User-centric authentication in Web 3.0

Björn Tackmann, DFINITY Foundation

Abstract

Humans are notoriously bad at managing secrets, which may be best witnessed by the ubiquity of problems with passwords in Web 2. The seed phrases used by most Web 3 wallet software, however, only amplify the problem. Seed phrases, unlike passwords, are practically non-memorable, so they have to be managed explicitly outside of the user's brain. On most blockchain platforms, there are also no built-in methods for key recovery or rotation, so failures in safely and securely managing the secrets are immediately catastrophic. So how can we enable non-expert users to safely participate in Web 3?

Internet Identity is a non-custodial and self-sovereign blockchain authentication system on ICP, which enables users to securely manage their identity across multiple devices without ever explicitly touching cryptographic secrets. Internet Identity achieves this by building on two main technical foundations:

- Web authentication: Modern devices support secure management of cryptographic key material using the FIDO and web authentication standards. Instead of remembering different passwords for each service, the user only requires a secure mechanism for unlocking their device, and the device securely manages the secrets.
- Chain-key cryptography: The cryptographic mechanisms implemented in ICP allow to untangle the single, static cryptographic key that represents the user's identity on the blockchain from the multiple, more ephemeral cryptographic keys held on the users devices and used for authentication.

The user experience of Internet Identity revolves around the user's *identity* to which a user can associate multiple devices or recovery mechanisms. The user can then use any one of their associated devices to authenticate toward dapps in a user flow resembling the smooth "sign in with Y" products in Web 2, while maintaining self-sovereignty.

This article describes the Internet Identity blockchain authentication system, its technical foundations, and the vision for future development. The article also discusses experiences from more than two years of production use.

1 Introduction

Transactions in blockchain networks are authenticated using digital signatures. Hereby, a digital signature scheme refers to a cryptographic mechanism in which a client signs a message using their private cryptographic key. The verification of the signature and thus the authenticity of the message, by contrast, is performed relative to a public key, which is derived from the client's private key. Digital signature schemes were first proposed by Diffie and Hellman as *one-way authentication schemes* in their seminal paper introducing public-key cryptography [Diffie and Hellman, 1976].

Assets on a blockchain are associated with an *address* of the user (or the smart contract) that holds the asset. For user-held assets, the address is derived from the user's public key. As a user's assets on a blockchain are controlled through transactions that are authorized via digital signatures, the security and accessibility of a user's assets is directly dependent on the user's ability to keep the private signature key both secure and accessible. If a user's private key was stored insecurely and exposed to some external party, that party would immediately be able to transfer the user's assets. Likewise, when a user loses access to their private key, they also lose the capability of controlling their assets.

The device or program used to store the user's private key is usually referred to as a *wallet*. There are two fundamentally different types of wallets:

- In the case of *custodial wallets*, the user delegates the management of cryptographic keys to a third party, the custodian. When the user intends to send a blockchain transaction, they have to interact with the custodian. During this interaction, the custodian usually authenticates the user. This authentication can be performed using with various methods ranging from standard Web 2 mechanisms (e.g., username and password) to video calls in which the custodian verifies the intent of the transaction with the user.

- A *non-custodial wallet* is a piece of software or hardware that is under control of the user and stores the cryptographic key. Examples of non-custodial wallets include browser extensions such as MetaMask[1] or hardware devices such as the Ledger Name[2] line of devices.

Custodial and non-custodial wallets have different characteristics and use cases, which are not further discussed in this article. This article focuses on the case of non-custodial wallets.

1.1 The problem of managing secrets

A non-custodial wallet, at its core, stores the user's private signature key. Since the key may have significant assets linked to it, the management of this key has two somewhat conflicting requirements: First, the key must be kept *secure*, meaning that it must be protected from access by people other than the legitimate owner. Second, the key must be kept *accessible*, meaning that the owner must still be able to access or recover the key even in case they, e.g., lose or break their devices that store the key.

For backup purposes, most wallets support a standard called BIP-0039.[3] Following this standard, a secret seed from which the cryptographic keys are derived is encoded as a phrase consisting of 12 or 24 common words. Upon replacing their device, the user initializes the new device using the seed phrase, the device decodes the seed and computes the cryptographic keys from it. The mechanism was invented for Bitcoin but is nowadays supported universally in the Web 3 ecosystem.

[1]`https://metamask.io/`

[2]`https://www.ledger.com/`

[3]`https://github.com/bitcoin/bips/blob/master/bip-0039.mediawiki`

The problem with BIP-0039 is that it requires the user to record the seed phrase in a way that is:

1. protected from access by other people, but
2. can be reliably retrieved in case the user needs to recover the wallet after a software upgrade or on a different device.

For tech savvy users that set up a system for high-value transactions, this is usually not a problem. The user will prepare a sheet of paper (or a more durable material such as metal), which will later be stored in a safe or a bank locker for backup access after the seed phrase has been recorded. For more casual use, however, the seed phrase flow is a major obstacle: users that spontaneously set up a new wallet may not have a way to record the seed phrase in that moment at all, or they may record the phrase in a way that is either not properly protected and can be stolen or (more likely in practice) they will not remember where they kept the seed phrase when they need to recover it later.

What is thus needed is a method that is easy to set up and maintain and does not require the user to explicitly manage cryptographic key material. Internet Identity (in short: *II*), the blockchain authentication system described in this paper, solves this problem based on web standards as well as the features of the ICP blockchain.

Outline

Section 2 introduces basic terminology required for the main part of the paper. Section 3 describes the user perspective on Internet Identity; how an identity is created, used, and maintained. The system architecture is then detailed in Section 4, before Section 5 provides a high-level overview of the cryptographic protocols. Section 6 describes some security properties of Internet Identity. Section 7 contains insights from more than two years of production use, and Section 8 lists planned extensions and improvements for future releases.

2 Background

2.1 Web authentication and FIDO

Web authentication[4] is a standard published by the W3C that enables web applications to access secure cryptographic authentication mechanisms. Web authentication was initially designed in the context of two-factor authentication (2FA), where the first authentication factor—typically a password—is complemented with a second factor—typically a device that the user holds. This device could be the user's mobile phone or computer, or a dedicated security device that is attached to the user's phone or computer via USB or NFC.

Web authentication is particularly well-suited to counter phishing attacks, in which a user is tricked into entering their credentials for some trusted web site into a fraudulent web site owned by an attacker. The attacker records the user's credentials, and can subsequently use them to perform malicious transactions on the user's behalf. Web authentication prevents this attack by binding the 2FA authenticator to the domain of the web site: As long as the attacker cannot obtain a valid TLS certificate for the domain of the trusted web site, the binding reliably prevents the user from accidentally using the authenticator on the attacker's fraudulent web site.

The cryptographic keys used in web authentication were initially thought of as being tied to the authenticator device. Services offering 2FA would then offer a method for replacing the authenticator to deal with the case In which the device got lost or stolen. Since Apple and Google developed protocols for securely synchronizing these cryptographic keys across multiple devices, web authentication has recently been suggested as primary user authentication mechanism, replacing the password entirely. In this context, the web authentication keys are often referred to as *passkeys*, which is also the terminology we adopt in the remainder of this work.

[4]`https://www.w3.org/TR/webauthn-2/`

2.2 Threshold cryptography

In a *threshold cryptosystem*, a group of n parties jointly maintains a cryptographic key in a way that for some specified value $t \leq n$, referred to as *the threshold*, any t-out-of-n parties can jointly perform cryptographic operations, but any less-than-t parties cannot. A *threshold (digital) signature* is a cryptographic protocol in which n parties jointly hold a private key, such that for a given threshold t, any t-out-of-n parties can jointly sign a message, but any less-than-t cannot. The concept of threshold signature dates back to [Desmedt and Frankel, 1991].

2.3 ICP and chain-key cryptography

ICP—the Internet Computer Protocol [The DFINITY Team, 2022]—is a blockchain protocol that makes heavy use of threshold cryptography, especially the BLS cryptosystem proposed by [Boneh et al., 2004] and its threshold variant proposed by [Boldyreva, 2003]. A practical deployment of threshold signatures, however, requires a lot more than a bare threshold signature scheme: it also needs protocols for securely generating the key shared among the participants in a way that no single party ever controls the key, as well as for re-sharing the key upon membership changes as old nodes disappear and new nodes come online. Such protocols can be built based on *distributed key generation* protocols such as the one originally proposed by [Pedersen, 1991]. In the context of ICP, the entire suite of protocols used for maintaining and using the threshold signature key is usually referred to as *chain-key cryptography*, alluding to the fact that the blockchain protocol maintains the private signature key.

One main consequence of chain-key cryptography is that external parties do not need to maintain a copy of the entire blockchain to validate artifacts processed by the chain, such as output values computed by smart contracts. Everything the external parties need in order to validate an artifact is the public key of the chain and a signed certificate for the artifact. In a bit more technical detail, after processing one round of transactions, ICP nodes threshold-sign the computed state, which includes the outputs computed by all canisters. (For efficiency, individual outputs computed, e.g., in the same round can be authenticated together by means of a Merkle tree [Merkle, 1987]. The certificate then contains the threshold signature and the Merkle tree path for the relevant artifact.) This not only enables blockchain applications that achieve full security while running on constrained devices or within a web browser, it also enables horizontal scaling of the blockchain protocol: The ICP network consists of multiple subnets, each of which is based on its own blockchain, and these subnets can interact securely with very low overhead.

2.4 Terminology

To remain consistent with existing literature on ICP, we adopt the same terminology. Smart contracts on ICP are referred to as *canisters*. Canisters do have more general properties than smart contracts on other platforms, such as an explicit notion of controller, but these differences will not be relevant in the context of this paper. The API of a canister consists of functions that can be called with arguments. Transactions sent to ICP are referred to as *(ingress) messages*. Each ingress message invokes a function on a canister with a specified argument. Each user and canister has a *principal*, which is the same concept often referred to as *address* in other blockchain platforms. The principal of the *caller* of a function, which may be a user or another canister, is exposed to the invoked canister.

3 The user perspective

The user experience of Internet Identity resembles the one known from Web 2. The user's internet identity, which is identified by a sequence number, looks and feels almost like an account in Web 2. This is despite the fact that II is a fully self-sovereign method of authentication, with the record of the user data relevant for verifying the authenticity of the user stored on the blockchain. More technically, the user's record contains the public keys corresponding to each of the user's passkeys. The user can use the II frontend[5] to manage their internet identity, or rather the record of their data stored on the blockchain. For instance, the user can

[5]`https://identity.ic0.app/`

add or remove additional passkeys or account recovery mechanisms to or from the user record. The frontend also allows to recover the account using any one of the registered mechanisms.

3.1 The creation flow

A user that visits the II frontend for the first time is asked to create a new internet identity. This involves, as a first step, the creation of a new passkey on the user's device. In this process, the device will prompt the user to provide an authentication gesture, which may be presenting the face or a fingerprint or simply touching an external security device. As a next step, the user is prompted to solve a simple CAPTCHA, which serves as a simple countermeasure against bots. Finally, the user is presented a (currently 7-digit) number, which serves as the identifier of the user's internet identity. The creation flow is usually completed in less than one minute.

3.2 The authentication flow

Web applications that support authentication via II generally display a button labeled "sign in with Internet Identity," or similar. Upon clicking the button, the II frontend opens in a new browser tab, in which the user then approves the authentication attempt as well as the use of the passkey. Using the passkey will require the user to provide the same authentication gesture as creating it (cf. Section 3.1). After the user is authenticated, the browser tab with the II frontend closes and the user is directed back to the application.

3.3 The management flow

If a user's internet identity is controlled through a single passkey, accessing this identity may be impossible if the device on which the passkey is stored is lost or stolen. Therefore, it is strongly suggested to add multiple passkeys or a recovery mechanism to the identity.

Users can maintain their internet identities by visiting the II frontend. After selecting the desired identifier, the user is asked to approve the use of the associated passkey, after which the user is forwarded to a management page displaying the currently registered passkeys and recovery mechanisms. On this page, new passkeys or recovery devices can be added, which is described in Section 3.4. Passkeys and recovery devices that are no longer functional or needed can be deleted.

3.4 The device addition flow

In case a user's passkeys are not automatically synchronized across all the user's devices, multiple passkeys can be associated to the same identity. All passkeys have equivalent capabilities, which implies that the user can use web applications from all registered devices seamlessly. The flow for adding a new passkey can be initiated from the II frontend on either (i.e., existing or new) device. When initiated from the existing device, a link (and QR code) is displayed that needs to be visited on the new device. Upon visiting the link, the new device generates a new passkey (authenticating the user in the process) and displays a 6-digit code, which then has to be entered on the existing device. Once this step is completed, the internet identity can be accessed from both devices.

II urges the user to additionally add a *recovery mechanism*. Currently, two types of mechanisms are supported. The first type is using an additional passkey, such as one stored on an external security device. The second type is using a BIP-0039 seed phrase, which is generated in the II frontend.

Supporting BIP-0039 seed phrases as recovery mechanisms may seem surprising; wasn't one goal of II to liberate the user from the burden of maintaining seed phrases? Indeed, and the addition of a seed phrase is entirely optional, the identity can be safely maintained via multiple passkeys. Also, the recovery seed phrase can be set up and replaced at any point in time, whenever it is convenient for the user.

4 System architecture

The II blockchain authentication system consists of three core software components: The *backend*, a canister smart contract running on ICP; the *frontend*, which runs as a web application in the user's browser; and the *authentication client*, a library used by web applications that support authentication with II.

The backend canister. The backend canister stores all data relevant for user authentication, which includes the public keys associated with the users' passkeys and some additional metadata (e.g., whether a passkey is used for authentication or recovery). The backend canister serves as smart contract that encodes the rules for modification of a user's records. The canister also serves the frontend application into the user's browser.

The frontend application. The II frontend runs in the user's web browser. It enables the user to modify the data in the backend canister, such as by adding or removing passkeys and recovery mechanisms. The frontend also supports the authentication flow used by applications. Passkeys used for II are associated with the URL of the II frontend, `https://identity.ic0.app/`.

The authentication client. Web applications that integrate with II can use a library referred to as authentication client. The library offers a simple interface for application developers and manages the in-browser interaction with the II frontend during the authentication flow.

5 The Internet Identity protocol

As described in Section 1, blockchain transactions are authenticated by digital signatures, and the principal of the user sending the transaction is derived from the cryptographic key used to sign the transaction. This means that assets and capabilities are bound to the cryptographic keys that the transactions are signed with. In order to support the control of assets from multiple devices with (potentially) different passkeys, II has to dissociate the passkey stored on the user's device from the principal used to control assets on chain.

II achieves this dissociation by introducing a layer of indirection: The passkeys stored on the user's devices serve as a means of authentication toward the II backend canister. All the user's assets are then associated to a principal that is under the control of that backend canister. The remainder of this section describes how the II protocol enables the backend canister to achieve its functionality.

5.1 Canister signatures

ICP subnets use threshold signatures to certify the state of the subnet after each round of computation (cf. Section 2.3). The subnet state contains so-called *certified variables* that can be written by canisters. A certified variable can then be efficiently verified by anyone using the subnet public key and the certificate for the variable written by a canister.

Certified variables can be used to define a (pseudo-)signature scheme for canisters, which is referred to as *canister signature*[6]: The canister writes the message it intends to sign into a certified variable, and the certificate for the variable becomes the digital signature in the signature scheme. The public key relative to which this new signature can be verified consists of the subnet's public key and the signing canister's principal, along with some canister-chosen auxiliary data.

Canister signatures can be used to authenticate ICP transactions. The caller principal of such a transaction is derived from the canister principal and the specified auxiliary data. Looking forward, the auxiliary data allows the II backend canister to generate different principals for different users and different contexts. Beyond that, however, the functionality of signing transactions by itself may not seem particularly useful; in the end, a canister can send messages to other canisters directly on chain, so why would it be helpful for the canister to sign a transaction? This becomes clear in the following section.

5.2 Delegations

Recall that the caller principal of an ingress message is derived from the signature public key that the message is signed with. ICP additionally supports a notion of *delegation* from one public key A to a second public key B. This allows the user to sign a message with public key B and send it to the blockchain together with

[6]`https://internetcomputer.org/docs/current/references/ic-interface-spec#signatures`

the delegation from A to B, which will result in the call being executed with the caller being set to the address derived from public key A.

The II backend canister uses the concept of delegation to delegate from the canister-controlled user (pseudo) public key to an actual key controlled by the web application running in the user's browser. Prior to the authentication flow, the web application generates a fresh signature key pair that corresponds to key B above. During the authentication flow, this session key is sent to the II backend canister, which signs a delegation from the (pseudo) public key associated with the user (which corresponds to key A above) to the session key. The delegation is returned to the web application, which can then sign further ingress messages with the session key.

5.3 Protocol flow

The complete protocol flow (on an abstract, cryptographic perspective) is then as follows:

- The user visits the web application. Upon the user clicking the button "sign in with Internet Identity," the II frontend opens in a new browser tab. The application generates a fresh signature key pair, holds the private key in the browser memory, and sends the public key to the II frontend via an in-browser message passing protocol.
- The II frontend generates a message that includes the newly generated public key. Upon the users approval, the message is signed using web authentication and sent to the II backend canister.
- The II backend canister validates the authority of the user and signs, using canister signatures with the public key associated to the user, a delegation toward the session public key provided in the ingress message.
- The II frontend retrieves the signed delegation from the II backend canister and forwards it to the application, again using the in-browser message passing protocol.

- The application signs blockchain transactions with the session public key, and includes the delegation from the II backend canister. This ensures that the *caller* attribute of the respective transactions is set to the user's identity that is controlled through the II backend canister.

6 Security

While a full analysis of the security if Internet Identity is beyond the scope of this paper, a few relevant security properties should be pointed out.

Isolation between different applications. As described in Section 5, an application that supports authentication with II receives, as result of the authentication flow, a delegation that allows the application to send messages using the user's principal to canisters on ICP. If multiple applications were to use the same user principal, security problems would arise: Suppose one of the applications is entrusted with maintaining valuable assets, while some other one is not. The second application, however, would have equal access to all assets maintained by the first application. Therefore, II derives a different principal for each application even for the same user. This is achieved by including the domain name of the application in the auxiliary data mentioned in Section 5.1. The use of the domain name as a separating property between different applications is consistent with the browser security model, which determines access based on the notion of *origin* that also includes the domain name.

The use of different user principals for different applications also impedes traceability of the same user across multiple devices and thus provides a certain level of privacy.

Storage of cryptographic keys. Devices that support web authentication generally store passkeys in specific secure chips, and the private keys cannot be exported to the operating system or even applications. Assuming the security of the secure hardware chips, the passkeys cannot be extracted even if the user's device gets infected with malware or stolen.

The fact that keys are stored securely does, however, not entirely rule out attacks via malware: As the user has no possibility to securely validate the message that is signed, malware on the user's device could replace a legitimate message that a user intends to sign with a fraudulent one before it is signed by the secure chip. In that sense, the security level is lower than with specialized hardware wallets that allow the user to validate the transaction details.

Security of the II canister. The II backend canister is developed as open-source software,[7] and upgrades to the canister are rolled out through ICP's decentralized governance system. This process ensures a high level of transparency.

7 Practical experience

Internet Identity was developed by the DFINITY Foundation and launched in May 2021 together with the ICP blockchain. Since then, the II canister is controlled through ICP's decentralized governance system. The further development of the Internet Identity protocol and its implementation is performed by DFINITY in collaboration with the ICP community. The subsequent paragraphs describe a few of the learnings the DFINITY team made since then.

Apple devices deleted web authentication keys. In Apple's implementation of web authentication up to iOS 15, cryptographic keys used in web authentication were strictly bound to the device on which they were generated. Somewhat surprisingly, upon clearing the browser history and cache, the web authentication keys would also get deleted. The effect of this behavior was that users would get locked out of their internet identities, requiring recovery.

The treatment of these keys changed completely in iOS 16 with Apple's introduction of the term *passkey*. Since then, the keys are synchronized between different devices via iCloud. They are also no longer deleted upon clearing the browser history and cache.

[7] `https://github.com/dfinity/internet-identity/`

Windows is different. Most platforms, including iOS, macOS, Android, as well as external security devices, generally use the ECDSA signature scheme, which is nowadays used ubiquitously on the web. Microsoft's Windows Hello, by contrast, is based on the older RSA signatures, whose use is otherwise generally discouraged. RSA signatures were not supported in the initial deployment of II, support was added in a later release.[8]

Users do not read warnings. Initial revisions of the II frontend supported unsafe operations such as the removal of all passkeys from an internet identity. While these operations were only performed after the user acknowledged multiple warnings about the deletion of the current and last passkey, several users contacted DFINITY for help after still performing these actions. Current revisions instead entirely block such unsafe behavior.

8 Future directions

Internet Identity has seen significant improvements since the first release in 2021, especially in terms of user experience. Several further features are currently planned or under development.

Attribute support. The II protocol as discussed in this paper is an authentication mechanism, but it currently falls short of being a full *identity* solution: The reason is that no attributes (such as age, nationality, academic credentials) can currently be assigned to the user's principal. Canisters can rely on the *caller* attribute of transactions to be set securely, but they do not know anything else about the user. Work toward supporting W3C verifiable credentials[9] in II is underway and close to completion.

Cryptographic privacy. As discussed in Section 6, the same user has different principals when using different applications. While this impedes traceability

[8]`https://medium.com/dfinity/windows-hello-support-added-to-internet-identity-e9021f74afe9`

[9]`https://www.w3.org/TR/vc-data-model/`

across different applications, it does not achieve anonymity in a strong, cryptographic sense. Based on advanced threshold cryptography that is currently being integrated in ICP, full cryptographic anonymity will be possible in the future [Cerulli et al., 2023].

Additional recovery mechanisms. The currently supported recovery operations (designated passkey, seed phrase) achieve a high level of self-sovereignty, but come at the cost of shifting significant operational responsibility to the user. Additional mechanisms such as social recovery (delegating authority to one or more friends) or support for professional recovery providers can be added to the protocol without major technical complications. By keeping the use of such features optional, II can serve both the (technically more proficient) users that have a preference for full self-sovereignty and the users that are willing to compromise on self-sovereignty for the benefit of easier management.

Acknowledgment

I would like to thank all researchers and engineers that contributed to the protocol and code, including Bartosz Przydatek, Christoph Hegemann, Frederik Rothenberger, Islam El-Ashi, Joachim Breitner, Mary Dwyer, Michel Abdalla, and Nicolas Mattia.

References

[Boldyreva, 2003] Boldyreva, A. (2003). Threshold signatures, multisignatures and blind signatures based on the gap-diffie-hellman-group signature scheme. In Desmedt, Y., editor, *Public Key Cryptography*, volume 2567 of *Lecture Notes in Computer Science*, pages 31–46. Springer.

[Boneh et al., 2004] Boneh, D., Lynn, B., and Shacham, H. (2004). Short signatures from the weil pairing. *J. Cryptol.*, 17(4):297–319.

[Cerulli et al., 2023] Cerulli, A., Connolly, A., Neven, G., Preiss, F.-S., and Shoup, V. (2023). vetKeys: How a blockchain can keep many secrets. Cryptology ePrint Archive, Paper 2023/616. `https://eprint.iacr.org/2023/616`.

[Desmedt and Frankel, 1991] Desmedt, Y. and Frankel, Y. (1991). Shared generation of authenticators and signatures (extended abstract). In Feigenbaum, J., editor, *CRYPTO*, volume 576 of *Lecture Notes in Computer Science*, pages 457–469. Springer.

[Diffie and Hellman, 1976] Diffie, W. and Hellman, M. E. (1976). New directions in cryptography. *IEEE Transactions on Information Theory*, 22(6):644–654.

[Merkle, 1987] Merkle, R. C. (1987). A digital signature based on a conventional encryption function. In Pomerance, C., editor, *CRYPTO*, volume 293 of *Lecture Notes in Computer Science*, pages 369–378. Springer.

[Pedersen, 1991] Pedersen, T. P. (1991). A threshold cryptosystem without a trusted party (extended abstract). In Davies, D. W., editor, *EUROCRYPT*, volume 547 of *Lecture Notes in Computer Science*, pages 522–526. Springer.

[The DFINITY Team, 2022] The DFINITY Team (2022). The internet computer for geeks. Cryptology ePrint Archive, Paper 2022/087. `https://eprint.iacr.org/2022/087`.