Decentralized Anonymous Credentials

Christina Garman, Matthew Green, Ian Miers

The Johns Hopkins University Department of Computer Science, Baltimore, USA
{*cgarman, mgreen, imiers*}*@cs.jhu.edu*

October 1, 2013

Abstract

Anonymous credentials provide a powerful tool for making assertions about identity while maintaining privacy. However, a limitation of today's anonymous credential systems is the need for a trusted credential issuer — which is both a single point of failure and a target for compromise. Furthermore, the need for such a trusted issuer can make it challenging to deploy credential systems in practice, particularly in the *ad hoc* network setting (e.g., anonymous peer-to-peer networks) where no single party can be trusted with this responsibility.

In this work we propose a novel anonymous credential scheme that eliminates the need for a trusted credential issuer. Our approach builds on recent results in the area of electronic cash and uses techniques — such as the calculation of a distributed transaction ledger — that are currently in widespread deployment in the Bitcoin payment system. Using this decentralized ledger and standard cryptographic primitives, we propose and provide a proof of security for a basic anonymous credential system that allows users to make flexible identity assertions with strong privacy guarantees. Finally, we discuss a number of practical applications for our techniques, including resource management in ad hoc networks and prevention of Sybil attacks. We implement our scheme and measure its efficiency.

1 Introduction

Traditionally, making statements about identity on the Internet, whether actual assertions of identity ("I am Spartacus") or about one's identity ("I am a gladiator") involves centralized providers who issue a credential attesting to that verification. These organizations, which include Certificate Authorities, DNS maintainers, or login providers like Google and Facebook, play a large role in securing internet infrastructure, email, and financial transactions. Our increasing reliance on these providers raises concerns about privacy and trust.

Anonymous credentials, introduced by Chaum [21] and developed in a line of subsequent works [4, 9, 13, 14, 16], represent a powerful solution to this privacy concern: they deprive even colluding credential issuers and verifiers of the ability to identify and track their users. Although credentials may involve direct assertions of identity, they may also be used for a large range of useful assertions, such as "my TPM says my computer is secure", "I have a valid subscription for content", "I have a certain reputation", or "I am eligible to vote."

Indeed, anonymous credentials have already seen several practical applications. The most widely deployed example is the Direct Anonymous Attestation (DAA) portion of the Trusted Platform Module specification [2, 10]. DAA extends the standard attestation capabilities of the Trusted Platform Module to allow for anonymous attestations of TPM state and to admit *pseudonyms* that are cryptographically bound to the TPM's internal identity certificate.

Unfortunately current anonymous credential systems such as DAA have a fundamental limitation: while identity certification itself can be performed by a variety of centralized and decentralized processes, all existing anonymous credential systems employ blind signatures and thus require the appointment of a central, *trusted* party to issue the credentials. This issuer represents a single point of failure and an obvious target for compromise, both of which can seriously damage the reliability of the credential system. Moreover, compromise or issuer malfeasance can be particularly difficult to detect in an anonymous credential system. As a result, in distributed settings such as ad hoc or peer-to-peer networks, it may be challenging to identify parties who can be trusted to play this critical role or verify that the trust is well placed.

These challenges raise the following question: is it possible to build practical *decentralized* anonymous credential systems, where the process of issuing credentials — if not the establishment of identity itself — no longer depends on a trusted party?

Our contribution. In this paper we answer this question in the affirmative, proposing a new technique for constructing anonymous credentials in a decentralized setting. Our approach extends prior work on anonymous credentials but removes the need to appoint a trusted credential issuer. A consequence of this result is that our credential system can be instantiated on-demand and operated by an *ad hoc* group of mistrustful peers. We further show how to extend our credential scheme to create *updatable* (e.g., stateful) anonymous credentials in which users obtain new credentials based on changing properties of their identity.

As a basic ingredient, our protocols require the existence of a *distributed public append-only ledger*, a technology which has most famously been deployed in distributed electronic currencies such as Bitcoin [39]. This ledger can be employed by individual nodes to make assertions about identity in a fully anonymous fashion *without* the assistance of a credential issuer.

We show that our techniques have several immediate applications. They include:

- **Decentralized Direct Anonymous Attestation.** We show how to decentralize the Direct Anonymous Attestation protocol, allowing individual collections of nodes in an ad hoc or distributed system to securely assert properties of their system state. Our approach removes the need for a single party to manage DAA credentials and therefore eliminates a major limitation of the existing system. We provide an exemplary description of our decentralized (dDAA) construction.

- **Anonymous resource management in ad hoc networks.** Peer-to-peer networks are vulnerable to impersonation attacks, where a single party simulates many different peers in order to gain advantage against the network [28]. We show that our credentials may be useful in mitigating these attacks. The basic approach is to construct an *anonymous subscription service* [12, 24, 35] where parties would establish unique or costly pseudonyms (for example by submitting a valid TPM credential or paying a sum of digital currency). They can then assert possession on their identity under a specific set of restrictions, e.g., a limit to the number of requests they can make in each time period.

- **Auditable credentials.** Our techniques may also be used to extend existing centralized credential systems by allowing for public audit of issued credentials. This helps to guard against compromised credential issuers and allows the network to easily detect and revoke inappropriate credential grants. For example, in Direct Anonymous Attestation (DAA) one might want to prevent a malicious DAA authority from covertly granting certificates to users who do not have a TPM or whose TPM did not attest.

Is decentralized credential issuance valuable? Before proceeding to describe our protocols, it is worth asking whether decentralizing the issuance of anonymous credentials is a useful goal at all. After all, identity credentialing is frequently a centralized process. One might ask: what do we gain by decentralizing the issuance of *anonymous* credentials?

A first response to this question is that most anonymous credential systems separate identity certification from the process of issuing anonymous credentials. For example, while each TPM ships with a certified identity in the form of an Endorsement Key (EK) certificate, the corresponding anonymous credential is not included within the TPM itself. In order to use the DAA system, organizations must configure a local server to validate identity certifications and issue the corresponding credential. This adds a cumbersome step to the anonymous attestation system and also introduces a point of failure. Indeed, this pattern of a trusted party transforming existing credentials into an *anonymous* credential repeats in many settings. Allowing for the distributed issue of anonymous credentials, even if they can only certify centrally validated assertions, removes this additional point of trust.

A more interesting question is whether identity certification itself can be decentralized. At least for certain claims, this seems like a promising direction. For example, non-extended validation SSL certificates are simply an assertion that the bearer controls the specified domain.[1] Similarly, DNS names are generally an assertion that the owner was the first to register that name and wants it mapped to certain values (e.g., an IP address). In both cases, since these claims are publicly verifiable by simple criteria, a distributed set of entities can easily validate these claims for themselves.

In fact, a fork of Bitcoin, Namecoin [40], uses Bitcoin's append-only ledger mechanism to maintain such first-come first-serve name-value mappings. Individuals register a name and an owning public key. Provided they are the first to register that name, they can make arbitrary updates to the associated values by signing them with the registered key. A DNS system built atop this — DotBIT — is already in experimental deployment. Namecoin can also be used to maintain mappings from names to public keys. One could imagine more complex semantics for allowing name registration — e.g., proofs of work, proofs of payment, TPM attestations, publicly verifiable proofs of storage and retrievability of files [49] — supporting more sophisticated functionality than simple DNS.

1.1 Overview of Our Construction

We now provide a brief overview for our construction, which is inspired by the electronic cash proposals of Sander and Ta-Shma [47] and Miers et al. [38].

Issuing and showing credentials. The ability to establish identities and bind them to a public key ensures that users can assert their identity in a non-anonymous fashion, simply by issuing signatures from the corresponding secret key. Unfortunately, this does not immediately show us how to construct anonymous credentials, since traditional anonymous credentials consist of a signature computed by a credential issuer. Since no central party exists to compute the credential signature, this approach does not seem feasible without elaborate (and inefficient) use of threshold cryptography.[2]

We instead take a different approach. To issue a new credential in our decentralized system, the user establishes an identity and related attributes as described above. She then attaches a vector commitment to her secret key sk_U along with the identity and attribute strings that are contained within her identity assertion. Finally, she includes a non-interactive proof that the credential is correctly constructed, i.e., that the attributes in the commitment correspond to those revealed in the identity assertion. The network will accept the identity assertion if and only if the assertion is considered correct and the attached proof is valid.

At a later point an individual can prove possession of such a credential by proving the following two statements in zero-knowledge:

1. She knows a commitment C_i in the set $(C_1, \ldots, C_N)$ of all credentials previously accepted to the block chain.
2. She knows the opening (randomness) for the commitment.

In addition to this proof, the user may simultaneously prove additional statements about the identity and attributes contained within the commitment C_i. The challenge in the above construction is to efficiently prove statements (1) and (2), i.e., without producing a proof that scales with N. Our solution, which adapts techniques from distributed e-cash systems [38], circumvents this problem by using an efficient publicly-verifiable accumulator [14] to gather the set of all previous commitments together. Using this accumulator in combination with an efficient membership proof due to Camenisch and Lysyanskaya in [15], we are able to reduce the size of this proof to $O(\lambda)$ for security parameter λ, rather than the $O(N \cdot \lambda)$ proofs that would result from a naive OR proof.

[1]In practice, CA's usually verify that the bearer controls some administrator email such as admin@domain or webmaster@domain.

[2]A possibility is to use ring signatures [46], which do not require a single trusted signer. Unfortunately these signatures grow with the number of participating signers and require expensive communication to generate.

Of course, merely applying these techniques does not lead to a *practical* credential system. A key contribution of this work is to supply a concrete instantiation of the above idea under well-studied assumptions and to prove that our construction provides for consistency of credentials (ensuring multiple users cannot pool their credentials), the establishment of pseudonyms, and a long set of extensions built upon anonymous credentials. Last but not least we need to formally define and prove the security of a distributed anonymous credential scheme and provide some model for the distributed ledger. Our instantiation requires a single trusted setup phase, after which the trusted party is no longer required.[3]

1.2 Outline of This Work

The remainder of this work is organized as follows. In the next section we discuss how to get a distributed bulletin board. In §3 we discuss specific applications for decentralized anonymous credentials and argue that these systems can be used to solve a variety of problems in peer-to-peer networks. In §5 we describe the cryptographic building blocks of our construction, and in §4 we define the notion of a *decentralized anonymous credential scheme* and provide an ideal-world security definition. In §6 we provide an overview of our basic construction as well as a specific instantiation based on the Discrete Logarithm and Strong RSA assumptions. In §7 we extend our basic construction to add a variety of useful features, including k-show credentials, stateful credentials, and credentials with hidden attributes. In §8 we describe the implementation and performance of a prototype library realizing our credential system. Finally, in §9, we show how to use our library to build a distributed version of anonymous attestation.

2 Real-world Distributed Bulletin Boards

A core component of our system is a distributed append-only bulletin board we can use to post issued credentials. The board must provide three strong security guarantees: (1) that credentials must not be tampered with once added to the board, (2) no party can control the addition of credentials to the board, and (3) all parties will share a consistent view of the board. While there are several ways to distribute such a service, relatively few have been proven efficient and secure in a highly adversarial environment.

A notable exception comes from the Bitcoin distributed currency system [39], which has grown since 2009 to handle between \$2-\$5 million USD/day in transaction volume in a highly adversarial environment. The heart of Bitcoin is the block chain, which serves as an append-only bulletin board maintained in a distributed fashion by the Bitcoin peers. The block chain consists of a series of blocks connected in a hash chain.[4] Every Bitcoin block memorializes a set of transactions (containing an amount of bitcoin, a sender, and a recipient) that are collected from the Bitcoin broadcast network. Thus the network maintains a consensus about what transactions have occurred and how much money each user has.

Bitcoin peers, who are free to enter and leave the network, compete to generate the next block by trying to calculate $\mathsf{H}(block \mid\mid nonce) < t$ where H is a secure hash function and t is an adjustable parameter. This process is known as *mining*, and the difficulty level t is adjusted so that a block is created on average every 10 minutes. When a block is generated, it is broadcast to the network and, if valid, accepted as the next entry in the block chain. Bitcoin and related systems provide two incentives to miners: (1) mining a block (i.e., completing the proof of work) entitles them to a reward[5] and (2) nodes can collect fees from every transaction in a block they mine.

While Bitcoin uses the hash chain for the specific purpose of implementing an electronic currency, the usefulness of the Bitcoin bulletin board already been recognized by several related applications. One spinoff of the Bitcoin concept is Namecoin [40], a fork of Bitcoin that uses the block chain to maintain key-value mappings. Namecoin is currently being used to implement an experimental DNS replacement, dotBit [27]. Users pay a small fee to register a key-value pair along with a controlling public key. They can then make

[3]In §7 we discuss techniques for removing this trusted setup requirement.
[4]For efficiency reasons, the hash chain is actually a Merkle Tree.
[5]For Bitcoin this reward is set at 25 BTC but will eventually diminish and be eliminated.

updates to the pair provided (1) the updates are signed by that key and (2) if necessary, they pay a transaction fee.[6] Due to this flexibility we use the Namecoin software in our implementations, but we stress that the same techniques can be used with nearly any hash chain based network, including mature deployments such as Bitcoin.

3 Applications

In this section we discuss several of the applications facilitated by decentralized anonymous credentials. While we believe that these credential systems may have applications in a variety of environments, we focus specifically on settings where trusting a central credential issuer is not an option or where issued credentials must be publicly audited.

Mitigating Sybil attacks in ad hoc networks. Impersonation attacks can have grave consequences for both the security and resource allocation capabilities of ad hoc networks. A variety of solutions have been proposed to address this problem. One common approach is to require that clients solve *proofs of work*: resource-consuming challenges that typically involve either storage or computation [28]. Unfortunately it can be challenging to re-use a single proof of work in an anonymous fashion, i.e., without either identifying participants or allowing other users to clone the solution.

One solution to this problem is to use k-show anonymous credentials. In this approach, peers establish a single credential by solving a proof of work. This allows the peer to obtain a credential that can be used a limited number of times or a limited number of times within a given time period. When a peer exceeds the k-use threshold (e.g., by cloning the credential for a Sybil attack), the credential can be identified and revoked. We note that this proposal is a distributed variant of the *anonymous subscription service* concept, which was first explored in [12, 24].

Managing resource usage. In networks where peers both contribute and consume resources, ensuring fair resource utilization can be challenging. For example, a storage network might wish to ensure peers provide as much storage as they consume [42] or ensure that peers fairly use network bandwith [43]. This can be problematic in networks that provide anonymity services (e.g., Tor) or VoIP[7], where peers may be reluctant to identify which traffic they originated. An anonymous credential system allows peers to identify their contributions to routing traffic in exchange for a credential which they can then use to originate traffic. This helps to ensure that peers contribute resources back to the network while preserving their anonymity.

4 Decentralized Anonymous Credentials

A traditional anonymous credential system has two types of participants: users and organizations. Users, who each have a secret key sk_U, are known by pseudonyms both to each other and organizations. Nym_A^O, for example, is the pseudonym of user A to organization O. Decentralized anonymous credentials have no single party representing the organization. Instead, this party is replaced with a quorum of users who enforce a specific *credential issuing policy* and collaboratively maintain a list of credentials thus far issued. For consistency with prior work, we retain the term "organization" for this group.

A distributed anonymous credential system consists of a global transaction ledger, a set of transaction semantics, as well as the following (possibly probabilistic) algorithms:

- $\mathsf{Setup}(1^\lambda) \rightarrow params$. Generates the system parameters.
- $\mathsf{KeyGen}(params) \rightarrow sk_U$. Run by a user to generate her secret key.
- $\mathsf{FormNym}(params, U, E, sk_U) \rightarrow (Nym_U^E, sk_{Nym_U^E})$. Run by a user to generate a pseudonym Nym_U^E and an authentication key $sk_{Nym_U^E}$ between a user U and some entity (either a user or an organization) E.

[6]Currently, neither Namecoin nor Bitcoin require significant transaction fees.
[7]Prior its acquisition by Microsoft, Skype used a peer-to-peer overlay network.

- $\mathsf{MintCred}(params,\ sk_U,\ Nym_U^O,\ sk_{Nym_U^O},\ attrs,\ aux) \rightarrow (c, sk_c, \pi_M)$. Run by a user to generate a request for a credential from organization O. The request consists of a candidate credential c containing public attributes $attrs$; the user's key sk_U; auxiliary data aux justifying the granting of the credential; and a proof π_M that (1) Nym_U^O was issued to the same sk_U and (2) the credential embeds $attrs$.
- $\mathsf{MintVerify}(params, c, Nym_U^O, aux, \pi_M) \rightarrow \{0,1\}$. Run by nodes in the organization to validate a credential. Returns 1 if π_M is valid, 0 otherwise.
- $\mathsf{Show}(params, sk_U, Nym_U^V, sk_{Nym_U^V}, c, sk_c, \mathbf{C}_O) \rightarrow \pi_S$. Run by a user to non-interactively prove that a given set of attributes are in a credential c in the set of issued credentials $\mathbf{C}_O$ and that c was issued to the same person who owns Nym_U^V. Generates and returns a proof π_S.
- $\mathsf{ShowVerify}(params, Nym_U^V, \pi_S, \mathbf{C}_O) \rightarrow \{0,1\}$. Run by a verifier to validate a shown credential. Return 1 if π_S is valid for Nym_U^V, 0 otherwise.

We now describe how these algorithms are used in the context of an anonymous credential system.

4.1 Overview of the Protocol Semantics

To realize the full anonymous credential system, we integrate the above algorithms with a decentralized hash chain based bulletin board as follows. In this section we assume a bulletin board such as Namecoin that provides a means for storing arbitrary key-value pairs.[8] We provide a concrete realization of our protocols in Sections 6 and 8.

Formulating a pseudonym. Prior to requesting a new credential, the user executes the KeyGen algorithm to obtain sk_U and then runs the FormNym algorithm to obtain a pseudonym for use with this organization. This requires no interaction with the bulletin board, hence the user can perform these actions offline.

Obtaining a credential. To obtain a credential, the user places the organization name and some public identity assertion — for example, a TPM attestation and AIK certificate chain — into the auxiliary data field aux, then executes the MintCred routine to obtain a credential and a signature of knowledge on that information. She then formulates a transaction including both the resulting credential and the auxiliary data and broadcasts it into the hash chain network, along with (optionally) some sum of digital currency to pay for the transaction fees. She retains the secret portion of the credential.

Once received by the network, all parties can verify the correctness of the credential and the identity assertion using the MintVerify routine and whatever external procedures are needed to verify the auxiliary data. This process can be conducted directly by the network nodes, or it can be validated after the fact by individual credential verifiers.

Showing a credential. When a user wishes to show a credential to some Verifier, she first scans through the bulletin board to obtain a set $\mathbf{C}_O$ consisting of all candidate credentials belonging to a specific organization. She next verifies each credential using the MintVerify routine (if she has not already done so) and validates the auxiliary identity certification information. She then runs the Show algorithm to generate a credential, which she transmits directly to the Verifier. The Verifier also collects the set of credentials in $\mathbf{C}_O$ and validates the credential using the ShowVerify routine. She accepts the credential certification if this routine outputs 1.

4.2 Security

We define our system in terms of an ideal functionality implemented by a trusted party T_P that plays the role that our cryptographic constructions play in the real system. All communication takes place through this ideal trusted party. Security and correctness for our system comes from a proof that this ideal model is indistinguishable from the real model provided the cryptographic assumptions hold. Our ideal functionality is outlined in Figure 1.

[8]While this functionality is supported by default in Namecoin, it is also possible to store arbitrary data in existing block chains such as the Bitcoin chain.

- $RegNym(Nym_U^O, U, O)$: U logs into T_P with sk_U to register a nym with organization O. If she does not have an account, she first creates one. She gives T_P a unique random string Nym_U^O for use as her nym with O. T_P checks that the string is indeed unique and if so stores (Nym_U^O, U, O) and informs U.
- $MintCred(Nym_U^O, O, attrs, aux)$: U logs into T_P authenticating with sk_U. If Nym_U^O is not U's nym with O or sk_U is wrong, reject. Otherwise, T_P checks that aux justifies issuing a credential under O's issuing policy and if so generates a unique random id ID and stores $(Nym_U^O, U, ID, attrs)$. It then adds ID to its public list of issued credentials for O.
- $ShowOnNym(Nym_U^O, Nym_U^V, O, V, attrs, \mathbf{C})$: U logs into T_P with sk_U. If Nym_U^O is not U's nym with O or Nym_U^V is not U's nym with V, reject. Else, T_P checks if the tuple (Nym_U^O, U) exists, if ID associated with that tuple is in the set of credentials $\mathbf{C}$ that U provided, and if the given attributes $attrs$ match the attributes associated with that tuple. If all conditions hold, T_P informs V that Nym_U^V has a credential from O in the set $\mathbf{C}$. V then retrieves the set of credentials $\mathbf{C}_O$ issued by O from T_P and accepts T_P's assertion if and only if $\mathbf{C} \subseteq \mathbf{C}_O$ and O's issuing policy is valid $\forall c' \in \mathbf{C}_O$.
- $GetCredList(O)$: T_P retrieves the list of credentials for organization O and returns it.

Figure 1: Ideal Functionality. Security of a basic distributed anonymous credential system.

It consists of organizations who issue credentials and users who both prove that they have these credentials and verify such proofs. Organizations have only two things: 1) an efficient and publicly evaluable policy, $policy_O$, for granting credentials and 2) an append-only list of credentials meeting that policy maintained by the trusted party.

4.3 Trusting the Ledger

An obvious question is whether the append-only transaction ledger is necessary at all. Indeed, if the list of valid credentials can be evaluated by a set of untrusted nodes, then it seems that a user (Prover) could simply maintain a credential list compiled from network broadcasts and provide this list to the Verifier during a credential show. However, this approach can enable sophisticated attacks where a malicious Verifier manipulates the Prover's view of the network to include a poisoned-pill credential that — although valid by the issuing heuristic — was not broadcast to anyone else. When the prover authenticates, she has completely identified herself.

The distributed transaction ledgers employed by networks such as Bitcoin and Namecoin provide a solution to this problem, as their primary purpose is to ensure a shared view among a large number of nodes in an adversarial network. In practice this is accomplished by maintaining a high degree of network connectivity and employing computational *proofs of work* to compute a hash chain.

For an attacker to execute the poisoned credential attack against such a ledger, she would need to both generate and maintain a false view of the network to delude the prover. This entails both simulating the prover's view of the rest of the network complete with *all* its computational power and forging any assurances the prover might expect from known peers about the present state of the network. If the prover has a reasonable estimate of the actual network's power (e.g., she assumes it monotonically increases), then an attacker must actually have equivalent computational power to the entirety of the network to mount such an attack. For the purposes of this paper we assume such active attacks are impossible even if the attacker controls a simple majority of the computational power. Attackers are still free to attempt any and all methods of retroactively identifying a user and mount any other active attacks.

5 Preliminaries

We make use of the following complexity assumptions and cryptographic building blocks to construct our scheme.

5.1 Complexity Assumptions

The security of our scheme relies on the following two complexity assumptions:

Strong RSA Assumption [3, 30]. Given a randomly generated RSA modulus n and a random element $y \in \mathbb{Z}_n^*$, it is hard to compute $x \in \mathbb{Z}_n^*$ and integer exponent $e > 1$ such that $x^e \equiv y \bmod n$. We can restrict the RSA modulus to those of the form pq, where $p = 2p' + 1$ and $q = 2q' + 1$ are safe primes.

Discrete Logarithm (DL) Assumption [25]. Let G be a cyclic group with generator g. Given $h \in G$, it is hard to compute x such that $h = g^x$.

5.2 Cryptographic Building Blocks

Zero-knowledge proofs. In a zero-knowledge protocol [32] a user (the prover) proves a statement to another party (the verifier) without revealing anything about the statement other than that it is true. Our constructions use zero-knowledge proofs that can be instantiated using the technique of Schnorr [48], with extensions due to, e.g., [8, 17, 19, 23]. We convert these into *non-interactive* proofs by applying the Fiat-Shamir heuristic [29]. When we use these proofs to authenticate auxiliary data, we refer to the resulting non-interactive proofs as *signatures of knowledge* as defined in [20].

When referring to these proofs we will use the notation of Camenisch and Stadler [18]. For instance, $\mathsf{NIZKPoK}\{(x, y) : h = g^x \ \wedge \ c = g^y\}$ denotes a non-interactive zero-knowledge proof of knowledge of the elements x and y that satisfy both $h = g^x$ and $c = g^y$. All values not enclosed in ()'s are assumed to be known to the verifier. Similarly, the extension $\mathsf{ZKSoK}[m]\{(x, y) : h = g^x \ \wedge \ c = g^y\}$ indicates a *signature of knowledge* on message m.

Accumulators [38]. An accumulator allows us to combine many values into one smaller value (the accumulator). We then have a single element, called the witness, that allows us to attest to the fact that a given value is actually part of the accumulator. Our constructions use an accumulator based on the Strong RSA assumption. The accumulator we use was first proposed by Benaloh and de Mare [6] and later improved by Baric and Pfitzmann [3] and Camenisch and Lysyanskaya [14]. We describe the accumulator using the following algorithms:

- $\mathsf{AccumSetup}(\lambda) \to params$. On input a security parameter, sample primes p, q (with polynomial dependence on the security parameter), compute $N = pq$, and sample a seed value $u \in QR_N, u \neq 1$. Output (N, u) as $params$.
- $\mathsf{Accumulate}(params, \mathbf{C}) \to A$. On input $params$ (N, u) and a set of prime numbers $\mathbf{C} = \{c_1, \ldots, c_i \mid c \in [\mathcal{A}, \mathcal{B}]\}$,[9] compute the accumulator A as $u^{c_1 c_2 \cdots c_n} \bmod N$.
- $\mathsf{GenWitness}(params, v, \mathbf{C}) \to \omega$. On input $params$ (N, u), a set of prime numbers $\mathbf{C}$ as described above, and a value $v \in \mathbf{C}$, the witness ω is the accumulation of all the values in $\mathbf{C}$ besides v, i.e., $\omega = \mathsf{Accumulate}(params, \mathbf{C} \setminus \{v\})$.
- $\mathsf{AccVerify}(params, A, v, \omega) \to \{0, 1\}$. On input $params$ (N, u), an element v, and witness ω, compute $A' \equiv \omega^v \bmod N$ and output 1 if and only if $A' = A$, v is prime, and $v \in [\mathcal{A}, \mathcal{B}]$ as defined previously.

[9] "Where $\mathcal{A}$ and $\mathcal{B}$ can be chosen with arbitrary polynomial dependence on the security parameter, as long as $2 < \mathcal{A}$ and $\mathcal{B} < \mathcal{A}^2$." [15] For a full description, see [15, §3.2 and §3.3].

For simplicity, the description above uses the full calculation of A. Camenisch and Lysyanskaya [14] observe that the accumulator may also be *incrementally* updated, i.e., given an existing accumulator A_n it is possible to add an element x and produce a new accumulator value A_{n+1} by computing $A_{n+1} = A_n^x \bmod N$.[10]

Camenisch and Lysyanskaya [14] show that the accumulator satisfies a strong *collision-resistance* property if the Strong RSA assumption is hard. Informally, this ensures that no *p.p.t.* adversary can produce a pair (v, ω) such that $v \notin \mathbf{C}$ and yet $\mathsf{AccVerify}$ is satisfied. Additionally, they describe an efficient zero-knowledge proof of knowledge that a committed value is in an accumulator. We convert this into a non-interactive proof using the Fiat-Shamir transform and refer to the resulting proof using the following notation:

$$\mathsf{NIZKPoK}\{(v, \omega) : \mathsf{AccVerify}((N, u), A, v, \omega) = 1\}.$$

Verifiable Random Functions. A pseudorandom function (PRF) [31] is an efficiently computable function whose output cannot be distinguished (with non-negligible advantage) from random by a computationally bounded adversary. We denote the pseudorandom function as $f_k(\cdot)$, where k is a randomly chosen key. A number of PRFs possess efficient proofs that a value is the output of a PRF on a set of related public parameters. Two examples of this are the Dodis-Yampolskiy (DY) PRF [26] and the Naor-Reingold PRF [41].

Pedersen Commitments. A commitment scheme allows a user to bind herself to a chosen value without revealing that value to the recipient of the commitment. This commitment to the value ensures that the user cannot change her choice (i.e., *binding*), while simultaneously ensuring that the recipient of the commitment does not learn anything about the value it contains (i.e., *hiding*) [22]. In Pedersen commitments [45], the public parameters are a group G of prime order q, and generators $(g_0, \ldots, g_m)$. In order to commit to the values $(v_1, \ldots, v_m) \in \mathbb{Z}_q^m$, pick a random $r \in \mathbb{Z}_q$ and set $C = \mathrm{PedCom}(v_1, \ldots, v_m; r) = g_0^r \prod_{i=1}^m g_i^{v_i}$.

6 A Concrete Instantiation

We now provide a concrete instantiation of our construction and prove the security of our construction under the Discrete Logarithm and Strong RSA assumptions.

6.1 Overview of the Construction

Alice's pseudonym with a given organization/user is an arbitrary identity that she claims in a transaction. She tags this value with a Pedersen commitment to her secret key sk and *signs* the resulting transaction using a signature of knowledge that she knows the secret key. There is no separate process for registering a pseudonym: instead they are simply used in issue and show to allow operations to be linked if necessary. Alice's credential c is a vector Pedersen commitment to both sk and a set of public attributes $attrs = a_0, \ldots, a_m$, which Alice also includes in her credential. To issue a credential, Alice provides the network with a credential, a pseudonym, her attributes, optionally some auxiliary data justifying the credential issue (e.g., a proof of work that Alice is not a Sybil), and a proof that (1) the commitment and the pseudonym contain the same secret key and (2) the attributes are in some allowed set. If all of this validates, the entry is added to the ledger. Alice shows the credential under a different pseudonym by proving in zero-knowledge that (1) she knows a credential on the ledger from the organization, (2) the credential opens to the same sk as her pseudonym, and (3) it has some attributes.

6.2 The Construction

The full construction is provided in Figure 2. We use Pedersen commitments and a Strong RSA based accumulator to instantiate the core of the protocol. The proofs of knowledge in the Show algorithm are conducted using Schnorr-style proofs modified using the Fiat-Shamir heuristic as in previous work [14, 48]. The implementation of the proofs are similar to those used by Miers et al. in [38].

[10]This allows the network to maintain a running value of the accumulator and prevents individual nodes from having to recompute it [38].

- $\mathsf{Setup}(1^\lambda) \rightarrow params$. On input a security parameter λ, run $\mathsf{AccumSetup}(1^\lambda)$ to obtain the values (N, u). Next generate primes p, q such that $p = 2^w q+1$ for $w \geq 1$. Let $\mathbb{G}$ be an order-q subgroup of $\mathbb{Z}_p^\star$, and select random generators $g_0, \ldots, g_n$ such that $\mathbb{G} = \langle g_0 \rangle = \cdots = \langle g_n \rangle$. Output $params = (N, u, p, q, g_0, \ldots, g_n)$.

- $\mathsf{KeyGen}(params) \rightarrow sk$. On input a set of parameters $params$, select and output a random master secret $sk \in \mathbb{Z}_q$.

- $\mathsf{FormNym}(params, sk) \rightarrow (Nym, sk_{Nym})$. Given a user's master secret sk, select a random $r \in \mathbb{Z}_q$ and compute $Nym = g_0^r g_1^{sk}$. Set $sk_{Nym} = r$ and output (Nym, sk_{Nym}).

- $\mathsf{MintCred}(params, sk, Nym_U^O, sk_{Nym_U^O}, attrs, aux) \rightarrow (c, sk_c, \pi_M)$. Given a nym Nym_U^O and its secret key $sk_{Nym_U^O}$; attributes $attrs = (a_0, \ldots, a_m) \in \mathbb{Z}_q$; and auxiliary data aux, select a random $r' \in \mathbb{Z}_q$ and compute $c = g_0^{r'} g_1^{sk} \prod_{i=0}^{m} g_{i+2}^{a_i}$. Set $sk_c = r'$ and output (c, sk_c, π_M) where π_M is a signature of knowledge on aux that the nym and the credential both belong to the same master secret sk, i.e.:

$$\pi_M = \mathsf{ZKSoK}[aux]\{(sk, r', r) : \\ c = g_0^{r'} g_1^{sk} \prod_{i=0}^{m} g_{i+2}^{a_i} \ \wedge \ Nym_U^O = g_0^r g_1^{sk}\}$$

  Finally, submit the resulting values $(c, \pi_M, attrs)$ to the public transaction ledger.

- $\mathsf{MintVerify}(params, c, attrs, Nym_U^O, aux, \pi_M) \rightarrow \{0, 1\}$. Given a credential c, attributes $attrs$, a nym Nym_U^O, and proof π_M, verify that π_M is the signature of knowledge on aux. If the proof verifies successfully, output 1, otherwise output 0. The organization nodes should accept the credential to the ledger if and only if this algorithm returns 1.

- $\mathsf{Show}(params, sk, Nym_U^V, sk_{Nym_U^V}, c, attrs, sk_c, \mathbf{C}_O) \rightarrow \pi_S$. Given a user's master secret sk; a nym Nym_U^V between the user and the verifier and its secret key $sk_{Nym_U^V}$; a credential c and its secret key sk_c; the attributes $(a_0, \ldots, a_m)$ used in the credential; and a set of credentials $\mathbf{C}$, compute $A = \mathsf{Accumulate}(params, \mathbf{C}_O)$ and $\omega = \mathsf{GenWitness}(params, c, \mathbf{C}_O)$ and output the following proof of knowledge:

$$\pi_S = \mathsf{NIZKPoK}\{(sk, \omega, r', c, r, Nym_U^V) : \\ \mathsf{AccVerify}(params, A, c, \omega) = 1 \ \wedge \ c = g_0^{r'} g_1^{sk} \prod_{i=0}^{m} g_{i+2}^{a_i} \ \wedge \ Nym_U^V = g_0^r g_1^{sk}\}$$

- $\mathsf{ShowVerify}(params, Nym_U^V, \pi_S, \mathbf{C}_O) \rightarrow \{0, 1\}$. Given a nym Nym_U^V, proof of possession of a credential π_S, and the set of credentials issued by organization O $\mathbf{C}_O$, first compute $A = \mathsf{Accumulate}(params, \mathbf{C}_O)$. Then verify that π_S is the aforementioned proof of knowledge on c, $\mathbf{C}_O$, and Nym_U^V using the known public values. If the proof verifies successfully, output 1, otherwise output 0.

Figure 2: Our basic decentralized anonymous credential scheme.

Theorem 6.1 *The basic distributed anonymous credential system described in Figure 2 is secure in the random oracle model under the Strong RSA and the Discrete Logarithm assumptions.*

We provide a sketch of the proof of Theorem 6.1 in Appendix A.

7 Extensions

We consider extending the basic system in several ways.

7.1 k-show Credentials

Damgård et al. [24] first suggested a credential system where users could only authenticate once per time period. Camenisch et al. [12] independently proposed a significantly more efficient construction that allows for up to k authentications per time period, with the ability to revoke all cloned credentials if a credential was used beyond this limit. Camenisch et al. suggested that these techniques might be used to build anonymous subscription services, allowing users to access a resource (such as a website) within reasonable bounds. We briefly show that these same techniques can be applied to our basic credential system.

In the system of [12] an authority issues a credential on a user's secret seed s. To show a credential for the i^{th} time in validity period t, the user generates a serial number S using a verifiable random function (VRF) as $S = f_s(0||t||i)$. She also includes a non-interactive zero-knowledge proof that this serial number is correctly structured.[11] This technique can be applied to our construction provided we can securely store a seed for the VRF. This is easy: the user simply generates a random seed s and includes this value in the commitment she stores in the transaction ledger. We note that for the trivial case of one-time show credentials, we can simply reveal the seed. For k-show, the user provably evaluates the VRF on the seed plus a secret counter.[12]

7.2 Credentials with Hidden Attributes

In our basic construction of §6, users provide a full list of attributes when requesting and showing credentials. While this is sufficient for many applications, there exist cases where a user might wish to conceal the attributes requested or shown, opting instead to prove statements about them, e.g., proving knowledge of a secret key or proving that an attribute is within a certain range. There are two simple ways to do this. First, we can simply use multi-message commitments where each message is an attribute. This increases the size of our zero-knowledge proofs (they are linear in the number of messages in a commitment) but does not change our schemes. A more efficient construction is to encode the attributes in one single value and then prove statements about that committed value rather than reveal it. For example, one could prove that a given bit corresponding to a certain attribute was set. One could also use the first x bits for attribute one, the next x bits for attribute two, etc. and use range proofs [7, 11, 33, 36] to reveal only those attributes we want to display.

7.3 Stateful Credentials

A stateful anonymous credential system [22] is a variant of an anonymous credential system where credential attributes encode some state that can be updated by issuing new credentials. This credential issuance is typically conditioned on the user showing a previous credential and offering proof that the new credential should be updated as a function of the original.

Intuitively, we can already have this capability quite easily due to the fact that our credentials are non-interactively issued. We can make stateful credentials simply by changing the policy by which we issue

[11]The re-use of a credential would result in a repeated serial number, and yet the nature of the VRF's output (for an honest user) ensures that attackers cannot link individual shows.

[12]Camenisch et al. [12] describe a further extension that reveals the user's identity in the event of a credential double-show. We omit the details here for space reasons but observe that the same technique can be applied to our construction.

- $\mathsf{Update}(params, sk, c, sk_c, \mathbf{C}_O, update\_relation, state') \rightarrow (c', sk'_c, \pi_u)$. Given a credential c and associated secret key sk_c, a set of credentials $\mathbf{C}_O$, an updated state $state' = (s'_0, \ldots, s'_m) \in \mathbb{Z}_q$, and an update relation $update\_relation$, generate a fresh random serial number $S' \in \mathbb{Z}_q$ and random value $r' \in \mathbb{Z}_q$ to form a new credential $c' = g_0^{r'} g_1^{sk} g_2^{S'} \prod_{i=0}^{m} g_{i+3}^{s'_i}$. Compute $A = \mathsf{Accumulate}(params, \mathbf{C}_O)$ and $\omega = \mathsf{GenWitness}(params, c, \mathbf{C}_O)$. Output (c', sk'_c, π_u) where $sk'_c = (S', state', r')$ and

$$\begin{aligned}\pi_u = \mathsf{NIZKPoK}\{(sk, \omega, c, state, r, c', state', r') :\\ &\mathsf{AccVerify}(params, A, c, \omega) = 1\\ &\wedge\, c = g_0^{r} g_1^{sk} g_2^{S} \prod_{i=0}^{m} g_{i+3}^{s_i} \ \wedge\ c' = g_0^{r'} g_1^{sk} g_2^{S'} \prod_{i=0}^{m} g_{i+3}^{s'_i}\\ &\wedge\ update\_relation(state, state') = 1\}\end{aligned}$$

- $\mathsf{UpdateVerify}(params, c, \mathbf{C}_O, \pi_u) \rightarrow \{0, 1\}$. Given a stateful credential c, a credential set $\mathbf{C}_O$, and proof π_u, output 1 if π_u is correct, the proved state transition is a legal one, and the serial number S was not previously used. Otherwise 0.

Figure 3: Extensions for a stateful anonymous credential system. $update\_relation(\ldots) = 1$ denotes that the update encodes some arbitrary state transition (e.g. $\forall i\ s'_i = s_i + 1$).

credentials: to issue a credential in a new state s_1, we require a user to demonstrate that they had a credential in state s_0 and discard it by revealing its single use serial number.

We construct a "single show" credential c embedding some state $state$ in the attributes and a serial number S. Users are free to show c as many times as they like without revealing the serial number. However, to update the state of the credential, they must author a transaction that shows the original credential and reveals the serial number S and "mint" a new candidate credential c' containing the updated state $state'$ (hidden inside of a commitment) and a proof that there exists a valid relationship between the state encoded in c and the new state in c' (for example, that the attributes have been incremented).

This requires only minor extensions to our basic scheme composing the existing secure functionality. In this case we add an Update algorithm that operates similarly to MintCred but includes the earlier credential and a proof of its construction. A valid proof of the existing credential now becomes a condition for the organization accepting the updated credential into the ledger. We provide a description of this new algorithm in Figure 3.

8 Integrating with Proof-of-work Bulletin Boards

We provide a basic implementation of our credential scheme as a library and construct a basic example using Namecoin as the bulletin board. Our prototype system allows users to prove they have a (fresh) commitment to some attributes in an issued credential. For our purposes it is sufficient to merely reveal the content of that commitment (the attributes) in its entirety during a show. However, selectively disclosable attributes are trivially realizable, see §7.2.

8.1 Integration

Namecoin integration is straightforward. Namecoin provides a built in mechanism for storing key-value pairs which, by convention, have a namespace as a prefix. It also provides a basic functionality to scan the list of existing names. Thus we can scan for credentials, validate them, and then accumulate them. It is then simply matter of generating and verifying proofs against that computed accumulator value.

For Alice to obtain a credential, she:

1. Pays a very small fee (currently 0.0064 USD) to purchase some name in the system's namespace by registering a public key as the owner of the name. This corresponds to a transaction looking like:

   ```
   1 665a...  OP_2DROP
   OP_DUP  OP_HASH160 6c1abe34
   OP_EQUALVERIFY  OP_CHECKSIG
   ```

2. Prepares a fresh credential with some attributes and any supporting documentation necessary for her identity claim and stores the private portion of the credential.

3. Updates, using the public key from step 1, her registered name to contain a credential and its supporting documentation.

   ```
   2 642f7...  7b...
   OP_2DROP OP_2DROP
   OP_DUP OP_HASH160
   14d...  OP_EQUALVERIFY OP_CHECKSIG
   ```

Once this update is confirmed, Alice has a fully formed credential.

To show the credential to Bob, Alice:

1. Scans through the list of added names and retrieves all candidate credentials.

2. Checks the supporting documentation for each candidate and puts valid ones in **C**.

3. Runs Show with the public parameters, the private portion of her credentials, and **C** and sends the result to Bob.

4. Bob does steps 1 and 2 and computes **C** himself.

5. Bob runs ShowVerify on Alice's supplied credential and **C** to verify it.

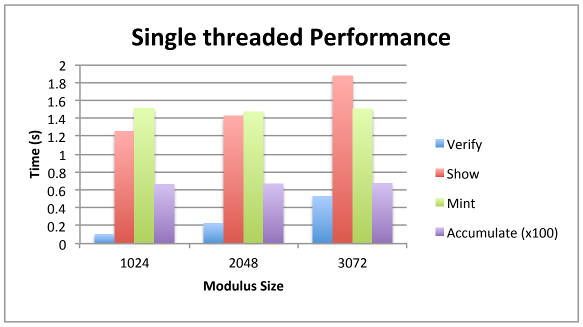
Alice has now proved she has a credential to Bob.

What the supporting documentation is and how it is verified is an application specific problem. For some applications, merely having paid the tiny registration fee may be sufficient and no verification is necessary. For others, some digital signature may need to be verified or some assertion about resource management (e.g., a proof of storage/retrievability) may need to be verified. Without modifications to Namecoin/Bitcoin, any assertion must be verifiable by all participants.[13] We consider one such application in the next section.

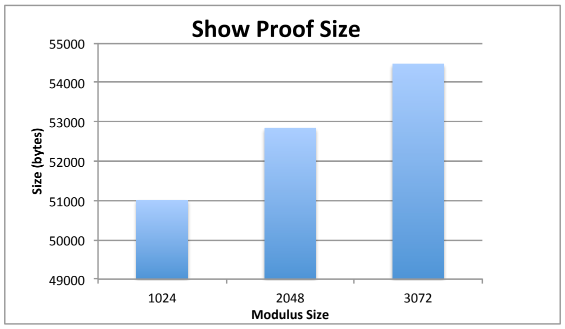
8.2 Operating Cost

Namecoin is not free to use as purchasing a name costs a small (less than 0.01 USD as of 7/31/2013) amount of money. This fee is necessary both to prevent mass name hoarding and to provide an economy to pay the miners who maintain the block chain. This cost must minimally be paid by users when creating a credential. For certain applications (e.g., k-anonymous credentials), relying parties must also post data on the block chain (e.g., double spend tags and serial numbers). This, again, costs a small fee. As such, there are monetary costs to using such an identity scheme.

[13]With modifications, identity assertions can be validated as part of the consensus protocol, abrogating relying parties from validating credential issue and allowing the use of ephemeral supporting documentation.



(a) Times for operations measured in seconds. These operations do not include the time required to compute the accumulator.



(b) Library show proof sizes measured in bytes as a function of RSA modulus size.

Figure 4: Library performance as a function of parameter size.

8.3 Latency

A third consideration for the limited show credentials is the latency of inserting items into the block chain. Because completely meaningful proofs of work take time, some time must elapse in any such system. Namecoin and Bitcoin both aim to create blocks every 10 minutes. Thus, the naive wait time from a block is about 5 minutes. Propagation delays in the network and transaction volume, however, skew this distribution. While historical data for Namecoin is not available, for Bitcoin it takes slightly less than 9 minutes for a transaction to first be confirmed. In practice, it then takes multiple confirmations to solidify the transaction's place in the block chain. Variants of Bitcoin operate with faster confirmation times (e.g., Feathercoin, which aims to get a block every 2.5 minutes), though it is not clear this is entirely stable.

Given these latency constraints, our system, at least built on top of proof of work based bulletin boards, is not suitable for applications that require fast credential issue or quick detection of multi-spends across mutually distrusting parties.[14] A side effect of this is that double spend prevention mechanisms for fast transactions need to rely on detection and punishment (e.g., forfeiture of an escrowed value), not prevention.

8.4 Performance

We now examine the performance of our anonymous credential system. There are four underlying operations: minting a credential, verifying that the mint is correct, showing a credential, and verifying that show. Showing and verifying credentials also entail computing the accumulation of all or all but one of the current credentials. This process has time complexity $O(n)$ where n is the number of credentials. We measure the accumulation cost separately and run our other benchmarks with a precomputed witness and accumulator. We also give measurements for our performance with different security parameters. See Figure 4.

All experiments were conducted on a 2010 MacPro with 16GB of RAM and two 2.4GHz quad core Xeon E5620 processors running OSX 10.8.3. Experiments were measured in seconds via wall clock run time and were repeated for 500 iterations. Because of the speed of accumulating elements, we measure accumulator time in seconds per 100 accumulations.

The primary performance bottleneck for our library is the complexity of the proof of knowledge generated during the credential show. Because this double discrete logarithm proof uses cut-and-choose techniques, we need to perform between 80 and 128 iterations depending on the security parameter. This entails approximately 800-1000 exponentiations. Luckily, the same cryptographic requirements that force these iterations also mandate that such computations are independent and hence they can easily be parallelized. To

[14]Obviously, parties could cooperate and maintain a faster store of double spend tags, alleviating this problem.

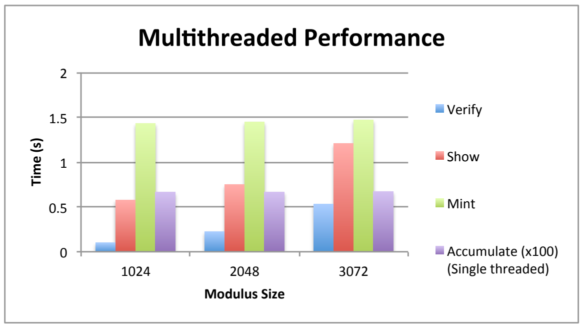


Figure 5: Multithreaded library performance as a function of parameter size.

exploit this, we make use of OpenMP to parallelize proof generation and verification. As shown in Figure 5, this offers significant performance benefits.

Unfortunately, OpenSSL, which we use for the computations underpinning our system, is not fully parallelizable due to the fact that its PRNG is synchronous. The resulting locks around RNG usage prevent us from effectively parallelizing portions of our code for showing a credential. It also causes problems when minting a credential. The resource intensive portion of credential mint is creating commitments and then testing if they are prime. This requires random numbers both directly for commitment generation and indirectly for primality testing which uses randomized Miller-Rabin. We believe further performance gains could be realized by using a parallelizable RNG (e.g., Intel's RDRand instruction).

9 Example Application: Distributed Direct Anonymous Attestation (dDAA)

In the original TPM 1.1b specification [1], attestations were signed by a TPM's Attestation Identity Key (AIK). The AIK's public key was itself signed using the TPM's baked in Endorsement Key (EK), creating an EK certification. Given an attestation, the public half of the AIK, the EK certificate, and the appropriate certificate chain from the manufacturer, it was possible to verify an attestation. However, the use of EKs creates a privacy problem, since these keys are globally unique and thus identify the user. The specification addressed this problem by introducing a privacy CA that prevented verifiers from learning the TPM's identity.

Direct Anonymous Attestation (DAA) [10], replaced the privacy CA with a cryptographically sound group signature scheme. Instead of signing attestations with an AIK, a TPM signs attestations with a private key for a group signature scheme that preserves the signer's anonymity. The TPM obtains the group signing key from a DAA authority in a manner that ensures the key is issued to a legitimate TPM and that it never leaves the TPM unencrypted. Unfortunately the integrity of this process depends fundamentally on the integrity of the software running in the DAA authority. This makes deploying a DAA instance somewhat problematic: each organization is responsible for deploying and securing this DAA authority, and any compromise of this server opens the door for credential theft or denial of service. Given the critical role envisioned for TPM attestations, this may inhibit the deployment of DAA systems.

Given a trusted execution environment (such as the environment provided by the TPM for DAA), we now show how to use our anonymous credential scheme as a replacement for DAA. To obtain a credential in the new scheme, the TPM runs the MintCred routine, securely store the resulting sk and transmitting the resulting credential up to the block chain along with a signature under the TPM's baked in Endorsement Key or a platform certificate. This signature authenticates the credential as having been generated by a valid TPM. Once the credential and signature are validated, they can be accumulated by verifiers. The TPM can later attest to a particular configuration by running a modified version of Show that ensures π_S is a signature of knowledge on the attestation values (i.e., the program configuration registers (PCRs) and an optional nonce). Running ShowVerify with the appropriate modifications for checking the signature of knowledge validates the attestation.

Although the existing TPM devices do not provide native support for our proposed scheme, we can emulate this support by implementing the cryptography in trusted software. In this design the TPM attests directly (i.e., non-anonymously) that some trusted code generated a specific credential and utilized the TPM's sealed storage functionality to ensure that only that code has access to the sk. This attestation, along with the credential, is placed in the block chain. After it is validated, the same trusted code can issue a signature of knowledge on the PCR values using Show. Verification is the same as with hardware generation.

We can achieve this trusted software execution using an Isolated Execution Environment. Available on recent AMD and Intel chips, Isolated Execution Environments allow code to execute without exposing itself or its data to the rest of the computer's software or hardware. Combined with the TPM's sealed storage and attestation functionality, code can execute in a secure environment, store data that only it can access, and attest to third parties not just that it did so but that the execution produced a specified output. Flicker [37] leverages this functionality to allow Pieces of Application Logic (PALs) to run and attest to their execution in one of these environments. Provided our code runs in run of these PALs, we get the same resilience to software attacks that DAA does. However, we are still vulnerable to hardware attacks while the PAL is executing (e.g., a cold boot attack [34]). We now describe the credential generation and show operations in greater detail assuming we operate in trusted software.

Obtaining a dDAA Credential. To obtain a credential, the user invokes the cred PAL, via Flicker, to generate a credential and stores sk_U securely. The user gets back only the public credential and an attestation signed by the TPM's AIK that the credential was generated by the PAL inside an isolated execution environment. She then assembles a `credential_blob` comprising: the credential output by the PAL, the signed attestation, the AIK, an Endorsement Key Certificate (EKC) for the AIK, and a certificate chain for validating the EKC. She places `credential_blob` in the block chain.

Any party can now validate the resulting information by running MintVerify on the resulting credential, verifying that the attestation is signed by the AIK, and finally validating the EK certificate and the rest of the certificate chain for the AIK. Provided these steps verify, the credential is accepted into the set $\mathbf{C}_O$.

Showing a dDAA Credential. Once the user has established the dDAA credential, she can attest to her state at any subsequent time. To do this she runs the cred PAL, and the PAL reads the TPM values and creates a signature of knowledge on them using Show. The resulting signature of knowledge can be sent to a verifier and validated using ShowVerify. Because every valid credential was accompanied by a direct attestation that it could only be accessed by the trusted PAL code, we know that any signature of knowledge using a valid credential came from the same code. Hence, assuming we trust the code, we know it is an accurate attestation to the system's configuration.

10 Related Work

Anonymous credentials. Introduced by Chaum [21] and developed in a line of subsequent works (e.g., [9, 13, 16]), anonymous credentials allow a user to prove that she has a credential issued by some organization, without revealing anything about herself other than that she has the credential. Under standard security definitions, even if the verifier and credential issuer collude, they cannot determine when the credential was issued, who it was issued to, or when it was or will be used. A common construction involves issuing a credential by obtaining a signature from an organization on a committed value (e.g., using the signature scheme of [14]) then proving in zero-knowledge that one has a signature under the organization's public key on that value. The contents of the commitment may be revealed outright or various properties can proved on the committed values (e.g., Alice can prove she is over 21 years old). Extensions to this work describe credentials that can only be shown anonymously a limited number of times [12] or delegated to others [4]. All of these schemes require issuing organizations to maintain a secret key.

Bitcoin and append-only ledgers. Our construction relies on the existence of a *distributed append-only transaction ledger*, a technology that makes up the core component of the Bitcoin distributed currency: the log of all currency transactions called the block chain [39]. These ledgers are maintained by an ad hoc group of network nodes who are free to enter and leave the network (there is no key provisioning necessary for them to join). A typical transaction ledger consists of a sequence of blocks of data that are widely replicated among the participating nodes, with each block connected to the previous block using a hash chain. Nodes compete for the opportunity to add new blocks of transactions to the ledger by producing a partial hash collision over the new data and the hash of the last block in the chain. The hash collision serves two purposes: first, it is a computationally-difficult-to-forge authenticator of the ledger and second, since finding a partial hash collision involves substantial computational effort, the peer who finds it is chosen "at random" with a probability proportional to the rate at which he can compute such partial collisions. As a result, an ad hoc group of mutually distrusting and potentially dishonest peers can correctly manage such a ledger provided that a majority of their computational power is held by honest parties. Recent experience with Bitcoin and Namecoin provides evidence that this assumption holds in practice.

Namecoin. Namecoin [40] is a decentralized identity system that uses the same block chain technology as Bitcoin. Although Namecoin is in part a currency system — its internal currency, the namecoin (NMC), is used to purchase a name in the same way one pays for a domain name — its primary purpose is to associate names with arbitrary data. A user can claim a name provided (1) they pay the price in NMC for it and (2) it is unclaimed. At that point, an entry is inserted into the block chain mapping the name to a public key and some arbitrary data. The public key allows the owner to update the data by signing a new record. The data allows for various uses. If it is an IP address, then one has a distributed DNS system (such a system, .bit, is already deployed). On the other hand, if it is a public key, the result is a basic PKI. The first-come first-served nature of Namecoin seems somewhat anachronistic, however it replicates in miniature the way normal DNS names are generally assigned, where the first person to claim the name gets it. Similarly, standard (non-extended validation) SSL certificates for a domain are typically issued to anyone who can demonstrate control of a domain (usually via an email to admin@domain).

11 Conclusion

In this work we constructed a distributed anonymous credential system and several extensions. Our constructions are secure in the random oracle model under standard cryptographic assumptions provided there exists a trustworthy global append-only ledger. To realize such a ledger we propose using the block chain system already in real world use with the distributed cryptographic currency Bitcoin. Although we are limited in the class of identity assertions we can certify, we argue that several basic assertions are of particular use in peer-to-peer systems, as they can be used to mitigate Sybil attacks, ensure fair resource usage, and protect users' anonymity while verifying their computer's correctness.

Future work. We leave two open problems for future work. First, the proofs in this work assumed the security of a transaction ledger. We leave a precise formal model of the ledger, which attacks are allowable, and what bounds may be placed on their consequence as an open problem. Second, the efficiency of our construction can be improved. Although all of our algorithms are efficient (in that they do not scale with the size of the ledger), the need for double-discrete logarithm proofs leads to somewhat large proof sizes when showing a credential (roughly 50KB for modest parameters). Our construction may be optimized for certain applications that do not require the full flexibility of our construction. For example, schemes not requiring selective disclosure of credentials require about half that proof size. At the same time, we hope that advances in bilinear accumulators, mercurial commitments, or lattice based techniques may provide a more efficient construction. We are particularly hopeful that generic work in verifiable computation [5, 44] will offer drastically smaller proof sizes without resorting to bespoke proofs and protocols.

References

[1] TPM Main Specification v1.1.

[2] TPM Main Specification v1.2.

[3] N. Barić and B. Pfitzmann. Collision-free accumulators and fail-stop signature schemes without trees. In *EUROCRYPT*, 1997.

[4] Mira Belenkiy, Melissa Chase, Markulf Kohlweiss, and Anna Lysyanskaya. P-signatures and noninteractive anonymous credentials. In *Theory of Cryptography*. 2008.

[5] Eli Ben-Sasson, Alessandro Chiesa, Daniel Genkin, Eran Tromer, and Madars Virza. Snarks for c: Verifying program executions succinctly and in zero knowledge.

[6] Josh Benaloh and Michael de Mare. One-way accumulators: a decentralized alternative to digital signatures. In *EUROCRYPT*, 1994.

[7] Fabrice Boudot. Efficient proofs that a committed number lies in an interval. In *Advances in CryptologyEUROCRYPT 2000*, pages 431–444. Springer, 2000.

[8] Stefan Brands. Rapid demonstration of linear relations connected by boolean operators. In *EUROCRYPT*, 1997.

[9] Stefan A. Brands. *Rethinking Public Key Infrastructures and Digital Certificates: Building in Privacy*. MIT Press, Cambridge, MA, USA, 2000.

[10] Ernie Brickell, Jan Camenisch, and Liqun Chen. Direct anonymous attestation. In *CCS '04*.

[11] Jan Camenisch, Rafik Chaabouni, et al. Efficient protocols for set membership and range proofs. In *Advances in Cryptology-ASIACRYPT 2008*, pages 234–252. Springer, 2008.

[12] Jan Camenisch, Susan Hohenberger, Markulf Kohlweiss, Anna Lysyanskaya, and Mira Meyerovich. How to win the clonewars: efficient periodic n-times anonymous authentication. CCS, 2006.

[13] Jan Camenisch and Anna Lysyanskaya. An efficient system for non-transferable anonymous credentials with optional anonymity revocation. EUROCRYPT, 2001.

[14] Jan Camenisch and Anna Lysyanskaya. Dynamic accumulators and application to efficient revocation of anonymous credentials. In *CRYPTO*, 2002.

[15] Jan Camenisch and Anna Lysyanskaya. Dynamic accumulators and application to efficient revocation of anonymous credentials. In *CRYPTO*, 2002. Extended Abstract.

[16] Jan Camenisch and Anna Lysyanskaya. A signature scheme with efficient protocols. SCN'02, 2003.

[17] Jan Camenisch and Markus Michels. Proving in zero-knowledge that a number n is the product of two safe primes. In *EUROCRYPT*, 1999.

[18] Jan Camenisch and M. Stadler. Efficient group signature schemes for large groups. In *CRYPTO*, 1997.

[19] Jan Leonhard Camenisch. *Group Signature Schemes and Payment Systems Based on the Discrete Logarithm Problem*. PhD thesis, ETH Zürich, 1998.

[20] Melissa Chase and Anna Lysyanskaya. On signatures of knowledge. In *CRYPTO*, volume 4117 of LNCS, pages 78–96, 2006.

[21] David Chaum. Security without identification: transaction systems to make big brother obsolete. *Communications of the ACM*, 1985.

[22] S. Coull, M. Green, and S. Hohenberger. Access controls for oblivious and anonymous systems. In *TISSEC*, 2011.

[23] Ronald Cramer, Ivan Damgård, and Berry Schoenmakers. Proofs of partial knowledge and simplified design of witness hiding protocols. In *CRYPTO*, 1994.

[24] Ivan Damgård, Kasper Dupont, and Michael Østergaard Pedersen. Unclonable group identification. EUROCRYPT, 2006.

[25] W. Diffie and M.E. Hellman. New directions in cryptography. *IEEE Transactions on Information Theory*, 1976.

[26] Y. Dodis and A. Yampolskiy. A verifiable random function with short proofs and keys. PKC, 2005.

[27] Dot-bit. Available at `http://dot-bit.org/`.

[28] John R. Douceur. The sybil attack. In *Peer-to-Peer Systems*. 2002.

[29] Amos Fiat and Adi Shamir. How to prove yourself: Practical solutions to identification and signature problems. In *CRYPTO*, 1986.

[30] E. Fujisaki and T. Okamoto. Statistical zero knowledge protocols to prove modular polynomial relations. In *CRYPTO*, 1997.

[31] O. Goldreich, S. Goldwasser, and S. Micali. How to construct random functions. *Journal of the ACM*, 1986.

[32] O. Goldreich, S. Micali, and A. Wigderson. Proofs that yield nothing but their validity and a methodology of cryptographic protocol design. In *FOCS*, 1986.

[33] Jens Groth. Non-interactive zero-knowledge arguments for voting. In *Applied Cryptography and Network Security*, pages 467–482. Springer, 2005.

[34] J Alex Halderman, Seth D Schoen, Nadia Heninger, William Clarkson, William Paul, Joseph A Calandrino, Ariel J Feldman, Jacob Appelbaum, and Edward W Felten. Lest we remember: cold-boot attacks on encryption keys. *Communications of the ACM*, 52(5):91–98, 2009.

[35] Michael Z Lee, Alan M Dunn, Brent Waters, Emmett Witchel, and Jonathan Katz. Anon-pass: Practical anonymous subscriptions. In *IEEE Security and Privacy*, 2013.

[36] Helger Lipmaa. On diophantine complexity and statistical zero-knowledge arguments. In *Advances in Cryptology-ASIACRYPT 2003*, pages 398–415. Springer, 2003.

[37] Jonathan M McCune, Bryan J Parno, Adrian Perrig, Michael K Reiter, and Hiroshi Isozaki. Flicker: An execution infrastructure for tcb minimization. In *ACM SIGOPS Operating Systems Review*, volume 42, pages 315–328. ACM, 2008.

[38] I. Miers, C. Garman, M. Green, and A. Rubin. Zerocoin: Anonymous distributed e-cash from bitcoin. In *IEEE Security and Privacy*, 2013.

[39] Satoshi Nakamoto. Bitcoin: A peer-to-peer electronic cash system. 2008.

[40] Namecoin. Available at `http://namecoin.info/`.

[41] Moni Naor and Omer Reingold. Number-theoretic constructions of efficient pseudo-random functions. *Journal of the ACM (JACM)*, 2004.

[42] Tsuen-Wan Johnny Ngan, Dan S. Wallach, and Peter Druschel. Enforcing fair sharing of peer-to-peer resources. In *Peer-to-Peer Systems II*. 2003.

[43] Daniel Obenshain, Tom Tantillo, Andrew Newell, Cristina Nita-Rotaru, and Yair Amir. Intrusion-tolerant cloud monitoring and control. In *LADIS*. 2012.

[44] Bryan Parno and Mariana Raykova. Pinocchio: Nearly practical verifiable computation.

[45] T. Pedersen. Non-interactive and information-theoretic secure verifiable secret sharing. In *CRYPTO*, 1991.

[46] Ronald L Rivest, Adi Shamir, and Yael Tauman. How to leak a secret. In *ASIACRYPT*. 2001.

[47] Tomas Sander and Amnon Ta-Shma. Auditable, anonymous electronic cash (extended abstract). In *CRYPTO*, 1999.

[48] Claus-Peter Schnorr. Efficient signature generation for smart cards. *Journal of Cryptology*, 4(3):239–252, 1991.

[49] Hovav Shacham and Brent Waters. Compact proofs of retrievability. In *Advances in Cryptology-ASIACRYPT 2008*, pages 90–107. Springer, 2008.

A Proof Sketch of Security for Our Basic System

We now provide a sketch of the proof of security for our basic distributed anonymous credentials system.

Our basic approach is to show that for every real-world adversary $\mathcal{A}$ against the credential system, we can construct an ideal-world adversary $\mathcal{S}$ against the ideal-world system such that the transcript of $\mathcal{A}$ interacting with the real system is computationally indistinguishable from the transcript produced by $\mathcal{A}$ interacting with $\mathcal{S}$. We assume a static corruption model in which the adversary controls some set of users and leave a proof in the adaptive corruption model for future work. For this sketch we also assume that our zero-knowledge signatures of knowledge include an efficient extractor and simulator and that the *params* are created using a trusted setup process. Note that in the random oracle model this assumption holds for the Fiat-Shamir proofs we employ, provided we conduct the proof sequentially.

Our proof assumes the existence of a global, trusted transaction ledger, which we use as a black box. We leave a complete proof that considers this construction and models it to future work.

We begin by sketching the simulator $\mathcal{S}$ for our system.

A.1 Description of the Simulator

Minting a credential. When a user controlled by the adversary with nym Nym_U^O wants a credential, the user first generates $(c, \pi_M, attrs)$. When the simulator receives notification of this, it first verifies that the credential and proof are valid and meet the organization's policy. If so it employs the knowledge extractor for the signature of knowledge on π_M to get (sk, aux).

The simulator then checks if it has a record of (U, sk, Nym_U^O) on its list of users. If the user with key sk and nym Nym_U^O exists, then $\mathcal{S}$ retrieves sk_U associated with (U, sk, Nym_U^O) and proceeds. If it is not on the list, the simulator checks if it has previously seen a user with key sk. If the user with key sk is not present, then the simulator creates a user U and runs $RegNym(Nym_U^O, U, O)$ to register Nym_U^O and obtain sk_U for further interactions with T_P. $\mathcal{S}$ then stores (U, sk, sk_U, Nym_U^O) in its list of users controlled by the adversary. If a user U with key sk exists, then it runs $RegNym(Nym_U^O, U, O)$ to register Nym_U^O and adds Nym_U^O to U's record.

Once the simulator has registered the nym or verified it already exists, it runs $MintCred(Nym_U^O, O, attrs, aux)$. The simulator then transmits the credential information to the trusted store and acknowledges the credential's issuance. $\mathcal{S}$ stores $(sk, Nym_U^O, attrs, aux, c, \pi_M)$ in its list of granted credentials.

When an honest user, through T_P, wants to establish a credential, the simulator creates a credential c (using the publicly available $attrs$) and uses the simulator for the signature of knowledge π_M to simulate the associated proof. It then transmits the credential information $(c, \pi_M, attrs)$ to the trusted store.

Showing a credential. When a user controlled by the adversary wants to show a credential from organization O to verifier V with which it has nyms Nym_U^O and Nym_U^V respectively, the user first generates π_S. When the simulator receives notification of this, it verifies the proof as in the real protocol (rejecting if it is invalid). If the show verifies, it runs the knowledge extractor for the proof of knowledge on π_S to get sk.

The simulator then checks if it has a record of $(U, sk, Nym_U^O, Nym_U^V)$ on its list of users. If the user with key sk and nyms Nym_U^O and Nym_U^V exists, then $\mathcal{S}$ retrieves sk_U associated with (U, sk, Nym_U^O) and proceeds. If the record does not exist, either in part or in full, the simulator checks if it has previously seen a user with key sk. If the user with key sk is not present, then the simulator creates a user U and runs $RegNym(Nym_U^O, U, O)$ and $RegNym(Nym_U^V, U, V)$ to register Nym_U^O and Nym_U^V and obtain sk_U for further interactions with T_P. $\mathcal{S}$ then stores $(U, sk, sk_U, Nym_U^O, Nym_U^V)$ in its list of users controlled by the adversary. If a user U with key sk exists, then it runs $RegNym(Nym_U^O, U, O)$ (resp. $RegNym(Nym_U^V, U, V)$) to register Nym_U^O (resp. Nym_U^V) and adds Nym_U^O (resp. Nym_U^V) to U's record.

Now, the simulator $\mathcal{S}$ runs $ShowOnNym(Nym_U^O, Nym_U^V, O, V, \mathbf{C})$ where $\mathbf{C}$ is obtained by the simulator through a call to $GetCredList(O)$.

When an honest user (through T_P) wants to show a credential to a verifier V controlled by the adversary, the simulator runs the zero-knowledge simulator for π_S to simulate a proof that it then sends to V.

A.1.1 Proof (sketch) of a Successful Simulation

Our simulation is computationally indistinguishable from the real protocol if the Strong RSA and the Discrete Logarithm assumptions hold. While we leave a full proof to the final version, we do provide an overview of the argument for security.

We first begin by discussing the signatures/proofs π_M and π_S. Under the Discrete Logarithm assumption, π_M is a computational zero-knowledge signature of knowledge on aux of the values sk, r, and r' such that the nym Nym_U^O and the credential c both belong to the same master secret sk. The proof is clearly zero-knowledge because it can be constructed using standard techniques [48]. One can see that the only way to forge this proof would be to cause a collision on the commitments, which occurs with negligible probability under the Discrete Logarithm assumption [45]. Under the Strong RSA and Discrete Logarithm assumptions, π_S is a statistical non-interactive zero-knowledge proof of knowledge of the values sk, ω, c, Nym_U^V, r, and r' such that ω is a witness that c is in the accumulator A and nym Nym_U^V and the credential c both belong to the same master secret sk. We can again see that this proof can be constructed using known techniques [14, 48] similar to the proofs used by Miers et al. in [38]. In order to forge such a proof, the adversary would need to either find a collision on the commitments or forge an accumulator membership witness. We previously discussed how the first case occurs with negligible probability. The second case occurs with negligible probability under the Strong RSA assumption due to [14]. See the final version of the paper for a formal treatment/reduction of these statements.

Intuitively, we can now see that the simulator will fail with at most negligible probability because it deals solely with zero-knowledge signatures of knowledge and zero-knowledge proofs of knowledge, which have efficient extractors and simulators. Our proofs π_M and π_S have knowledge extractors that succeed with probability $1 - \nu(\lambda)$ for some negligible function $\nu(\cdot)$. Since signatures and proofs are the sole point of failure for our simulator described above, it fails with negligible probability. Because the only thing the adversary sees is the zero-knowledge proofs and signatures, the simulated signatures and proofs are computationally indistinguishable from legitimate ones, and the adversary cannot cause our simulator to fail except with negligible probability, the adversary cannot distinguish between an interaction with the simulator and the real protocol.