The Keep Network:
A Privacy Layer for Public Blockchains

Matt Luongo
mhluongo@gmail.com

Corbin Pon
corbin.pon@gmail.com

Abstract

We introduce the keep, a new privacy primitive for developing smart contracts on public blockchains, enabling secure storage and usage of secrets, as well as supporting infrastructure, including the keep market and token.

Our incremental approach to privacy infrastructure can be brought to market on the Ethereum public network, iterated on, and adapted for other public blockchains and cross-blockchain use.

1 Motivation

1.1 The irony of public blockchains

Public blockchains have brought unprecedented transparency and auditability to financial technology. Records are immutable, verifiable, and censorship-resistant.

Unfortunately, these strengths are also weaknesses for many potential users.

For every financial use case a public blockchain enables, its public status restricts another. Bitcoin is touted as a more private payment method than the traditional financial system, but those familiar with the technology know that while it may be censorship-resistant, it's certainly not private by default [1]. Developers introduced to Ethereum quickly learn to adjust their expectations [2]- all contract state is published to the blockchain, and can be easily read by competing interests.

These issues are recognized by developers of the Bitcoin and Ethereum projects.

Confidential transactions [3] is an ongoing effort to bring better privacy, and therefore fungibility, to Bitcoin via sidechains [4]. The Zerocash project [5] applied zero-knowledge proofs to Bitcoin, leading to the creation of Zcash [6], a cryptocurrency using zk-SNARKs to ensure transaction privacy.

As early as December 2014, Vitalik Buterin, one of the founders of Ethereum, explored solving this problem with secure multi party computation (sMPC) [7]. In more recent writing, Buterin shares that "when [he] and others talk to companies about building their applications on a blockchain, two primary issues always come up: scalability and privacy" [8].

Scalability of public blockchains is a hurdle to mainstream adoption. Some of the best minds in the cryptocurrency space [9] [10] [11] are working on multiple order-of-magnitude improvements. Privacy, however, hasn't garnered the same attention, especially in smart contracts.

Basic use cases of smart contracts, including publishing secrets after certain criteria are met, assessing borrower risk for a loan, and signing messages and transactions, are incredibly difficult on today's public blockchains.

1.2 Existing approaches

In practice, developers have found a number of ways to build decentralized applications that use private data.

1.2.1 The hash-reveal pattern

A common pattern on public blockchains is to keep private data with the application's users. Contracts can receive and manipulate hashes of private data, more generally called commitments [12], while users withold the original until revealing the private data off-chain. We call this the "hash-reveal" pattern.

For many applications, this approach is satisfactory. There's a clear privacy benefit over typical web applications- no centralized third-party database is at risk. Spreading storage across many users means a distributed, diverse

target for attackers.

There are significant downsides, however. The hash-reveal pattern requires that all users party to a transaction be online, monitoring the system, providing private data when necessary, and validating hashes against private data provided by other users.

This requirement makes the hash-reveal pattern inflexible for complex protocols, and unsuitable for systems that don't revolve solely around active human participants, like decentralized autonomous organizations (DAOs).

1.2.2 Private blockchains

Another response to privacy restrictions, primarily from the finance industry, has been to build private blockchains, or so-called "permissioned ledgers".

These systems operate in a trusted or semi-trusted manner. Instead of using proof-of-work or other consensus mechanisms designed with an adversarial network in mind, they can use systems like RAFT to reach consensus.

One such system, J.P. Morgan's Quorum [13], is a fork of Ethereum supporting private contract state and messaging between network participants. Another, Microsoft's recently announced Coco Framework [14], provides data privacy atop an existing private blockchain.

These systems solve privacy at the expense of many of the benefits of a public blockchain- trustlessness, public accountability, censorship-resistance, and permissionless innovation.

1.2.3 Zero-knowledge proofs

Zero-knowledge proofs have been leveraged to maintain privacy on public blockchains- most famously, by the Zcash [6] project.

Zero-knowledge proofs allow one party, the prover, to prove a statement to another party, the verifier, without revealing the knowledge used to prove that statement. For example, a prover could show that they have access to a private key by encrypting a message chosen by a verifier. The proof can be trivially checked by the verifier by decrypting the cyphertext with the public key. The private key, however, remains secret.

More relevant to the domain, zero-knowledge proofs can be used for a party to prove they have access to funds, or in the case of Zcash, for a party to prove to miners that a transaction is valid according to the consensus rules of the network.

Zero-knowledge proofs can be used to construct private financial systems on a public blockchain. On their own, however, they stop short of allowing safe delegation of private data from one party to another, and suffer the same always-online requirements of the "hash-reveal" pattern.

Zero-knowledge proofs are a powerful cryptographic tool, and can be used in conjunction with other techniques to safely delegate secret access and computation (see section 3.1).

1.3 Public applications, private data

None of these techniques adequately address how to build a publicly verifiable, decentralized, censorship-resistant application that makes use of private data.

Consider contracts to reveal a secret in case of a dispute between two parties, to sign a message verifying contract identity off-chain, or to securely encrypt files [1].

2 Introducing keeps

To solve this mismatch between the transparency of public blockchains, and the need of many autonomous smart contracts for private data, we introduce the *keep*.

A keep is an off-chain container for private data. Keeps allow contracts to manage and use private data without exposing the data to the public blockchain.

2.1 Keep operations

Though keeps maintain state off-chain, they are provisioned and messaged by contracts on-chain. We will describe the keep in terms of these on-chain operations. The practical implementation of keeps, including security guarantees, is covered in sections 3 and 4.

2.1.1 Creation and population

A contract, $Contract_{owner}$, requests a keep by publishing a request to the blockchain. Once a keep, $Keep_{accepted}$, has accepted a request and finished initializing off-chain, it will respond to

[1] We go over applications in more depth later in section 8

Keep operations	
Create:	$Contract_{owner}$ publishes a creation request, including an initial deposit and a public key, $K_{Contract_{owner}}$.
Accept:	A keep, $Keep_{accepted}$, publishes one or more public keys $K_{Keep_{accepted_i}}$ signalling readiness.
Populate:	$Contract_{owner}$ publishes an initial secret on-chain, encrypted in total or in shares by one or more $K_{Keep_{accepted_i}}$, or a specification for a secret to be generated.
Grant:	$Contract_{owner}$ publishes another contract address, $Contract_{delegate}$, and a permission level, P_{read} or P_{admin}.
Compute:	$Contract_{owner}$ or $Contract_{delegate}$ publishes a function to compute over the secret, $F(S, ...)$, as well as other arguments to F. Initially $F \in \{f_{identity}, f_{rsa}, f_{ecdsa}\}$, though additional functions are planned.
Results:	$Keep_{accepted}$ publishes the results of its computation, either in whole or in part, over one or more invocations.
Shutdown:	$Contract_{owner}$ or $Contract_{delegate}$ with permission P_{admin} publishes a shutdown request.

the request with a set of public keys the calling contract can use to communicate privately with the keep.

Once the keep has been created, it can be populated in a number of ways. dApps can publish secret data to the blockchain, encrypted by the keep's public keys, or send the data to the keep off-chain. Alternatively, a keep can self-populate with pseudorandom data.

2.1.2 Publishing data on-chain

The purpose of a keep is to compute a function over its secret and publish the results to the blockchain.

Initially, keeps will support publishing their secrets on-chain, unmodified or encrypted with a public key provided by $Contract_{owner}$. This enables functionality that's difficult in today's public smart contracts, like a secret-exposing dead man switch, useful in a variety of decentralized market schemes.

Keeps can be extended to use their secret in a variety of other ways, including as key material for symmetric encryption and signing.

2.1.3 Access management

The owning contract $Contract_{owner}$ of a keep can delegate access to the keep to other contracts.

Read and admin access can each be granted, allowing another contract i($Contract_{delegate}$) to request that a keep's content be published (read permission, P_{read}), or to delegate further access to other contracts (admin permission, P_{admin}). Owners ($Contract_{owner}$) can also revoke their own access.

Access management enables multi-party secret escrow and auditability of secret access.

2.1.4 Destruction

Depending on the use case, keeps can be long- or short-lived. Contracts can request that a keep shut down, and should also handle keeps that are terminated unexpectedly, scenarios which are covered in more detail later in section 5.2.

3 Eliminating third-party risk

We've described a simple black box for off-chain data storage. The standardization of this method of secret management will enable secrets to be bought, sold, and transferred on a public blockchain, but doesn't inherently solve third-party risks.

Next, we'll describe techniques to eliminate third-party risk.

3.1 Secure multi party computation

Secure multi party computation (sMPC) is a type of cryptographic system where a computation is distributed across multiple participants, some of which may be dishonest. Each participant is initially given access to a share

of a secret by a dealer, and computes a function over that share. The outputs are then reported to the dealer, who can assemble the final output, without any participant learning more than their initial secret share.

Intuitively, sMPC works like this:

1. A dealer D wants to compute a function F over a secret, S.
2. The dealer selects n parties to the computation, sending each of them a share of the secret, s_i.
3. Each party computes a function over their share $f_i(s_i)$ and reports the result to the dealer.
4. The dealer combines these outputs, such that $G(f_1(s_1), f_2(s_2), ...f_n(s_n)) = F(S)$

The shares s_i should be chosen in such a way that exposing any share does not jeopardize the secret S. A common approach is to use Shamir's secret sharing [15], such that details about the secret remain confidential in the face of $n-1$ dishonest parties.

This explanation holds for all F including addition, subtraction, and multiplication by a known constant. To achieve general computation, however, we also need to be able to multiply secrets securely.

Multiplication adds what the literature calls "rounds"- communication between the parties, rather than just the dealer D.

To multiply two secrets, each party P_i of the n chosen by the dealer splits its share, s_i, into two components, s_{i1} and s_{i2}. The party multiplies those two components, resulting in $s_{i'}$. Each P_i then acts as a dealer among the the remaining parties, splitting $s_{i'}$ into $n-1$ pieces.

Each P_i can now resolve their resulting share of the round of multiplication, s'_i, given their access to $n-1$ shares of $s_{i'}$.

With addition and multiplication, sMPC can securely execute general computation, at the expense of communication overhead between the computing parties.

3.2 sMPC and the blockchain

sMPC was originally conceived in 1982 [16], but its practical application has been limited due to restrictions on the security model. Existing sMPC solutions only maintain security in the face of an honest majority of parties.

The advent of the blockchain enables secure usage of sMPC in adversarial scenarios. By using a public blockchain as an immutable ledger, sMPC can be made secure in the face of a dishonest supermajority [17], and, with the requirement of a network token, can be made strongly Sybil-resistant (see section 5).

For these reasons, sMPC and blockchains are a natural fit. In the smart contract space, sMPC has been proposed before as a privacy mechanic.

In 2014, Vitalik Buterin gave a strong introduction to the subject in an early blog post on privacy on the Ethereum public blockchain [7]. In 2016, a team from UMD designed Hawk [18], a system that marries public and private smart contracts via sMPC, and the Enigma project out of MIT describes a system related to ours [19], with a wider focus on general private computation.

The Keep network will incorporate these ideas into the first production-ready sMPC system for a public blockchain.

4 Keep providers

The Keep network includes a number of different provider types, each with their own strengths and tradeoffs. The most important provider, however, is a novel application of secure multi party computation.

4.1 Simple sMPC

Simple sMPC keeps are backed by n nodes, each of which maintain a share of the provided secret, such that the secret can't be reconstructed without all n nodes colluding.

These keeps can be populated securely by divvying up a secret into shares via Shamir secret sharing [15], and encrypting each share with its respective node's public key. The encrypted shares can then be published to the public blockchain, or communicated off-chain.

The only computation these keeps will run is an implementation of distributed RSA [20] on sMPC, used to publish encrypted data to the blockchain.

4.2 Signing sMPC

The next provider will extend the sMPC keep with two new operations- securely generating pseudorandom numbers, and signing and encrypting data, using the keep's contents as a key.

In addition to simple pseudorandom numbers, signing keeps will be able to generate RSA [20] and Bitcoin [21, 22] key pairs, or be populated with them via secret sharing.

This means signing keeps will be able to sign and secure contract communications on- and off-chain, as well as sign transactions for Bitcoin, Ethereum, and other cryptocurrencies.

Finally, signing keeps can act as pRNG oracles, significantly improving current methods of random number generation on public blockchains.

4.3 Future providers

The off-chain keep pattern is flexible enough to include a variety of other pluggable providers, each with their own unique benefits.

4.3.1 Secure hardware

Keeps backed by secure hardware can be used to lower the cost of securing private data by verifying that only signed code is run against privileged data.

Instead of requiring n nodes to safely split a secret, a secret can be sent to a single node that's properly responded to a challenge, proving it's running signed code. Not only are fewer nodes required, but these keeps wouldn't suffer the computation overhead of secure multi-party computation.

This sort of security is fundamentally weaker than that provided by secure multi-party computation. If a single secure hardware manufacturer is compromised, it puts all nodes using that hardware at risk, shifting the threat model. The cost and benefit of this approach will depend on the application.

4.3.2 Private smart contracts

Unlike related work on systems like Enigma [19] or Hawk [18], which use sMPC to build off-chain and alternative-chain computation networks for private smart contracts, we've chosen to restrict the initial sMPC keeps to generating, securing, storing, encrypting, and transmitting secrets. Such restrictions help to minimize the attack surface on keeps in a production network.

In later work, sMPC schemes can be used to build more feature-rich keeps. These keeps will enable complex use cases, like operating private ledgers against public blockchains, or running third-party code trustlessly on private data.

5 Incentivizing keep providers

Providers need to be incentivized to maintain capacity on the network. Running and securing keeps should be a profitable way to use excess compute and storage resources.

Consumer contracts, on the other hand, need keeps that will provide:

- High availability
- Robustness against data loss
- Maintenance of confidentiality
- Data integrity

5.1 Paying for keeps

The best payment structure for keep providers will reward highly available keeps, and punish poor performance.

¡sequence diagram of deposit + per-operation payment¿

The two primary costs providers incur are storage and compute, which map naturally to paying keeps per block and per operation.

Payment per block can be accomplished via a deposit to the managing contract at the time of keep initialization, metered out over the lifetime of the keep, and refilled occasionally by the calling contract. Though this seems like a good fit for payment channels, minimizing on-chain fees, the security ramifications differ from typical two-party channels. These differences are discussed further in the next section.

Payment per operation is simpler. Each request to publish a keep's contents will require payment of an amount agreed to at the initialization of a keep.

5.2 Concerns with uptime and reliability

Because availability is vital to using keeps in practice, improper termination must be disincentivized.

¡proper shutdown protocol¿

Any keep that doesn't respond properly within a certain block count threshold to a request will be considered aborted. Aborted keeps will forfeit all client deposits that have yet to be disbursed. To avoid skewing client incentives, the deposits that have been earned, but not yet disbursed, will be burned, and the unearned deposits will be returned to the client.

Volatility in the crypto currency markets can provide a strong incentive for a keep provider to improperly terminate a keep. If the value of the paid currency drops significantly relative to the cost of running a keep, it's in a provider's best interest to devote their limited resources to a better-paying client.

To counter this issue, keep providers will need a protocol to optionally re-negotiate fees for a running keep.

5.3 Concerns with active attacks

Existing open-source sMPC frameworks, such as VIFF [23], are secure against active attacks in the presence of a $\frac{3}{4}$ supermajority of honest nodes. In such an attack, keeps can be forced to return malformed data, but secrets can't be compromised unless all nodes with a unique share backing an sMPC keep are colluding- an extremely high bar for a Sybil attack.

Recent approaches using SPDZ proofs [17] anchored on the blockchain [24, 25] make such correctness attacks impossible, even if all nodes backing a keep are compromised. sMPC keeps will publish proofs to the public blockchain that can be used to verify correctness. The threat of active attacks is then reduced to disrupting keep availability, rather than returning malformed data.

We address the issue of network disruption by introducing two incentives to keep providers, making active attacks on data availability impractically expensive.

First, keep providers will be required to prove their holdings in a token native to the system. Significant disruption of the network should lead to a drop in the value of the token, incentivizing provider honesty, lest they devalue their holdings. This scheme also provides resistance to Sybil attacks— an active attacker would need to obtain an outsize portion of all tokens locked up by keep providers to ensure their overwhelming selection backing new keeps.

Second, keep redundancy can be used to further minimize availability disruptions [25]. All nodes can be required to include a deposit when they publish their results. If their results can't be verified by the included SPDZ proof, their deposit is forfeit to competing nodes.

6 High-level network design

Deploying sMPC-based privacy on a public blockchain requires supporting infrastructure. To build a functional privacy network against Ethereum, our first target blockchain, we'll introduce components to ensure fair keep node selection, report results, and incentivize network actors.

6.1 The Keep network token

The native network token, *KEEP*, will be required for providers to participate.

To be chosen to provide a node for a new keep, a provider must lock up a minimum stake in KEEP tokens, using a shared staking contract.

At any time, a provider can choose to retrieve their stake— for example, to liquidate their position. All withdrawals, however, will be subject to a two-week waiting period to disincentivize providers from quickly staking and withdrawing their position, which could have adverse effects on running keeps and fair keep selection.

Requiring a native token, rather than the underlying blockchain's currency, means providers will suffer from negative externalities in the presence of malicious behavior (see section 5.3). This sort of staking also strengthens the system against Sybil attacks (see section 6.2).

6.2 Ensuring fair keep selection

Contracts requesting keeps and keep providers need to be matched. An ideal system would enable price discovery, incentivizing new providers to join if capacity is low, across different keep types.

This matching problem is a great fit for a market. Unfortunately, on-chain markets are a difficult problem, prone to complexity, miner frontrunning, and orderbook manipulation. A clever attacker could manipulate a

market, giving them an unfair advantage to be chosen for a particular keep. Essentially, a two-sided market would expose the network to Sybil attacks.

In lieu of a market, we need a fair keep selection mechanism.

6.2.1 Random beacons

The best way to select providers for a new keep is with a fair coin toss. Unfortunately, Ethereum only supports deterministic functions. Contracts that require a random number often rely on a trusted oracle.

A system is only as decentralized as its most centralized component. Relying on a trusted third party for such a core function of the project isn't an acceptable risk.

Instead, we can utilize our keep providers as a decentralized source of entropy. All staked providers can be required to take part in the random number generation process.

There are a few design considerations for such a system:

- Providers can't be allowed an unfair advantage over each other in the node selection process.
- Each block on the public chain will require at least one random number of sufficient size. Today's Ethereum block time is 25 seconds, but that will likely change significantly in the future. The RNG process needs to be fast enough to support much shorter block time, if necessary.
- RNG needs to be resilient to node failure. Failure in production means no new keeps can be created, so resilience to partitions between providers as well as against active denial of service attacks is desirable.
- While not a hard system requirement, providing the Ethereum network with a trusted source of randomness will also be a great boon to other projects.

Most distributed key generation schemes are too slow or prone to manipulation to be considered. Any scheme we choose should provide good performance, regardless of the number of participating providers. Instead, most generation schemes require rounds of communication between participants, slowing down the key generation process and providing a large surface for communication failure.

Fortunately, the Dfinity team has solved these issues with their random beacon design, based on a concept they call threshold relay [26].

6.2.2 Threshold relay

This work relies on the idea of threshold secret sharing schemes—secret sharing schemes that retain confidentiality up to some threshold t of honest actors.

Threshold signatures are a related idea. A threshold signature is a signature across n parties that requires some minimum t actively participating to sign. It's a similar idea to "multi-sig" as deployed in cryptocurrencies today.

Traditional multi-sig, however, requires a smart contract on the blockchain to validate each signature and release funds. Threshold signature schemes actually require a threshold t to construct a signature at all, removing a layer of complexity and coordination between parties.

The use of threshold signatures means a number of participating signers in a signing group can be unavailable, and the signature will still succeed in the presence of t functioning signers. This provides some of our beacon's required resilience in the face of failing or misbehaving nodes.

If threshold signatures sound familiar, it might be because they're a core functionality keeps provide. For example, a keep signing a Bitcoin transaction does so using threshold ECDSA.

A threshold relay is a way to chain threshold signatures to create a random beacon. Participants in a threshold relay form threshold groups. These groups generate new public keys that identify the group and correspond to a newly generated threshold private key, split across the participants.

As providers join the network, they will form threshold groups. These groups will then sign a piece of random data, initially provided by early network contributors, to bootstrap the relay. The resulting signature provides the random data for the next iteration, which can be verified by the rest of the network participants and rejected if invalid. Each iteration, a

Iterative threshold signatures for randomness on existing chains	
Registration:	As providers join the network, they register with at least one threshold group G_i of all groups G, generating a share of the group's private key, s_i. Threshold groups are capped at c members, and may intersect. Groups that have reached this maximum size publish their public key to the blockchain. We'll designate such groups as $G_{registered}$.
Trusted setup:	A trusted party posts a random value r_0 to the blockchain as the beacon's first output.
Bootstrapping:	$mod(r_0, \|G_{registered}\|)$ is used to select a registered threshold group, G_i, from $G_{registered}$. G_i signs r_0 and publishes the result, $r_1 = threshold(r_0, s_{0 \to t})$ where $s_{0 \to t}$ is the minimal shares necessary for the group to produce a signature. Note that $threshold(...)$ must be a deterministic signature scheme to avoid share withholding attacks leading to a biased output.
Iteration:	Each block published on the chain will include a signature from $G_{registered}$ of the random value r_i. As the chain grows, the signing threshold groups will change based on provider availability. If any group is non-responsive up to its threshold t, the group is removed from $G_{registered}$.
Failure:	Each iteration is an opportunity for a group to fail to generate a valid signature. If a group G_i fails to sign the last iteration's random value, G_{i+1} will be used instead.

new signing group is chosen by the previous iteration's random value. As all groups sign the previous iteration's value, if a signature that's chosen is invalid, the signature from the next group in line can be chosen instead.

Importantly, the threshold signature scheme needs to be deterministic to prevent individual shareholders from biasing the signature outcome in their favor. BLS signatures [27] have been used in related work.

6.2.3 Keep selection group

Our threshold relay system will be composed of keep providers seeking to be chosen to back a new keep, capturing the fees from that keep.

Each block will include a random signature, published by the nominated keep selection group. Any keeps that require new nodes will have their providers chosen randomly, using the beacon value from the last block.

In this way, we can ensure fair chances to all staked keep providers, keeping the cost of a Sybil attack high.

7 The result registry

Keeps will offer a number of methods to publish to the public blockchain. In the case where keeps publish to a smart contract provided by the keep owner, coordination is simple. In uses that don't have a natural contract to communicate with, a result registry will be provided as a default to simplify keep and owner coordination.

8 Applications

8.1 Dead man switch

A dead man switch is a device that is automatically activated in case its owner becomes incapacitated. Keeps enable a particular kind of dead man switch- publishing a secret, under certain contract conditions.

Examples of dead man switch applications with keeps include automated inheritance ("send my beneficiary my private key if I don't check in quarterly"), arbitration with time limits ("if no decision is made in 10 blocks, publish a shared secret"), as well as protection for leakers ("publish a key to these insurance files if I don't check in").

8.2 Marketplaces for digital goods

Buying and selling digital goods on public blockchains today requires settling off-chain. Keeps make marketplaces for digital goods, like audio and video files, straightforward.

Without keeps, each transfer of a private digital good requires one or more hash-reveal constructions on-chain. More complex scenarios that require escrow, arbitrators, and other parties who might need access to the transfered digital good will need n^2 on-chain transactions to maintain security. They also require each party to be online to participate.

Keeps obviate always-online requirements, and simplify the hash-reveal protocol to access management. All keep access is auditable, and participants can have access to a keep without viewing its contents, allowing further optimization.

Without an always-online requirement or complex reveal protocols, keeps can efficiently support services like iTunes on the blockchain.

8.3 Pseudorandomness oracle

Since keeps can populate themselves with random data, they can act as pseudorandomness oracles, improving on currently popular methods [28]. sMPC and other secure keeps are a good fit for decentralized lotteries and other games of chance, as well as offering a building block for other on-chain algorithms that require tamper-resistant pRNG.

This capability is an important component of advanced keep uses, like decentralized signing.

8.4 Decentralized signing service

Signing sMPC keep providers are able to sign messages, including blockchain transactions, using a generated or provided private key.

For the first time, contracts will be able to assert their identity off-chain, without requiring the recipient's awareness of blockchain state.

Consider a decentralized signing service for Bitcoin transactions. The service can participate in multi-signature transactions, only signing transactions that follow a strict set of rules, including daily spending limits and recipient whitelists.

Other uses for such a service include second-factor authentication, where a contract can answer a challenge-response protocol based on rules on the blockchain.

8.5 Custodial wallets and cross-chain trading

As a special case of a signing service, contracts can use keeps to generate their own cryptocurrency wallets, taking full custody of any received funds.

For example, a contract can generate a Bitcoin wallet, and sign Bitcoin transactions in response to receiving assets on the contract's native blockchain.

8.6 Encryption service for blockchain storage

Services like Filecoin [29] and Storj [30] are being built to provide cheap, ubiquitous storage, accessible globally, via smart contracts and traditional storage interfaces.

These services offer few privacy guarantees by default, leaving the onus of file encryption on users. Keeps can provide a private bridge to blockchain storage. By generating an AES key at keep initialization and providing off-chain data access to the keep, smart contracts can use keeps to secure files stored on decentralized services.

8.7 Banking on public blockchains

As more keep providers are developed, more applications that once required a private blockchain can be built against public networks.

Traditional finance offers many examples. Consider lending, a basic service provided by most banks.

There are a number of sensitive variables involved in the lending process. Borrower credit scores are sensitive; risk assessment is highly competitive; the terms of a loan aren't typically made public.

Keep providers that execute generic private smart contracts can protect scores and the risk assessment process, while maintaining auditability and all other benefits of a public blockchain.

References

[1] Stan Higgins. Bitcoin's privacy gets 'failing grade' in 2016 threat report. https://www.coindesk.com/bitcoins-privacy-2016-threat-report/. Accessed: 2017-08-16.

[2] Ethereum StackExchange private info on ethereum. https://ethereum.stackexchange.com/questions/2624/private-info-on-ethereum. Accessed: 2017-08-16.

[3] Greg Maxwell. Confidential transactions. https://people.xiph.org/~greg/confidential_values.txt.

[4] The Elements Project confidential transactions. https://elementsproject.org/elements/confidential-transactions/. Accessed: 2017-08-16.

[5] Eli Ben Sasson, Alessandro Chiesa, Christina Garman, Matthew Green, Ian Miers, Eran Tromer, and Madars Virza. Zerocash: Decentralized anonymous payments from bitcoin. In *Security and Privacy (SP), 2014 IEEE Symposium on*, pages 459–474. IEEE, 2014.

[6] Zcash: All coins are created equal. https://z.cash/. Accessed: 2017-08-16.

[7] Secret sharing DAOs: crypto 2.0. https://blog.ethereum.org/2014/12/26/secret-sharing-daos-crypto-2-0/. Accessed: 2017-08-16.

[8] Vitalik Buterin. Privacy on the blockchain. https://blog.ethereum.org/2016/01/15/privacy-on-the-blockchain/, 2016. Accessed: 2017-08-16.

[9] Joseph Poon and Thaddeus Dryja. The bitcoin lightning network: Scalable off-chain instant payments. 2016. https://lightning.network/lightning-network-paper.pdf Accessed: 2017-08-16, Draft Version 0.5.9.2.

[10] On sharding blockchains. https://github.com/ethereum/wiki/wiki/Sharding-FAQ. Accessed: 2017-08-16.

[11] Joseph Poon and Vitalik Buterin. Plasma: Scalable autonomous smart contracts. https://plasma.io/plasma.pdf Accessed: 2017-08-16.

[12] Commitment scheme - Wikipedia. https://en.wikipedia.org/wiki/Commitment_scheme. Accessed: 2017-09-02.

[13] Quorum. https://github.com/jpmorganchase/quorum. Accessed: 2017-08-16.

[14] The Coco Framework: Technical Overview. https://aka.ms/cocopaper. Accessed: 2017-08-21.

[15] Shamir's secret sharing. https://en.wikipedia.org/wiki/Shamir%27s_Secret_Sharing. Accessed: 2017-08-16.

[16] Andrew C Yao. Protocols for secure computations. In *Foundations of Computer Science, 1982. SFCS'08. 23rd Annual Symposium on*, pages 160–164. IEEE, 1982.

[17] Multiparty computation with spdz online phase and mascot offline phase. https://github.com/bristolcrypto/SPDZ-2. Accessed: 2017-08-16.

[18] Ahmed Kosba, Andrew Miller, Elaine Shi, Zikai Wen, and Charalampos Papamanthou. Hawk: The blockchain model of cryptography and privacy-preserving smart contracts. In *Security and Privacy (SP), 2016 IEEE Symposium on*, pages 839–858. IEEE, 2016. http://download.xuebalib.com/xuebalib.com.27267.pdf.

[19] Guy Zyskind, Oz Nathan, and Alex Pentland. Enigma: Decentralized computation platform with guaranteed privacy. *arXiv preprint arXiv:1506.03471*, 2015. https://arxiv.org/pdf/1506.03471.pdf.

[20] Atle Mauland. Realizing distributed rsa using secure multiparty computations. Master's thesis, Institutt for telematikk, 2009. https://brage.bibsys.no/xmlui/bitstream/handle/11250/261858/347875_FULLTEXT01.pdf Accessed: 2017-08-20.

[21] Rosario Gennaro, Steven Goldfeder, and Arvind Narayanan. Threshold-optimal

dsa/ecdsa signatures and an application to bitcoin wallet security. In *International Conference on Applied Cryptography and Network Security*, pages 156–174. Springer, 2016. https://eprint.iacr.org/2016/013.pdf.

[22] Jan Henrik Ziegeldorf, Fred Grossmann, Martin Henze, Nicolas Inden, and Klaus Wehrle. Coinparty: Secure multi-party mixing of bitcoins. In *Proceedings of the 5th ACM Conference on Data and Application Security and Privacy*, pages 75–86. ACM, 2015. https://www.comsys.rwth-aachen.de/fileadmin/papers/2015/2015-ziegeldorf-codaspy-coinparty.pdf.

[23] Viff, the virtual ideal functionality framework. http://viff.dk/. Accessed: 2017-08-16.

[24] Marcin Andrychowicz, Stefan Dziembowski, Daniel Malinowski, and Lukasz Mazurek. Secure multiparty computations on bitcoin. In *Security and Privacy (SP), 2014 IEEE Symposium on*, pages 443–458. IEEE, 2014. https://eprint.iacr.org/2013/784.pdf.

[25] Charles Noyes. Blockchain multiparty computation markets at scale. https://www.overleaf.com/articles/blockchain-multiparty-computation-markets-at-scale/mwjgmsyybxvw/viewer.pdf Accessed: 2017-08-16.

[26] Threshold relay: how to achieve near-instant finality in public blockchains using a vrf. https://dfinity.network/pdfs/viewer.html?file=../library/threshold-relay-blockchain-stanford.pdf. Accessed: 2017-09-02.

[27] Dan Boneh, Ben Lynn, and Hovav Shacham. Short signatures from the weil pairing. *Advances in Cryptology—ASIACRYPT 2001*, pages 514–532, 2001. https://www.iacr.org/archive/asiacrypt2001/22480516.pdf Accessed: 2017-09-06.

[28] Ethereum StackExchange: How can I securely generate a random number in my smart contract? https://ethereum.stackexchange.com/a/207. Accessed: 2017-08-16.

[29] Filecoin: A Decentralized Storage Network. https://filecoin.io/filecoin.pdf. Accessed: 2017-08-16.

[30] Storj: A Peer-to-Peer Cloud Storage Network. https://storj.io/storj.pdf. Accessed: 2017-08-16.