Blockstack

Blockstack Technical Whitepaper v 2.0

May 2019

By Blockstack PBC

Parts of this whitepaper were published earlier in the following peer-reviewed conferences and magazine:

- M. Ali, J. Nelson, R. Shea and M. J. Freedman, *"Blockstack: A Global Naming and Storage System Secured by Blockchains"*, 2016 USENIX Annual Technical Conference, Denver, CO, June 2016.
- J. Nelson, M. Ali, R. Shea and M. J. Freedman, *"Extending Existing Blockchains with Virtualchain"*, Workshop on Distributed Cryptocurrencies and Consensus Ledgers, Chicago, IL, July 2016.
- M. Ali, J. Nelson, R. Shea and M. J. Freedman, *"Bootstrapping Trust in Distributed Systems with Blockchains"*, USENIX ;login: Issue: Vol. 41, No. 3, Pages 52-58, Fall 2016.

This version (2.0) of the whitepaper describes major changes since the whitepaper 1.0 released in 2017. The earlier whitepaper can be referenced as:

- M. Ali, R. Shea, J. Nelson and M. J. Freedman, *"Blockstack: A New Internet for Decentralized Applications"*, Whitepaper Version 1.1, Oct 2017.

Some systems and concepts described in this whitepaper were also discussed in the following doctoral dissertations of the respective authors of this whitepaper:

- M. Ali, *Trust-to-Trust Design of a New Internet*, PhD dissertation, Princeton University, June 2017.
- J. Nelson, *Wide-area Software-defined Storage*, PhD dissertation, Princeton University, June 2018.

DISCLAIMER: The Blockstack Tokens, "Stacks Tokens" or "Stacks", are a crypto asset that is currently being developed by Blockstack Token LLC, a Delaware limited liability company, whose website can be found at www.stackstoken.com.

This whitepaper does not constitute an offer or sale of Stacks tokens or any other mechanism for purchasing Stacks. Any offer or sale of Stacks tokens or any related instrument will occur only based on definitive offering documents for the Stacks tokens.

This communication may be deemed "testing the waters" material under Regulation A under the Securities Act of 1933. We are not under any obligation to complete an offering under Regulation A. We may choose to make an offering to some, but not all, of the people who indicate an interest in investing, and that offering might not be made under Regulation A. We will only be able to make sales after the Securities and Exchange Commission (SEC) has "qualified" the offering statement that we have filed with the SEC. The information in that offering statement will be more complete than the information we are providing now, and could differ in important ways. You must read the documents filed with the SEC before investing.

No money or other consideration is being solicited, and if sent in response, will not be accepted. No offer to buy the securities can be accepted and no part of the purchase price can be received until the offering statement filed by the company with the SEC has been qualified by the SEC. Any such offer may be withdrawn or revoked, without obligation or commitment of any kind, at any time before notice of acceptance given after the date of qualification.

An indication of interest involves no obligation or commitment of any kind.

Any person interested in investing in any offering of Stacks Tokens should review our disclosures and the publicly filed offering statement relating to that offering, a copy of which is available at www.sec.gov.

Blockstack is not registered, licensed or supervised as a broker dealer or investment adviser by the Securities and Exchange Commission (SEC), the Financial Industry Regulatory Authority (FINRA) or any other financial regulatory authority or licensed to provide any financial advice or services.

The Blockstack Decentralized Computing Network

Muneeb Ali Jude Nelson Aaron Blankstein
Ryan Shea Michael J. Freedman*

https://blockstack.org
Whitepaper Version 2.0

May 30, 2019

Abstract

In this paper, we present the Blockstack decentralized computing network. Blockstack provides a full-stack alternative to traditional cloud computing for building secure, private applications. Compared to traditional internet applications, a key differentiator of decentralized applications as enabled by Blockstack is that most business logic and data processing runs on the client, instead of on centralized servers hosted by application providers. In many ways, a transition to decentralized computing is similar to a transition from mainframes to desktop computing in the 1980s.

Blockstack follows the end-to-end design principle to keep the core of the network as simple as possible while pushing complexity to the edges, i.e., user devices and user-controlled storage. At the foundation of our network is the Stacks blockchain, which is designed to (a) scale decentralized applications, and (b) incentivize developers to build high-quality applications on the network. The Stacks blockchain uses a novel Tunable Proofs election system to securely bootstrap a new blockchain. Our new smart contracting language, Clarity, enables powerful on-chain expressibility while optimizing for security and predictability of smart contracts and admitting static analysis for all transactions.

A key component of our architecture is a highly scalable and performant decentralized storage system, called Gaia, that enables user-controlled private data lockers. Users connect their private data lockers to their Blockstack client software, and applications write user data to the data locker directly. A universal ID and authentication system, called Blockstack Auth, removes both the need to sign up separately for each application as well as the need for password-based logins, which are known to be less secure than cryptographic authentication.

Our SDKs and developer tools make developing a Blockstack application no more complicated than developing traditional internet applications, and developers using Blockstack don't need to worry about running servers or databases. Blockstack has been operating in production

*Professor of Computer Science at Princeton University and technical advisor to Blockstack PBC.

with over 100 independent applications built on it as of early 2019. The architecture of Blockstack has evolved after years of production experience and feedback from application developers. This paper, which describes the result of that architectural evolution, is a major revision of our earlier 2017 whitepaper.

1 Introduction

The internet, designed more than 40 years ago, has grown from little more than a research project to something that touches almost all digital interactions in the world. Although the core, lower-layer internet protocols have remained somewhat consistent since the 1990s, the internet's application layer and server infrastructure have evolved tremendously to support the massive growth of internet applications.

The primary model for building internet applications is the client/server model [1], popularized in the 90s. This model was a short-term blessing with long-term negative consequences. It enabled the Web to take off but caused Web services to become increasingly dependent on remote servers. Cloud computing is an evolution of the basic client/server model. Today, cloud providers stores private user data, run application logic and computations, manage access credentials, and so on.

In the last decade, we've started seeing the negative consequences of cloud computing, which has called into question the entire model of building software while relying on the client/server model. Mass data breaches [2], loss of user privacy [3], lack of data portability, and the broader mistrust of tech giants [4] stems from the core design of the client/server model. Given the increasing importance of computing in human society, we cannot let outdated computing models define how we live our lives.

The next evolution of cloud computing will leverage more powerful client devices, edge computation, and global connectivity to reduce reliance on these centralized platforms. This evolution towards decentralized computing has already started, and we believe that it is the most significant technological shift for the computing industry since the arrival of desktops after mainframes. Decentralized computing can change how software gets built and used. It gives developers a set of new tools to work with and changes the relationship consumers have with software: the software exists to protect the users and optimizes the benefit of users over anything else.

Blockstack is an open-source effort to design, develop, and grow a decentralized computing network that provides a full-stack alternative to traditional cloud computing. Blockstack is re-imagining the application layer of the traditional internet and provides a new network for decentralized applications; applications built on Blockstack enable users to own and control their data directly [5]. Blockstack uses the existing internet transport layer and underlying communication protocols while removing points of centralization in the application layer. We follow the end-to-end design principle [6, 7] to keep the core of the network simple while pushing complexity to the clients. To scale the applications, we minimize global state changes and provide a reliable decentralized storage system that gives comparable performance to cloud

storage. Further, our full-stack approach gives default options for all developer stack components necessary to build decentralized applications. Blockstack is modular, and developers can easily customize it and integrate alternative technologies.

This paper is a major revision of our earlier 2017 whitepaper and incorporates the evolution of our design as informed by lessons from production deployments and feedback from application developers. Parts of our 2016 peer-reviewed publications [8, 9, 10] are also outdated, and we encourage the readers to refer to this paper for the latest Blockstack design. This paper introduces the design of our new *Stacks* blockchain, which is designed to scale decentralized applications and provide incentives to developers for building high-quality applications (Section 2). We present a new smart contracting language, *Clarity*, that optimizes for security and predictability (Section 3). We outline the design of the *Gaia* decentralized storage system (Section 4), our authentication protocol (Section 5), developer tools (Section 6), and highlight some ways in which application developers are currently using Blockstack (Section 7).

1.1 Decentralized Computing Overview

Decentralized systems are a particular type of distributed system where no single entity is in control of the underlying infrastructure, and nodes have economic incentives to participate in the network. Recent interest in decentralized networks started with the release of the Bitcoin whitepaper [11]. Blockchains and cryptocurrencies play a central role in contemporary decentralized systems. We recommend readers to see [12] for background on blockchains and cryptocurrencies.

There are many different types of decentralized systems in production today. The primary goal of Bitcoin, the first and currently the largest blockchain network, is to track and resolve the ownership of the Bitcoin digital currency. The goal of Ethereum [13] is more general purpose: to construct a "world computer" to enable smart contracts and decentralized applications. Filecoin [14] is an attempt to construct a network for decentralized file hosting and storage. In contrast, Blockstack attempts to realize *a full-stack* for decentralized computing, focusing on enabling secure, private applications where the blockchain layer handles minimal state and logic.

1.2 Design Goals

The design of Blockstack optimizes for the following properties:

1. **Ease of Use.** Decentralized applications should be as easy to use for end users as current internet applications. Moreover, decentralized applications should be as easy to develop as developing on cloud computing is today.
2. **Scalability.** Decentralized applications should support users at internet-scale, i.e., hundreds of millions to billions of users. To do so, the network (including the blockchain) must scale with the number of users and applications it runs.

3. **User Control.** Applications that use decentralized computing should put users in control by default. Instead of relying on servers operated by applications, users should be able to provide their computation and storage resources.

With these design goals in mind, Blockstack makes design choices that differentiate it from contemporary decentralized computing approaches with "heavy" blockchains and "world computer" design philosophy [13, 15, 16, 17].

Minimal logic and state at the blockchain layer: To achieve *scalability*, Blockstack minimizes application logic and data at our "light" blockchain layer. Using blockchain operations for application logic and storage is inherently slower than "off-chain" approaches; the need to synchronize and validate state across a wide range of networks and devices imposes significant limits on the throughput of such operations. The limiting factor is underlying bandwidth for global connectivity and memory/storage available at typical network nodes, i.e., physical limitations (vs. any protocol limits).

Localized state changes vs. global state changes: The Blockstack network uses the full-stack approach to ensure that applications built on Blockstack are *scalable*: interactions in applications result in local state changes vs. global state changes whenever possible. Because of this, our storage system (Gaia, see Section 4) and authentication protocol (see Section 5) are fundamental components of our network— they enable applications to interact with a user's private data locker and authenticate a user without ever issuing a blockchain transaction. The Stacks blockchain is only used to coordinate global state transitions in a consistent way (such as registering a globally-unique username) in a decentralized fashion.

Reliable cloud-like storage vs. peer storage: Applications built on Blockstack store data with the user (using their private data lockers) and don't need to store any user data or access credentials at the server side. This approach not only puts users in control of their data but also reduces complexity for developers: developers no longer need to run servers and databases and pay cloud infrastructure bills on behalf of their users. Moreover, we avoid reliability and performance issues inherent with peer-to-peer storage [18] and repurpose existing cloud storage providers in a decentralized wide-area file system — the blockchain layer only stores pointers to user's data lockers.

Full-stack SDKs for developers: Blockstack takes a "full-stack" approach and provides default options for all the layers required to develop decentralized applications. Developer SDKs abstract away the complexity of the blockchain and other technologies at work; application developers can build their applications with ease using interfaces of SDKs (Section 6). Various layers of the developer stack are modular and can be used with other technologies as needed.

In addition to these differences from contemporary decentralized computing approaches, our smart contract language also makes unique design decisions to optimize for security and predictability of smart contracts (see Section 3 for details).

1.3 A New Model for Applications

Blockstack provides developers with a new model for constructing applications, ensuring that the applications are decentralized and put the users in control by default:

1. **No opaque databases:** In the client/server model, databases are a core part of any application because the server-side needs to store and query large amounts of user data. In decentralized computing, developers don't need to worry about maintaining and securing databases since they do not host data in the first place. Developers mostly focus on their app logic; users download the apps and plug-in their private data lockers. Databases, if used, are functionally equivalent to "search indexers" on the old internet— services which index public data. Anyone can create these indexes using the underlying (decentralized) data.
2. **No servers:** In the client/server model, apps scale by adding more servers as computations for all users execute on the server side. In decentralized computing, apps run client-side, and each new user brings their computation and storage capacity to the network (rather than relying on the app developers). Developers only need to supply minimal infrastructure for hosting the application code, since each user brings the storage and computing resources they need to use the app.
3. **Smart contracts:** In the client/server model, global state changes are coordinated by a central server which functions as the sole authority of truth in the network. In decentralized computing, these state changes occur through smart contracts executing on an open blockchain.
4. **Decentralized authentication:** In the traditional internet, users authenticate using some trusted authentication process. If an application maintains a user database, the application authenticates the user with a password and sometimes a second factor. If an application relies on a third-party identity service, like Google or Facebook, it will use the OAuth [19] protocol to obtain an assertion from that identity service. Of course, all these approaches remove control of the process from the users themselves. In decentralized computing, authentication is performed by the user's client, by cryptographically signing a statement proving control over a particular username anchored to the blockchain. Any application can independently verify these proofs.
5. **Native tokens:** In traditional internet applications, payment activities are usually performed using third-party services like credit cards. Digital tokens are a native asset of decentralized computing platforms like Blockstack and Ethereum. Users have direct ownership of these tokens and can use them directly to register digital assets and smart contracts, as well as pay for executing smart contracts. Use of such native tokens can be programmed through smart contracts to build subscription services and automate other app functionality. Such programmable tokens were traditionally not available to developers of traditional internet apps.

1.4 Layers of Decentralized Computing

The Blockstack decentralized computing network logically exists at the "application layer" in the traditional internet design. However, the Blockstack network itself is composed of multiple systems which together provide the necessary components for implementing decentralized applications:

1. **Stacks Blockchain:** The foundation for the Blockstack network is the Stacks blockchain which enables users to register and control digital assets like universal usernames and register/execute smart contracts. Digital assets like universal usernames, in turn, allow users to control their data storage and more—users link their access credentials for private data lockers with their universal usernames.
2. **Gaia:** The Gaia storage system is a user-controlled storage system that enables applications to interact with private data lockers. Users can host these encrypted data lockers on a cloud-provider, local disk, or remote storage. Importantly, the *user* controls the choice of the underlying provider. Data on Gaia is encrypted and signed client-side by the user's cryptographic keys. Data lockers for users are discovered by looking up information on the Stacks blockchain.
3. **Blockstack Authentication:** The Blockstack Authentication protocol is a protocol for decentralized authentication with applications. This protocol enables users to authenticate using identities that they own and provide information about which Gaia location should be used to store that user's application data.
4. **Blockstack Libraries and SDKs:** At the top of the software stack are the developer libraries and SDKs through which application developers and users interact with the various components of the Blockstack network. For example, Blockstack client software allows users to register and manage their own identities. Blockstack's developer libraries make it *as easy* for developers to build Blockstack applications as it is to create traditional web applications.

2 Stacks Blockchain

The Blockstack network's foundational layer is the Stacks blockchain. The Stacks blockchain provides the global consensus and coordination layer for the network and implements the native token of the Blockstack network called the *Stacks token*. Stacks tokens are consumed as "fuel" when users register digital assets like universal usernames, software licenses, pointers to storage lockers, etc. They are also used to pay miners for registering/executing smart contracts.

In this section, we present the high-level design of the Stacks blockchain. For details on how these designs are implemented and are evolving, we recommend reading the Stacks Improvement Proposals (SIPs) for the various components[1]. We plan to update

[1] Available at `https://github.com/blockstack/blockstack-core/tree/develop/sip`

this paper as more SIPs are accepted through the Stacks improvement process. The Stacks blockchain incorporates the following design decisions:

1. A Tunable Proofs mechanism for leader election.
2. A Proof-of-Burn mining algorithm to reuse hashpower of existing blockchains.
3. A novel peer network (Atlas) which uses random graph walks for peer connectivity and reduces the amount of data required to achieve consensus.
4. A smart contracting language (Clarity) that is non-Turing complete and *interpreted*.

Blockchain Versions: The current Stacks blockchain is at "Version 1" which is an initial implementation to deploy basic functionality. Stacks blockchain v1 uses the Bitcoin network to implement its consensus algorithm and support Stacks token operations like the transfer operations. Stacks blockchain v1 implements smart contracts for use cases like the Blockstack Naming System [8]. For more details on the implementation and functioning of version 1, see the implementation available on Github [20]. The remainder of this section discusses the design of "Version 2" of the Stacks blockchain. Stacks blockchain v2 implements the full functionality of our new consensus algorithm and smart contract language and will be a major upgrade from version 1.

2.1 Leader Election

Blockstack's first-generation blockchain operated logically on top of Layer-1 (L1), and each transaction was in 1-to-1 correspondence with an L1 Bitcoin transaction. The reason for doing this was to ensure that the the difficulty of reorganizing Blockstack's blockchain is just as hard as reorganizing Bitcoin's blockchain – a lesson learned from security problems of smaller blockchain networks like Namecoin [8].

The Stacks blockchain uses a Tunable Proofs mechanism for the leader election process. The Tunable Proofs mechanism is a leader election system that can take input from multiple mechanisms and adapt the relative weight given to each input. For example, with Tunable Proofs we can combine a native Proof-of-Work algorithm with the added functionality to reuse hash-power from another, more established, blockchain. Our goal with Tunable Proofs is to securely bootstrap a new blockchain and slowly transition to using the native PoW mechanism. The current Tunable Proofs mechanism has two parts (a) native Proof-of-Work and (b) Proof-of-Burn of another cryptocurrency.

Initially, the Proof-of-Burn part of mining has more weight. With Proof-of-Burn miners burn cryptocurrency to indicate their interest in participating in the mining process. To be elected as a leader, a candidate burns the underlying cryptocurrency (i.e., Bitcoin) and commits to an initial set of transactions in the leader's would-be block. This commitment *also* serves as the leader's fork selection: the block's consensus hash must include the prior block header. In the event of multiple competing forks,

leaders who choose to "mine" on losing forks do not receive block rewards, transaction fees, or recover their burned cryptocurrency.

The proof-of-burn mechanism used in the Stacks blockchain allows for:

High validation throughput. The number of Stacks transactions processed is decoupled from the transaction processing rate of the underlying "burn chain" (i.e., Bitcoin). Using Proof-of-Burn elections allows for *entire blocks* of Stacks transactions to confirm with each new block in the underlying burn chain.

Low-latency block inclusion. By enabling a single leader election, our Proof-of-Burn consensus algorithm allows the current leader to *immediately* include a new transaction from the mempool in their Stacks block. This *block streaming* model allows users to learn that a block includes a transaction within seconds.

Open leadership set. The Proof-of-Burn election allows *anyone* to become a leader. This mechanism ensures that the Stacks blockchain is an open blockchain (as opposed to closed blockchains which rely on fixed leadership sets, or delegated Proof-of-Stake systems which behave functionally as closed sets). Further, by performing *single-leader election*, our consensus algorithm ensures that would-be leaders don't need to coordinate with each other.

Participation without mining hardware. The work required for participation as a leader involves *burning* a cryptocurrency, rather than a traditional Proof-of-Work mining scheme. Because of this, mining hardware is not necessary to participate as a leader. Anyone who can acquire the burn cryptocurrency can participate in mining, even if they can only afford a minimal amount.

Fair mining pools. The Stacks blockchain natively supports fair mining pools. Anyone participating in the network can burn a cryptocurrency in support of the election of a given leader. Users who commit such "user support burns" share in equal proportion as the leader in a given Stacks block's rewards.

Ability to failover. This design ensures that in the event of the burn chain becoming unstable or otherwise unsuitable for mining the Stacks chain, the Stacks chain can use a different burn chain.

More details on our Proof-of-Burn component are available at [21]. It's possible that the Proof-of-Burn component may not be needed in the future once there is enough native hash-power on the Stacks blockchain.

2.2 Tunable Proofs

The Stacks blockchain includes a native Proof-of-Work (PoW) component in the consensus algorithm in addition to the Proof-of-Burn part. This combination enables sharing the security responsibility of the chain with the Proof-of-Burn election system described in SIP-001. This combination of native PoW and Proof-of-Burn is the current implementation of Tunable Proofs in our system and allows for a responsible introduction of native PoW mining, where Proof-of-Burn ensures chain stability, even while Proof-of-Work interest is low. The tunable aspect opens a path towards migration if the underlying burn chain deteriorates. Tunable Proofs also enable us to research other Proof-of-Work or Proof-of-Stake mechanisms and slowly introduce them over the years in a tunable fashion.

The current native PoW component in leader election works by allowing leader candidates to include a Proof-of-Work nonce in their burn transaction optionally. The amount of work required to produce that nonce (i.e., some function the number of leading zeros in the resulting hash) will count towards the candidate's "burn amount". Initially, there will a 5% cap on the native PoW (relative to the submitted burn amount). The native PoW component is still undergoing substantial development and design. As more details are fleshed out, this section (and a corresponding SIP) will be updated.

2.3 Atlas Peer Network

The Atlas Peer Network is a content-addressable peer network which implements a gossip-protocol where each peer keeps track of which other peers are in the network, and each peer attempts to store a full replica of all the data in the network. The capacity of the network is rate-limited by the Stacks blockchain: any new entry in the data set must be associated with a transaction on the Stacks blockchain. The Atlas Peer Network works as a subsystem of the Stacks blockchain. It's designed to be an unstructured peer network to avoid issues with nodes joining and leaving the network [18, 22]. Further, since all nodes keep a replica of all data, and an index of data is available from the Stacks blockchain, new Atlas nodes can quickly sync on data they need to store as they know in advance what data they should store from other peers (a properly generally not available to nodes in peer-to-peer networks).

The Atlas network functions as an "extended storage" subsystem for the Stacks blockchain. Our design approach is to rely *as little as possible* on interacting directly with the Stacks blockchain itself and store as little data on it as possible. For many applications on Blockstack, such as the Blockstack Naming System (BNS) smart contract [8], it is essential to have a mechanism for storing *immutable* and *timestamped* data. In BNS, this is used to associate usernames with routing information used to discover that user's profile and application data. In most blockchains, storing this kind of data is done *directly* on the blockchain itself. However, we chose instead to store hashes on the blockchain (where space is expensive) and implement a separate peering network for exchanging the data which corresponds to those hashes.

2.4 Stacks Token Usage

The native Stacks token implemented by the Stacks blockchain enables several foundational operations on the Blockstack network:

1. **Fuel to register digital assets.** The Stacks token is used to register different kinds of digital assets like usernames, domain names, software licenses, podcasts, and several others.
2. **Fuel to register/execute smart contracts.** Executing smart contracts requires fuel to fund the cost of verifying the correctness of the smart contract and executing them. Stacks tokens are also used to cover the cost of storing the smart contract in the Stacks blockchain.
3. **Transaction fees.** Stacks tokens are used to pay transaction fees for including a transaction in the Stacks blockchain.
4. **Anchored app chains.** For apps that become massively popular on Blockstack, our blockchain has a scalability on-ramp where an app can initialize its blockchain on top of the Stacks blockchain. Such an "app chain" burns Stacks for their mining and progress.

The above list is not exhaustive — as the Blockstack network matures, we expect network participants will discover and invent other uses for the Stacks tokens. We're currently actively researching a "App Staking" mechanism where token holders can potentially participate in our developer incentive program called "App Mining" [23].

3 The Clarity Smart Contracting Language

The Stacks blockchain supports the launching and execution of smart contracts for programmatic control of digital assets. This new smart contracting language, called *Clarity* optimizes for security and predictability, which informed some key design goals that differentiate it from prior smart contracting systems:

1. The language must readily permit fast and accurate static analysis for runtime and space requirements. To support this, the language is non-Turing complete over the execution of a single transaction. However, when taken over the entire history of transactions, the language is Turing complete.
2. Smart contracts should be *interpreted* by our VM, not compiled. The code, as written by developers, must be deployed directly on the blockchain.

To achieve the above two properties, we created a new LISP-variant, specially designed for the writing of smart contracts. For a more detailed discussion of the design of Clarity, see SIP-002 [24].

3.0.1 Language Overview.

Clarity is similar to other LISP-variants (e.g., Scheme), but with the following differences:

1. Recursion is illegal and there is no `lambda` function.
2. Looping may only be performed via `map`, `filter`, or `fold`
3. The only atomic types are booleans, integers, fixed length buffers, and principals
4. There is additional support for lists of the atomic types, however the only variable length lists in Clarity appear as function inputs (i.e., there is no support for list operations like append or join). We also support named-and-typed tuples.
5. Variables may only be created via `let` binding and there is no support for mutating functions like `set`.
6. Defining of constants and functions are allowed for simplifying code using `define` statements. However, these are purely syntactic. If a definition cannot be inlined, the contract will be rejected as illegal. These definitions are also private, in that functions defined this way may only be called by other functions defined in the given smart contract.
7. Functions specified via `define-public` statements are public functions. Arguments to these functions must specify their types.

Smart contracts have the power to:

1. Call public functions from other smart contracts. These smart contracts are identified by their hash, and must already exist at the time the calling smart contract is published. This, paired with the illegality of recursion, prevents function reentrancy, which is a common attack vector in existing smart contracting platforms.
2. Own and control digital assets. Smart contracts are first-order principals just like public keys or multi-signature addresses.

Each smart contract has its own data-space. Data within this data-space are stored in maps. These stores relate a typed-tuple to another typed-tuple (almost like a typed key-value store). As opposed to a table data structure, a map will only associate a given key with exactly one value.

Any smart contract may fetch data from any other smart contracts maps. However, only a smart contract may directly update data within its own maps.

We chose to use data maps as opposed to other data structures for two reasons:

1. The simplicity of data maps allows for both a simple implementation within the VM, and easier reasoning about functions. By inspecting a given function definition, it is clear which maps will be modified and even within those maps, which keys are affected by a given invocation.

2. The interface of data maps ensures that the return types of map operations are fixed length, which is a requirement for static analysis of smart contracts' runtime, costs, and other properties.

3.0.2 Turing Incompleteness and Static Analysis

Creating a non-Turing complete language was an essential design consideration. There are a multitude of benefits from this for programming in the hostile environment of blockchains.

1. Turing incompleteness enables static analysis to determine the cost of executing a given transaction. This allows the network to know *a priori* exactly how much of a fee to charge a given transaction. This improves client behavior as well, because the cost of broadcasting a transaction is well-known to the client, and can be easily conveyed to the user.
2. Turing incompleteness allows static analysis to quickly determine important properties such as which other contracts a single transaction may ever invoke. This improves the user experience, because clients can warn users about any possible side-effects from a given transaction.
3. Improved, accurate, static analysis will allow programmers to confidently analyze their smart contracts for any possible faults and mistakes before those smart contracts launch.

Fundamentally, we believe that it is a mistake to treat smart contract programming like other forms of programming. The properties of blockchains make the particulars of smart contracts very important and we believe that trading off programming ease for increased human and machine comprehension of smart contract behavior is a good trade off. Existing practical uses of smart contracts bear this out— the history of Turing-complete smart contracts is basically the history of smart contract bugs.

In Clarity, static analyses run before ever broadcasting the smart contract can provide information such as:

1. The cost to broadcast a given transaction as a function of input size.
2. The sets of transactions which will be able to modify any particular table.

Future work could support even more advanced analysis features, such as the ability to automatically check proofs on smart contracting code.

3.0.3 Interpreted Languages vs. Compiled

A second key design decision in Clarity is opting for an interpreted language, rather than a compiled one (e.g., compiling to WASM). Our design decision to not use a compiler is a fundamental difference from contemporary approaches. The main reason for this design decision is the ability to reason about implementation bugs.

Implementation bugs are a fact of life, and, even with the best coding standards, they are inescapable. This applies to smart contract bugs (blockchains) just as well as it applies to other code. Smart contract bugs are more complicated to handle. Various blockchain communities adhere to the "code is law" philosophy and rules committed to a blockchain are the source of ultimate truth. Developers writing smart contracts express their intent through source code, however compilation translates their intent into the actual rules. This leads to the situation where the actual rules differ from developer intent because of bugs in the compiler. This leads to nasty situations where people argue over if the developer intent was more important or the rules are more important. In the Stacks blockchain we avoid this situation by removing the compilation step and directly committing developer intent to the blockchain so that developer intent never diverges from the rules.

Let's consider the case of a bug in the implementation of the smart contracting language (i.e., the VM). If the smart contracting language uses an interpreter, the bugfix is relatively easy to apply. All of the worlds contract code is on the blockchain itself, and you can just apply a bugfix to the interpreter, and restart the blockchain nodes to boot again from genesis (re-applying all of the transactions).

But if the smart contracting language is compiled and the bug resides in the compiler, not the VM, then the remedy is much less obvious, and therefore likely to be much more contentious. This is because a bug in the compiler can cause the generated code (which is what is ultimately broadcast on the blockchain) to behave differently than the intended code from the developer. With the "code is law" philosophy seen in crypto communities this situation is more complicated. The code written by the developer is correct, but the generated transactions on the blockchain itself are wrong. It's not realistic to collect every developer's source code and recompile it, especially when you cannot verify that the source code wasnt changed. We suspect that, in practice, in such situations the code published on the blockchain is the ultimate source of truth in most cases. In which case, developers should be reasoning about and verifying that code, not their source code. We believe that using a high-level, interpreted language is of paramount importance for ensuring correct smart contract execution.

3.0.4 Related work.

The design of the Clarity language was informed by a wide range of existing smart contracting languages. Work in other non-Turing complete languages such as Pact [25] and Simplicity [26] informed our own design choices in Clarity— notably in Clarity's non-Turing completeness. While Clarity is syntactically similar to Pact, our languages differ significantly in how they interact with stored data and handle types (making Clarity more readily analyzed by various static analyses).

4 Gaia: User-Controlled Storage

Blockstack gives users control of their data using the *Gaia* storage system, a user-controlled storage system that enables applications to interact with private data lockers. Users can host these data lockers on a cloud-provider or other data storage options like private hosting. Importantly, the user controls which provider to use. Data on Gaia is encrypted and signed by user-controlled cryptographic keys. Logically, Gaia works as a wide-area file system which can be mounted to store files.

With the Gaia storage system, users designate the location of a Gaia storage location which stores data. The Stacks blockchain (and Atlas subsystem) only stores the "pointers" to Gaia locations. When users log in to applications and services using the Blockstack authentication protocol (see Sec. 5), they pass that location to the application; with this information, applications know how to communicate with the specified Gaia storage locker such that the application stores data on storage specified by the user.

Gaia's design philosophy is to reuse existing cloud providers and infrastructure in a way that end-users don't need to trust the underlying cloud providers. We treat cloud storage providers (like Amazon S3, and Google Cloud Storage, or even just a local disk) as "dumb drives" and store encrypted and/or signed data on them. The cloud providers have no visibility into the user's data; they only see encrypted data blobs. Further, since the associated public keys or data hashes are discoverable through the Stacks blockchain, cloud providers cannot tamper with users' data.

Writing data to a Gaia hub involves `POST`-ing it to the appropriate location on that server. The hub validates these `POST`s by checking that any such write request carries a signed *authentication* token. This token is signed by the private key which controls the particular bucket being written to. To provision separate buckets for each application a user might use, the user derives different private keys for each of those applications. Each of these private keys only grants access to specific buckets on the Gaia server.

In Gaia, the user's blockchain-verified routing information contains a URL that points to a signed JSON object (signed by the owner key of the username). This signed JSON object contains URLs that point the user's Gaia data locker. Once an application knows the location of the user's Gaia data locker, they simply request a file from that location using a standard HTTP request. To look up a file created by a *different* user, an application can perform these sequential lookups entirely client-side. While this imposes a latency penalty on initial lookups, much of this routing information will be cached locally by browsers (or native applications) such that subsequent lookups will be as fast as any traditional data fetches on the internet.

Figure 1 shows an overview of Gaia. Looking up data for a name, like `werner.id`, works as follows:

1. Lookup the $name$ in the Stacks blockchain to get the $(name, hash)$ pair.
2. Lookup the $hash(name)$ in Blockstack's Atlas peer network to obtain the name's routing information file.

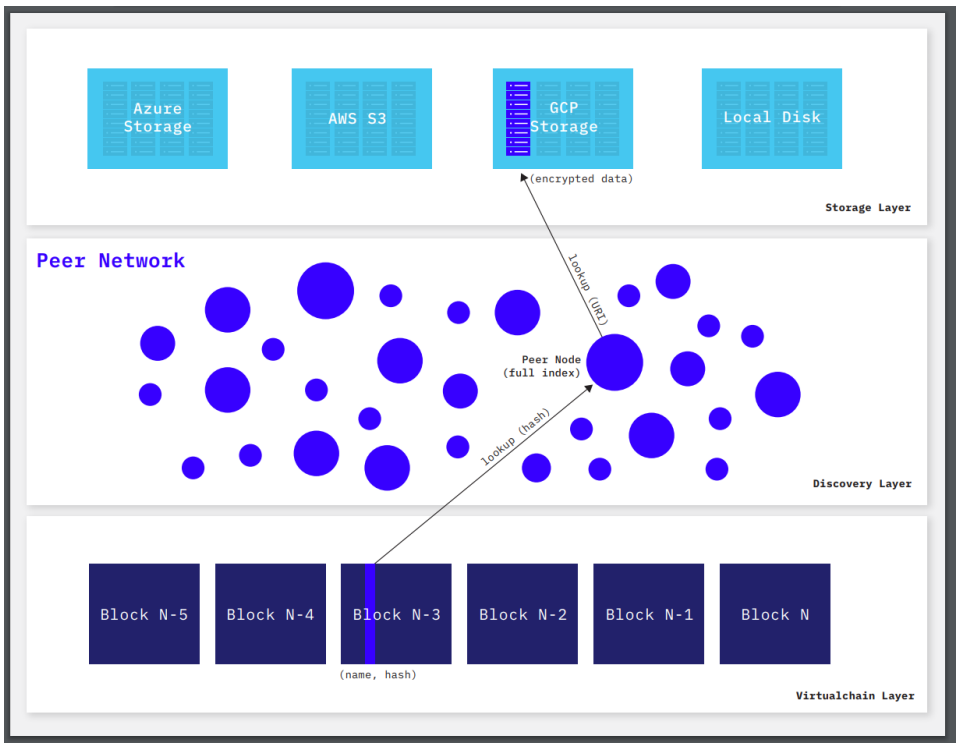

Figure 1: Overview of Gaia and steps for looking up data.

3. Get the user's Gaia URL from the routing file and look up the URL to connect to the storage backend.
4. GET/PUT data from the designated Gaia service (decrypt it if needed and if the reader has access rights) and verify the respective signature or hash.

Steps 1 and 2 above are performed with a single call to `blockstack-core` at the `/v1/names/<name>` endpoint. These iterative reads and writes are handled automatically in our developer libraries.

Performance. The goal of our architecture is to give comparable performance to traditional internet applications built on cloud providers. We introduce meaningful security and fault-tolerance benefits by removing central points of control and failure—paying a small overhead on read/write performance is worthwhile so long as the overhead is not significant nor noticeable to average users. We evaluated the performance of reads and writes of Gaia to demonstrate that it reads and writes files at competitive rates with the underlying storage. Gaia adds a negligible constant storage space overhead per file due to encryption (roughly 5% larger files). There is a CPU overhead for encryption, but because the file size difference is minimal, the network performance for reads/writes is similar to directly accessing the underlying storage service.

System Scalability. The storage layer of our architecture is not a scalability bottleneck. Contemporary cloud storage systems are highly scalable [27]. The Atlas subsystem also scales well because it does not index individual user files or file-chunks but indexes pointers to user's storage backends. The storage backends deal with the bulk of data read/writes, and the Atlas network is involved only when (a) a user is changing or updating her storage backends or public key mappings, or (b) new users are registered on the system. When registering new domains/usernames, routing file hashes must be announced on the Stacks blockchain. While the blockchain could be a bottleneck on scalability (relative to the Atlas subsystem), it is written to extremely infrequently for any user. Further, the use of *off-chain* name registrars enables over a hundred users to register in a single blockchain transaction, which could support hundreds of thousands of user registrations per day (comparable to the number of new users per day on traditional cloud-based platforms). Scaling Gaia to billions of users in practice will likely uncover scalability issues that are not obvious right now, and addressing these challenges is an area of ongoing research and future work.

5 Authentication

User accounts are essential to using internet applications. Blockstack provides users with a universal username that works across all applications without the need for any passwords. Instead of password-based authentication, users authenticate via public-key cryptography: a locally-running software client handles sign-in requests from respective applications and signs authentication requests.

Our authentication protocol, Blockstack Auth, also connects apps with the user's Gaia hub and any app-specific private keys. This information is used by applications to store user data with the users and verify that data produced by other users is authentic.

5.1 Single Sign-On

Blockstack Auth uses public-key cryptography for authentication. The user signs into an application to give the app the ability to generate and store signed data, which other users can read and authenticate. This, in turn, proves to other users that the signed-in user is legitimate.

In Blockstack, the purpose of sign-in is to provide the application client with enough information to generate and store authentication data. This means that the auth functionality *can run solely on the user's computer* in the form of an authenticator app. Because all names are registered on the Stacks blockchain, each and every application and authenticator app always has an up-to-date view of (1) all names that exist, and (2) all of their public keys and Gaia hubs. This obviates the need for a server-side identity provider.

An application client only needs to be able to contact a Stacks blockchain peer to authenticate user data. The user provides the application with the network address of

their preferred Stacks peer on sign-in to do this.

A user signs into a Blockstack application by clicking a "login" button. The application (via the *blockstack.js* SDK) redirects the user to their Blockstack authenticator app with a request to sign in. The user is presented with the choice of Blockstack IDs to use to sign in, as well as a list of which permissions the application needs from the user. Upon selecting an ID, the authenticator directs the user back into the application and passes the application three pieces of information:

1. The user's username (or the hash of their public key if they do not yet have a name).
2. An *application-specific* private key for encrypting and signing the user's data. This key is deterministically generated from the user's master private key, the ID they used to sign in with, and the application's HTTP Origin.
3. The URLs to the user's Gaia hub and preferred Stacks blockchain peer to use for looking up other users and their data.

In doing so, the user presents their username and instructs the application as to where their data can be found and stored. From there, the app can both read and write application-specific data persistently, and access other users' application-specific data—all without needing to provide its own storage or identity solution.

The act of signing out is simply to clear the application's local state, thereby causing the Web browser and the client to forget the application-specific private key.

6 Blockstack Libraries & SDKs

Blockstack PBC, a Public Benefit Corp, along with open-source contributors develop the core protocols and developer libraries for Blockstack. The developer libraries make it easier for developers to build applications on the Blockstack network and Blockstack clients allow users to interact with the various components of the Blockstack network and the various applications.

6.1 Developer Libraries

Blockstack is designed to make developing decentralized apps as easy as possible for developers. Most of the complexity of interacting with the Stacks blockchain or decentralzied storage is hidden from app developers and they can focus on just their app logic. The Blockstack open-source repositories contain developer libraries for a number of different platforms: a Javascript Web SDK (`blockstack.js`) and mobile SDKs for iOS and Android. All of these libraries are provided under the terms of the MIT license, and available via `https://github.com/blockstack`.

These libraries provide all the necessary APIs and code for implementing our authentication protocol, directly interacting with Gaia servers, and generating Stacks

transactions. Using these libraries allows developers to create decentralized applications that respect user's security and privacy as easily as if they were developing traditional applications.

Radiks For applications that wish to share data across complex social graphs, it is often useful and most efficient to build *indexes* over that data. The Radiks system is a server and client library for building and interacting with such an index. The Radiks libraries enable developers to create cross-user structured data collections within the app that can be queried by field values. This requires a server-side component that processes indexes and queries, but crucially, is *not* part of users' trusted computing base. It only sees the data ciphertext and some metadata necessary for building and answering queries over the index. More information on Radiks is available at [28].

6.2 User Software

While application developers will use SDKs and libraries to interact with the Blockstack network, users also require software to perform functions like registering usernames, designating their Gaia servers, and authenticating with applications. The Blockstack ecosystem currently provides two open-source projects that allow users to interact with the network:

1. **Blockstack Browser.** This is currently the reference open-source implementation of an authenticator app, and it allows users to browse through available Blockstack applications, register usernames, and authenticate with applications. It is available for installing locally on desktops, as well as a web-deployed form.
2. **Blockstack CLI.** This is a command-line utility that allows power users and developers to interact with Blockstack's protocols. In addition to providing authentication functionality, it allows users to generate raw transactions and engage in advanced data management tasks with Gaia.

6.3 Documentation and Community Resources

The Blockstack open-source community maintains tutorials, API documentation, and system design documents which are available on Github and at `https://docs.blockstack.org`.

Blockstack users and developers have the following official community resources available to them.

- **Github**: All software development takes place through Github at `https://github.com/blockstack`.
- **Forum**: Power users and developers often answer technical questions, share ideas, and help each other on the Blockstack Forum available at `https://forum.blockstack.org`.

- **Slack**: Blockstack community uses a public Slack group for real-time chat available at `https://blockstack.slack.com`.

7 Apps and Services

As of early 2019, there are more than 100 applications built on Blockstack. Developers are building various different types of applications and a full listing of the growing number of Blockstack applications can be found at `app.co`. Because Blockstack is modular, different applications can make use of different components independently. Below we give a brief overview of some example uses cases.

Current office productivity applications on Blockstack [29, 30, 31, 32] use Blockstack Auth and Gaia storage to enable users to create, edit, and share documents. To help users discover each other's documents, these apps make use of a Blockstack profile search indexer. This search indexer is decentralized— because the set of profiles is globally visible and discoverable, anyone can deploy and run a profile search indexer.

The Blockstack ecosystem also contains a number of social applications [33, 34, 35, 36]. These typically use Blockstack Auth in combination with a Radiks server deployment to enable users to efficiently discover and fetch other users' data. In at least one case, the app uses a dedicated relay channel to route encrypted messages across many users [33]. Publishing and storage applications on Blockstack [37, 38, 39, 40, 41, 42], use Gaia not only to store user data, but also to share it with non-Blockstack users via conventional HTTP URLs.

Developer Incentives The Stacks blockchain broadens the concept of mining where app developers can "mine" Stacks tokens by publishing high-quality application on the network. This mechanism, called App Mining, is designed as an incentive mechanism to get high-quality applications on the network. The App Mining program is currently operated by Blockstack PBC with several independent reviewers. Developers can submit their application to be reviewed once a month and receive a payout based on how well the app performs in the app ranking mechanism. The applications are reviewed by a set of independent reviewers, each of which have their own criteria for what makes a good application. The application's *aggregate score* determines its ranking. The rankings and payouts are published monthly. Details on the App Mining program are outside the scope of this paper and readers should see `https://app.co/mining` for more details.

8 Conclusion

Blockstack is a decentralized computing network that provides a full-stack to developers for building decentralized applications. To date, over 100 decentralized applications have been built on the network. Blockstack removes the need for developers to

run servers and databases: apps write data to user-controlled private data lockers instead. This decentralized storage system gives comparable performance to traditional cloud storage and only introduces a small overhead for encryption/decryption. Our authentication protocol removes the need for password-based logins which are known to be less secure than cryptographic authentication. Users are able to use a single account across services and applications, removing the need to continuously create new accounts new services. Our developer libraries make the development of decentralized apps on this platform as easy as building traditional internet applications.

In this paper, we presented the latest design of Blockstack. Since earlier production implementation in 2016 and 2017, the core design of Blockstack has evolved and incorporates lessons learned from production deployments and feedback from developers of decentralized applications. The main changes from the earlier (2017) whitepaper include (a) description of the Stacks blockchain which uses a new Tunable Proofs mechanism to security bootstrap a new blockchain, and (b) description of a new smart contract language that focuses on security and predictability of smart contracts. We've released Blockstack as open-source [43].

Acknowledgements

Over the years many people have contributed to the design and implementation of Blockstack. We would like to thank Larry Salibra, Ken Liao, Guy Lepage, Patrick Stanley, and John Light for their early contributions and ideas. Hank Stoever, Shreyas Thiagaraj, and Matthew Little for their work on developer libraries and SDKs. Jeff Domke, Mark Hendrickson, Thomas Osmonson, Jasper Jansz, and Mary Anthony for product designs and developer docs; Jesse Wiley, Virginia Hickox, and Tim Wells for infrastructure development; Brittany Laughlin and Diwaker Gupta for their helpful comments and feedback. More Blockstack contributors can be found at `https://github.com/blockstack` but in general so many people have contributed to this project that it's not realistic to explicitly thank them – we're grateful for the support of the entire Blockstack open-source community.

References

[1] S. Sulyman, "Client-server model," *IOSR Journal of Computer Engineering*, vol. 16, pp. 57–71, 01 2014.

[2] N. Perlroth, "Yahoo says hackers stole data on 500 million users in 2014," Sept. 2016. `http://nyti.ms/2oAqn0G`.

[3] K. Granville, "Facebook and cambridge analytica: What you need to know as fallout widens," Mar. 2018. `https://nyti.ms/2HP4Dr3`.

[4] R. Mcnamee, "I mentored mark zuckerberg. i loved facebook. but i can't stay silent about what's happening.," Jan. 2019. `http://time.com/5505441/mark-zuckerberg-mentor-facebook-downfall/`.

[5] "Blockstack website," 2019. `http://blockstack.org`.

[6] J. H. Saltzer, D. P. Reed, and D. D. Clark, "End-to-end arguments in system design," *ACM Trans. Comput. Syst.*, vol. 2, pp. 277–288, Nov. 1984.

[7] D. D. Clark and M. S. Blumenthal, "The end-to-end argument and application design: The role of trust," *Federal Comm. Law Journal*, vol. 63, no. 2, 2011.

[8] M. Ali, J. Nelson, R. Shea, and M. Freedman, "Blockstack: A global naming and storage system secured by blockchains," in *Proc. USENIX Annual Technical Conference (ATC '16)*, June 2016.

[9] J. Nelson, M. Ali, R. Shea, and M. J. Freedman, "Extending existing blockchains with virtualchain," in *Workshop on Distributed Cryptocurrencies and Consensus Ledgers (DCCL'16)*, (Chicago, IL), June 2016.

[10] M. Ali, J. Nelson, R. Shea, and M. J. Freedman, "Bootstrapping trust in distributed systems with blockchains," *USENIX ;login:*, vol. 41, no. 3, pp. 52–58, 2016.

[11] Satoshi Nakamoto, "Bitcoin: A peer-to-peer electronic cash system," tech report, 2009. https://bitcoin.org/bitcoin.pdf.

[12] A. Narayanan, J. Bonneau, E. Felten, A. Miller, and S. Goldfeder, *Bitcoin and Cryptocurrency Technologies: A Comprehensive Introduction*. Princeton, NJ, USA: Princeton University Press, 2016.

[13] V. Buterin, "A next-generation smart contract and decentralized application platform," tech. rep., 2017. https://github.com/ethereum/wiki/wiki/White-Paper.

[14] "Filecoin: A Cryptocurrency Operated File Network," tech report, 2014. http://filecoin.io/filecoin.pdf.

[15] https://eos.io.

[16] T. Hanke, M. Movahedi, and D. William, "Dfinity technology overview series consensus system rev.1," 2018. https://dfinity.org.

[17] "Ethereum 2.0 specifications," 2019. https://github.com/ethereum/eth2.0-specs.

[18] Eng Keong Lua, J. Crowcroft, M. Pias, R. Sharma, and S. Lim, "A survey and comparison of peer-to-peer overlay network schemes," *IEEE Communications Surveys Tutorials*, vol. 7, pp. 72–93, Second 2005.

[19] "Oauth." https://oauth.net.

[20] "Blockstack Core: Stacks blockchain v1," 2018. https://github.com/blockstack/blockstack-core/tree/v20.0.8.1.

[21] "SIP 001: Burn Election," 2019. https://github.com/blockstack/blockstack-core/blob/develop/sip/sip-001-burn-election.md.

[22] I. Stoica, R. Morris, D. Karger, M. F. Kaashoek, and H. Balakrishnan, "Chord: A scalable peer-to-peer lookup service for internet applications," in *Proceedings of the 2001 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications*, SIGCOMM '01, pp. 149–160, 2001.

[23] "App mining." https://app.co/mining/.

[24] "SIP 002: Smart Contract Language," 2019. https://github.com/blockstack/blockstack-core/blob/develop/sip/sip-002-smart-contract-language.md.

[25] S. Popejoy, "The Pact Smart-Contract Language," June 2017. https://kadena.io/docs/Kadena-PactWhitepaper.pdf.

[26] R. O'Connor, "Simplicity: A New Language For Blockchains," December 2017. https://blockstream.com/simplicity.pdf.

[27] "Google Cloud Storage SLA." Retrieved from https://cloud.google.com/storage/sla in May 2017.

[28] "Radiks." https://github.com/blockstack-radiks.

[29] J. E. Hunter, "Graphite docs," Feb. 2019. https://app.graphite-docs.com.

[30] D. Travino, "Noteriot," Feb. 2019. https://note.riot.ai/.

[31] "Forms.id," Feb. 2019. https://forms.id.

[32] "Blockusign," Feb. 2019. https://blockusign.io.

[33] P. Bhardwaj and A. Carreira, "Stealthy," Feb. 2019. https://www.stealthy.im.

[34] A. Sewrathan, R. Adjei, and F. Madutsa, "Afari," Feb. 2019. https://afari.io.

[35] T. Alves, "Recall," Feb. 2019. https://app.recall.photos/.

[36] T. Alves, "Travelstack," Feb. 2019. https://app.travelstack.club.

[37] J. E. Hunter, "Graphite publishing," Feb. 2019. https://publishing.graphitedocs.com.

[38] "Decs," Feb. 2019. https://app.decs.xyz.

[39] "Sigle," Feb. 2019. https://app.sigle.io.

[40] "Xorbrowser," Feb. 2019. https://xorbrowser.com.

[41] "Mevaul," Feb. 2019. https://mevaul.com/.

[42] "Xordrive," Feb. 2019. https://xordrive.io.

[43] "Blockstack source code release v20.0.8," 2019. http://github.com/blockstack/blockstack-core.