v1.0, October 2015, available at http://bitfury.com/white-papers-research.

[4] J. Camenisch and A. Lysyanskaya, "A Signature Scheme with Efficient Protocols," in *Security in Communication Networks (SCN2002) (LNCS 2576)*, S. Cimato, G. Persiano, and C. Galdi, Eds. Springer, 2002, pp. 268–289.

[5] E. Brickell, J. Camenisch, and L. Chen, "Direct Anonymous Attestation," in *Proceedings of the 11th ACM Conference on Computer and Communications Security CCS2004*. ACM, 2004, pp. 132–145.

[6] E. Brickell and J. Li, "Enhanced Privacy ID from Bilinear Pairing for Hardware Authentication and Attestation," in *Proceedings of the IEEE International Conference on Social Computing (SocialCom 2010)*. IEEE, 2010, pp. 768–775.

[7] D. Boneh, X. Boyen, and H. Shacham, "Short Group Signatures," in *In Advances in Cryptology - Proceedings CRYPTO ?04 (LNCS 3152)*. Springer, 2004, pp. 41–55.

[8] D. Boneh and H. Shacham, "Group signatures with verifier-local revocation," in *Proceedings of the 11th ACM conference on Computer and communications security*. ACM, 2004, pp. 168–177.

[9] Trusted Computing Group, "TPM Main – Specification Version 1.2," Trusted Computing Group, TCG Published Specification, October 2003, http://www.trustedcomputinggroup.org/ resources/ tpm_main_specification.

[10] ——, "TPM Main – Part 1 Design Principles – Specification Version 1.2," Trusted Computing Group, TCG Published Specification, October 2003, http://www.trustedcomputinggroup.org/ resources/ tpm_main_specification.

[11] Microsoft Corp, "Trusted Platform Module and Bitlocker Drive Encryption," https://msdn.microsoft.com/en-us/library/windows/hardware/dn653315.

[12] Trusted Computing Group, "TCG Storage Security Subsystem Class: Opal (v1.0)," Trusted Computing Group, TCG Published Specification, January 2009, http://www.trustedcomputinggroup.org/ resources.

[13] T. Hardjono, "Infrastructure for Trusted Computing," in *Proceedings of ACSAC Workshop on Trusted Computing*, December 2004, available at https://www.acsac.org/2004/workshop/Thomas-Hardjono.pdf.

[14] T. Hardjono and N. Smith (Eds), "TCG Infrastructure Reference Architecture for Interoperability (Part 1) – Specification Version 1.0 Rev 1.0," June 2005, http://www.trustedcomputinggroup.org/ resources.

[15] ——, “TCG Infrastructure Working Group Architecture (Part 2) – Integrity Management – Specification Version 1.0 Rev 1.0,” November 2006, http://www.trustedcomputinggroup.org/ resources.

[16] ISO/IEC, “Information Technology – Security Techniques – Anonymous Digital Signatures,” International Organization for Standardization (ISO), International Standard ISO/IEC 20008-1, November 2013.

[17] ——, “Information Technology – Security Techniques – Anonymous Entity Authentication,” International Organization for Standardization (ISO), International Standard ISO/IEC 20009-1, August 2013.

[18] Trusted Computing Group, “TCG Interoperability Specifications for Backup and Migration Services (v1.0),” Trusted Computing Group, TCG Published Specification, June 2005, http://www.trustedcomputinggroup.org/ resources.

[19] T. Hardjono and G. Kazmierczak, “Overview of the TPM Key Management Standard,” 2008, available on http://www.trustedcomputinggroup.org/ files/ resource_files/.

[20] J. Camenisch and E. Van Herreweghen, “Design and implementation of the Idemix anonymous credential system,” in *Proceedings of the 9th ACM conference on Computer and communications security*. ACM, 2002, pp. 21–30.

[21] Microsoft, “U-Prove Cryptographic Specification v1.1 (Rev 3),” Microsoft Corporation, http://research.microsoft.com/en-us/projects/u-prove, 2014.

[22] L. Chen and J. Li, “Flexible and Scalable Digital Signatures in TPM 2.0,” in *Proceedings of the 2013 ACM SIGSAC Conference on Computer and Communications Security CCS2013*. ACM, 2013, pp. 37–48.

[23] L. Chen and R. Urian, “DAA-A: Direct Anonymous Attestation with Attributes,” in *Proceedings of TRUST 2015 (LNCS 9229)*. Springer, 2013, pp. 228–245.

[24] R. Housley, W. Polk, W. Ford, and D. Solo, “Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (crl) Profile,” RFC 3280 (Standards Track), April 2002.

[25] M. Myers, R. Ankney, A. Malpani, S. Galperin, and C. Adams, “X.509 Internet Public Key Infrastructure Online Certificate Status Protocol (OCSP),” RFC 2560(Standards Track), June 1999.

[26] J. Walker and J. Li, “Key Exchange with Anonymous Authentication using DAA-SIGMA Protocol,” in *Trusted Systems (INTRUST2010) (LNCS 6802)*, L. Chen and M. Yung, Eds. Springer, 2010, pp. 108–127.

[27] R. Housley, W. Ford, W. Polk, and D. Solo, “Internet X.509 Public Key Infrastructure Certificate and CRL Profile,” RFC 2459 (Proposed Standard), Internet Engineering Task Force, Jan. 1999, obsoleted by RFC 3280. [Online]. Available: http://www.ietf.org/rfc/rfc2459.txt

[28] D. Crocker, J. Vittal, K. Pogran, and D. Henderson, “Standard for the format of ARPA network text messages,” RFC 733, Internet Engineering Task Force, Nov. 1977, obsoleted by RFC 822. [Online]. Available: http://www.ietf.org/rfc/rfc733.txt

[29] OASIS, “Assertions and Protocols for the OASIS Security Assertion Markup Language (SAML) V2.0,” http://docs.oasis-open.org/security/saml/v2.0/saml-core-2.0-os.pdf, March 2005.

[30] N. Sakimura, J. Bradley, M. Jones, B. de Medeiros, and C. Mortimore, “OpenID Connect Core 1.0,” OpenID Foundation, Technical Specification v1.0 – Errata Set 1, November 2014, http://openid.net/specs/openid-connect-core-1_0.html.

[31] T. Hardjono, E. Maler, M. Machulak, and D. Catalano, “User-Managed Access (UMA) Profile of OAuth2.0 – Specification Version 1.0,” April 2015, https://docs.kantarainitiative.org/uma/rec-uma-core.html.

[32] D. Hardt, “The OAuth 2.0 Authorization Framework,” RFC 6749 (Proposed Standard), Internet Engineering Task Force, Oct. 2012. [Online]. Available: http://www.ietf.org/rfc/rfc6749.txt

[33] OASIS, “ Profiles for the OASIS Security Assertion Markup Language (SAML) V2.0,” http://docs.oasis-open.org/security/saml/v2.0/saml-profiles-2.0-os.pdf, March 2005.

[34] US General Services Administration, “U.S. Federal Identity, Credential and Access Management (FICAM) Program,” 2013, http://info.idmanagement.gov.

[35] SAFE-BioPharma Association, “Trust Framework Provider Services,” 2014, http://www.safe-biopharma.org/SAFE_Trust_Framework.htm.

[36] Z. Shelby, K. Hartke, and C. Bormann, “The Constrained Application Protocol (CoAP),” RFC 7252 (Proposed Standard), Internet Engineering Task Force, Jun. 2014. [Online]. Available: http://www.ietf.org/rfc/ rfc7252.txt

[37] T. Bray, “The JavaScript Object Notation (JSON) Data Interchange Format,” RFC 7159 (Proposed Standard), Internet Engineering Task Force, Mar. 2014. [Online]. Available: http://www.ietf.org/rfc/rfc7159.txt

[38] M. Jones, J. Bradley, and N. Sakimura, “JSON Web Token (JWT),” RFC 7519 (Proposed Standard), Internet Engineering Task Force, May 2015. [Online]. Available: http://www.ietf.org/rfc/rfc7519.txt

[39] ——, “JSON Web Signature (JWS),” RFC 7515 (Proposed Standard), Internet Engineering Task Force, May 2015. [Online]. Available: http://www.ietf.org/rfc/rfc7515.txt

[40] M. Jones and J. Hildebrand, “JSON Web Encryption (JWE),” RFC 7516 (Proposed Standard), Internet Engineering Task Force, May 2015. [Online]. Available: http://www.ietf.org/rfc/rfc7516.txt