Who Am I? Secure Identity Registration on Distributed Ledgers

Sarah Azouvi, Mustafa Al-Bassam, and Sarah Meiklejohn

University College London
`{sarah.azouvi.13,mustafa.al-bassam.16,s.meiklejohn}@ucl.ac.uk`

Abstract. Bitcoin is a decentralized cryptocurrency that uses a ledger (or "blockchain") to keep track of the transactions made between its users. Because it is a fully decentralized system and anyone can join, every transaction is by necessity public. Thus, to preserve some semblance of privacy, users in the system are represented not by their real-world identities but by pseudonyms. While pseudonyms are acceptable for a standalone cryptocurrency, the emergence of other potential blockchain-based applications — e.g., using them to administer benefits and pensions — poses a need to associate certain attributes with the users of the system. In this paper, we address the question of how to register identities and attributes in a system built on globally visible ledgers. We propose a variety of possible solutions and in each case, we analyze the tradeoff our solution provides between privacy (ensuring that no one can associate the user's real-world identity with the pseudonym or other attributes they use on the ledger), usability (ensuring that verification of their attributes poses the lowest possible burden to users), and integrity (ensuring that no one can impersonate a user). We also present an implementation of one of our solution using Ethereum.

1 Introduction

Distributed ledgers, or "blockchains," have received a lot of attention for their potential applications: in addition to being used as the underlying architecture for cryptocurrencies such as Bitcoin, they have been discussed for achieving decentralized versions of identity management, DNS and public-key infrastructures, notary publics, and file storage. While centralized versions of these systems already exist, the attraction of distributed ledgers is that they minimize the extent to which users must place trust in a single entity such as a certificate authority.

In all existing deployments of distributed ledgers, users identify themselves using *pseudonyms* — or even more anonymous identifiers, as in the cryptocurrency Zcash [4] — that they create themselves. The use of pseudonyms is important for two reasons: first, all existing distributed ledgers are *transparent*, meaning their contents are globally visible, so having users reveal their real-world identities would completely violate their privacy. Second, allowing users to generate their own identifiers is necessary to preserve the openness of the system and allow anyone to join.

While these "on-chain" pseudonyms are thus seemingly quite useful (and to some extent necessary) in public distributed ledgers, there are certain cases in which it may be necessary for someone to know some quality of the owner of a pseudonym, e.g., gambling services would like to know that their users are over 18. As a more involved example, we consider the case of governments administering pensions or benefits on a distributed ledger; the argument that has been made for doing this is that it could provide recipients with better visibility into their spending and reduce fraud [13], but such programs have recently come under significant scrutiny [9, 19] due to the fact that they allow the government to identify its recipients on the ledger and thus track and monitor their spending. In all of these settings, we would thus like the user to not be forced to reveal to anyone the tie between their real-world identity and their pseudonym(s), but rather to have some information that proves that the real-world user associated with their pseudonym has been registered for some scheme (e.g., a pension) or is associated with some required set of attributes (e.g., is over 18).

Our focus in this paper is on the role that registration of identity can play in public distributed ledgers. While certain settings such as the ones described above might require a centralized registration protocol (e.g., only the government can decide whether or not a user is eligible for a pension), we also consider more informal notions of registration such as the so-called "web of trust." The web-of-trust concept has historically been used solely within the setting of certificate issuance, wherein users sign each others' PGP identity certificates to vouch for their authenticity, but has recently been discussed for the more general concept of identity in distributed ledgers. These decentralized settings are particularly appealing, as they remove the need for a single trusted party and provide an opportunity to improve privacy for users.

Our contributions In this paper, we propose methods for achieving registration in decentralized settings — such as the web of trust — in which multiple entities, in potentially flexible configurations, can act to validate attributes of a user's identity. We consider the registration of users' pseudonyms, unless stated otherwise. Our results focus on public open (or "permissionless") ledgers, but the same results would hold in the more restricted setting of "permissioned" ledgers.

Before presenting these methods, in Section 4 we consider both the functional and security properties that we hope to achieve. In particular, we consider how to provide *privacy* for users, so that even the registrar who sees their real-world identity and signs off on their attributes cannot subsequently link that identity to the pseudonyms that the user goes on to adopt within the ledger.

Due to space constraints, we relegate our centralized constructions to a full version of the paper. In the decentralized setting, in Section 5, we begin with a registration protocol in the style of the web of trust (but again, leveraging some of the key properties of distributed ledgers), and then build off of it to achieve protocols that provide better privacy and overall security.

Finally, in Section 6, we present an implementation of a decentralized registration protocol — that most closely resembles the web of trust, but allows for the *blinding* of attributes — as an Ethereum smart contract. In this setting, users can publish certain attributes (e.g., their Twitter handle) associated with their Ethereum address. Other users or institutions can then publish a signature on these attributes, reflecting a certain belief in its veracity. For attributes that the user may not want to directly link to their real-world identity (e.g., a particular Bitcoin or other cryptocurrency address), we provide a blind signing protocol in which users can publish blinded attributes on the blockchain and other users can sign them (and then the user can unblind them locally).

2 Related Work

In the setting of certificate issuance, our proposed systems are related to the idea of a public-key infrastructure (PKI). Some of our proposed registration protocols rely on a fixed set of specified registrars. These are related to the decentralized PKIs proposed by Fromknecht et al. [14] and the ARPKI system [3], which both distribute the process of certificate issuance to not only provide transparency into the process but also prevent misbehavior in the first place. In the more ad-hoc setting in which we allow any user to act as a registrar, our protocols are related to the idea of the web of trust.

The notion of accessing a service in a privacy-preserving manner can seemingly be achieved by anonymous credentials [11, 7, 8], which allow an issuer to create credentials that vouch for a user's identity or other generic attribute (e.g., their age). These credentials can then be shown to a verifier in a way that doesn't reveal anything to the verifier beyond the fact that the user possesses the attribute (e.g., is over 18 years old). The idea of issuing anonymous credentials has also been explored in the decentralized setting [15]. Our goal in this paper, however, is to allow users to not only access services but also to openly engage in existing blockchain-based systems using a registered identifier that — despite being vouched for by some registrar — cannot be linked to their real-world identity. To the best of our knowledge, this goal cannot be achieved directly by any solution based on anonymous credentials, at least not in an efficient manner: even if credentials could be issued on-chain, they would be larger than a blockchain address and issuance would consume a prohibitively high amount of gas.

Finally, a lot of recent work, both in the academic literature and in the broader community, has focused on the question of using the blockchain to establish and manage identities (see, e.g., `https://github.com/peacekeeper/blockchain-identity` for a comprehensive list). The ChainAnchor project [17] presents a system for identity and access control, with the purpose of having anonymous but verified on-chain identities, and of providing incentives to miners to include only transactions from verified users. While some of the techniques used are similar to our own, as they also adopt a form of registration, their focus is on permissioned ledgers and on requiring registration for all users (which is useful in, e.g., the setting of providing compliance with know-your-customer and anti-money-laundering regulations). In terms of industrial solutions, uPort [12] is a web identity management system that links an Ethereum address with a name, profile picture, and other information like an email address or Twitter account, and OneName is a similar initiative that does the same with Bitcoin addresses. MIT also recently introduced its Digital Certificates Project [20] using the Bitcoin blockchain, with the goal of making "certificates transferable and more easily verifiable." These solutions have seen some level of adoption and we borrow some useful features from each of them (e.g., we use a similar technique to achieve revocation as the Digital Certificates project), but add the benefit of additional points of comparison, and a security framework and analysis.

3 Background

3.1 The web of trust

The web of trust is a public-key authentication system established by PGP. In this setting, if Alice trusts that a certain key belongs to Bob (e.g., they have met in person), she can demonstrate this by signing his public key. The more signatures associated with Bob's public key, the more confident another user can be that this public key does indeed belong to him and not to someone who wants to impersonate him in order to intercept his communications.

In this system, one must of course be careful that it achieves some notion of Sybil resistance; i.e., that an adversary has not simply created alternate identities in order to vouch for their own impersonated key. To do this, users in the web of trust can form a *trust path*. For example, if Alice trusts Bob's public key, and Bob trusts Dave's public key, then there is a trust path from Alice to Dave and she can have added confidence in Dave's public key (as Bob's public key, which she trusts, was used to sign it). The shorter the trust path, the stronger the trust can be in the associated public keys.

3.2 Distributed ledgers

Bitcoin relies on a peer-to-peer network to process transactions. Within the system, users are represented by *addresses* addr, each of which is uniquely linked to a pair of public and private ECDSA keys (pk, sk). We denote by $\mathsf{addr}(pk)$ the address associated with pk. Every time Alice wants to pay Bob using Bitcoin she generates a *transaction* $\mathsf{tx}(\mathsf{addr}(pk_A) \rightarrow \mathsf{addr}(pk_B))$ and signs it with her private key sk_A. (More generally, Bitcoin transactions can have arbitrarily many input and output addresses, in which case the transaction must be signed by all private keys associated with the input addresses, or even m-of-n multi-signature transactions, in which a transaction must be signed by the private keys associated with at least m of the input addresses.) She then broadcasts the signed transaction to the network, which checks its validity and if applicable, adds it to the *blockchain*, which acts as a public ledger of all such transactions.

To achieve more general functionality, Ethereum is a decentralized platform that operates with the same underlying blockchain technology as Bitcoin, except that it provides a Turing-complete scripting language. In Ethereum, a smart contract consists of program code, a storage file, and an account balance. The program's code is executed by the network, which is responsible for maintaining a consistent view of the state of every contract in the blockchain. Users can call the contract by sending transactions to its address, which updates the state of the contract in the blockchain. Moreover, the execution of a program's instructions induces a cost; the currency used to pay for it is called gas.

3.3 Cryptographic Primitives and Notation

Following standard cryptographic notation, we use $x \overset{\$}{\leftarrow} S$ to denote the process of sampling a member uniformly from S and assigning it to x. In particular, we use $x \overset{\$}{\leftarrow} [n]$ to denote sampling x uniformly from $\{1, \ldots, n\}$. We use $y \leftarrow A(x_1, \ldots, x_n; R)$ to denote running algorithm A on inputs $x_1, \ldots, x_n$ and random coins R and assigning its output to y. By $y \overset{\$}{\leftarrow} A(x_1, \ldots, x_n)$ we denote $y \leftarrow A(x_1, \ldots, x_n; R)$ for R sampled uniformly at random.

Both Bitcoin and Ethereum rely on ECDSA for signing. In what follows we use $(pk, sk) \overset{\$}{\leftarrow} \mathsf{Sig.KeyGen}(1^\lambda)$ to denote key generation, $\sigma \overset{\$}{\leftarrow} \mathsf{Sig.Sign}(sk, m)$ to denote signing, and $0/1 \leftarrow \mathsf{Sig.Verify}(pk, m, \sigma)$ to denote verification.

Some of our decentralized registration protocols make use of public-key encryption; here we denote the appropriate generic algorithms as $c \overset{\$}{\leftarrow} \mathsf{Enc}(pk, m)$ (for encryption) and $m \leftarrow \mathsf{Dec}(sk, c)$ (for decryption). In order to maintain compatibility with Bitcoin and Ethereum, the Elliptic Curve Integrated Encryption Scheme (ECIES) provides an encryption scheme that is compatible with ECDSA; i.e., one that allows for the encryption of ECDSA secret keys.

Finally, some of our schemes also make use of blind signatures. As initially defined by Chaum [10], a blind signature provides an interaction — denoted $\mathsf{U}(pk, m) \leftrightarrow \mathsf{S}(sk)$ — wherein a user U obtains a signature from a signer S on a message without the signer learning anything about the message. One commonly used construction is the RSA blind signature [16], which we use in our constructions due to the lack — to the best of our knowledge — of any provably secure blind signatures that are compatible with ECDSA.

4 Definitions and Threat Model

We consider a setting in which *users* maintain *attributes* about themselves and require *registrars* to vouch for these attributes. For example, in order to register the attribute "over 18 years old," a user reveals their identity to the government, who verifies their age. If they are over 18, the government registers the user's pseudonym, and they are now able to use it directly on the blockchain. For Bitcoin, we consider only the registration of pseudonyms, but in Ethereum, we consider the registration of more general types of attributes. Confirmation that the user possesses a given pseudonym may in turn be carried out by *verifiers* in order for the user to gain access to a particular service; i.e. for the users to interact with the service using their registered pseudonyms (e.g., use it to receive a pension from the government). We break this system down into four phases: (1) *setup*, in which various actors may initialize certain information about themselves (e.g., keys); (2) *registration*, in which the user interacts with the registrar(s) to register their pseudonym(s) and receive some evidence of this; (3) *verification*, in which the user interacts with the verifier to convince them that certain pseudonyms have been registered; and (4) *revocation*, in which either the registrar or (in some cases) the user revokes the registration of their pseudonym.

In order for the system to function, we must have a way for verifiers to check certain information about users without the intervention of the registrar. Let's assume, for example, that the user wants to register as an attribute the fact that they are over 18 years old so they can use a gambling service. If the registrar must intervene in order to confirm this attribute — as in the recently proposed brokered identification systems proposed in the US and UK [21, 18, 5] — then the registrar must be online at all times and can link the user's identity with their usage of certain services, neither of which is desirable. If instead this information is stored on a blockchain, then the verification step can happen in a non-interactive, or *passive*, fashion, as the verifier can simply check for themselves if the user's pseudonym has been registered or not. If evidence of the registration is not stored on the ledger, or if additional information is needed to "unlock" it (e.g., it is encrypted), then it may be necessary for the user to send additional information to — or otherwise interact with — the verifier. We capture these two functional properties as follows:

Definition 1 (Passive/active verification). *The verification process is* passive *if any verifier with access to the shared ledger can determine whether or not a given user has registered a particular attribute. The verification process is instead* active *if verifiers require additional information beyond what is available on the shared ledger.*

In order for the system to be secure, we would like to ensure that users are able to register only accurate attributes about themselves; e.g., they can register only for services, such as a pension scheme, that they are eligible to use. We must also ensure that the individual identities of users are protected and cannot be impersonated by anyone else. Once the user has completed the registration process and is interacting within the system using only their registered pseudonyms (e.g., their Bitcoin address), we should be able to ensure *privacy*; i.e., that the registrar cannot link the user's real-world and "on-chain" identifiers (even across separate attributes). We consider the different types of security we would like to achieve as follows:

Definition 2 (Attribute integrity). Attribute integrity *holds if attributes are registered only to those users to whom they belong; i.e., in the presence of an honest registrar, malicious users are unable to either register a fake attribute or one that otherwise does not belong to them, and malicious registrars are unable to impersonate an individual honest user.*

Definition 3 (Attribute privacy). Attribute privacy *holds if malicious entities (i.e., registrars and verifiers who are allowed to collude) are unable to link*

	Verification		Attribute integrity	Privacy
	passive	active		
Basic web of trust	●		◐	
Blinded web of trust (with revocation)		●	◐	◐
Blinded web of trust (without)	●		◐	●
Multi-Casascius	●		●	●
Mix-network	●		●	●

Table 1. The different properties of a blockchain-based registration protocol and whether or not they are satisfied by our various constructions. No circle indicates that the property is not satisfied, a filled circle indicates it is, and a partially filled circle indicates it is partially satisfied.

the attributes a user claims within the system to their identity. In particular, after the registration process is complete, malicious registrars are unable to distinguish the behavior of two users within the system that have different real-world identities but the same set of attributes.

As we will see in our constructions, while revocation is useful and often necessary — as keys are frequently compromised or lost — it also tends to require active verification, as registrations cannot be deleted from the ledger (because it is immutable) and it is difficult to efficiently prove the absence of a revocation entry. To thus separate out these complexities, we analyze our protocols separately in the cases where revocation is and isn't supported.

5 Decentralized Registration

The "web of trust" reputation system can be considered a decentralized registration process in which any user can act as a registrar. The more signatures one accumulates for a particular attribute, the more *trusted* that attribute can be considered. In the PGP web of trust, however, the system still uses a central website to provide the lookup and signing services. In our constructions below, we use the blockchain to provide these two services. We also consider additional decentralized protocols that provide more robust properties or are useful in settings outside of the web of trust.

5.1 Basic web of trust

One simple way of translating the web of trust into the setting of blockchains is to have users create transactions that vouch for each others' attributes. This can be done either individually or — if a user knows in advance which other users will vouch for their attribute — as a multi-input transaction.

Construction In the setup phase, the user optionally chooses a set of peers to validate their attribute and act as registrars. We assume each registrar creates and publishes an on-chain identity addr_R.

In the registration phase, the user sends their identity id and address $\mathsf{addr}_{\mathsf{id}}$ to each registrar, who determines if the address belongs to id (or just if id is a valid identity), using some off-chain mechanisms that we omit here. If it does, each registrar R_i creates a revocation keypair $(pk_{\mathsf{rev}}^{(i)}, sk_{\mathsf{rev}}^{(i)}) \xleftarrow{\$} \mathsf{Sig.KeyGen}(1^\lambda)$, and publishes to the blockchain a transaction $\mathsf{tx}(\mathsf{addr}(\mathsf{R}_i) \to \{\mathsf{addr}_{\mathsf{id}}, \mathsf{addr}(pk_{\mathsf{rev}}^{(i)})\})$. In a basic system like Bitcoin this could involve sending a specific amount of bitcoins to both the attribute and revocation addresses, while in a more sophisticated system like Ethereum it could be a registration smart contract. Alternatively, if the set of registrars is fixed ahead of time, a user can create an n-input $n+1$-output transaction and, after collecting signatures on it from each registrar, publish it to the blockchain.

In the verification phase, when the user wishes to prove that they have registered the pseudonym, the verifier checks for the existence of these transactions in the blockchain, and that the output address $\mathsf{addr}(pk_{\mathsf{rev}})$ has not spent its contents. (While this may seem inefficient, if we associate with the ledger a list of unspent transaction outputs, or utxos, then it becomes significantly faster.)

Our approach to revocation here and in what follows is inspired by the approach of the MIT Digital Certificates project [20]. In the revocation phase, a registrar R_i can revoke their registration by spending the contents of $\mathsf{addr}(pk_{\mathsf{rev}}^{(i)})$.

Security analysis

Verification is passive, as the verifier needs to check only whether or not certain transactions are in the blockchain.

Attribute integrity is partially satisfied: restricting ourselves to the setting of on-chain pseudonyms, no registrar is able to impersonate the user, as they don't know the private key corresponding to a user's $\mathsf{addr}_{\mathsf{id}}$. We could strengthen integrity by requiring the user to also send a signature to prove its ownership of $\mathsf{addr}_{\mathsf{id}}$. Because the user can pick its own set of registrars, however, we cannot unilaterally guarantee that a user can't register a fake attribute, as a malicious coalition of users could act to register each other's fake identities or attributes. This is the same problem faced in the web of trust, however, and it can be mitigated by having the verifier place trust only in registrars with whom they can create a trust path of a certain (short) length (see Section 3.1). If malicious registrars can place themselves along this trust path with a certain proximity to the verifier, this is analogous to launching a Sybil attack, which can be prevented or detected in a variety of ways [2]. Thus, if the verifier sets a low threshold for the required length of the trust path and a high threshold for the number of registrars required to have registered the attribute, we can argue that the probability that malicious users can register fake attributes is low.

Privacy is not satisfied, as every registrar sees both id and $\mathsf{addr}_{\mathsf{id}}$ at the same time.

5.2 Blinded web of trust

We provide a blinded version of the web of trust in which the user collects blind signatures from a set of nodes and the verifier then verifies the unblinded signatures. In Section 6, we present the results of an implementation and deployment of this approach on Ethereum.

Construction In the setup phase, each registrar maintains as before a public on-chain identity $\mathsf{addr}_{\mathsf{R}}$ linked to a public signing key pk_{R}.

In the registration phase, the user sends their identity id to a registrar, who determines whether or not they believe the user is eligible for the service. If they do, the user and registrar engage in the blind signing protocol $\mathsf{U}(\mathsf{addr}_{\mathsf{R}}, pk) \leftrightarrow \mathsf{R}(sk_{\mathsf{R}})$ at the end of which the user obtains a signature σ such that $\mathsf{Sig.Verify}(pk_{\mathsf{R}}, pk, \sigma) = 1$ and the registrar learns nothing about pk. The registrar also creates a revocation keypair $(pk_{\mathsf{rev}}, sk_{\mathsf{rev}}) \xleftarrow{\$} \mathsf{Sig.KeyGen}(1^{\lambda})$, sends it to the user, publishes to the blockchain a transaction $\mathsf{tx}(\mathsf{addr}_{\mathsf{R}} \rightarrow \mathsf{addr}(pk_{\mathsf{rev}}))$, and maintains the mapping from id to pk_{rev}. The user repeats this process with every registrar. In the verification phase, the verifier verifies the unblinded signatures, and the user proves they control the revocation address pk_{rev} by signing a message using sk_{rev}. The verifier verifies this signature, checks the existence of the revocation transaction in the blockchain, and checks that $\mathsf{addr}(pk_{\mathsf{rev}})$ has not yet spent its contents.

In the revocation phase, the registrar spends the contents of $\mathsf{addr}(pk_{\mathsf{rev}})$.

Security analysis

Verification is active, as the user must provide the signatures to the verifier.
To allow for passive verification without revocation, the user could, after some delay, send pk and σ back to the registrar. The registrar would then check its own signature and, if it verifies, publish to the ledger a transaction of the form $\mathsf{tx}(\mathsf{addr}_{\mathsf{R}} \rightarrow \mathsf{addr}(pk))$. The verifier would, in this case, simply check for a the transaction $\mathsf{tx}(\mathsf{addr}_{\mathsf{R}} \rightarrow \mathsf{addr}(pk))$ in the blockchain to verify the registration, making it passive.
If we require revocation, however, we cannot achieve passive verification. The user would need to prove to the verifier that the coins in their revocation address are unspent, but to do that they would still need to prove that they know the secret key associated with their revocation address — as otherwise they could find and use any revocation address in the ledger — which requires active participation.

Attribute integrity is partially satisfied, as malicious registrars cannot impersonate users since they do not know the private key associated with the public key they register. While the unforgeability of the blind signature guarantees that a malicious user cannot fake the approval of an honest registrar, we cannot guarantee that malicious users and registrars cannot collude to register fake attributes. Instead, we can diminish the probability of this by requiring short trust paths and numerous registrars.

Privacy is satisfied, as the unlinkability of the blind signature means that malicious registrars are unable to link id and pk. If we consider malicious verifiers as well, however, then the verifier could collude with the registrar and use pk_{rev} to de-anonymize the user. If we ignore revocation then privacy is (fully) satisfied.

5.3 Multi-Casascius

In this setting, we assume that the registrar consists of several entities (e.g., different certificate authorities) that are assumed to have some level of trust in each other; in particular, one registrar must be trusted by the others to correctly verify the identity of the user. As an improvement over the previous construction, these multiple entities make the registration process anonymous and provide passive verification, even in the case where revocation is necessary.

Our solution is based on the two-factor key generation protocol used to generate physical Casascius coins [6]. In this process, the manufacturer (Casascius) encodes on a physical coin a public key and a share of the associated secret key. (Traditional Casascius coins have the full secret key, meaning the manufacturer knows it and is able to spend the contents in the same way as the person who bought it.) The user who purchases the product can then fold in their own share of the secret key (which has been communicated to Casascius in the obfuscated form of an "intermediate code"), which yields the full secret key needed to spend the coins stored in the public key; thus, only the user and not the manufacturer can spend the coins. Our solution attempts to retain this property, which allows for attribute integrity, but provides a decentralized version for use in a wider variety of settings.

Construction In the setup phase, each registrar R_i establishes some on-chain identity $\mathsf{addr}_{\mathsf{R}_i}$ associated with a public key pk_i, and the user creates a keypair $(pk_{pub}, sk_{pub}) \xleftarrow{\$} \mathsf{Sig.KeyGen}(1^\lambda)$. The user chooses a set of registrars with whom they want to register, as well as the order in which the registrars will proceed. (This can be thought of as either a property of the system, or as a choice made by the user that they communicate to the registrars.)

The registration phase proceeds in two phases. First, the user sends pk_{pub} and their real world identity id to R_1. This registrar verifies that the user is legitimate; if so, it picks a random secret key sk_1, sends $pk_1 \leftarrow (pk_{pub})^{sk_1}$ to R_2, and keeps (for use in the second phase) the mapping $sk_1 \mapsto pk_{pub}$. Now, for all i, $2 \leq i < n$, registrar R_i picks a random secret key sk_i, sends $pk_i \leftarrow (pk_{i-1})^{sk_i}$ to registrar R_{i+1}, and keeps the mapping $sk_i \mapsto pk_{i-1}$. Upon receiving pk_{n-1}, registrar R_n also picks a random secret key sk_n and forms $pk_n \leftarrow (pk_{n-1})^{sk_n}$. It then creates a revocation keypair $(pk_{\mathsf{rev}}, sk_{\mathsf{rev}}) \xleftarrow{\$} \mathsf{Sig.KeyGen}(1^\lambda)$ and publishes a transaction $\mathsf{tx}(\mathsf{addr}_{\mathsf{R}_n} \rightarrow \{\mathsf{addr}(pk_n), \mathsf{addr}(pk_{\mathsf{rev}})\})$ that acts as a registration.

In the second phase, the registrars create an *onion* to send the secret keys back to the user. In particular, R_n encrypts sk_n using pk_{n-1} and sends $c_n \xleftarrow{\$}$

$\mathsf{Enc}(pk_{n-1}, (sk_n, \perp))$ to R_{n-1}. Now, for all i, $n > i \geq 2$, R_i folds their own secret key into the onion by sending $c_i \xleftarrow{\$} \mathsf{Enc}(pk_{i-1}, (sk_i, c_{i+1}))$ to R_{i-1}. At the end, R_1 creates $c_1 \xleftarrow{\$} \mathsf{Enc}(pk_{pub}, (sk_1, c_2))$ and sends this to the user. The user can now recover all the individual sk_i values by computing $(sk_i, c_{i+1}) \leftarrow \mathsf{Dec}(sk_{i-1}, c_i)$ for all i, $1 \leq i \leq n$ (where $sk_0 = sk_{pub}$), and can thus reconstruct the public key pk_n as

$$pk_n \leftarrow (pk_{pub})^{\prod_{i=1}^{n} sk_i},$$

and the private key as $sk_n \leftarrow sk_{pub} \cdot \prod_{i=1}^{n} sk_i$.

In the verification phase, the verifier checks for the existence of the transaction $\mathsf{tx}(\mathsf{addr}_{\mathsf{R}_n} \to \{\mathsf{addr}(pk_n), \mathsf{addr}(pk_{\mathsf{rev}})\})$ in the blockchain, and verifies that the contents of pk_{rev} are unspent.

In the revocation phase, R_1 is the only registrar that can initiate revocation (as it is the only one that knows id), but R_n is the only registrar that can spend the contents of $\mathsf{addr}(pk_{\mathsf{rev}})$. Thus, R_1 starts by sending a revocation request for key pk_1 to R_2. In turn, using the mapping $sk_i \mapsto pk_{i+1}$, R_i sends a revocation request for key pk_i to R_{i+1} for all i, $2 \leq i < n$. When the request reaches R_n, they can revoke the registration by spending the coins in $\mathsf{addr}(pk_{\mathsf{rev}})$.

Security analysis

Verification is passive, as the verifier needs to check only whether or not certain transactions exist in the blockchain.

Attribute integrity is satisfied. The first registrar R_1 checks for the validity of id, so a user cannot register a fake or ineligible identity as long as R_1 is honest. Similarly, because the user sends a value pk_{pub} to the registrar that involves a partial secret key sk_{pub} known only to them, even if all the registrars collude the user is still the only entity who knows the full secret key associated with pk_n, which means they cannot be impersonated. We could require the user to send a signature under sk_{pub} to additionally prove their ownership of pk_{pub}.

Privacy is satisfied, as long as at least one registrar is honest. For all i, $2 \leq i \leq n$, each registrar R_i knows the mapping between pk_{i-1} and pk_i, and R_1 knows the mapping between id and pk_1. If all the registrars collude, they can thus learn the mapping between id and pk_n, but as long as one registrar doesn't collude with the others and $n \geq 3$, the user cannot be de-anonymized. Assuming that R_1 does not know which node is acting as R_n, which is plausible as it communicates directly only with R_2, R_1 cannot de-anonymize the user by observing the transactions published in the blockchain. Timing attacks can be mitigated by adding some random delays in the publication of the registration transaction. Our next protocol will completely thwart this attack.

5.4 Mix-network

While the blinded web of trust protocol in Section 5.2 and the multi-Casascius protocol in Section 5.3 provide strong privacy guarantees, the former has the drawback that verification required active participation on behalf of the user, and the latter has the drawback that all registrars must trust the initial one to verify the identities and allows timing attacks. Here, we try to maintain the advantages of these protocols but eliminate these drawbacks.

Without adopting the time delay from our protocol in Section 5.2, we cannot achieve passive verification unilaterally. Instead, we consider how to provide passive verification in a setting in which multiple users register at the same time through the same set of nodes (e.g., voter registration), which also allows us to provide each registrar with the ability to verify the set of identities for themselves without violating privacy. As we will see, if k users register at the same time then this provides each user with an anonymity set of size k.

Construction In the setup phase, each user j creates a keypair $(pk_{pub}^{(j)}, sk_{pub}^{(j)}) \xleftarrow{\$} \mathsf{Sig.KeyGen}(1^\lambda)$, and each registrar R_i maintains some on-chain identity $\mathsf{addr}_{\mathsf{R}_i}$. The order of registrars is determined beforehand.

The registration phase is similar to the two-phase process in Section 5.3. First, each user j sends its public key $pk_{pub}^{(j)}$ and $\mathsf{id}^{(j)}$ to R_1. This first registrar then verifies that all the identities are legitimate; if not it drops the illegitimate identities and waits to receive a legitimate set of k users. For each user, R_1 then picks a random secret key $sk_1^{(j)}$, computes $pk_1^{(j)} \leftarrow (pk_{pub}^{(j)})^{sk_1^{(j)}}$, and keeps the mapping $sk_1^{(j)} \mapsto pk_{pub}^{(j)}$. It then performs a permutation π_1 on the identities and sends the public keys $\{pk_1^{(j)}\}_{j=1}^{k}$ and the permuted identities $\pi_1(\{\mathsf{id}^{(j)}\}_{j=1}^{k})$ to R_2. For all i, $2 \leq i < n$, R_i verifies for itself the set of identities and, if they are eligible, picks for each user j a random secret key $sk_i^{(j)}$, computes

$$pk_i^{(j)} \leftarrow (pk_{i-1}^{(j)})^{sk_i^{(j)}},$$

and keeps the mapping $sk_i^{(j)} \mapsto pk_{i-1}^{(j)}$. It then applies its own permutation π_i to the mapping and sends the public keys $\{pk_i^{(j)}\}_{j=1}^{k}$ and the permuted identities $\pi_i \circ \cdots \pi_1(\{\mathsf{id}^{(j)}\}_{j=1}^{k})$ to R_{i+1}. Finally, R_n creates k revocation keypairs $(pk_{\mathsf{rev}}^{(j)}, sk_{\mathsf{rev}}^{(j)}) \xleftarrow{\$} \mathsf{Sig.KeyGen}(1^\lambda)$ and k transactions

$$\mathsf{tx}(\{\mathsf{addr}_{\mathsf{R}_1}, \ldots, \mathsf{addr}_{\mathsf{R}_n}\} \to \{\mathsf{addr}(pk_n^{(j)}), \mathsf{addr}(pk_{\mathsf{rev}}^{(j)})\}). \qquad (1)$$

It signs each transaction $\mathsf{tx}^{(j)}$ of this form with its private key.

In the second phase, the registrars must now jointly create the transactions to publish to the blockchain, and create an onion (as in Section 5.3) to send the keys back to the users. So, R_n first signs each transaction $\mathsf{tx}^{(j)}$ of the form specified in Equation 1 with its private key. It then encrypts $sk_n^{(j)}$ with $pk_{n-1}^{(j)}$ to form $c_n^{(j)} \xleftarrow{\$} \mathsf{Enc}(pk_{n-1}^{(j)}, (sk_n^{(j)}, \perp))$ and sends the set $\{\mathsf{tx}^{(j)}, c_n^{(j)}\}_{j=1}^{k}$ to R_{n-1}.

For all i, $n > i \geq 2$, R_i incorporates its own signature into the transactions $tx^{(j)}$, encrypts $sk_i^{(j)}$ with $pk_{i-1}^{(j)}$ to form $c_i^{(j)} \xleftarrow{\$} \mathsf{Enc}(pk_{i-1}^{(j)}, (sk_i^{(j)}, c_{i+1}))$, and sends $\{\mathsf{tx}^{(j)}, c_i^{(j)}\}_{j=1}^{k}$ to R_{i-1}.

Finally, R_1 incorporates its own signature into the transactions $\mathsf{tx}^{(j)}$ and, now that they have the full set of signatures needed for validity, publishes these transactions to the blockchain. It also creates $c_1^{(j)} \xleftarrow{\$} \mathsf{Enc}(pk_{pub}^{(j)}, (sk_1^{(j)}, c_2^{(j)}))$ and sends $c_1^{(j)}$ to each user j.

At the end, user j recovers the secret key shares $sk_i^{(j)}$ in the same manner as in in Section 5.3; i.e., they compute $(sk_i^{(j)}, c_{i+1}^{(j)}) \leftarrow \mathsf{Dec}(sk_{i-1}^{(j)}, c_i^{(j)})$ for all i, $1 \leq i \leq n$ (using $sk_0 = sk_{pub}$), and computes the secret key as $sk_n^{(j)} \leftarrow sk_{pub}^{(j)} \cdot \prod_{i=1}^{n} sk_i^{(j)}$ and the public key as

$$pk_n^{(j)} \leftarrow (pk_{pub}^{(j)})^{\prod_{i=1}^{n} sk_i^{(j)}}.$$

In the verification phase, the verifier check for the existence of the transaction in the blockchain and verifies that the contents of $pk_{\mathsf{rev}}^{(j)}$ are unspent.

As in Section 5.3, the revocation request can be initiated only by R_1, but revocation can be carried out only by R_n. R_1 can initiate the process by sending a revocation request for $pk_1^{(j)}$ (which represents $\mathsf{id}^{(j)}$) to R_2. In turn, R_i transmits the revocation request to R_{i+1} using their partial key $pk_i^{(j)}$ and their knowledge of the mapping $pk_{i-1}^{(j)} \mapsto pk_i^{(j)}$. Once this reaches R_n, it can spend the coins in $pk_{\mathsf{rev}}^{(j)}$ to revoke the registration.

Security analysis

Verification is passive, as the verifier needs to check only whether or not certain transactions exist in the blockchain.

Attribute integrity is satisfied, as long as one registrar is honest: every registrar verifies the set of identities $\{\mathsf{id}^{(j)}\}_{j=1}^{k}$ for themselves, so if one registrar is honest then it will drop any fake identities and users cannot register fake ones. As in our previous protocols, malicious registrars cannot impersonate a user as they do not have access to the private key.

Privacy is satisfied, as k-anonymity is provided as long as one registrar is honest. In particular, R_i knows only the mapping between $pk_{i-1}^{(j)}$ and $pk_i^{(j)}$, and only R_1 knows the mapping between $\mathsf{id}^{(j)}$ and $pk_1^{(j)}$. Thus, as long as not all registrars collude, $\mathsf{id}^{(j)}$ and $pk_n^{(j)}$ are unlinkable.

6 Implementation and Deployment

We now present an implementation of the decentralized registration systems described in Sections 5.1 and 5.2; i.e., a system that allow for decentralized registration in both a standard (in which no privacy is achieved) and blinded (in which privacy is achieved) fashion. Our implementation is built on top of SCPKI [1], which implements a basic web of trust system on the Ethereum blockchain. We extended SCPKI to supported a blinded web of trust.

6.1 Overview

We have developed an identity management system based on blind signatures and deployed it on the Ethereum blockchain as a smart contract. As in SCPKI, each user has their own identity on the blockchain that corresponds to an Ethereum address. Using the methods of the smart contract, users can add attributes to their Ethereum address, sign attributes, and revoke signatures. The system also provides a way for users to search and retrieve attributes, by producing Ethereum events, which allow clients to efficiently watch the blockchain for new changes by a smart contract.

Due to the expensive fees of Ethereum data storage, data associated with attributes may be stored off the blockchain but authenticated on the blockchain. This can be done by adding an address (e.g., a URI) for the location of the data instead of the data itself along with its cryptographic hash if necessary for authenticity. The smart contract allows for the ability to store data using IPFS (`https://ipfs.io/`) where the cryptographic hash of the data is also its address.

The signing and verification of signature validity is performed client-side. As described in Section 5.2, when checking a signature, the client must also look for the existence of a revocation transaction as well as check the optional signature expiry date. Because of the incompatibility discussed in Section 5.2 between revocation and privacy, our implementation does not allow for the revocation of blind signatures — only standard signatures can be revoked.

6.2 Technical specification

The smart contract is written in Solidity, a high-level language for writing Ethereum contracts, and the client is written in Python. Our open-source implementation, based on SCPKI, consists of 1502 lines of Python and Solidity code. The client is a command line console application and provides access to the smart contract's methods and functionality to search for user attributes, retrieve attributes, retrieve signatures, and verify signatures. For the blind signature, we use the RSA blind signature scheme with 2048-bit RSA keys.

Simple signing In the setup phase, the user generates their own Ethereum address. To obtain a simple signature the user first adds an attribute to their Ethereum address, by calling the method **addAttribute** and specifying the attribute type and data. This creates an **AttributeAdded** event on the blockchain containing the attribute properties, which can be detected by the client. Because Ethereum events are indexable, the client can easily search for attributes. In the registration phase, the registrar signs the attribute by calling the method **signAttribute** and specifying the ID of the attribute to sign and optionally an expiry date of its signature. This creates an **AttributeSigned** event containing the signature properties, including the Ethereum address of the signer. Because only the owner of the private key of an Ethereum address can create transactions originating from that address, this cryptographically proves that a specific

Ethereum address signed an attribute. In the verification phase, the verifier checks the published signature on the user's attributes, checking that there are no revocations and that the signature has not expired.

Blind signing If a user wants to obtain a blind signature in order to anonymously register a public key, they first publish a blinded public key attribute using the method **addBlindedAttribute**, providing the data for the blinded key and specifying the ID of the registrar's public key attribute on the blockchain that the key is blinded for; i.e., specifying which registrar the user wants to blindly sign the key. This creates a **BlindedAttributeAdded** event that can be detected by the owner of the signing public key attribute. To blindly sign an attribute, the registrar calls the method **signBlindedAttribute** on the blinded public key attribute previously added by the user, providing the data of the signature, thnis is done client-side. This creates a **AttributeBlindSigned** event. On receiving the event, the user can then unblind the signature client-side. In the verification phase, the user shows the unblinded signature to the verifier (as described in Section 5.2).

6.3 Costs

In Ethereum, every operation has a cost paid using gas. As of May 2017, this cost can be translated into ether and USD using the exchange rate of 1 gas = 0.00000002 ether, and 1 ether = \$192.00. Table 2 shows the cost of each operation

operation	gas	ether	USD
publish contract	786,586	0.0157	3.01
add standard RSA attribute	70,952	0.0014	0.27
add standard RSA attribute (IPFS)	40,713	0.0008	0.15
sign standard attribute	49,904	0.001	0.19
revoke standard attribute	28,514	0.0006	0.12
add blinded RSA attribute	60,173	0.0012	0.23
add blinded RSA attribute (IPFS)	38,303	0.0008	0.15
sign blinded RSA attribute	58,012	0.0012	0.23
sign blinded RSA attribute (IPFS)	36,079	0.0007	0.13

Table 2. Cost for operations, where all data is stored on the blockchain.

when data is stored on and off the blockchain. Aside from the observation that operations are relatively cheap — publishing the contract is the most expensive step, at about \$3, and all of the operations involving individual attributes cost a few cents — we also see that the operations that involve adding and signing attributes are significantly cheaper when the data representing attributes and blind signatures is stored on IPFS.

7 Conclusions and Open Problems

In this paper, we have proposed different methods for achieving registration in public distributed ledgers. We presented a decentralized setting, where registration is potentially flexible and can be done by several entities. For each case we presented the trade-offs between security (in the form of privacy and integrity), usability (in the form of passive or active verification), and efficiency. Moreover, all our solutions use only lightweight cryptographic primitives, as opposed to approaches that adopt zero-knowledge proofs or other advanced cryptography. We have also implemented a decentralized registration process that operates on the Ethereum blockchain and evaluated its costs and efficiency.

Our system doesn't provide a mechanism for key recovery, but we view this as an important open problem and an avenue for future research, especially in the setting in which a user has accumulated many signatures on an attribute and built up a robust on-chain identity.

Acknowledgements

This project was supported in part by EPSRC Grant EP/N028104/1.

References

1. M. Al-Bassam. SCPKI: A smart contract-based PKI and identity system. In *Proceedings of the ACM Workshop on Blockchain, Cryptocurrencies and Contracts*, BCC '17, pages 35–40, New York, NY, USA, 2017. ACM.
2. L. Alvisi, A. Clement, A. Epasto, S. Lattanzi, and A. Panconesi. SoK: The evolution of sybil defense via social networks. In *2013 IEEE Symposium on Security and Privacy*, pages 382–396, Berkeley, California, USA, May 19–22, 2013. IEEE Computer Society Press.
3. D. Basin, C. Cremers, T. H.-J. Kim, A. Perrig, R. Sasse, and P. Szalachowski. ARPKI: Attack Resilient Public-Key Infrastructure. In *Proceedings of ACM CCS 2014*, pages 382–393, 2014.
4. E. Ben-Sasson, A. Chiesa, C. Garman, M. Green, I. Miers, E. Tromer, and M. Virza. Zerocash: Decentralized anonymous payments from Bitcoin. In *Proceedings of the IEEE Symposium on Security and Privacy*, 2014.
5. L. T. A. N. Brandão, N. Christin, G. Danezis, and Anonymous. Towards mending two nation-scale brokered identification systems. In *Proceedings on Privacy Enhancing Technologies*, 2015.
6. M. Caldwell and A. Voisine. *Passphrase-protected private key*, 2016.
7. J. Camenisch and A. Lysyanskaya. An efficient system for non-transferable anonymous credentials with optional anonymity revocation. In B. Pfitzmann, editor, *EUROCRYPT 2001*, volume 2045 of *LNCS*, pages 93–118, Innsbruck, Austria, May 6–10, 2001. Springer, Berlin, Germany.
8. J. Camenisch and A. Lysyanskaya. Signature schemes and anonymous credentials from bilinear maps. In M. Franklin, editor, *CRYPTO 2004*, volume 3152 of *LNCS*, pages 56–72, Santa Barbara, CA, USA, Aug. 15–19, 2004. Springer, Berlin, Germany.

9. R. Cellan-Jones. Blockchain and benefits - a dangerous mix? `http://www.bbc.com/news/technology-36785872`. Accessed: 2016-08-04.
10. D. Chaum. Blind signatures for untraceable payments. In D. Chaum, R. L. Rivest, and A. T. Sherman, editors, *CRYPTO'82*, pages 199–203, Santa Barbara, CA, USA, 1982. Plenum Press, New York, USA.
11. D. Chaum. Security without identification: Transaction systems to make big brother obsolete. *Communications of the ACM*, 28(10):1030–1044, 1985.
12. Consensys. uport: The wallet is the new browser. `https://medium.com/@ConsenSys/uport-the-wallet-is-the-new-browser-b133a83fe73#.jquv8q5u3`. Accessed: 2016-08-04.
13. L. Evenstad. Dwp trials blockchain technology for benefit payments. `http://www.computerweekly.com/news/450300034/DWP-trials-blockchain-technology-for-benefit-payments`. Accessed: 2016-08-04.
14. C. Fromknecht, D. Velicanu, and S. Yakoubov. A decentralized public key infrastructure with identity retention. IACR Cryptology ePrint Archive, Report 2014/803, 2014. `http://eprint.iacr.org/2014/803.pdf`.
15. C. Garman, M. Green, and I. Miers. Decentralized anonymous credentials. In *Proceedings of the NDSS Symposium 2014*, 2014.
16. S. Goldwasser and M. Bellare. Lecture notes on cryptography. `http://cseweb.ucsd.edu/~mihir/papers/gb.pdf`, 2000.
17. T. Hardjono and A. S. Pentland. Verifiable anonymous identities and access control in permissioned blockchains, 2016. `http://www.mit-trust.org/s/ChainAnchor-Identities-04172016.pdf`.
18. U. C. Office and G. D. Service. Introducing GOV.UK Verify, Sept. 2015. `https://www.gov.uk/government/publications/introducing-govuk-verify`.
19. G. Plimmer. Use of bitcoin tech to pay UK benefits sparks privacy concerns. `http://www.ft.com/cms/s/0/33d5b3fc-4767-11e6-b387-64ab0a67014c.html`.
20. P. Schmidt. *Certificates, Reputation, and the Blockchain*, 2015.
21. U. S. P. Service. Federal cloud credential exchange (fccx), Aug. 2013. `https://www.fbo.gov/spg/USPS/SSP/HQP/1B-13-A-0003/listing.html`.