Anonymous Identities for Permissioned Blockchains

(DRAFT v06C – 1/24/2016 – Please Do Not Distribute)

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.

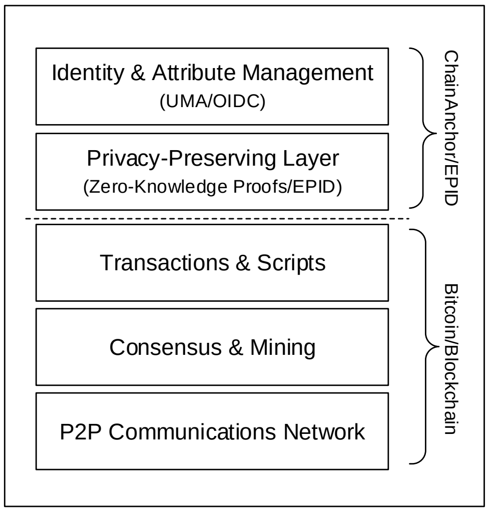

Fig. 1. ChainAnchor Layers

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 the blockchains. ChainAnchor adds an *anonymous identity verification* step using the EPID zero-knowledge scheme [2] to a permissioned blockchain that limits access to the blockchain only to verified members.

As a deployment mode of ChainAnchor, we introduce the notion of *semi-permissioned blockchains* where only members will have their transactions processed, but where these transactions can be publicly validated. We use this term 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 his/her private-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 (capital "B") in the Bitcoin system as the backdrop for our discussions regarding ChainAnchor, even though ChainAnchor is aimed at permissioned (private) blockchain. This is because the current Blockchain is the only operational blockchain system that has achieved the scale of several thousand nodes in a true peer-to-peer (P2P) network topology. The Bitcoin system and its Blockchain has received much attention, and therefore well understood compared to other proposed blockchain systems.

As such, designing ChainAnchor following the operational model of the public permissionless Blockchain will allow ChainAnchor to be used more readily in the more restricted permissioned (private) blockchain systems – which may have a different transaction/block format and may have different consensus algorithms. Hence our *functional independence* design requirement and our use of the term *transaction keys* generically to refer to public-key pair that the user self-issues and employs.

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

In Section III we present the proposed ChainAnchor architecture and protocols in detail. The cryptographic protocols are summarized in the Appendix. As such, it is useful to read Section III in conjunction with the Appendix.

Section IV presents two deployment modes of ChainAnchor, namely the private permissioned deployment mode and the semi-permissioned (overlay) deployment mode. We also briefly discuss some possible remuneration models for Miner who participate in ChainAnchor.

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 using the zero knowledge proof protocols and mechanisms of the *Enhanced Privacy ID* (EPID) scheme [2]. This scheme itself is an extension of the *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 Architecture

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 a *Permissioned Group* and therefore has the privilege to have his/her transactions processed and be added to the permissioned-blockchain. We refer to this zero-knowledge proof as the *anonymous identity verification* proof. The permissioned-group is created by a *Permissions Issuer* entity on behalf of the group Owner (i.er. an organization or person).

In the anonymous identity verification, the User has to also cryptographically "bind" his or her transaction public-key to the zero-knowledge proof sent to Permissions Verifier. This results in that transaction public-key being recognized as a valid "identity" that has obtained 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. Similarly, a Miner who wishes to participate the permissioned-blockchain must perform the same anonymous identity verification process with the Permissions Verifier.

Depending on the mode of operation (see Section **??**) 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

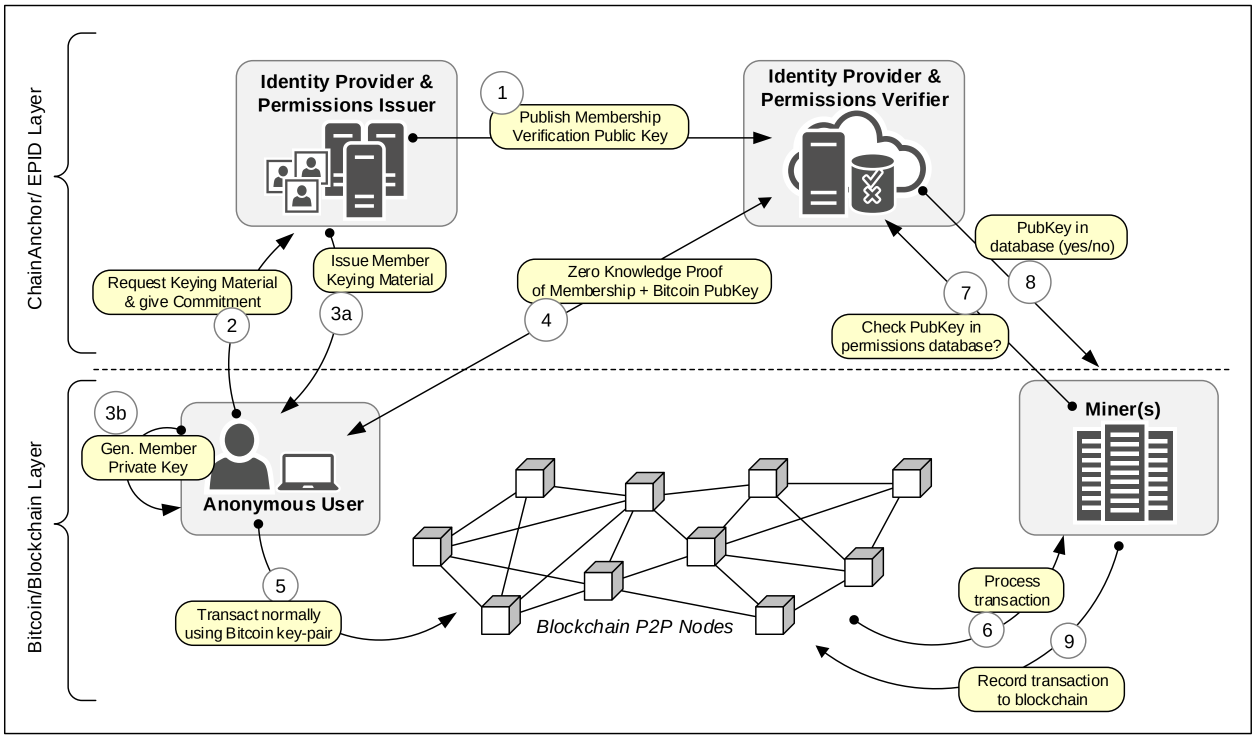
ChainAnchor/ EPID Layer
Bitcoin/Blockchain Layer
Identity Provider & Permissions Issuer
Identity Provider & Permissions Verifier
1
Publish Membership Verification Public Key
Request Keying Material & give Commitment
2
Issue Member Keying Material
3a
3b
Gen. Member Private Key
Zero Knowledge Proof of Membership + Bitcoin PubKey
4
7
Check PubKey in permissions database?
PubKey in database (yes/no)
8
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

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 and PGP [26] where a User can use any number of self-issued public key pairs.

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 the EPID zero-knowledge proof protocol to be deployable within permisisoned-blockchains we propose to converge the roles in EPID with those found Identity Provider (IdP) services.

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

- *Permisisoned-Transaction*:
  We define a (fully) 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 belonging to 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.

  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 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 assumption of the DAA feature in the TPMv1.2 hardware.
  The Permissions Verifier maintains the *Verified Identities Database* containing all the transaction public-keys belonging to the anonymous Users who have successfully undergone the ChainAnchor anonymous identity verification. The database only contains transaction public-keys and the time-stamp of the successful zero-knowledge proof protocol completion. No other identifying information is stored by the Permissions Verifier.
  For each permissioned-group corresponding to a permissioned blockchain there is one distinct Verified Identities Database. The Permissions Issuer has read access into the database at the IdP-PV, but not write-access. This is to maintain the business integrity of the IdP-PV as a service provider.
  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 mines for a permissioned-block and records it on the permissioned blockchain. When reading from the pool of unconfirmed transactions, the Miner composes a block of transactions referred to as the "candidate" block. To compose a block of permissioned-transactions, for each unconfirmed transaction the Miner must check that the public-keys of the transaction are in the Verified Identities Database.
  Similar to the User, a Miner wishing to participate in a given permissioned-group must perform the anonymous identity verification with the Permissions Verifier (IdP-PV) and have the Miner's transaction public-key added to the database for 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 later claim the reward for mining permissioned-blocks of the group. Section **??** discusses the modes of operation of ChainAnchor and the possible remuneration models.

- *User*:
  The User in ChainAnchor has the same function as the originator/recipient of a transaction in Bitcoin. The User can have any number of self-issued transaction public-key pairs. However, in order to participate in a ChainAnchor permissioned-group the User must perform the anonymous identity verification to the Permissions Verifier (IdP-PV) and have its transaction public-key added by the IdP-PV to the Verified Identities Database for that permissioned-group.

- *Owner*:
  Although not shown explicitly in Figure 2, a permissioned-group must be created and owned by an organization or individual. We refer to this entity or person as the Owner of a permissioned-group that implements the permissioned blockchain.
  In initiating the creation of a permissioned-group to implement a permissioned blockchain, the Owner employs the services of the IdP-PI. Although IdP-PI and IdP-PV are service providers that together implements this permissioned-group, only the IdP-PI knows the real identity of the Owner as a customer.

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. We denote this key as K_{MIPK}.

- *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. We denote the Membership Verification Public Key as K_{MVPK}.

- *User's Transaction Public-Key Pair*:
  This is the transaction public-key pair that the User employs to transact on the permissioned blockchain. For simplicity and ease of comparison with the current Bitcoin public-key pairs, we denote the User's transaction

public-key pair as $(K_{bitcoin}, K_{bitcoin}^{-1})$, with the public key being $K_{bitcoin}$.

- *Long Term Pair-wise Shared Key (PSK)*:
  When a User (Miner) executes the zero knowledge proof protocol with the Permissions Verifier entity, one by-product of a successful proof is the establishment of a *pairwise shared key* (PSK) between the User (Miner) and the IdP-PV. This key is a symmetric key.
  We denote a PSK shared between a User and the IdP-PV as $k_{U,PV}$, while the PSK shared between a Miner and the IdP-PV as $k_{M,PV}$.
  The long-term PSK shared between the User (Miner) and the IdP-PV is used to provide the User (Miner) with an authenticated read-access to the Verified Identities Database at the IdP-PV. The authenticated read-access is done through proof of possession (POP) of the shared symmetric key. Note that session-keys derived from the long-term PSK may be used instead, but this topic is outside the scope of discussion of the current work.

- *IdP-PI and IdP-PV Certificates*:
  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 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 Shares Verification Public Key with IdP-PV***:
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 Authenticates & Requests Membership***
A User obtains permission to transact on the permissioned blockchain by requesting membership to the permissioned-group that implements the permissioned blockchain. The User must first authenticate itself to the IdP-PI (namely the Identity Provider endpoint of the IdP-PI) and obtain authorization to join the the permissioned-group. The method used to authenticate is external to the blockchain and is outside the scope of the current paper.

At this point in the ChainAnchor protocol the User is not anonymous to the IdP-PI, and the IdP-PI knows the user (e.g. has an account at the IdP-PI or the user is an employee of an Enterprise deploying the IdP-PI). As such, the User is assumed to be using a traditonal Internet identity (e.g. `alice@gmail.com`) that is well known.

To join the permissioned-group 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 employs 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 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.

At this point onwards in the ChainAnchor protocol the User becomes anonymous to the IdP-PI and 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):

- **Step 4.1**: 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 $K_{bitcoin}$.

- **Step 4.2**: 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.

- **Step 4.3**: 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

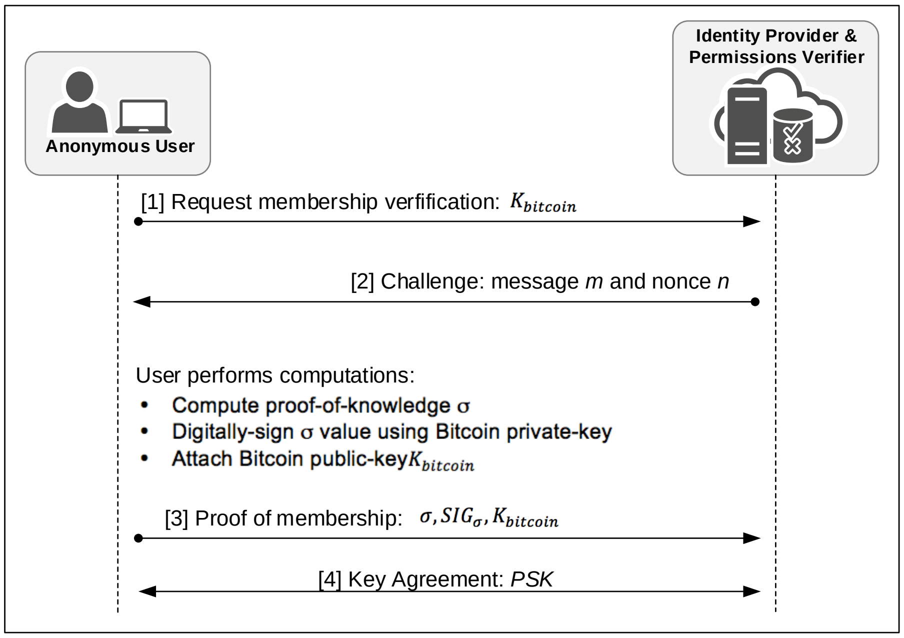
Anonymous User
Identity Provider & Permissions Verifier
[1] Request membership verfification: $K_{bitcoin}$
[2] Challenge: message *m* and nonce *n*
User performs computations:
• Compute proof-of-knowledge σ
• Digitally-sign σ value using Bitcoin private-key
• Attach Bitcoin public-key$K_{bitcoin}$
[3]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

Fig. 3. User Anonymously Proves Membership to IdP-PV (Steps 4.1 – 4.4)

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 together with some parameters to establish a pair-wise shared key (PSK) between the User and the IdP-PV. The IdP-PV then adds the User's transaction public key $K_{bitcoin}$ to the Verified Identities Database.

- **Step 4.4**: The User and the IdP-PV engage in a key agreement subprotocol that results in a pair-wise shared key (PSK) denoted as $k_{U,PV}$. This PSK is shared between the User (who is anonymous throughout Step 4) and the IdP-PV.

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 [27], 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 $K_{bitcoin}^{-1}$ 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. We discuss mining and consensus further in Section IV.

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 [28]).

  Note that the IdP-PV will only respond to revocation 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 permissioned-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 identity verification (in Step 4) the User employs one of the user's self-issued transaction public-key pairs. As such, the User remains anonymous to the Permissions Verifier.

- *User remain anonymous to IdP-PI*: In requesting membership to the IdP-PI (in Step 1), the User most likely may have used his/her own personal public-key pair and therefore made known their real-world identity to the IdP-PI.

  However, after 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). Since the IdP-PI is not involved in Step 4 onwards, the IdP-PI has no knowledge of which transaction public-key pairs are owned by the User.

- *Simple binding to transaction key pair*: We have used a digital signature as a simple binding mechanism between

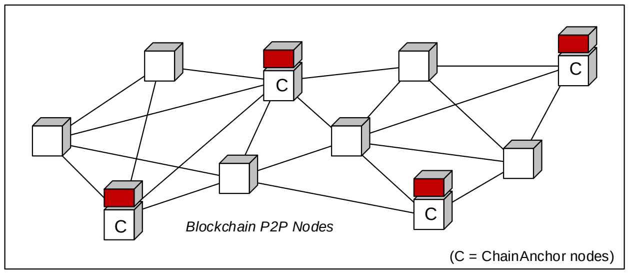

Fig. 4. The Semi-Permissioned Overlay

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).

In the following sections we discuss the application of ChainAnchor to fully private permissioned blockchains and to "semi-private" blockchains – which we refer to as *semi-permissioned* blockchains.

F. Deployment Modes

- *Private Permisioned Blockchains*:
  In this deployment mode the blockchain is privately run system that is separate from the permissionless blockchain in Bitcoin. As such it may use its own non-standard transaction protocol and/or block payload format. ChainAnchor Users remain anonymous to each other.

- *Semi-Permissioned Blockchains (Overlay)*:
  In this mode of deployment, ChainAnchor is deployed as an *overlay* above the current permissionless public Blockchain in the Bitcoin system (Figure 4). This mode is discussed further in Section IV.

G. Transaction Processing Modes

Although a detailed discussion is beyond the scope of the current work, we note that other modes of processing of transactions can be used in ChainAnchor:

- *Originator-only permissioned-transactions*: The Miner verifies that the originator of a transaction is a member of the permissioned-group.

- *Recipient-only permissioned-transactions*: The Miner verifies that the recipient of a transaction is a member of the permissioned-group.

- *Cross-ledger permissioned-transactions*: Here the originator and recipient of the transaction are members of two different permissioned-groups, but are granted the right to transact across the groups.

- *Fully verified permissioned-transaction*: Here the Miner verifies that both the originator and recipient of the transaction are members of the same permissioned-group.

In the remainder of the paper, we focus only on fully verified permissioned-transactions that make-up permissioned-blocks.

IV. The Semi-Permissioned Overlay

An interesting and promising deployment mode for ChainAnchor that does not require the creation of a separate blockchain system with private nodes in the *semi-permissioned overlay*.

Here ChainAnchor is deployed as an overlay above the current public and permissionless Blockchain. The goal of the overlay approach is not to create a separate chain, but rather use the current permissionless Blockchain (in Bitcoin) to carry permissioned-transactions relating to Users in ChainAnchor in such a way that non-ChainAnchor nodes are oblivious to the transactions belonging to a permissioned-group. We use the example of the current Bitcoin blockchain as the underlying blockchain due to the fact that today it is the only operational blockchain that has achieved scale.

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

A. Mining & Consensus

Mining and consensus over a block of permissioned-transactions is achieved the same way as in Bitcoin transactions. Indeed, permissioned-transactions *are* Bitcoin transactions and have no difference. Validated blocks of permissioned-transactions are added to the Blockchain in the same way as validated ordinary Bitcoin blocks of transactions, using the same protocols and mechanism (Figure 6). This is one of the advantages of the semi-permissioned overlay approach.

In Bitcoin a successful miner receives 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 (Figure 5). Being a participant in a ChainAnchor permissioned-group, the miner 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 miner can be selective in choosing transactions to

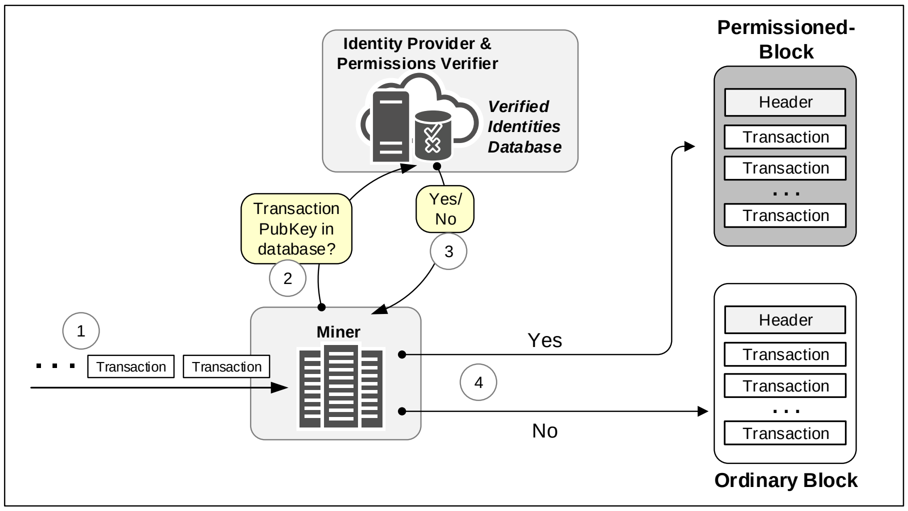
Identity Provider & Permissions Verifier
Verified Identities Database
Transaction PubKey in database?
Yes/ No
1
2
3
4
Transaction
Transaction
Miner
Yes
No
Permissioned-Block
Header
Transaction
Ordinary Block

Fig. 5. Overview of Mining Permissioned-Blocks

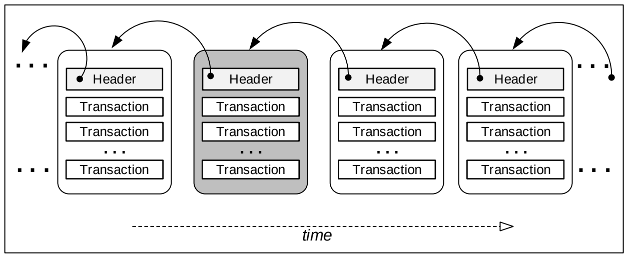

Fig. 6. Permisioned-Blocks among ordinary blocks in the Blockchain

process and prioritizing permissioned-transactions with higher rewards over ordinary transactions.

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.

B. Block Validation by IdP-PI and IdP-PV

The in semi-permissioned overlay, both the Permissions Issuer entity (IdP-PI) and Permissions Verifier entity (IdP-PV) are full-nodes on the underlying Blockchain. This means that although they do not perform mining for rewards, they must validate all blocks of transactions and keep a copy of the validated blocks (the blockchain) themselves. This ensures they can be independent from other nodes on the network.

In addition to independently validating all blocks in the usual Bitcoin manner, IdP-PI and IdP-PV entities have additional steps that they need to perform. For every block of transaction they validate they must also pick-out those which are permissioned-blocks:

- *Validate permissioned-blocks*: Verify that each transaction in a validated block is a permissioned-transaction. If this is true, this means that the block is in fact a permissioned-block. For such permissioned-blocks, the IdP-PI and IdP-PV may take note as to their location in the Blockchain.
- *Identify Miner of permissioned-blocks*: For a validated permissioned-block, IdP-PI and IdP-PV must verify that the successful Miner's public-key is found in the Verified Identities Database. That is, they both must ensure that the Miner has executed the zero knowledge proof protocol with the IdP-PV prior to mining for ChainAnchor permissioned-transactions.

This last step is needed in order for the Miner to be remunerated by the Owner of the permissioned-group through the IdP-PI.

C. Preventing Collusion: Multiple Permisions Verifiers

The business objective of the Permissions Issuer and the Permissions Verifier is to realize the execution of a permisisoned-group on behalf of the paying Group Owner. As such, it is in the interest of the IdP-PI and IdP-PV to ensure that only Users that have been authorized by the Group Owner get access to the permissioned-group.

However, given that Users and Miners are anonymous and that they are recognized only through the presence of their transaction public-keys in the Verified Identities Database, there is the possibility of Users or Miners colluding with the

IdP-PV to get their public-keys listed in the database without previously performing the zero knowledge proof protocol.

To counter such a scenario, the Permissions Issuer has the option of engaging multiple Permissions Verifiers in relation to the same permissioned-group. Each Permissions Verifier would therefore build-up its own version of the Verified Identities Database which is read-accessible to the IdP-PI. This allows the IdP-PI to check multiple independent databases, and detect inconsistent entries.

In this configuration, a User or Miner must execute a zero knowledge proof protocol separately to each IdP-PV entity. In each case, the Use/Miner must use different parameters for their Commitment values (Step 2 of Section III-C) and the generation of their User-Member Private Key K_{UMPK} (Step 3 of Section III-C). However, in each case he or she must bind the same transaction public-key ($K_{bitcoin}$) to each of the IdP-PV entities respectively. This ensures that the same transaction public-key will be listed in each of the distinct databases respectively.

D. Discovery of Permissioned-Groups

Information regarding the existence of a permissioned-group at the IdP-PI must be made known by the IdP-PI to potentials Users and Miners. The precise mechanism is out of scope for the current work, but it should carry enough identifying information to allow a User or Miner to identify the desired permissioned-group and to request membership (Step 2 of Section III-C). Some of these information may include:

- *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 & IdP-PV*: This is the identity (e.g. X.509 certificate) and possibly the transaction public-keys of both entities. Note that the IdP-PI and IdP-PV are not anonymous entities, and must be well known legal entities in order to obtain trust by their customers (namely the Group Owners).

Additional authentication and authorization mechanisms maybe implemented by the Identity Provider function of the IdP-PI. These are external to the Blockchain and will be deployment-specific.

Similarly, the reward fee for completing a ChainAnchor permissioned-block must be advertised out-of-band by the Permissions Issuer entity (IdP-PI) to ensure that Miners participate in mining permissioned-blocks.

E. Remuneration for Miners

In the ChainAnchor semi-permissioned overlay 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 Issuer (IdP-PI) on behalf of the Owner of the permissioned-group.

If the IdP-PI rewards the miner using Bitcoins, this act will be visible to other members of the permissioned-group by looking at the underlying Blockchain (ie. search through confirmed transactions). Note that the transaction public-key of the IdP-PI is known to all members of a permissioned-group.

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 semi-permissioned overlay 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. 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 adds an *anonymous identity verification* step using the EPID zero-knowledge scheme [2] to a permissioned blockchain that limits access to the blockchain only to verified members.

In the anonymous identity verification, the User has to also cryptographically "bind" his or her transaction public-key to the zero-knowledge proof sent to Permissions Verifier. This results in that transaction public-key being recognized as a valid "identity" that has obtained 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. Similarly, a Miner who wishes to participate the permissioned-blockchain must perform the same anonymous identity verification process with the Permissions Verifier.

The database only contains transaction public-keys and the time-stamp of the successful zero-knowledge proof protocol completion. 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. For each permissioned-group corresponding to a permissioned blockchain there is one distinct Verified Identities Database.

We suggest two modes of deployment, with differing possible incentive models. In the first deployment mode – namely the *private permissioned blockchains* – the ChainAnchor blockchain is a privately run system that is separate from the public permissionless Blockchain in Bitcoin. 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 *semi-permissioned overlay* – ChainAnchor is deployed as an overlay above the current permissionless public Blockchain in Bitcoin. The goal of the overlay approach is not to create a separate chain, but rather use the current permissionless Blockchain to carry permissioned-transactions relating to Users in ChainAnchor in such a way that non-ChainAnchor nodes are oblivious to the transactions belonging to a permissioned-group. Mining and consensus over a block of permissioned-transactions is achieved the same way as in Bitcoin transactions. Additional reward is given by the IdP-PI to a miner that successfully mines a permissioned-block.

The semi-permissioned overlay 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 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 miner's overall income.

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:

- *Cross-ledger permissioned-transactions*: The case of cross-ledger transactions is one where the originator and recipient of the transaction are members of different permissioned-groups, but are granted the right to transact across different groups.

  There are several interesting deployment scenarios for ChainAnchor cross-ledger transactions. Examples include different fiat currencies that are legally exchanged with Bitcoin, and where a separate permissioned-group is established for each fiat currency. The Permissions Issuer entity could be implemented within an existing regulated financial institution (e.g. Bank), while one or more Permissions Verifier entities could be realized by the emergent Bitcoin Exchanges.

- *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 *semi-permissioned overlay* mode can be used for AML purposes. 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 or value passing through the Bitcoin network.

  Here ChainAnchor could be used to establish a permissioned-group for "verified" (AML-friendly) and "unverified" transactions. Using ChainAnchor members of the AML-friendly permissioned-group can remain anonymous but have the option to voluntarily disclose their identity when legally challenged regarding suspicious transactions. This means disclosing the link between their Internet identity (as known by the Permissions Issuer entity) and their transaction (Bitcoin) public-key. Such legal challenges should come from the appropriate financial authorities.

  Note that when a user discloses one of their ChainAnchor anonymous public-keys it does not affect the user's remaining undisclosed anonymous public-keys, so long as these public-keys were bound to distinct instances of the anonymous zero-knowledge proof protocol exection (even if these distinct executions were for 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 $q \mid (p-1)$, $q \nmid \frac{(p-1)}{q}$ 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. Callas, L. Donnerhacke, H. Finney, and R. Thayer, “OpenPGP Message Format,” RFC 2440 (Proposed Standard), Internet Engineering Task Force, Nov. 1998, obsoleted by RFC 4880. [Online]. Available: http://www.ietf.org/rfc/rfc2440.txt

[27] 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.

[28] 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