NuID: A Model for Trustless, Decentralized Authentication and Self-Sovereign Identity

info@nuid.io

Table of Contents

1. Introduction and Motivation

1.0 What is NuID?

NuID is a trustless and decentralized authentication service that facilitates participation in a decentralized identity model. The service uses zero knowledge proofs (ZKPs) and distributed ledger technology (DLT) to enable applications and web services to authenticate users without ever having to see, store, or therefore be liable for securing users' authentication data, thereby eliminating the risk of mass credential breaches. Users effectively own their own credentials and can prove ownership of them without ever having to entrust them to another party, including NuID. In addition to strengthening traditional authentication flows and reducing service vulnerabilities, NuID unlocks a wide range of new approaches for single sign-on (SSO) and self-sovereign identity (SSI). The decentralized nature of the architecture eliminates traditional siloed databases of credentials, creating an inherently portable and user-owned identity platform on which users can secure and choose to share other forms of data as well.

On the front-end, NuID looks like any other login experience. Just like the traditional "shared secret" model, NuID takes standard username and password credentials as inputs and, if authenticated, provides the user access to the desired privileged resources. It is when the user pushes 'login' that NuID departs from traditional authentication approaches. The NuID service uses lightweight client libraries to convert the user credentials into public ZKP parameters. These public parameters are sent via POST request to NuID's API and are then appended to the Ethereum distributed ledger by NuID. During authentication, the API pulls the user's public ZKP parameters and executes the ZKP protocol with the client device. Finally, the NuID API returns either a success or failure response to the service, indicating if the user has provided the correct credential (a more detailed description of the protocol is provided in Section 2). It is important to note that during NuID's authentication flow, user credentials are never stored or even transmitted from the user's device. The information that is stored on the distributed ledger can be publicly viewed, similar to a public key in traditional public key infrastructure (PKI). Furthermore, the

NuID service does not rely on any aspect inherent to the device (e.g. device ID), and therefore is also portable between devices (not just services). In this way, NuID sets the basis for a truly abstracted identity layer, pulling user authentication out of both the device and web service silos.

Flexibility is central to NuID's architecture. Beyond the flexibilities inherent to the decentralized authentication approach (device, service), NuID is designed to be maximally flexible with respect to both credential types and storage. The core authentication engine is interoperable with any reproducible authentication secret (password, PIN, private key, software or hardware token, etc., each optionally protected under device-local biometrics). On the other side, the storage mechanism is also flexible, with the NuID service fully agnostic to the storage approach (public ledger, private ledger, local database, etc.). The NuID service is also interoperable and complementary to authentication and identity standards such as OpenID Connect (OIDC), OAuth, Decentralized Identifiers (DID), and SAML.

NuID is currently deployed in production environments and has been cryptographically verified by Professor Jonathan Katz et al. at George Mason University.[1] More technical detail on how NuID works and how you can integrate it into your web service or identity model is provided in Section 2 and Section 3.

1.1 What Problems Does NuID Solve?

At NuID, we believe that the future of the internet belongs to the user, and the NuID decentralized identity architecture was designed to support that future. The current state of authentication and digital identity fundamentally limit the ability of users to control their own identity and manage their own risk. NuID strives to set the foundation for an alternative identity architecture which releases web services from these limitations and creates a framework to return control of data and identity to the user.

[1] Security Analysis of the NuID Decentralized Identification Protocol

1.1.1 Eliminate the Risk of Mass Credential Breaches

By far the most common authentication approach in production today involves the use of "shared secrets" between the user (client) and the application (server). In this model, the client sends the user's plaintext password to the server, where it is stored for future comparison. In practice there are many ways to strengthen this process—for example, using encrypted channels like SSL, salting and hashing passwords at the server before storing, employing two-factor authentication, etc. While useful, these are ultimately bandages to an inherently flawed process. By sharing a secret, the user loses control. He or she must trust the service to safely manage and store his or her credentials. As has been demonstrated time and again, this trust is not well-founded, as holding user secrets in a centralized fashion is inherently risky. Breaches and the mass compromise of user credentials have become commonplace in recent years.[2] This is not a risk reserved for the techno-illiterate, as most of the technological leaders have also encountered major incidents of credential breaches in the recent past.[3]

These breaches are bad for users and businesses alike. Businesses are often encumbered with large fines[4] and the loss of trust from their customer base.[5] Users face ongoing risk as their personally identifiable information (PII) is exposed to the public. Not only can hackers and criminals conduct fraudulent activity at the breached company, they can use this information to establish new vectors of attack into completely unrelated services.

This exposes the critical issue: in the shared secret model users are forced to trust services which are inherently vulnerable, and they do so at great personal risk. NuID solves the source of this problem by eliminating the need to share any secrets whatsoever, and therefore put an end to the requirement of trust (hence "trustless"). The decentralized storage of public parameters plays an important role in this objective by protecting the integrity and access of the public parameters without the reliance on any one source.

[2] ITRC 2019 End of Year Data Breach Report
[3] LinkedIn, Facebook, Google, Twitter
[4] IBM Cost of a Data Breach Report
[5] Ping Identity 2019 Consumer Survey

1.1.2 Remove Credentials and Identities from Service Provider Silos

The web as it exists today is riddled with identity silos. Every time a user signs up for a web service he or she instantiates a duplicative copy of his or her identity. At a minimum, this identity consists of a username (or email), password, and the user's activity on the relevant web service. Depending on the type of service, this identity may also include a range of additional personal information. At the moment the user registers, this information is transferred to the servers of the service provider, where it is stored under their full control. This single central store presents security risks, the prevailing of which have already been mentioned. However, the user's digital existence is not confined to a single central service, but instead quite the opposite. The average individual's digital life typically consists of a wide range of online services. Over time, this leads to the existence of numerous, parallel identities across the user's many web services. This identity fragmentation adds significant points of user experience friction, such as the necessity to remember multiple unique passwords or the difficulty of ensuring that changes in personal information are propagated to every service.

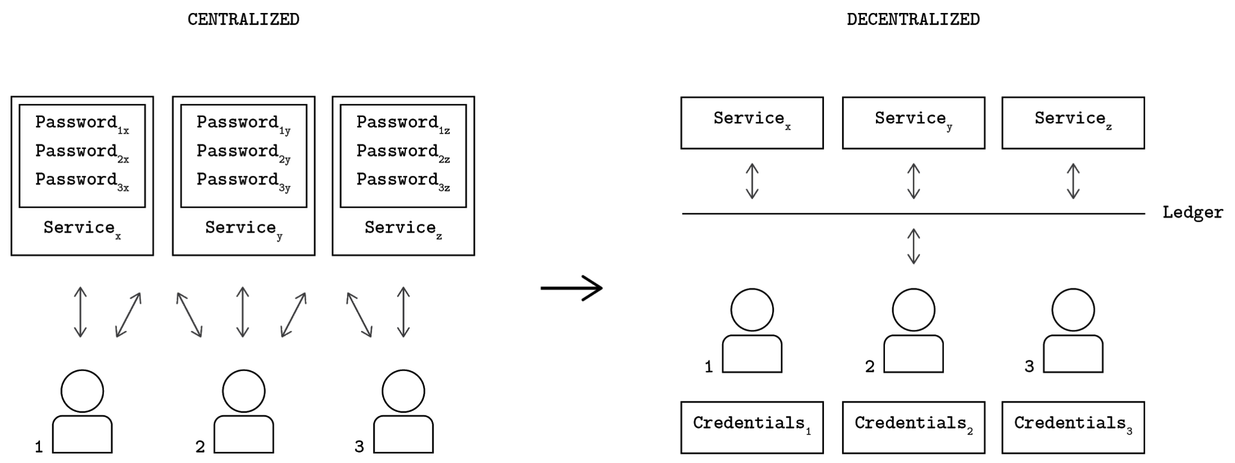

Figure 1: *A decentralized identity removes the need for service-specific siloes.*

Federated identity solutions have been introduced in an effort to help consolidate this fragmented user experience. While effective at this objective in some cases, these services, unfortunately, come at the cost of even further centralization of control and perpetuate the requirement of trust. The third-party identity provider, typically a large social media service, becomes a single point of authority (and failure) for users' identities across many services. The user has some control over how his or her information is shared, but the power and data still sit in the walled garden of the federated identity provider. In this way, the federated identity providers help to compress the silos, but persist the shared secret, trust-based model. This trust has been increasingly strained as of late as users have become more aware of the breadth of activities conducted with their information by these social media providers behind the scenes.[6]

The distributed nature of NuID helps to eliminate these silos and create a single source of truth for user identity. With public ZKP parameters stored on a public ledger, NuID credentials are accessible to anyone, and therefore exist outside of the traditional web service silos. Critically, in this architecture the user becomes the central administrator of his or her credentials, keeping all secrets locally on his or her devices—or in their mind—and holding the sole power to amend the central source of truth. NuID's role in this interchange is that of the blind facilitator. NuID never sees or holds the user secret and does not maintain any privileged access to the storage location. Furthermore, every interaction the NuID service facilitates—credential registration, storage, and verification—can be publicly validated against the backing ledger network. It is critical to the decentralization of the architecture that an organization may eject the NuID dependency from their technology stack with uninterrupted support for new and previously registered identities.

1.1.3 Abstract Digital Identity From the Device

The existence and proliferation of service silos has been a feature of the internet since its inception. A newer phenomenon, coinciding with the proliferation of mobile smart devices, is the increasing importance of a user's device in establishing his or her digital identity. In many instances, the device parameters have become essential to the function of public key infrastructure (PKI). Protocols such as leading cryptocurrencies, passwordless authentication (e.g., WebAuthn approaches), and cellular network authentication are founded on either native device IDs or private keys stored on a device. By using the device as a proxy for user identity these services

[6] 2018 Edelman Trust Barometer Global Report

have introduced a rigidity in the authentication process and potentially excluded users or applications without the necessary device access.

Unlike these solely device-based identity approaches, the zero knowledge protocol leveraged by NuID can be used to authenticate any repeatedly producible secret value. This flexibility allows NuID to support both “something you know” (e.g. passwords) and “something you have” (e.g. device-based private keys) authentication factors. As with other device-based authentication technologies, a private key used as a NuID factor can be additionally protected by device-local biometrics, or “something you are.” The result is a digital identity that is authenticated by a composable set of credentials stored either in the user’s mind, on their devices, or both, depending on the needs and constraints of the authentication context. *Figure 2* illustrates the flexible and independent nature of the NuID protocol, and how NuID can help to set the foundation for a portable, persistent, and user-controlled digital identity layer.

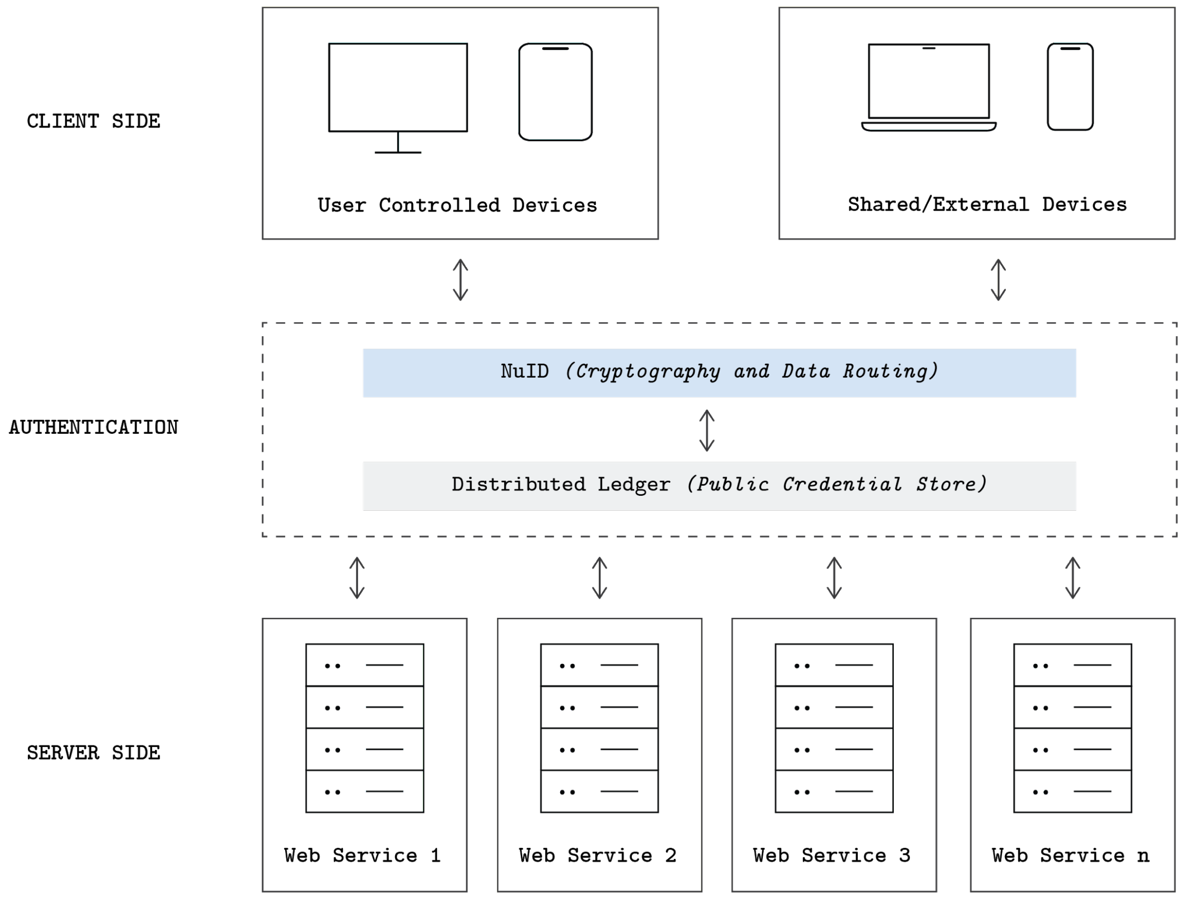

Figure 2: *NuID introduces a distinct and independent layer for authentication.*

1.2 How Can NuID be Used?

There are essentially two broad use-cases for the NuID protocol and service. The first of which is a traditional authentication use-case. In this application NuID essentially replaces a service's traditional authentication flow. The benefits of this have been enumerated already, but essentially distill down to reduced enterprise risk, increased credential portability, and control of credentials returned to the user. By not being responsible for holding and protecting user credentials, services can materially decrease their risk exposure. Users benefit as they do not need to trust any third parties, gaining full control of their own credentials. Further, as credential silos are eliminated, the risks associated with breaches of these silos are mitigated. Businesses can leverage NuID to accomplish this simply by incorporating the relevant client libraries into their service and connecting them to the NuID REST API (API access is available at https://portal.nuid.io). It is worth noting that NuID is currently operating in this capacity in enterprise production environments.

The second, and most impactful, use-case for NuID is as a foundation to self-sovereign digital identity models. Authentication is a fundamental component of any digital identity model, serving as the critical link between individuals and their digital identities. *Figure 3* below shows the general components of digital identity, and the important role played by authentication.

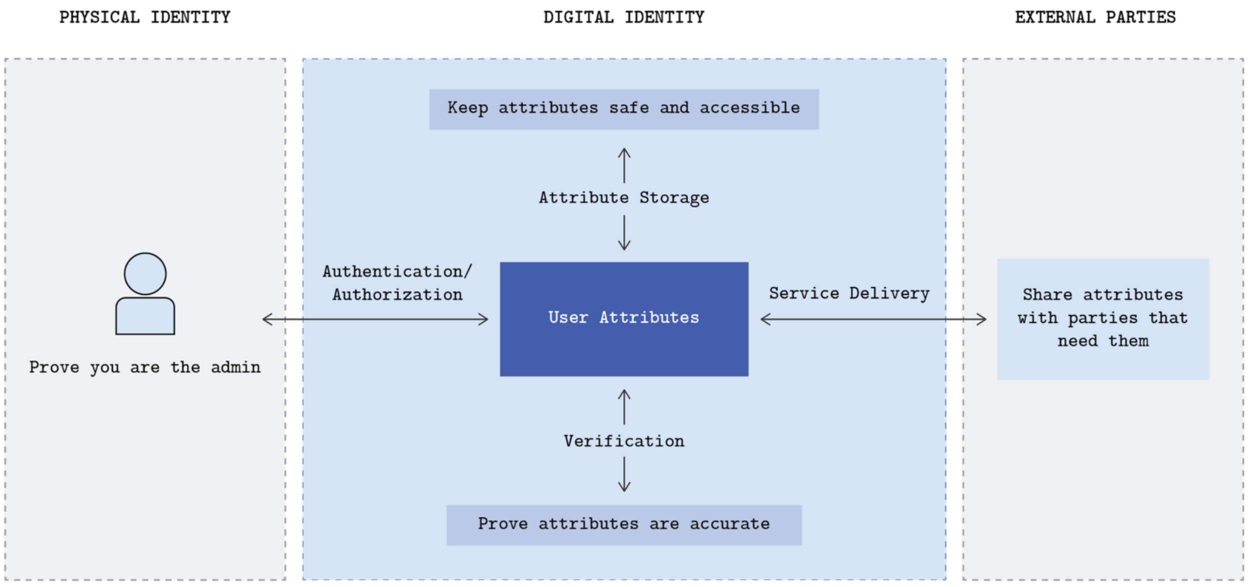

Figure 3: *Authentication links the user to the digital world.*

As previously mentioned, existing authentication protocols either silo the user identity inside a specific service, device, or both. These limitations to the current authentication protocols fundamentally limit the ability of self-sovereign identity to reach its idealistic potential. For a user to control their identity they must not rely on any 3rd party gatekeeper, and they should be able to access the resources they require in any situation. Only with a decentralized authentication architecture like the one described herein can the user truly serve as the gatekeeper to their own digital identity.

By lifting the user's credentials into this distinct and independent layer, the NuID protocol unlocks a host of potential identity applications. At the most basic level, this creates the potential for a single sign on (SSO) service which does not require a central trusted entity to hold these critical credentials. Beyond basic login applications, a decentralized authentication protocol sets the stage for full-fledged decentralized identity models, which in turn can enable applications previously not possible. Developers and service providers will eventually be able to use NuID as an anchor to secure, store, and access relevant user attributes. These attributes will be stored in distributed or centralized databases, and only be accessed by a decentralized authentication process. This end-to-end approach describes an identity model in which attributes are secured by one's credentials and access to them is in the exclusive control of the user. Attributes are of course a general term referring to any potential information about an individual. The generality of such a solution creates a wide range of potential applications—including a basic data locker with verifiable attributes (e.g., age, state of residence, etc.), financial institution "know-your-customer" (KYC) verification information, portable electronic medical records (EMR), digital asset ownership, and many more. The generalized applicability of such a model is the motivation behind its development and the motivation behind this paper.

The NuID solution was developed with the objective of supporting self-sovereign identity, and it is therefore fully complementary and interoperable with key protocols and standards such as OpenID Connect (OIDC), OAuth, SAML, and Decentralized Identifiers (DIDs). These protocols do not set requirements on how the user is authenticated, but rather provide frameworks for authorizing and exchanging information between disparate web services after the user has already been authenticated by one of the services. For example, OIDC requires user authentication prior to issuing an identity assertion to a relying party, and it does not specify a particular authentication mechanism. Therefore, an OIDC provider could use decentralized authentication as presented herein to perform user authentication prior to issuing an OIDC identity assertion. Doing so would eliminate the need for such an OIDC provider to maintain and protect a password storage back-end, which is typically the case for current OIDC user authentication. While OIDC is used as a specific example, this same complementary relationship applies to all of the aforementioned protocols.

2. How NuID Works

2.0 Architecture Overview

The NuID service consists of four main components: (1) a zero knowledge proof (ZKP) authentication protocol, (2) a distributed ledger storage layer, (3) open-source client libraries, and (4) the NuID API and service infrastructure. Collectively these four components enable the trustless authentication and decentralized identity model described herein.

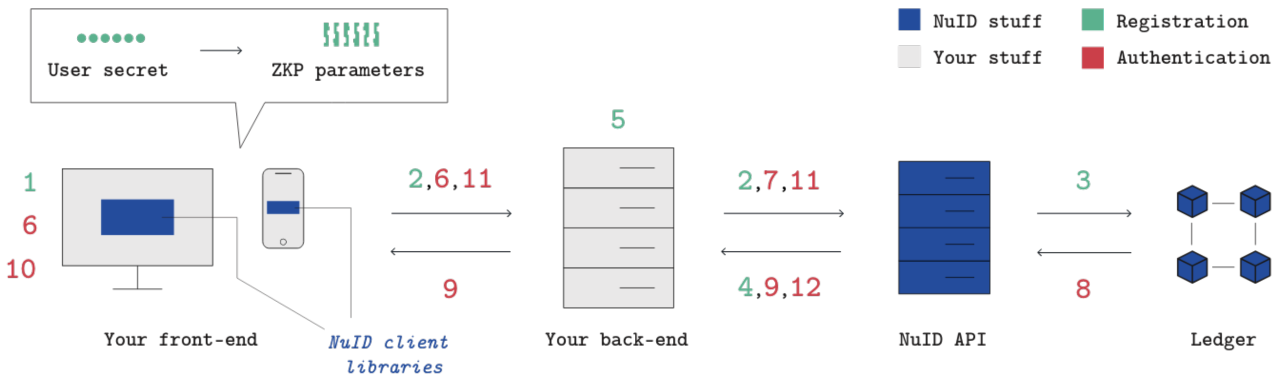

Figure 4: *NuID registration and authentication flow.*

Registration

1. User inputs a username/email and a new secret (e.g. a password), and the NuID client libraries generate ZKP parameters from the secret.
2. Username and ZKP parameters are sent to your back-end; parameters are forwarded to the NuID API.
3. ZKP parameters posted to the ledger.
4. NuID API returns a unique, persistent identifier for the new credential.
5. Username is associated with the persistent identifier; registration is complete.

Authentication

6. User inputs their username and secret and clicks ‘Login’; username is sent to your back-end.
7. Your back-end sends the associated identifier for that username to the NuID API to request a cryptographic challenge.
8. NuID API retrieves the ZKP parameters from the ledger.
9. NuID API returns a unique, one-time cryptographic challenge derived from the user’s ZKP parameters, which is forwarded to the client.
10. NuID client libraries use the secret and the challenge to generate a one-time ZKP of the secret.
11. ZKP is forwarded to the NuID API for verification.
12. NuID API returns the outcome of the verification (success/failure) to your back-end.

2.1 Authentication Protocol

2.1.1 Zero Knowledge Authentication in the Abstract

In general, a zero knowledge proof allows one party to prove they know a secret value to another party without revealing anything about the secret itself. This property of ZKPs creates the knowledge asymmetry necessary in constructing a trustless authentication service. Mechanically similar to PKI, knowledge asymmetry is what allows non-sensitive authentication parameters to be posted publicly, against which any verifier (e.g. any authenticating service, or provider such as NuID) may validate that the correct underlying authentication secret has been produced by the user or client agent without learning anything else about the secret or its nature.

Because the verifier learns nothing of the underlying secret, that secret may take any form, allowing the user to choose any deterministic authentication secret to authenticate their identity. A password, PIN, private key, software or hardware token, etc., are all indistinguishable to the verifying party in a zero knowledge authentication protocol. On devices that support biometrics, a client application may require device-local biometric authentication prior to releasing a securely stored deterministic value as input to the proof generation algorithm.

2.1.2 Zero Knowledge Authentication at NuID

NuID's zero knowledge proof authentication protocol is based on the Schnorr identification scheme[7] and has been subject to rigorous third-party cryptographic evaluation.[8] It is worth noting that Schnorr's identification scheme relies on the discrete log (DLOG) assumption and is therefore not quantum-safe. Section 2.1.3 illustrates why NuID's authentication service is not operationally bound to a single instance of a zero knowledge protocol, detailing the system's extensibility to support multiple zero knowledge authentication protocols side-by-side, including future quantum-safe approaches.

[7] Schnorr scheme: https://tools.ietf.org/html/rfc8235. NuID's protocol differs slightly from the Schnorr identification scheme, which does not use any nonce.

[8] Security Analysis of the NuID Decentralized Identification Protocol.

NuID's ZKP protocol consists of the algorithms KeyGen, P (proof generation), and V (proof verification), and uses a cyclic group G of prime order q with generator g, as defined in the secp256k1 elliptic curve specification.[9] In the notation below, H is a cryptographic hash function; NuID currently uses scrypt.[10]

The protocol steps for registration and authentication are detailed below. "Registration" is when a new secret is created and associated with a user, and "authentication" is any subsequent verification of that secret when a user attempts to login. We omit here the mapping of usernames to the storage location of public credentials to more clearly understand how the ZKP authentication process works.

<u>Registration</u>

1. The user generates a secret and enters it into the client.[11]

   ilovemydog! $=$ *secret*

2. The client uses *secret* as the input to the KeyGen algorithm and outputs the value *Pub*.

   KeyGen (*secret*) computes $x = \mathtt{H}(secret)$ and outputs $Pub = g^x$

3. The client sends *Pub* along with credential metadata (see Section 2.1.3 below) to the service back-end, which forwards them to the NuID API.[12]

4. The NuID API appends *Pub* and the credential metadata to the Ethereum ledger.

[9] https://www.secg.org/sec2-v2.pdf

[10] https://tools.ietf.org/html/rfc7914

[11] Currently, NuID's client libraries only support a user-generated password as the secret; however, future versions will allow a device-based software token to be registered as the secret, in addition to other authentication factors.

[12] *Pub* and the credential metadata are collectively referred to as "ZKP parameters" in *Figure 4*.

Authentication

1. The user inputs the same secret into the client.

   ilovemydog! $= \textit{secret}$

2. The client uses Pub, $secret$, and a nonce `nonce` as inputs to the proof generation algorithm P and outputs the proof $(c,\ s)$. The `nonce` is issued by the NuID API and is unique to each authentication attempt.[13]

   P first computes $x = \mathtt{H}(secret)$
   P then chooses uniform $r \leftarrow \mathbb{Z}_q$ and sets $A = g^r$
   P then computes $c = \mathtt{H}(Pub,\ \mathtt{nonce},\ A)$ and $s = c \cdot x + r \bmod q$

3. The client sends the proof $(c,\ s)$ to the service back-end which forwards it to the NuID API.

4. The NuID API uses Pub, `nonce`, and π, where $\pi = (c,\ s)$, as inputs to the proof verification algorithm V and outputs TRUE or FALSE.

   V first computes $A = g^s/Pub^c$
   V returns TRUE if and only if $\mathtt{H}(Pub,\ \mathtt{nonce},\ A) = c$

   To see that the scheme is correct, note that in an honest execution we have:

   $g^s/Pub^c = g^{c\cdot x+r}/g^{c\cdot x} = g^r = A$

   and so $\mathtt{H}(Pub,\ \mathtt{nonce},\ A) = c$ as required.

2.1.3 Self-describing Protocols: Interoperability and Extensibility

In the decentralized authentication architecture presented here, the non-sensitive authentication parameters posted publicly to the ledger are self-describing. That is, the parameters encode how the parameters themselves were produced. This implies that any authenticating service or decentralized authentication provider may efficiently dispatch the verification of a proof to the appropriate verification algorithm.

[13] The nonce is part of the "cryptographic challenge" referenced in *Figure 4*.

Concretely, this means that authentication credential metadata such as the zero knowledge protocol; elliptic curve; cryptographic hash function, H; and the key generation algorithm, KeyGen; are themselves embedded as public parameters on the ledger, alongside the user's public authentication parameters (Pub), so that any service may identify the appropriate verification algorithm for a given credential. The benefit of publishing authentication credential metadata publicly is two-fold. First, it allows any participating service to independently verify the shared body of authentication credentials (i.e. the authentication parameters posted to the ledger) without mutual trust. This implies that each participating service may be an independent verifier within the ecosystem should it choose to be, while maintaining interoperability with other verifiers without additional coordination. This allows providers such as NuID to be interchanged with uninterrupted support for all new and existing identities. It also implies that users may share a single persistent identity across participating services should they choose to. This has significant identity management implications that are out of scope of this overview.

Second, publishing the parameter metadata publicly allows authentication data produced by different zero knowledge protocols to be verified side-by-side, because the data itself contains the information necessary to programmatically identify the appropriate verification algorithm and dispatch accordingly. This implies that the system is extensible in what cryptographic primitives and protocols it supports.

2.2 Distributed Ledger Storage Layer

A distributed and decentralized storage mechanism allows authentication credentials to be independently verified by every participant without mutual trust between participants. This universal availability and lack of centralized control is crucial to achieving the decentralized identity benefits described in Sections 1.1 and 1.2. It also reduces the total amount of data any individual service operator must store and protect in order to authenticate users.

NuID uses the Ethereum ledger as a storage layer for authentication credentials. Ethereum was chosen due to its stability, robust decentralization, and maturity of developer tools and resources. However, other layers in the architecture don't depend on a specific credential storage implementation. Alternative storage layers, including both DLT and non-DLT options, may be supported by NuID in the future. It is worth noting that the storage layer determines the scope in which registered credentials are recognized: any party with read-access to the storage layer may independently verify the authentication data registered there, and therefore authenticate registered identities.

2.3 Open-source Client Libraries

As described in *Figure 4*, the NuID client libraries are used at the service front-end for zero knowledge proof generation. These libraries were designed to be lightweight and optimized for resource-constrained environments like the browser. Currently, the NuID client libraries are available in JavaScript. We plan to port these libraries to other native mobile and cross-platform ecosystems in the near future. Up-to-date information on available libraries can be found on the NuID GitHub page (https://github.com/NuID) and in the NuID Developer Portal (https://portal.nuid.io).

The NuID libraries for proof verification and challenge generation are also available with open-source licensing. These libraries serve as reference implementations, promote adoption, and support providers of the architecture. However, developers can greatly simplify their implementation of NuID by offloading these steps to the NuID API, as illustrated in *Figure 4*. NuID's highly-available and misuse-resistant API can be utilized for storing credentials on the Ethereum ledger, generating cryptographic challenges, and verifying ZKPs generated by the client.

NuID's own open-source libraries wrap well-recognized open-source dependencies, which are routinely reviewed by the broader cybersecurity and developer communities for correctness—an intrinsic benefit of the open-source model.

2.4 NuID API and Service Infrastructure

NuID exposes a cost-effective elastic endpoint REST API for credential registration and authentication. The NuID API allows developers and service operators to offload secure authentication as they would with traditional managed authentication services, and to participate in a decentralized authentication ecosystem without operating a node on the backing ledger network. There are currently two root endpoints in NuID's API:

- **/credential** — The credential endpoints allow developers to create and retrieve credentials and supports associating additional metadata to public identities.

- **/challenge** — The challenge endpoints allow developers to create single-use, time-bound challenges, as well as verify proofs generated against previously created challenges. JWTs are currently supported, and alternative bearer tokens and assets are on the immediate development roadmap.

NuID's open-source client libraries and example deployments automate all common interactions with the above service endpoints. There are a number of other developer convenience features relevant to the service API and supporting open-source libraries which are beyond the scope of this overview, but available to read and use on NuID's Developer Portal (https://portal.nuid.io).

2.4.1 Integrations

Many web-based technologies contain a plugin or module system which allows them to be configured and extended, especially for common interactions like user authentication and management.

NuID-supported integrations with popular web technologies aim to provide native modules to reduce developer friction and promote adoption of a decentralized authentication architecture. Popular frameworks in supported languages will be targeted for future development, including

Express (Node.js), Passport (Node.js), Rails (Ruby), Ring (Clojure), Macchiato (ClojureScript), Spark (Java), Spring (Java), Struts (Java), Django (Python), and Laravel (PHP). Extensible web content management systems such as WordPress are also on the development roadmap as an additional integration layer.

2.4.2 Credential Loss and Revocation

Once an authentication credential has been registered to a distributed, append-only public ledger, there is no way to delete or revoke it. This is a strength of the architecture in that registered identities cannot be centrally censored. The immutability of credentials also has implications on how users recover their application data at a given service in the case of credential loss or compromise. It also impacts how services might deny rights to certain identities within the policy terms and network boundaries of their service.

- **Loss and Compromise** — Restoring a lost or compromised authentication credential reduces to a user simply registering a new credential and following any out-of-band verification required by a service to reassociate their existing application data to the new authentication credential. This can take the form of a traditional email-based recovery, call center verification, or any other out-of-band process, including decentralized approaches aimed at savvy users. The old authentication credential remains in the ledger's history and becomes obsolete.

- **Revocation** — A service may deny rights to a given identity as defined by their terms of service within the bounds of their network by adding an authentication credential's identifier to a blacklist. This is similar to how such a process might work today, but crucially such decisions do not leak beyond the individual service's network and auxiliary systems, which becomes important in an ecosystem where persistent identities may be shared across many services with heterogeneous policies.

Put simply, these two processes can be made to look identical to the processes users are accustomed to, with minor mechanical differences "under the hood."

2.5 Web Services and Applications

The outermost and most interesting layer in a decentralized identity architecture is the services and applications that participate in it. At this layer, authentication internals are securely abstracted, and developers are able to focus on defining and implementing interactions specific to their platform that are more meaningful and important to users. While there are no required changes in user-experience or management patterns from the perspective of the end-user, they are able to more granularly define and manage how they are authenticated at each service, optionally sharing—not replicating—a persistent identity across services. The identity management implications of such an architecture are broad and are well-documented in resources relating to self-sovereign digital identity.

3. Implementing NuID

3.0 Who Stands to Benefit?

Ultimately, every participant in a decentralized authentication architecture—service providers, users, and developers—benefits from the resilience and availability intrinsic to decentralized systems. These systems eliminate the reliance on the continuity and benevolence of a third party to appropriately uphold identities, which reduces risk for service providers and users alike.

- **Services** — A decentralized authentication architecture provides digital service operators similar security, liability, operational, and economic benefits of any modern managed authentication and identity solution. As a decentralized technology upheld by open-source libraries, there is no concept of vendor lock-in. The architecture synergizes with existing and emerging identity standards such as OIDC and DIDs and is flexible to meet credential type and storage requirements.

- **Users** — For users, a decentralized authentication architecture provides the security of standards-based implementation, the flexibility to select any reproducible authentication gesture (password, PIN, private key, software or hardware token, etc., each optionally protected under device-local biometrics), and the pseudonymity and cross-service portability of self-sovereign identity. The architecture makes no assumption regarding the availability of a per-user smartphone or personal device, and therefore extends to resource-constrained contexts and applications. The model synergizes with the existing ecosystem of password managers, authenticator applications, and other digital identity protection and management tools.

- **Developers** — As participants in a decentralized authentication model, developers do not need to implement password storage to achieve secure authentication in their application. Application servers in a decentralized authentication architecture only ever handle public and non-sensitive authentication data, removing password storage and caching, request logging, and passive server compromise as vectors for credential leakage. Using open-source libraries and plugins, adoption is identical to language-native authentication frameworks, without the need to provide and protect a password storage back-end, and without outsourcing the same process to an OpenID Connect provider. The model is akin to traditional PKI, where users issue their own credentials, and identities are pseudonymous by default.

3.1 Implementing NuID into Your Service

Whether you intend to use NuID as an authentication solution for a greenfield project, to replace your existing authentication solution in a mature web service, or integrate it as a component in a broader identity model, the NuID team is available as a resource to support you in your implementation. NuID's level of support will scale to a customer's specific requirements—ranging from full managed implementation to full customer self-service. NuID maintains a self-service developer portal which contains the necessary documentation, API key, reference code, and client libraries to deploy NuID into a development or full production environment. This self-service portal can be accessed at https://portal.nuid.io. You can also find NuID on GitHub and npm. Regardless of your requirements, please feel free to contact us at info@nuid.io or join our slack channel with any questions or comments you may have.