Finprint: A consumer-controlled, decentralized financial profile

Mike Yu Eugene Marinelli Nima Ghamsari
Kenneth Schiller Arjun Baokar
`team@finprint.com`

Abstract

Digitized consumer financial data—structured and verified by trusted third parties—will drive almost all financial transactions in the future. The easy availability and fidelity of this data will eliminate the need for redundant verification, resulting in dramatic efficiency gains and cost savings for banks, lenders, and other financial institutions.

Today, a large and fragmented array of data providers package and sell this data to financial institutions, creating an unnecessarily complex system that is difficult for the consumer to understand and control.

To simplify this ecosystem, we introduce Finprint, a decentralized protocol to securely and privately manage consumer financial data on the blockchain. With this new protocol, consumers have a single hub to control their entire financial profile, which they use to simply and securely share information with financial institutions they trust. The consumer also benefits from full transparency into the data stored on their profile and how it is used.

1 Background

The last decade has seen a surge in data-driven processes within consumer finance that will soon change how consumers apply for financial products and how institutions make risk decisions. A large part of this shift has been driven by Blend, a key Finprint partner and today's leading consumer lending platform. Blend is already leveraging data and technology to help many of the largest lenders expand access to financial services and empower their customers.

In the Finprint team's experience building and deploying Blend, we've seen consumers and financial institutions alike take an increased interest in the aggregation of verified data for the origination of financial products. Historically, lenders have required documentation in the form of paper or digital statements to serve as verification of a consumer's financial situation. At the surface level, this is onerous for consumers, who must source statements individually from each institution. But the systemic effects are more profound: lenders often have hundreds or thousands of staff members whose jobs are described as "stare and compare", meaning much of their day is spent manually reading the provided documentation and reconciling it with other information provided by the consumer. Adding further cost, the lender will often undergo an expensive direct verification with the underlying institutions towards the end of the loan process to ensure the documentation wasn't forged.

By using financial data aggregation capabilities provided by technologies such as Blend, lenders can directly verify data with the underlying institutions in real time at the point of application. This is simpler for consumers in that it doesn't require them to bring together their financial statements. For the lender, the data now automatically populates their underwriting models and processes without the need for manual entry. Because verification happens up front, the lender can provide the consumer with an initial approval instantaneously [1]. Lastly, because the data comes from a third party with no incentives in the active transaction, and is not manually provided by the consumer or an intermediary, the data can be trusted to a greater degree by both the lender and any downstream participants.

Aggregating consumer data into a centralized data store is a promising approach, but has several key flaws. We've already seen security-related incidents related to centralized consumer data stores, in which over a hundred million consumers have had their personal data compromised. Centralized data stores for consumer information are high-value targets for malicious actors, and it is difficult for external parties to validate their security.

Second, the gatekeepers for such a centralized data store are likely to impose large fees for access to the high-value underlying data. This cost is passed on to the consumer, increasing costs and friction in the ecosystem, and ultimately disincentivizing usage of verified financial data. We have already seen how these high costs discourage lenders from using verified data early on in the origination process. Lenders typically collect and manually validate documents up front, and will only pull the expensive verified data after they're confident that the customer needs the financial product. This creates unnecessary work throughout the system.

A centralized system creates complexities around data management and security, which are top of mind for the industry and its regulators. In early 2018, the Consumer Financial Protection Bureau published a set of "Consumer Protection Principles" [2] to guide how the financial services industry manages consumer data. These principles include consumer access, consumer consent (and accompanying ability to revoke consent), security, and access transparency. A centralized model is fundamentally incompatible with these principles, as it demands that consumers trust third parties. These parties are then relied upon to provide consumer access, respect consumer consent, provide security, and access transparency. Whoever owns the database has the ability to read the data, edit audit logs, and resell the data without consumer knowledge. Finprint, on the other hand, leverages the blockchain to create transparency and control for the consumer, and naturally promotes and enforces the CFPB's principles.

Lastly, centralized data stores might not allow the consumer to freely reuse their own data across financial products or institutions, depending on the incentives of the provider of the data store. This could mean that consumers are required to re-pull and aggregate their data each time they wish to apply for a new financial product. In contrast, Finprint allows consumers to seamlessly reuse their verified personal data with a single tap when they have connected with any platform on the network. This reduces friction and costs for the consumer and lender while allowing for greater velocity of data access and usage by the ecosystem.

The combined weaknesses in the centralized system around security, costs, regulation, and re-use of data call for facilitation via a decentralized protocol. Finprint is that protocol, leveraging core constructs in blockchain and cryptography to build a simpler financial data ecosystem for consumers and financial institutions alike. Finprint uses a public, anonymized ledger to empower consumers to aggregate, share, and audit access to their personal financial data, moving us toward a world where financial processes are data-driven and completed in minutes, not weeks.

2 Objectives

Finprint aims to create a world in which each consumer has easy means to access their own verified financial data and share it with financial institutions of their choosing. The protocol solves three key problems:

1. **Simplicity for the consumer.** Today, consumers access their financial data through a plethora of systems, often relying on third parties to access source of truth information on their own assets, income,

and credit. When a consumer wants to share this information with financial institutions such as lenders, the situation becomes even more complicated, involving an even larger network of middlemen.

2. **Trust and verification of data.** Financial institutions cannot simply trust data provided to them by a consumer. As a result, institutions incur significant costs procuring and validating documentation as they attempt to verify consumer data when making a risk decision.
3. **Consumer consent and access transparency.** Consumer data is often shared between parties without the consumer having full knowledge of how their data is being shared. It is difficult for consumers to gain a clear understanding of who has viewed their data and when. In cases where this data is available to consumers (e.g. credit reports by the credit bureaus), the access log is maintained by and entrusted to a third party.

To further illustrate these three points, consider the example of getting a mortgage. Today, a consumer must prove to a lender that they have adequate assets, income, and creditworthiness to get a loan. In the extensive process of proving this, the consumer and lender will encounter our three problems as follows:

1. The consumer will have to work with separate systems to retrieve all of their own income, asset, and credit data. This process can be very time-intensive, as the consumer has to manage many sets of credentials for various systems. During this process, the lender may receive information about the consumer that the consumer has not previously seen. A common example is when a lender pulls a borrower's credit report. A 2013 Federal Trade Commission report [3] found that 5% of consumers had errors on at least one of their three major credit reports. These errors are particularly problematic when a consumer is unable to access the data before providing it to the lender. This can result in some consumers being unfairly denied loans. This also disadvantages lenders, who would prefer to have originated that loan with the accurate information.
2. The consumer will need to provide hundreds of pages of documentation, such as bank statements and tax transcripts, as evidence of the consumer's stated assets and income. The lender needs to process all of this documentation, typically by evaluating it manually. This adds cost, delay, and friction to the process for both the consumer and the lender.
3. The lender will often share the data with its secondary market investors as it attempts to determine fungibility on the loan. They also share subsets of the data with other data providers in order to verify information about the borrower. While this data sharing may be disclosed to the borrower in legal terms, it is not always clear to the borrower what is being shared, when, and for what purpose.

Our vision is that, in place of the lengthy process above, the consumer will be able to get a loan in a single click by authorizing the lender to access the needed information in their Finprint profile. This profile would have been previously constructed by various trusted providers via the Finprint network, and the lender would have the ability to identify the original writer of each piece of data and decide which data to trust as "verified." All access by the lender, or by any parties the lender wants to share the data with, would have to be explicitly granted by the consumer and would be published to the public audit ledger.

3 Data Sharing Protocol

3.1 Overview

Finprint enables consumers, data providers, and financial institutions to interact via a common protocol and a shared, distributed ledger in order to share consumer financial data. This ledger manages access and

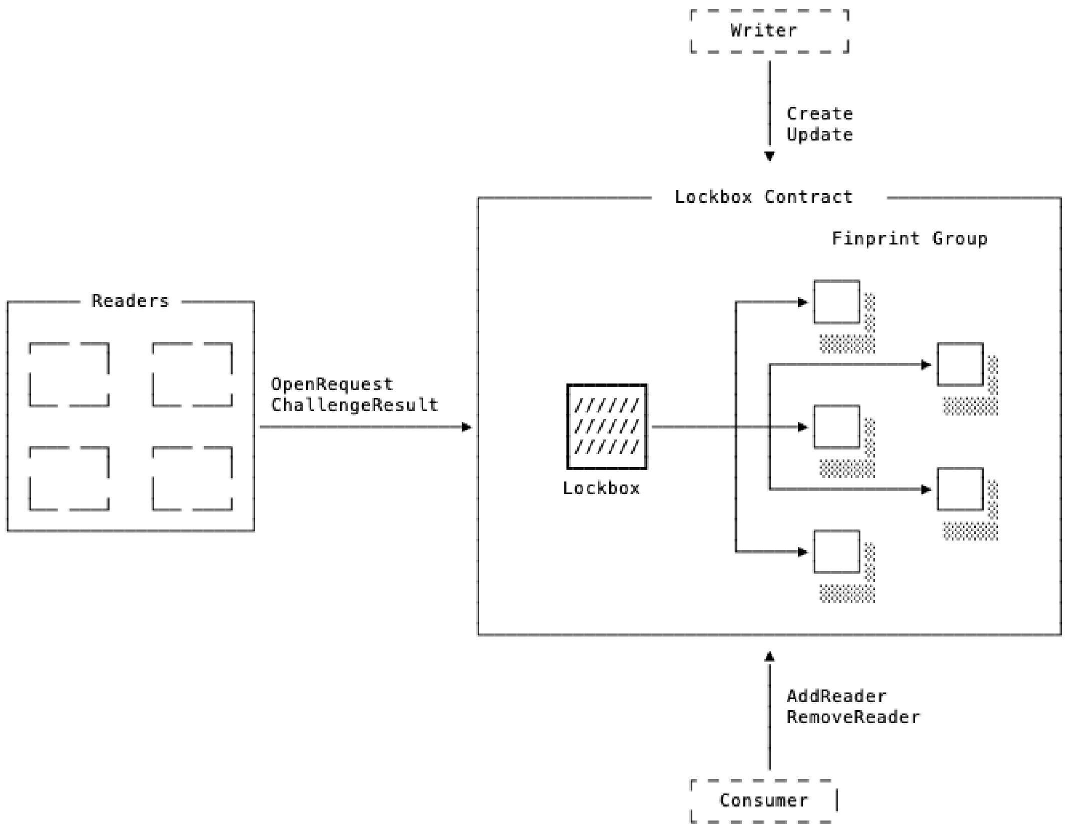

Figure 1: An overview of the major components of the Finprint protocol

payments so that the consumer has control of each party's permissions, the audit trail is immutable and public, and payment for data is built into the protocol to incentivize data providers to offer the highest quality data.

Finprint keeps the consumer's financial data private, while making it simple to share and fully auditable on the public blockchain. At a high level, this is achieved by encrypting and storing private data in decentralized storage, and secret sharing the decryption key across a group of "Finprint nodes" who hold an economic stake in the network. In this way, access to the secret is split up such that every node must coordinate in order to provide the data to another party, and individual nodes never gain information about the private data. Good behavior is enforced with a challenge mechanism and an economic stake requirement.

We use this to develop the concept of a lockbox, an abstraction that enables a writer to provide verified data for a consumer to share. It's worth noting that other cryptographic protocols besides the scheme described above could in theory be used to power lockboxes. These include secure multi-party computation (SMPC), which is leveraged by decentralized privacy protocols such as Keep [4] and Enigma [5], and proxy re-encryption schemes, such as those used by NuCypher [6]. We found the secret sharing approach to be the most suitable given the needs of Finprint and current limitations in the state of the art of SMPC.

The Finprint protocol has four key components. Figure 1 illustrates the interfaces between them. The components are:

1. **Consumer**. A consumer is a party named as the owner of a lockbox. Ownership means that the consumer has the ability to give others permission to view the lockbox's contents. Typically, the consumer may be a person applying for a financial product.

2. **Writer**. A writer is the creator of a lockbox. The writer has the ability to update the data in a lockbox, and receives payment each time the lockbox is read. Typically, the writer may be a financial institution or data provider who is responsible for providing accurate consumer data.

3. **Reader**. A reader is a party who pays to read the data contained in a lockbox. This may be a lender, investor, or institution that uses consumer data to make risk decisions or otherwise needs to observe the data.

4. **Lockbox**. A lockbox is a smart contract object containing a unit of data written by a writer about a consumer. The lockbox specifies a Finprint group that is responsible for maintaining the availability and security of the lockbox data. Interactions with the lockbox occur via the CREATE, UPDATE, ADDREADER, REMOVEREADER OPENREQUEST, CHALLENGERESULT, and LEAVELOCKBOX functions (described in §3.7). Among other things, these provide the mechanisms for a consumer to share the data in the lockbox with readers.

Each entity E who is a party to the protocol has an address a_E, an asymmetric key pair (pk_E, sk_E), and a token balance T_E, denominated in a stablecoin such as DAI [7] or USDC [8]. These stablecoins can be transferred between entities using a transfer function provided by the stablecoin; throughout the paper we will refer to this function as transferTo.

To describe the protocol, we will start by walking through an example use case in §3.2. We then examine each component in the protocol and their state and actions in sections 3.3–3.6. §3.7 describes in detail the functions available on a lockbox. §3.8 describes the management of Finprint groups. Finally, §3.9 disucsses incentive mechanisms including stablecoin transactions and the Finprint token, which is used to demonstrate ownership in the network and is required for participation in Finprint groups.

3.2 Example Use Case

Suppose that Alice, a consumer, is applying for a loan from Emblem Bank, and wishes to provide the necessary personal information via the Finprint protocol. An Emblem loan officer might tell Alice that Emblem Bank needs 60 days of asset history, proof of income, and credit history in order to make a decision.

All Alice needs to do is use her Finprint key to explicitly grant Emblem Bank access to view the data required.

Once Emblem Bank has been granted access, they send payment in stablecoin to data providers (say, A, B, and C for asset, income, and credit data respectively). The payment matches the price that the relevant providers have set for Alice's data. Once the payment is sent, the network validates the payment and Alice's authorization, and decrypts and delivers the data to Emblem Bank. Emblem Bank can read this data, verify via the blockchain that it was originally written by trusted providers, decision on it, and immediately grant Alice her loan.

Note that at any time (before or after Emblem Bank checks the data and makes a credit decision), Alice can take a look at the data and make sure it matches her expectations and is accurate.

Once Alice receives the loan, she may simply revoke Emblem Bank's permissions to read the lockbox, removing their ability to query it for more or updated data. If Alice wants to prove her creditworthiness to her insurance agent while shopping for auto insurance the following week, she can simply permit them to view the required data (in this case, perhaps just a credit history), and they can pay provider A and view the trusted data as well.

3.3 Consumers

A consumer is a party who owns one or more lockboxes, where ownership refers to the ability to grant and revoke access to the underlying data. Because Finprint is consumer-centric (i.e. it is meant to be a way for consumers to collect verified data for the purpose of obtaining financial products), we group lockboxes by consumer as a profile on the blockchain.

Any party can add a lockbox to any profile. Only the consumer who owns the profile can manage the permissions of readers on the lockboxes associated with said profile. The consumer can also view the data inside any lockbox associated with their profile.

3.4 Writers

A writer creates, but does not own, lockboxes and writes data into them. A writer may create thousands of lockboxes over time.

3.5 Readers

A reader doesn't need to maintain any state (other than a public key, which all parties have). A reader simply interacts with lockboxes when given permission to do so, and has read-only access to the data.

3.6 Lockboxes

A lockbox controls access to a consumer's financial data and consists of a smart contract object linked to a Finprint group. To access the data protected by the lockbox, a reader must provide the contract with payment. The contract will validate that the consumer has given permission for this reader to access the data before requesting the Finprint group to provide their shares, encrypted with the reader's session key.

For scalability, the consumer's data may be stored off the blockchain in some decentralized, content-addressable storage network. For example, the data could be encrypted and stored on a hash-keyed storage network, such as IPFS [9] or Swarm [10]. This means that it can be read by any party with the hash and decryption key, and maintains some of the nice properties of blockchain storage (availability, immutability, censorship-resistance) at lower cost. By decentralizing the responsibility of storing data, we also reduce the uptime requirements for the writers.

When creating a lockbox, the writer encrypts data D with a symmetric key k_D and stores the ciphertext on the storage network, keyed with hash H_D. (H_D, k_D) is sufficient to retrieve the data. Then, (H_D, k_D) is secret shared across the Finprint group such that the whole group is needed to retrieve the data for any reader. The reader can use H_D to verify that they receive the correct data from the storage network.

The data associated with the lockbox can be updated by a writer as frequently as a writer and consumer agree to do so—for instance, bank account information might be updated daily, while credit information might only be updated monthly.

The operation of lockboxes is maintained by two key components:

- **Smart contract**. A contract that maintains some publicly available state and validates that the necessary conditions have been met in order for a lockbox to return data to a reader.

- **Finprint group**. A set of service providers, each of whom has a share of the secret data's hash and decryption key. Together, this group can share the contents of the lockbox with a reader under the correct conditions.

3.6.1 Smart Contract

The lockbox smart contract maintains some public state and allows parties to execute a set of functions. The state maintained by the smart contract can be represented as:

```
type Lockbox {
  consumer: Address,
  writer: Address,
  permissionedReaders: Set[Address],
  paidReaders: Set[Address],

  finprintGroup: Array[Address], // Set of group members
  secretShareHashes: Array[String], // Respective hashes for the shares of the group members
  secretShares: String, // Decentralized storage address for encrypted shares of $(H_D, k_D)$

  price: Number, // Price reader must pay to access data, denominated in stablecoin
  transactionFee: Number, // Fee paid to each group member to access data, denominated in stablecoin -
      ↪  the total fee (transactionFee * group size) is a fixed fraction of the price
  lockboxDepositAmount: Number // Deposit amount each group member must store to participate,
      ↪ denominated in stablecoin - this is a fixed multiple of the price
}
```

The functions it exposes are:

- CREATE creates a lockbox.
- UPDATE allows the lockbox's writer to modify the data contained in the lockbox.
- ACKNOWLEDGESHARE allows Finprint nodes to confirm their appointment to a lockbox and commit to their shares.
- ADDREADER and REMOVEREADER allow a consumer to manage the set of permissioned readers.
- OPENREQUEST allows a permissioned reader of a lockbox to get the data it contains by submitting a session key with which the secret shares will be encrypted.
- POSTRESULT is called by each group member to submit their share of the secret, encrypted with the session key of the reader.
- CHALLENGERESULT is called by a reader to challenge the share posted by a Finprint group member if the share does not match the corresponding hash provided by the writer in *secretShareHashes*.
- LEAVELOCKBOX is called by a Finprint node to remove itself from the Finprint group for a lockbox.

The functions are described in detail in §3.7.

3.6.2 Finprint Group and Secret Sharing

The Finprint group is at the core of Finprint's data sharing protocol. The choices we made in the design of this group—and of the associated secret sharing and challenge mechanisms—were directly driven by the use cases we envision for Finprint.

Giving the data to either the consumer or the writer would create uptime requirements that shouldn't be imposed on either party. Using Finprint to share data should not require parties to synchronously respond to requests to read the data because consumers and writers might be smaller parties who are not interested in maintaining a 24/7 node on the network. A reader should be able to read a lockbox totally asynchronously of the consumer's consent or the writer's posting of the data.

There are weaknesses in the incentives with giving the data to the consumer or writer as well. Giving the verified data directly to the consumer would mean the writer has no guarantee of getting paid when the data is read and used. As the writer is providing a useful service by being an unbiased source of truth, as well as attesting to the data and providing it to the consumer, this would break down the ecosystem's incentives.

There are also incentive-related flaws with having the writer provide the data from the lockbox. If the writer has to serve the data, they may choose to withhold it from certain readers. For instance, a financial institution might not want to share asset data with a competitor if that data will be used for the consumer to move their bank account. Giving this role to the writer effectively moves too much control of the data from the consumer to the writer, which is antithetical to the purpose of Finprint.

Our solution is to encrypt and store the consumer's data in decentralized storage, and secret share the decryption key with a group of staked third parties—the Finprint group. When an authorized reader requests to read the data, each member of the group must encrypt its share of the secret for the reader's session key and publish it. The reader then decrypts and combines these shares to determine the full secret without revealing information to any other participant. Structurally, this is similar to a proxy re-encryption scheme, with the Finprint group acting as the proxy. Unless all the members of a sharing group conspire to combine their shares, none of the members gain information about the lockbox contents.

3.7 Lockbox Functions

In this section we describe the basic functions outlined in the protocol overview above and describe how they might be implemented.

3.7.1 Create

To post data about a consumer C, a writer W must first create a lockbox L. W first determines the Finprint group $\{F_1, \ldots, F_n\}$ via AssignGroup, outlined in §3.8. W then takes the desired data, encrypts it with a symmetric key k_D, and stores the encrypted data on a content-addressable decentralized storage network at hash H_D. W shards (H_D, k_D) into n shares using XOR secret sharing, encrypts each share s_i using F_i's public key, and publishes these encrypted shares as *secretShares*.

We don't actually post *secretShares* directly to the contract because the contract never needs to interact with it—only the group members do. Instead, we store it on the same storage network used for the lockbox data, and post a hash or address that can be used to retrieve the data to the smart contract. Additionally, W must post hashes of the individual shares as *secretShareHashes*. These hashes are used by the challenge mechanism described in §3.7.7.

The lockbox is not active, meaning it cannot receive requests, until each Finprint group member has acknowledged via the AcknowledgeShare function that its respective share matches the corresponding hash in *secretShareHashes*.

Algorithm 1 Create

```
1: procedure Create($L_{id}$, $L_{version}$, $C$, $finprintGroup$, $secretShares$, $secretShareHashes$, $price$, $W$=caller)
2:     assert($\forall F_i \in finprintGroup, F_i$.stakedToken $\geq stakeThreshold$)
3:     $LockboxVersionRegistry[L_{version}] := \langle$
           finprintGroup : $finprintGroup$,
           secretShares : $secretShares$,
           secretShareHashes : $secretShareHashes$,
           price : $price$,
           transactionFee : $price * transactionFeeFraction$,
           lockboxDepositAmount : $price * lockboxDepositMultiple$,
       $\rangle$
       $LockboxRegistry[L_{id}] := \langle$
           consumer : $C$,
           writer : $W$,
           currentVersion : null
           pendingVersion : $L_{version}$
       $\rangle$
4:     $ProfileRegistry[C]$.lockboxes.add($L_{id}$)
5: end procedure
```

Two parameters of the lockbox, *transactionFee* and *lockboxDepositAmount*, are calculated from the lockbox price according to fixed ratios specified by the protocol implementation.

3.7.2 Update

Update allows W to update L's content, price, or Finprint group. Updating the content or the Finprint group entails regenerating H_D and k_D, and therefore *secretShares* and *secretShareHahes* as well.

Algorithm 2 Update

```
1: procedure Update($L_{id}$, $L_{version}$, $finprintGroup$, $secretShares$, $secretShareHashes$, $price$, $W$=caller)
2:     assert($W = L$.writer)
3:     assert($\forall F_i \in finprintGroup, F_i$.stakedToken $\geq stakeThreshold$)
4:     $LockboxVersionRegistry[L_{version}] := \langle$
           finprintGroup : $finprintGroup$,
           secretShares : $secretShares$,
           secretShareHashes : $secretShareHashes$,
           price : $price$,
           transactionFee : $price * transactionFeeFraction$,
           lockboxDepositAmount : $price * lockboxDepositMultiple$,
       $\rangle$
5:     $L$.pendingVersion = $L_{version}$
6: end procedure
```

A lockbox's contents are versioned. Each lockbox has a $currentVersion$, which contains the content, price, and Finprint group that readers interact with when opening a request for data. UPDATE only changes a lockbox's $pendingVersion$, which contains the updated information W wishes to write. A $pendingVersion$ is promoted to become the $currentVersion$ after each Finprint group member named in $pendingVersion$ calls ACKNOWLEDGESHARE. Maintaining these two versions allows for data in L to be "hot swapped" without incurring downtime for L while waiting for the Finprint group to acknowledge its shares.

3.7.3 AcknowledgeShare

When a Finprint node F_i has been selected for L's Finprint group, it must follow two steps. First, the Finprint node must deposit *lockboxDepositAmount* stablecoin into L. Then, it must acknowledge its participation in the group by calling ACKNOWLEDGESHARE.

Once a lockbox is active, each Finprint node S_i in L.finprintGroup must respond to requests opened by readers by publishing s_i, encrypted for the reader. The challenge mechanism described in 3.7.7 will enforce that this published share matches the corresponding hash published in L.secretShareHashes. Therefore, before a lockbox becomes active, each S_i must confirm that the secret share s_i provided to it via L.secretShares matches the corresponding hash h_i in L.secretShareHashes.

A Finprint node should only call ACKNOWLEDGESHARE once it verifies that $\mathsf{hash}(s_i) = h_i$. It should not call this function otherwise.

Algorithm 3 ACKNOWLEDGESHARE

```
1: procedure AcknowledgeShare($L$, $L_{version}$, $S_i$=caller)
2:     assert($F_i \in L$.finprintGroup)
3:     assert($L_{version} = L$.pendingVersion)
4:     $L$.secretShareHashes[$F_i$].acknowledged = true
5:     assert($L$.deposits[$F_i$] $\geq L$.lockboxDepositAmount)
6:     if $\forall h_i \in L$.secretShareHashes, $h_i$.acknowledged = true then
7:         $L$.currentVersion = $L$.pendingVersion
8:         $L$.pendingVersion = null
9:     end if
10: end procedure
```

Requiring Finprint nodes to call ACKNOWLEDGESHARE before a lockbox accept requests helps to protect the Finprint nodes from being penalized in case the writer posts invalid share hashes. If a Finprint node does not call ACKNOWLEDGESHARE in a timely manner, the writer may call UPDATE to replace that node with another eligible Finprint node or fix malformed shares or share hashes.

3.7.4 Add/Remove Reader

C can manage who can see the data in L by adding or removing permissioned readers.

Algorithm 4 AddReader / RemoveReader

```
1: procedure AddReader($L$, $R$, $C$=caller)
2:     assert($C = L$.consumer)
3:     $L$.permissionedReaders.add($R$)
4: end procedure
1: procedure RemoveReader($L$, $R$, $C$=caller)
2:     assert($C = L$.consumer)
3:     $L$.permissionedReaders.remove($R$)
4: end procedure
```

3.7.5 OpenRequest

Once C grants it permission, a reader R can request L's content using OpenRequest. To do so, R generates an asymmetric key pair $(pk_{session}, sk_{session})$ for the request and passes $pk_{session}$ into OpenRequest. This is similar to usage of a session key in conventional encrypted communication. OpenRequest also requires R to pay L.price into L.

Algorithm 5 OpenRequest

```
1: procedure OpenRequest($L$, $pk_{session}$, $R$=caller)
2:     assert($R \in L$.permissionedReaders)
3:     assert($L$.currentVersion $\neq$ null)
4:     assert($L$.finprintGroup $\neq \emptyset$)
5:     if $R \notin L$.paidReaders $\cup \{L$.consumer, $L$.writer$\}$ then
6:         transferTo($R, L, L$.price)
7:         $L$.paidReaders.add($R$)
8:     end if
9:     transferTo($R, L, L$.transactionFee $* |L.finprintGroup|$)
10: end procedure
```

Note that C or W can act as a reader and call OpenRequest. In this case, C or W is not required to pay the portion of the price that would go to W. They only have to pay the *transactionFee* for the Finprint group members.

3.7.6 PostResult

Once R's payment of L.transactionFee to L for a request r has been recorded on the blockchain, each group member F_i re-encrypts their share of the secret with the session key $pk_{session}$ from the OpenRequest call and posts the encrypted share to the smart contract as res_i with PostResult.

Algorithm 6 PostResult

```
procedure PostResult($L$, $r$, $res_i$, $version$, $F_i$=caller)
    assert($F_i \in L$.finprintGroup)
    assert(no previous PostResult involving $L, F_i,$ and $r$))
    $L$.results$[r, F_i] = res_i$
    transferTo($L, F_i, L$.transactionFee)
    if $R \notin \{C, W\}$ and $F_i$ is the last member of $finprintGroup$ to post then
        transferTo($L, L$.writer, $L$.price)
    end if
end procedure
```

Once all of the res_i have been posted, W is paid their share of L.price. R can decrypt each res_i using $sk_{session}$ and recombine the results to recover (H_D, k_D). R can then use H_D to fetch the encrypted data from decentralized storage, and decrypt it using k_D to get D.

3.7.7 ChallengeResult

The protocol as presented so far assumes trust in all parties in the Finprint group to correctly and honestly post their shares, and thus present valid data to the reader. Rather than rely on this trust, we create an enforcement mechanism that allows R to claim that the result posted by a Finprint node F_i was malformed or not delivered at all. If the reader successfully proves this was the case, they receive the lockbox deposit F_i provided to L.

If R wants to claim a result was never posted, it can call ChallengeMissingResult, specifying the request r and which Finprint node F_i failed to respond. If R wants to claim a result is malformed or incorrect, it can call ChallengeInvalidResult, which requires it to additionally publish its secret session key $sk_{session}$. This allows the smart contract to verify publicly whether or not F_i's posted share matches the secret share hash posted by the writer.

Algorithm 7 ChallengeResult

```
procedure ChallengeMissingResult($r$, $F_i$, $R$=caller)
    assert($R$ opened $r$)
    assert($F_i \in L.finprintGroup$)
    if $L$.results$[r, F_i]$ = null $\wedge$ $currentTime > r$.openedAt + $responseWindow$ then
        transferTo($F_i, R, L$.lockboxDepositAmount)
        $L$.finprintGroup = $\emptyset$
    end if
end procedure

procedure ChallengeInvalidResult($r$, $F_i$, $sk_{session}$, $R$=caller)
    assert($R$ opened $r$)
    assert($F_i \in L.finprintGroup$)
    $res_i := L$.results$[r, F_i]$
    if isValidKeypair($r.pk_{session}, sk_{session}$) $\wedge$ $L$.secretShareHashes$[F_i] \neq$ hash(decrypt($res_i, sk_{session}$)) then
        transferTo($L, R, L$.lockboxDepositAmount)
        $L$.finprintGroup = $\emptyset$
    end if
end procedure
```

In response to a missing result challenge, the smart contract will check if a publicly known $responseWindow$ has passed since the request was opened. If $responseWindow$ has passed but F_i has not posted a result, F_i forfeits its lockbox deposit to R.

In response an invalid result challenge, the smart contract will first validate that $sk_{session}$ is the secret key that matches the previously posted $pk_{session}$ for that request. It then uses $sk_{session}$ to decrypt res_i. If res_i does not match its corresponding hash in L.secretShareHashes, F_i forfeits its lockbox deposit to R.

In the event F_i forfeits its deposit, it no longer has the lockbox deposit required to participate in L, and cannot continue being a member of that lockbox's Finprint group. With a single missing member, the group can no longer serve requests from readers, so L.finprintGroup is reset and the lockbox is rendered inactive until the writer calls UPDATE. This incentivizes writers to quickly replace unreliable Finprint group members.

Note that each Finprint node is simply responsible for encrypting and publishing the share that was originally provided to it. The Finprint nodes are not responsible for ensuring the quality of the data provided by the writer. If a writer provides bad or poor quality data or shares, readers can leverage the reputation protocol discussed in §4 to encourage others to avoid the lockbox, decreasing the writer's profits.

3.7.8 LeaveLockbox

It is important that Finprint group members are able to exit the group associated with any lockbox at will so that they are able to absolve themselves of availability requirements for that lockbox and recover their deposits. By exiting all groups they participate in, members can also reclaim their (much larger) stakes from the staking contract, as they no longer need to have "skin in the game" when they are no longer participating.

In order to do this, a Finprint group member s_i must broadcast intent to leave the group by calling INITIATELEAVELOCKBOX. A writer can call UPDATE to replace this Finprint group member with another that meets the requirements. In the event that a writer does not respond to this request for a protocol-specified period of time, $leaveRequestWindow$, the Finprint node can call FORCELEAVELOCKBOX to reset the Finprint group for that lockbox and remove all members from it.

Algorithm 8 LEAVELOCKBOX

```
   procedure INITIATELEAVELOCKBOX($L$, $F_i$=caller)
2:     assert($F_i \in L.finprintGroup$)
       $L$.leaveRequests[$F_i$].$initiatedAt = currentTime$
4: end procedure
1: procedure FORCELEAVELOCKBOX($L$, $F_i$=caller)
2:     assert($R = L$.requests[$requestId$].R)
3:     assert($F_i \in L.finprintGroup$)
4:     if $L$.leaveRequests[$F_i$].initiatedAt + $leaveRequestWindow < currentTime$ then ;
5:         $L$.finprintGroup = ø
6:         $L$.leaveRequests = ø
7:     end if
8: end procedure
```

If a Finprint node successfully calls FORCELEAVELOCKBOX and resets the Finprint group, the lockbox becomes inactive and no longer allows readers to open requests. A writer can reactivate it by calling UPDATE with a new Finprint group.

3.8 Finprint Group Management

Finprint group membership must be carefully controlled because the members of lockbox groups are ultimately responsible for maintaining the integrity and security of the network. Here, we define the on-ramps and off-ramps for Finprint group participation.

3.8.1 Finprint Group Entry

In designing a mechanism for lockboxes to select Finprint group members, there are a few factors that need to be considered:

- The security of the data is parameterized by the number of members of the group, transaction fee, and lockbox deposit. These parameters should be selected by the writer, as they know the value of the data and its sensitivity. Note that these parameters will likely vary by dataset—e.g. detailed asset history is probably worth more than an attestion that Alice is an employee of a specific company.
- It should be difficult for an attacker to deliberately target and gain control of a specific lockbox. To this end, the group selection should not be deterministic or predictable.
- Members of the group need to be incentivized to behave in line with the protocol, including to post valid results when called upon, and to protect the secret in the lockbox.
- For scalability purposes, we want to do as little of the selection as possible on the blockchain.

To address these factors, there are some requirements that a party must meet to be a member of a Finprint group:

- Finprint groups are opt-in—a party must want to be in a group in order to be selected. In particular, they must opt in to participate at a given fee F and lockbox deposit LD, both a fixed percentage of the data price P.
- Finprint group members must hold a large amount of Finprint token, staked in a staking contract from which the token can only be retrieved if the party is not a member to any lockbox's Finprint group. This disincentivizes bad network behavior with the threat of a devaluation of the network currency if the network is compromised.

In order to meet all of these conditions, we present the AssignGroup protocol. It proceeds in three phases:

1. **Search for candidates.** The writer W broadcasts the parameters of the Finprint group: the price P, the number of group members n, and the minimum number of candidates $m > n$ for the AssignGroup protocol to succeed.
2. **Declare candidacy.** Each party who wants to be considered for the group posts a signed response to the search. This response consists of a public hash function H computed over the hash of the current block in the blockchain and the address of the poster (to generate some pseudo-randomness in the selection process), as well as the address of a standard staking contract in which the candidate has committed a reasonably large stake of Finprint token.
3. **Select group.** If there are not at least m parties who declared candidacy, the protocol exits with no selection and the writer will need to try again, probably with a higher fee. Otherwise, the n lowest hashes of eligible parties (those with large enough holdings) are selected to be the group.

3.8.2 Finprint Group Exit

Finprint nodes can exit their respective Finprint groups using the process described in 3.7.8. When selecting the new group, the writer executes **Declare candidacy** and **Select group** to select the new members.

While leaving lockboxes is straightforward, Finprint group members cannot immediately recover their lockbox deposit upon leaving L to ensure that they can still be held responsible for reader challenges to their recently posted results. To recover its lockbox deposit, a F_i can have no requests opened against the L.finprintGroup it was a part of within the protocol-fixed $challengeWindow$ time period. Furthermore, to ensure that it isn't part of the new Finprint group for L, the node cannot have called AcknowledgeShare on the pending version for L. Once these conditions are met, the F_i can withdraw its deposit for L.

To reclaim its staked Finprint token, F_i simply has to ensure that it isn't part of any lockbox's Finprint group, current or pending, and that a protocol-specified amount of time has passed since it was part of one.

The smart contract provides these guards to protect readers for the promised time window while maintaining Finprint group flexibility.

3.9 Incentives

While the above components make our goals *possible*, incentives are needed to make them actually happen. When designing a decentralized protocol, especially one meant to handle consumer information and power the financial system, it's important to incentivize good behavior, such as the writing of truthful and high-quality data and the protection of each secret contained in a lockbox.

In order to drive these incentives, we use two forms of cryptocurrency. One is used in the data sharing protocol for the reader to pay the writer to retrieve data from the lockbox and to incentivize the secret sharing and re-encryption operation by paying fees to the group. This will be a stablecoin not native to the Finprint network.

We also introduce a native Finprint token, which must be held by group members in order for them to participate. Finprint group members need to have significant holdings of Finprint token so they are incentivized to maintain the integrity of the network, lest their holdings be significantly devalued.

All data, keys, and tokens (of either type) change hands in the Finprint protocol via transactions, which are recorded on the blockchain. This also creates the immutable audit log, allowing consumers to identify who has read what data and how often.

In this section, we analyze the incentive structures of the parties in the Finprint protocol.

3.9.1 Consumer Behavior

Consumers using Finprint are looking to get a financial product. Finprint is designed to bring simplicity and transparency to the consumer, so their incentives are most aligned with the overall protocol. Consumers only need to authorize readers to access their lockboxes, which they are naturally incentivized to do when they have reason to do so (applying for a loan, etc), and are incentivized not to do otherwise to protect their own privacy. Finprint is about empowering the consumer, and it seems clear that the consumer would be inclined to behave as expected.

3.9.2 Writer Behavior

Writers using Finprint are effectively selling data they have on the consumer. This data may have value for two reasons: (1) it comes from an unbiased and trustworthy third party, and (2) it is structured and high-fidelity. Finprint incentivizes writers to provide data with these two characteristics.

Finprint's per-read-request payment model incentivizes writers to provide data that is as valuable as possible. Even today, buyers of financial data deeply value high-fidelity trustworthy data, demonstrated by initiatives such as Fannie Mae's Day 1 Certainty program [11] to attest to the trustworthiness of data providers. Writers who are able to build strong reputations as high-quality data providers will see the usage of their lockboxes skyrocket, and their profits will rise accordingly. A future, on-chain representation of a writer's reputation would also incentivize this further.

3.9.3 Reader Behavior

The only way for the reader to get "trusted" data that they are confident was provided by the writer is to obtain it from the lockbox itself, which requires payment to the writer. We have seen that readers are increasingly willing to pay for "trusted" data, especially in the wake of financial disasters spurred by poor risk decisions such as the housing market collapse in 2007-2009. Consumer-stated data is no longer sufficient for many loan products, and this trend is expected to continue as the regulatory environment develops. Accordingly, we expect that this inability to validate the data as "trusted" without paying for it will be sufficient incentive for the reader to pay.

3.9.4 Finprint Group Behavior

The incentives of the Finprint group are the hardest to pin down, as this group must be incentivized to:

- Quickly post results via PostResult when a reader has successfully executed OpenRequest.
- Post accurate results that can be used to derive the correct secret.
- Not post results via PostResult at any other time, or give the secret to unauthorized parties using any other method.

To maintain these incentives, tokens are used in three ways:

1. Transaction fees are received for good behavior. These fees are received in stablecoin.
2. Members of the Finprint group must deposit tokens with a lockbox to be a part of it. This deposit can be lost to the reader for not posting accurate results. These deposits are also transacted in stablecoin.
3. Members of the Finprint group must prove a large stake of Finprint token, disincentivizing behavior that might devalue the network. This is the only use for the native Finprint token, which effectively serves as a license to participate in secret sharing and collect fees.

In order to incentivize the posting of results, a Finprint group member is paid a transaction fee when it makes a call to PostResult. This fee is a fixed fraction of the price specified by the writer when creating the lockbox, which is also when the Finprint group is selected.

To incentivize the posting of accurate results, the ChallengeResult protocol allows readers to punish Finprint group members for not doing so. We can settle disputes by having the network validate a group member's posted result by exposing the shares they each posted if the reader disputes the validity. If the shares do not match those the writer provided the group member, the challenging reader may demonstrate it and collect the member's deposit.

In order to prevent the secret from being given up, there are a number of incentives and safeguards in place. Because every component of the secret is necessary to compute it, we maintain confidentiality in the presence of $n - 1$ dishonest nodes, meaning all participants in the group would have to deviate in order to give up the secret. This protects the secret from all but the most comprehensive Sybil attacks. We further disincentivize each individual member from giving the secret to unauthorized parties by requiring that all group members prove a large stake of Finprint tokens. If the network appears insecure due to Finprint group members giving up secrets, the value of that Finprint token stake will be adversely affected. This means that a massive monetary gain would have to be realized in order for an group member to rationally give up their share of the secret, making it impractical for a party to attempt to collect the secret from the group members.

4 Reputation Protocol

When a financial institution makes a risk decision, it is important that the data being used is "trustworthy." Institutions have thresholds for what data they treat as trustworthy and what data they do not—trust is relative. Different institutions have different risk tolerances and different partnerships with providers, and accordingly have different levels of trust in each provider.

Finprint does not have an absolute measure of public trust, instead allowing each reader to identify the writer of the data in question and independently decide how to treat that data. However, Finprint will introduce support for public reputation, which is available, but not required, for readers to consider when determining the value of lockboxes and deciding what data to pay for.

This protocol uses similar paradigms to those found in token-curated registries [12] and allows consumers and readers to vote on the quality of data sources by staking Finprint tokens on the direction of the vote. This reputation protocol needs more work to be formalized.

5 Conclusion

The Finprint protocol offers a way to securely store and share financial data while ensuring the consumer maintains control over their data. By incentivizing the sources of truth to write data themselves, the protocol provides frictionless access to high-fidelity data to expedite risk decisions that today are slowed down by lengthy data verification steps.

Finprint also provides further guarantees which our experience leads us to believe would ease adoption of the protocol. We don't impose uptime requirements on consumers or financial institutions that wish to read or write data, and we do this without sacrificing data availability, thanks to the design of the Finprint group. This same mechanism protects readers from selective censorship by writers—if a consumer wants their data shared with a particular reader, a writer cannot block that request. Finally, a reader receiving invalid data can demand compensation via our challenge mechanism, and a reputation system incentivizes writers to provide accurate data. Finprint provides first steps toward the creation of a secure data sharing protocol that increases transparency, data fidelity, and simplicity for the consumer.

References

[1] Nima Ghamsari. The simplest mortgage approval ever. https://blend.com/blog/news/future-lending-one-tap/, 2019.

[2] Consumer Financial Protection Bureau. Consumer Protection Principles: Consumer-Authorized Financial Data Sharing and Aggregation. http://files.consumerfinance.gov/f/documents/cfpb_consumer-protection-principles_data-aggregation.pdf.

[3] Federal Trade Commission. In FTC Study, Five Percent of Consumers Had Errors on Their Credit Reports That Could Result in Less Favorable Terms for Loans. https://www.ftc.gov/news-events/press-releases/2013/02/ftc-study-five-percent-consumers-had-errors-their-credit-reports.

[4] Luongo and Pon. The Keep Network: A Privacy Layer for Public Blockchains. https://keep.network/whitepaper.

[5] Zyskind, Nathan, and Pentland. Decentralizing Privacy: Using Blockchain to Protect Personal Data. https://www.enigma.co/ZNP15.pdf.

[6] Egorov, Nuñez, and Wilkison. NuCypher: A proxy re-encryption network to empower privacy in decentralized systems. https://github.com/nucypher/whitepaper/blob/master/whitepaper.pdf, 2018. Accessed: 2019-11-20.

[7] The Maker Team. The Dai Stablecoin System. https://makerdao.com/whitepaper/Dai-Whitepaper-Dec17-en.pdf.

[8] The Centre Team. CENTRE. https://www.centre.io/pdfs/centre-whitepaper.pdf.

[9] IPFS - Content Addressed, Versioned, P2P File System (DRAFT 3). https://ipfs.io/ipfs/QmR7GSQM93Cx5eAg6a6yRzNde1FQv7uL6X1o4k7zrJa3LX/ipfs.draft3.pdf. Accessed: 2019-11-20.

[10] Swarm Documentation. http://swarm-guide.readthedocs.io/en/latest/index.html. Accessed: 2018-03-05.

[11] Fannie Mae. Day 1 Certainty. https://www.fanniemae.com/singlefamily/day-1-certainty, 2018. Accessed: 2019-11-20.

[12] Mike Goldin. Token-Curated Registries 1.0. https://medium.com/@ilovebagels/token-curated-registries-1-0-61a232f8dac7, 2017. Accessed: 2019-11-20.