Sancus: Cryptographic Audits for Virtual Currency Institutions

by

Ravi Rahman

SB, Electrical Engineering and Computer Science, Massachusetts Institute of Technology (2020)

Submitted to the Department of Electrical Engineering and Computer Science
in partial fulfillment of the requirements for the degree of

Master of Engineering in Electrical Engineering and Computer Science

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

June 2021

Author . . . .
Department of Electrical Engineering and Computer Science
May 20, 2021

Certified by. . . . .
Lalana Kagal
Principal Research Scientist
Thesis Supervisor

Accepted by . . . .
Katrina LaCurts
Chair, Master of Engineering Thesis Committee

Sancus: Cryptographic Audits for Virtual Currency Institutions

by

Ravi Rahman

Submitted to the Department of Electrical Engineering and Computer Science
on May 20, 2021, in partial fulfillment of the
requirements for the degree of
Master of Engineering in Electrical Engineering and Computer Science

Abstract

Sancus introduces fully accountable, privacy preserving, cryptographic audits for virtual currency institutions – entities that allow users to deposit, exchange, and withdraw blockchain-based funds. These audits, verifiable by the public, provide irrefutable proofs that institutions not only have accounted for all customer transactions but also own at least as much in blockchain assets as they owe to their users. Sancus addresses major limitations in previous works for blockchain auditing: it supports institutions that offer multiple currencies on multiple blockchains, including Bitcoin and Ethereum; it follows security best practices and uses offline wallets; it preserves privacy for the institutions and their customers by hiding transaction amounts and blockchain addresses; and it produces definitive proofs of solvency as individual customers take no part in the auditing process. Evaluation of our reference implementation of Sancus demonstrated that the audit generation time, audit validation time, and size of audits scale linearly with the number of users, number of transactions, and privacy parameters. With efficient runtimes for audit generation and validation in a multi-threaded environment and megabyte order-of-magnitude audit sizes, Sancus offers a promising, new approach for continuous auditing of virtual currency institutions.

Thesis Supervisor: Lalana Kagal
Title: Principal Research Scientist

Acknowledgments

First and foremost, I would like to thank my advisor, Dr. Lalana Kagal. I joined her lab in 2018 for my Advanced Undergraduate Researching Opportunities Project (SuperUROP) on representing legal agreements through blockchain smart contracts. This opportunity introduced me to blockchain systems and led to the idea for Sancus. I am grateful to have had such a thoughtful, supportive, and responsive advisor whose guidance was invaluable to this project.

I would also like to thank the undergraduate researchers (UROPs) – Qiong Huang, Lay Jain, Anne Ouyang, and Ben Wolz – who implemented key components of the system. Qiong built a fully-functioning web client which has allowed us to demonstrate the usability of Sancus. Lay's and Anne's research on developing extensions to support loans and enhance privacy contributed to a more secure and flexible system design. Ben's work on implementing the auditor, in addition to evaluating the entire system, was crucial to proving Sancus's implementation actually works. Building and demonstrating a full implementation of Sancus, in just one semester, would not having been possible without this dedicated and talented team.

Finally, I would like to thank my parents – Lynda Furash and Swapan Rahman – for their encouragement to pursue my Master of Engineering degree and support through this process. It has been a rewarding experience – one that will be the foundation for my future.

Contents

List of Figures

Chapter 1

Introduction

Public blockchains, such as Bitcoin [29] and Ethereum [41], introduced an alternative to the traditional financial system. Exchanges such as Coinbase [1] and Gemini [2] enable users to trade virtual currencies for fiat currencies. BitPay [3] and BlockCard [4] offer debit cards backed by virtual currency, which enable their customers to effectively use their virtual currency holdings as checking accounts. More recently, virtual currency lending platforms such as BlockFi [5] and Nexo [6] issue loans secured by virtual currency.

However, the virtual currency industry does not share the same oversight and protections extended to traditional financial institutions. In the United States, there is no federal regulatory framework for virtual currency institutions. Unlike bank deposits which are insured by the Federal Deposit Insurance Corporation [15], virtual currency deposits are not insured by government entities. Individual states have a patchwork of virtual currency regulation. For example, New York developed BitLicense [31]. In September 2020, the Conference of State Bank Supervisors (CSBS) introduced a multi-state regulatory framework for money services businesses, which include virtual currency firms [35]. This system aims to standardize all state virtual currency regulations. However, the CSBS framework does not offer the same pro-

[1]`https://coinbase.com`
[2]`https://gemini.com`
[3]`https://bitpay.com`
[4]`https://getblockcard.com`
[5]`https://blockfi.com`
[6]`https://nexo.io`

tections afforded by federal banking regulations, such as insurance on stored funds [16].

Moreover, the open design of public blockchains have enabled criminal hackers to steal funds from unsuspecting users. Blockchain accounts do not reveal real-world identities, and the decentralized control combined with a cryptographic protocol prevents transactions from being reverted. These technical limitations, combined with a lack of regulatory oversight, illustrate why the blockchain ecosystem must adopt new approaches for trust and accountability to compete with the traditional financial system.

We introduce Sancus, a system that produces irrefutable audits for virtual currency institutions. We define virtual currency institutions as entities that allow users to deposit, exchange, and withdraw blockchain-based funds. Audits produced by Sancus are verifiable by the public and prove that institutions not only have accounted for all customer transactions but also own at least as much in blockchain assets as they owe to their users. Unlike previous works, Sancus supports institutions that offer multiple currencies on multiple blockchains, including Bitcoin and Ethereum; it follows security best practices and uses offline wallets; it preserves privacy for the institutions and their customers; and it produces definitive proofs of solvency as individual customers take no part in the auditing process. Unlike traditional auditors, Sancus runs on off-the-shelf computing equipment, and audits can be performed as frequently as every new block (e.g. for Bitcoin, every 10 minutes).

Chapter 2 provides a background of blockchain systems and the challenges of a blockchain-based financial system. Chapter 3 describes related works in the field. Chapter 4 presents the capabilities and specification for Sancus. Chapter 5 discusses our reference implementation. We evaluate our implementation in chapter 6 and conclude with possible extensions in chapter 7.

Chapter 2

Background

Sancus extends upon public blockchains and zero knowledge proofs to audit virtual currency institutions – entities that allow users to deposit, exchange, and withdraw blockchain-based funds.

2.1 Blockchains

A blockchain is a trustless, decentralized, append-only, globally synchronized ledger. A network of independent nodes maintain a blockchain. Records appended to a blockchain, called *transactions*, are bundled together into *blocks*. While any type of data can be appended to a blockchain, records typically denote a transfer of funds between blockchain users. The nodes collectively guarantee that all new transactions are consistent with the existing chain – for example, that funds are not being double spent.

Blockchains offer a promising platform for a new financial system. The append-only design guarantees that transactions cannot be reverted. Combined with a globally synchronized state, blockchains prohibit users from spending the same funds twice. The public record of blockchain transactions enables anyone to validate that each appended record was a valid transition from the previous state. Competition among miners provides a competitive market to minimize transaction fees. Unlike the traditional financial system which requires users to trust many entities, such as banks, auditors, and courts, a blockchain-based financial system can replace this network of trusted entities with strong cryptographic proofs.

2.1.1 Bitcoin

The Bitcoin white-paper in 2009 [29] introduced a novel computational platform, called a blockchain.

Anyone can become a Bitcoin user by creating a random 256 bit private cryptographic key, computing the corresponding public key on the `secp256k1` elliptic curve, and following the Bitcoin specification to reduce the public key into a Bitcoin address [40]. This authentication scheme does not require any centralized authority to approve new users. Users must sign their transactions with their private keys. Via public key cryptography, users can share their public keys. These signatures prove that the owners authorized the transactions which spend funds associated with their addresses.

Most transactions involve transferring funds from one account to another account. However, the Bitcoin protocol, via Bitcoin Script, supports complex transaction authorization. For example, via a multi-signature wallet, one can specify that a transaction is only valid if a certain number of signatures have been obtained.

The Bitcoin protocol requires that the entire transaction history be public, so one can verify that there are sufficient funds available to be spent. While Bitcoin's public key authentication does not reveal real-world identities and thereby preserves anonymity, Bitcoin does not preserve privacy. Anyone can see any user's transactions, including transaction recipients and amounts.

A novel contribution from Bitcoin is its proof-of-work protocol. Proof-of-work requires presenting a solution to a computational hard-to-solve but easy-to-verify problem. Bitcoin uses the `hashcash` function, which involves finding a nonce, that when appended to an input, produces a hash where first k bits are 0 [39]. As long as honest miners contribute a majority of the computational power, it is not feasible for a minority of dishonest miners to permanently rewrite the transaction history.

To encourage users to become miners and maintain the state of the blockchain, the Bitcoin protocol rewards miners with Bitcoin for mining new blocks. In addition, individual transactions include transaction fees which go to the miners. This scheme enables market-based pricing for Bitcoin transactions.

2.1.2 Ethereum

Ethereum, launched in 2015 [41], introduced a new blockchain that supports arbitrary, Turing-complete transactions. These transactions, called smart contracts, support on-chain storage, can call functions in other smart contracts, and can spawn new smart contracts.

Smart contracts support many applications, including Ethereum-based tokens. The ERC-20 protocol defines a common interface for Ethereum-based tokens [36]. Such tokens typically represent one's claim to another asset, such as ownership in a company or hard currencies stored in an institution's account.

Similar to Bitcoin, Ethereum uses the `secp256k1` elleptic curve, has a public transaction history, and uses a proof of work protocol.

2.2 Challenges of Virtual Currency

There are multiple challenges with building a financial system on the Bitcoin or Ethereum blockchains.

2.2.1 Loss of Credentials

It is impossible to spend the virtual currency stored at a given address without the corresponding private key. Should users lose their private keys, their funds would also be lost – there is no equivalent of a "reset password" operation, as is standard in online banking. Likewise, should hackers acquire users' private keys, they would be able to steal users' virtual currency. Centralized virtual currency platforms overcame this challenge by requiring that customers store their virtual currency in accounts controlled by the platform. Thus, the platforms – and not the users themselves – are responsible for secure key storage.

2.2.2 Transaction Privacy

Suppose Alice wants to send virtual currency to Bob. When Alice asks for Bob's virtual currency address, she would be able to see every other transaction for Bob's address. Likewise, when Bob receives Alice's virtual currency, he would be able to see Alice's transaction history. While such privacy concerns are mitigated because the blockchain does not store real-world identities, and one can create infinitely many blockchain addresses, the public transaction graph of blockchains reveals related accounts and transaction amounts. Virtual currency institutions provide privacy by pooling customer funds together.

2.2.3 Transaction Fees

To encourage a miner to include a transaction in a block, the transaction must include a fee for the miner. The fee is independent of the transaction amount. As miners prioritize transactions by the fee, paying a higher fee results in faster transaction processing.

2.2.4 Transaction Throughput

Bitcoin and Ethereum collectively process fewer than 20 transactions per second, for a maximum daily transaction volume of 1.7 million [4, 5]. In comparison, in 2018, the ACH networks process 63 million transactions per day [28]. Ongoing improvements to the blockchain, such as lightning networks, promise to increase transaction throughput [33]. However, lightning networks require users to join and commit funds to the same network, thereby preventing funds from otherwise being spent on the blockchain.

2.2.5 Accountability

In traditional audits, such as those performed on publicly-traded financial institutions, trusted auditors view confidential records and summarize their non-confidential findings in public statements. However, few traditional auditors offer services for centralized virtual currency institutions, and privately held companies are not inclined to pay for auditors absent regulatory requirements. As such, centralized virtual currency institutions have limited oversight. While some institutions such as Coinbase and Gemini voluntarily maintain insurance to cover theft of virtual currency [9, 19], such insurance may be insufficient to cover all customer deposits. As such, users must rely on the public statements put forth by these companies and the reputation of their investors. This trust model is weak relative to the auditing and regulatory requirements imposed on banks.

2.3 Zero Knowledge Proofs

Zero knowledge proofs (ZKPs) enable one to attest to knowledge of a statement without revealing the statement itself. For example, suppose Alice has a computer secured by a password, which she tells to Bob. Charlie would like to know whether Bob can access Alice's computer. While it would be trivial for Bob to give Charlie the password, which Charlie could then verify, Charlie would then have access to all of Alice's files. Instead, Charlie could present a challenge to Bob, such as asking him for the content of the latest email Charlie sent to Alice. Bob could then log into Alice's computer, retrieve this email, and give it to Charlie. Charlie, knowing his own emails, could easily verify that Bob is telling the truth. In this example, Bob used a zero-knowledge proof to demonstrate that he knows a statement (Alice's password) without revealing what it is.

In practice, ZKPs use NP-hard algorithms that have trapdoor functions. These algorithms are identical to those used in public key cryptography.

2.3.1 Elliptic Curve Cryptography

We focus on elliptic curve cryptography (ECC) [22] zero knowledge proofs, as ECC is also used by blockchain systems. For convention, Sancus uses capital letters to represent elliptic curve points, lowercase letters to represent scalars, and polynomial notation to represent operations. The Elliptic Curve Discrete Logarithm Problem forms the basis of ECC security: there is no known efficient (polynomial) solution to solve for x given Y and G in the equation $Y = xG$. ECC permits efficient point multiplication ($Y = xG$) and addition ($Z = X + Y$). It also allows for homomorphic addition (given $Y_1 = x_1G_1 + x_2G_2$ and $Y_2 = x_3G_1 + x_4G_2$, then $Y_1 + Y_2 = (x_1 + x_3)G_1 + (x_2 + x_4)G_2$).

2.3.2 Pedersen Commitments

For an elliptic curve group $\mathbb{G}$ of prime order q, consider two generators (points) $\{G, H\} \in \mathbb{G}$ and scalars $\{x, r\} \in \mathbb{Z}_q$, where the relative discrete logarithm between G and H is unknown. The Pedersen commitment $Y = xG + rH$ represents a commitment to the value x with $r \overset{\$}{\leftarrow} \mathbb{Z}_q$ [32]. Since Y is any point on the curve, it is perfectly hiding; Y does not reveal any information about x. One can later "open" a commitment by revealing x and r. Because of the Elliptic Curve Discrete Logarithm Problem, it is infeasible to open the commitment without knowing both x and r.

Because Pedersen commitments are based upon ECC, they support point multiplication and homomorphic addition on committed values. Sancus uses these properties to build commitments, which are later used in zero-knowledge proofs.

2.3.3 Non-Interactive Zero Knowledge Proofs (NIZKs)

Traditional zero knowledge proofs require an interactive process, called the Sigma protocol. First, the prover commits to a random value. Second, the verifier presents a random challenge to the prover. Finally, the prover responds to the challenge to complete the proof. To eliminate the challenge-response step in the Sigma protocol, the prover can use the Fiat-Shamir characteristic to select an appropriate challenge [17]. The verifier then checks that the challenge was properly computed. The Sigma protocol otherwise proceeds as before. Sancus utilizes this technique to form non-interactive zero-knowledge proofs (NIZKs), as they can be computed once and verified by anyone.

Chapter 3

Related Works

Sancus extends upon existing works regarding zero-knowledge proofs, privacy preserving blockchains, and decentralized information storage and sharing.

3.1 Proofs of Solvency

A proof of solvency demonstrates that a financial institution has sufficient assets to satisfy its liabilities. This proof is generally broken into two components: a proof of liabilities and a proof of assets. The proof of liabilities ensures that the institution properly accounted for all customer-held funds; the proof of assets ensures that the institution has the private keys to spend its assets. Finally, the proof of solvency ensures that the amount from the proof of assets is equal to or greater than the amount from the proof of liabilities.

3.1.1 Maxwell Protocol

The Maxwell protocol (summarized in [12]) introduced a protocol for a proof of liabilities. It requires the institution to publish a Merkle tree. Each node on the tree contains a customer's balance and a customer-verifiable signature consisting of a nonce, the customer's balance, and the customer ID. Parent nodes sum together the balance of the children and concatenate the signatures. The root node contains the sum of all customer deposits, or institutional liabilities. This node is published in a well-known location.

For a customer to verify this proof of liabilities, the institution must reveal to the customer the customer's own ID, node nonce, and the hashes of the sibling nodes on the path between the customer's node and the root node. The customer then recomputes the Merkle tree hashes for the nodes on this path. So long as the signatures hold, then one knows that the institution properly included the node in the proof.

To prove sufficient assets, the institution demonstrates control of the private encryption keys that correspond to their virtual currency addresses. Methods to demonstrate control include moving well-selected amounts of virtual currencies from one address to another, or signing a message (such as the proof of liabilities) with the encryption keys. Such a proof is trivial to verify with public-key cryptography. One can compute the total assets by summing together the account balances that correspond to the addresses included in the proof. Account balances are public knowledge on the blockchain.

While simple to implement and efficient to verify, this protocol is weak and leaks information. Specifically, it requires each customer to verify that one's own balance was included, which novice users may not do. The protocol is also not foolproof. In the case of a disagreement about the customer's balance, it does not resolve whether the institution – or the customer – is right. In addition, the proof of liabilities reveals significant information about the institution – specifically, the number of customers and the balances of individual, anonymous sibling customers. The proof of assets requires institutions to publicize their account addresses, which can make them more susceptible to hacking, and to have the private keys available for each audit, which prohibits automated audits when using cold (offline) storage.

Notwithstanding these limitations, the Maxwell Protocol's simplicity enabled it to be used in practice. Kraken [21], a virtual currency exchange, used a deviation of this protocol in 2014 [24]. A trusted auditor computed the sum of Kraken's virtual currency reserves, and given a list of customer accounts and balances, verified that the total sum of customer deposits was approximately equal to the amount of the reserves. The auditor then published the root note of the Merkle tree used in the Maxwell protocol. Karekn offered users the amount reported to auditors, the hash function, and the hashes of adjacent nodes. Customers could then verify that their nodes' hashes are consistent with the root hash of the Merkle tree published by the auditor, and by extension, that their balances are properly included in the audit.

3.1.2 Provisions

Provisions [12] presents a privacy-preserving proof of solvency for Bitcoin exchanges. This protocol reveals neither customer balances nor the total liabilities of the institution. Moreover, it conceals exchanges' addresses and assets.

For the proof of liabilities, Provisions publishes a Pedersen commitment to each customer's balance, which customers can verify independently. The homomorphic sum of these commitments represents the cumulative liabilities of the institution. Provision's proof of assets utilizes anonymity sets, which contain all addresses that belong to the institution and "decoy" addresses that do not belong to the institution. Each address has a Pedersen commitment, which is to the address's balance only if the institution knows the corresponding private key. Otherwise, this commitment is to 0. NIZKs prove correct construction of these commitments and ensure that the "decoy" addresses do not contribute to the committed balance. The sum of these commitments forms the proof of assets. A zero-knowledge range proof (adapted from Mao [27]) between the proof of liabilities and assets completes the proof of solvency.

While Provisions introduced a novel, privacy-preserving approach, it has significant limitations. First, it only supports one blockchain; many institutions operate on multiple blockchains. Second, this protocol requires all customers to ensure that their balances are properly recorded in the proof of liabilities. Even if customers check their balances and notice a discrepancy, Provisions does not resolve who is correct: is it a greedy customer or a malicious exchange? Given the requirement for customer participation, it is impossible to resolve such a dispute in a privacy-preserving manner. These weaknesses render the protocol insufficient to replace the role of a traditional auditor.

3.2 Auditable Blockchains

zkLedger introduces a new blockchain that enables private transactions and audits on a public ledger [30]. It enables participants to transfer virtual currency among each-other without revealing transaction parties or amounts. The main contribution of zkLedger is the type of zero-knowledge audits it supports. Through generalized Schnorr proofs [34], participants on zkLedger can respond to complex auditing queries in a privacy-preserving manner.

Because of the overhead of on-chain zero-knowledge transactions, zkLedger throughput decreases significantly as the number of banks grows. With 10 banks, the throughput decreases to 2 transactions per second. In addition, zkLedger focuses on bank-level auditing and does not provide breakdowns for individual customers. Because of the limited transaction bandwidth, it is infeasible to represent each individual customer as a bank. Finally, zkLedger is incompatible with Bitcoin and Ethereum and cannot be used on top of existing blockchains. While an interesting design, zkLedger's limitations prevent its real-world use.

Chapter 4

System Design

We present the design of Sancus: a protocol for performing cryptographic audits for virtual currency institutions. Unlike other works, Sancus generates irrefutable proofs of solvency, and customers do not participate in the auditing process. In addition to a detailed description of Sancus, we briefly discuss the security and privacy of the design and offer techniques to mitigate its attack surface.

4.1 Features

4.1.1 Compatible with Bitcoin and Ethereum

Sancus supports the two largest blockchains by market capital, Bitcoin and Ethereum [10], and is extendable to other blockchains that use the SECP256K1 elliptic curve. For Ethereum, Sancus supports ETH and ERC-20 smart contract tokens [36].

4.1.2 Cold Wallets

To mitigate the risk of hacking, centralized virtual currency institutions store customer funds in offline, cold storage wallets. The private keys for such wallets are not stored on any internet-connected computer, thereby minimizing the attack surface. For example, Coinbase only stores 2% of virtual currencies in online accounts to support its typical transaction volume [9]. Should withdrawal requests exceed this figure, human intervention would be needed to transfer virtual currency into online wallets. Sancus supports cold-wallet storage and, unlike Provisions [12], does not require an online private key to prove solvency.

4.1.3 Privacy Preserving

Centralized virtual currency institutions mask the transaction patterns of their customers by issuing withdrawals from different accounts than those into which customers made deposits. As such, observers cannot distinguish individual customers. Moreover, because institutions store their funds in multiple accounts, blockchain users do not know the institutions' total assets, which may be considered proprietary.

Sancus offers strong privacy protections, where all account balances are encrypted via Pedersen commitments, and blockchain addresses are published as larger anonymity sets which provide plausible deniability. Sancus adapts the privacy-preserving techniques from Provisions [12].

4.1.4 Fully Accountable

Previous works, such as Provisions [12], provided a mechanism for proving the assets exchanges control are equal to or greater than the value of their self-reported customer deposits. However, this scheme does not address a scenario where an exchange and a customer disagree on one's balance. For example, an institution in Provisions could selectively ignore customer deposits, such that the reported liabilities would be artificially lower. The protocol does not provide a mechanism to resolve such a disagreement between a customer and an institution.

Sancus is fully accountable. It is impossible for an institution to hide customer deposits or steal customer funds without being detected. Unlike in previous works where customers participated in the auditing process and disputes would be left unresolved, Sancus does not require customer participation. As such, it produces irrefutable audits that fully account for all liabilities and assets of an institution.

4.2 Trust Model

Sancus applies a semi-honest trust model for institutions. In this trust model, institutions must follow the protocol, but they can attempt to cheat while doing so. Specifically, we assume that institutions would steal customer funds only if they would not be caught on any subsequent audit. However, because of the semi-honest trust model, we assume that they will always publish a properly-formatted audit on a regular cadence.

Auditors, who can be any member of the public, are assumed to be honest-but-curious. We assume that they will try to learn private information about the customers or institutions. Since anyone can verify the audit, we assume that auditors will validate the audit properly. One could easily check an auditor's work by validating the audit oneself.

Customers are not trusted. Sancus assumes that customers may provide malformed inputs or attempt to make illegal transactions (e.g. withdraw more funds than they have). The institution and auditors are responsible for validating all customer requests and rejecting those that are invalid.

4.3 Zero Knowledge Algorithms

Sancus defines five zero knowledge algorithms. They describe how to generate Pedersen commitments and accompanying NIZKs which are used throughout the audit.

All operations are in the SECP256K1 elliptic curve group $\mathbb{G}$ of prime order q with generator point $G \in \mathbb{G}$. $H \in \mathbb{G}$ denotes a second (constant) generator point, where it is presumed that the discrete logarithm between G and H is unknown. Proof statements are presented in Camenisch-Stadler notation [8]. For example, $PK\{(x) : Y = xG\}$ denotes a proof of knowledge of x given Y and G where $Y = xG$. We use the result of a proof statement to informally represent its non-interactive zero knowledge proof (NIZK), which can be serialized and later verified.

4.3.1 Key Permutation

Algorithm 1 picks a permutation r_{Y^*} and uses this value to permute a public key Y into Y^*. This algorithm also produces a zero knowledge proof of the permutation $NIZK_{Y^*}$. Proving $NIZK_{Y^*}$ follows from section 2.1 in Cramer et al. [11]. In addition, given the private key x corresponding to the public key Y (such that $Y = xG$), it follows that the permuted private key $x^* = r_{Y^*}x$, and $Y^* = r_{Y^*}xG$.

Algorithm 1: Key Permutation Algorithm

```
**Input:** $Y \in \mathbb{G}$, the public key
**Output:** $Y^*$, the permuted public key
**Output:** $r_{Y^*}$, the permutation
**Output:** $NIZK_{Y^*}$, the NIZK
begin
    $r_{Y^*} \xleftarrow{\$} \mathbb{Z}_q$;
    $Y^* \leftarrow r_{Y^*}Y$;
    $NIZK_{Y^*} \leftarrow PK\{(r_{Y^*}) : (Y^* = r_{Y^*}Y)\}$;
end
```

4.3.2 Key Bit

Algorithm 2 binds a binary value s with an elliptic curve point Y^* representing a permuted public key. It returns a binding Pedersen commitment L and a zero knowledge proof $NIZK_L$ demonstrating knowledge of s and that $s \in \{0, 1\}$. Proving $NIZK_L$ follows from Boudot [6], section 1.2.1. In Sancus, Y^* is always a permuted public key from the key permutation algorithm (algorithm 1).

4.3.3 Currency Conversion

Algorithm 3 uses a publicly-known exchange rate $\frac{e_n}{e_d}$ to convert a Pedersen commitment C representing a value x in one currency into a Pedersen commitment C' representing an equivalent value $x' = x\frac{e_d}{e_n}$ in a second currency. This procedure produces a zero knowledge proof $NIZK_{C'}$ demonstrating such conversion was done correctly. Specifically, $NIZK_{C'}$ demonstrates that $x' \in (x\frac{e_d}{e_n} - 1; x\frac{e_d}{e_n}]$. Since $x' \in \mathbb{Z}_q$, there is only one possible value in that range.

```
Algorithm 2: Key Bit Algorithm
  Input: $Y^* \in \mathbb{G}$, the permuted public key
  Input: $s \in \{0,1\}$, the secret bit
  Output: $L$, a Pedersen commitment for $s$
  Output: $r_L$, the random value for $L$
  Output: $NIZK_L$, the NIZK
  begin
      $r_L \xleftarrow{\$} \mathbb{Z}_q$;
      $L \leftarrow sY^* + r_L H$;
      $NIZK_L \leftarrow PK\{(s, r_L) : (L = sY^* + r_L H) \wedge (s \in \{0,1\})\}$;
  end
```

The exchange rate $\frac{e_n}{e_d}$ represents the *cost* for one unit of the second currency in terms of the first currency; hence, one receives $\frac{e_d}{e_n}$ units of the second currency for each unit of the first currency. Proving $NIZK_{C'}$ follows from Camenisch et al. [7].

```
Algorithm 3: Currency Conversion Algorithm
  Input: $e = \frac{e_n \in \mathbb{Z}_q}{e_d \in \mathbb{Z}_q}$, the exchange rate as a fraction
  Input: $\{C, x, r_C\} | C = xG + r_C H$, the revealed Pedersen commitment for the
         currency being converted
  Output: $\{C', x', r'_C\}$, the revealed Pedersen commitment for the resulting
          currency
  Output: $NIZK_{C'}$, the NIZK
  begin
      $x' \leftarrow x\frac{e_d}{e_n}$;
      $r'_C \xleftarrow{\$} \mathbb{Z}_q$;
      $C' \leftarrow x'G + r'_C H$;
      $x^* \leftarrow e_d x - e_n x'$;
      $r^*_C \leftarrow e_d r_C - e_n r'_C$;
      $C^* \leftarrow x^* G + r^*_C H$;
      $NIZK_{C'} \leftarrow PK\{(x^*, r^*_C) : (C^* = e_d C - e_n C') \wedge (C^* =$
        $x^* G + r^*_C H) \wedge (x^* \in [0, e_n))\}$;
  end
```

4.3.4 Blockchain Balance

Algorithm 4 takes a value $b \in \mathbb{Z}_q$ and produces a Pedersen commitment P which is to $b \in \mathbb{Z}_q$ only if a discrete logarithm $x^* \in \mathbb{Z}_q$ for $Y^* \in \mathbb{G}$ is known; otherwise,

P is a Pedersen commitment to 0. It also returns a zero knowledge proof $NIZK_P$ demonstrating that P was constructed correctly. Specifically, this proof demonstrates that P is a Pedersen commitment to the balance b secured by key x only if the institution knows the permuted private key x^*. If the institution does not know x^*, it is guaranteed that P is a Pedersen commitment to 0. Proving $NIZK_P$ follows from protocol 1 in Provisions [12]. However, since Sancus uses permuted keys from algorithm 1 and previously computed key bit Pedersen commitments from algorithm 2 as inputs, the validator must also validate that these proofs hold and that the commitments match.

Algorithm 4: Blockchain Balance Algorithm

```
Input: $b \in \mathbb{Z}_q$, the value of a balance corresponding to $Y$
Input: $Y^*$, the permuted public key for $Y$
Input: $s \in \{0, 1\}$, the bit representing whether the permuted private key $x^*$
       for $Y^*$ is known
Input: $\hat{x^*} = sx^*$, the permuted private key $x^*$ (if $x^*$ is known); otherwise 0
Input: $\{L, r_L\}$ from the key bit algorithm (algorithm 2) for $Y^*, s$
Output: $P$, the Pedersen commitment for $sb$
Output: $r_P$, the random value for $P$
Output: $NIZK_P$, the NIZK for $P$
begin
    $r_P \xleftarrow{\$} \mathbb{Z}_q$;
    $P \leftarrow sbG + r_PH$;
    $NIZK_P \leftarrow PK\{(s, \hat{x^*}, r_P, r_L) : (P = sbG + r_PH) \wedge (L =$
      $sY^* + r_LH) \wedge (L = \hat{x^*}G + r_LH)\}$;
end
```

4.3.5 Less Than Equal

Algorithm 5 demonstrates that the value $x \in [0, 2^{254})$ in one Pedersen commitment C is less than or equal to the value $x' \in [0, 2^{254})$ in another Pedersen commitment C'. It returns a zero knowledge proof $NIZK_{C'}$. Proving $NIZK_{C'}$ follows from Camenisch et al. [7]. As $(0 - (2^{254} - 1)) \mod q > 2^{254}$, this algorithm is safe from modular arithmetic overflow.

Algorithm 5: Less Than Equal Algorithm

```
Input: $\{C, x, r_C\} | C = xG + r_C H$, the revealed Pedersen commitment for the
       smaller value
Input: $\{C', x', r'_C\} | C' = x'G + r'_C H$, the revealed Pedersen commitment for
       the equal or larger value
Precondition : $x \in [0, 2^{254})$
Precondition : $x' \in [0, 2^{254})$
Output: $NIZK_{C'}$, the NZIK
begin
    $x^* \leftarrow x' - x$ ;
    $r^* \leftarrow r' - r$ ;
    $C^* \leftarrow C' - C$ ;
    $NIZK_{C'} \leftarrow PK\{(x^*, r^*) : (C^* = C' - C) \wedge (C^* = x^*G + r^*H) \wedge (x^* \in$
      $[0, 2^{254})\}$;
end
```

4.4 System Components

4.4.1 Blockchain Nodes, IPFS Nodes, and the Audit Smart Contract

Sancus requires nodes for Bitcoin [29], Ethereum [41], and the InterPlanatery File System (IPFS) [3] which are accessible to both the auditor and the institution. In a live deployment, the institution and each auditor should run their own nodes, as it is unsafe to trust the output of remote procedure calls from public nodes. Figure 4-1 illustrates how these components interact.

The institution publishes completed audits to the audit smart contract which has a well-known location on the blockchain. The smart contract guarantees that audits are immutable, and to prevent forgeries, only permits the institution to submit audits. As blockchain storage is expensive, Sancus stores the actual audit data files on the InterPlanatery File System (IPFS), an immutable, content-addressable file system [3]. The 32-byte IPFS address is a SHA-256 based hash of the content. Recording the IPFS address in a blockchain both timestamps the audit and prevents the institution from later changing the audit. IPFS allows anyone to host any content, providing resiliency should the institution's IPFS node be unavailable. The audit contains public

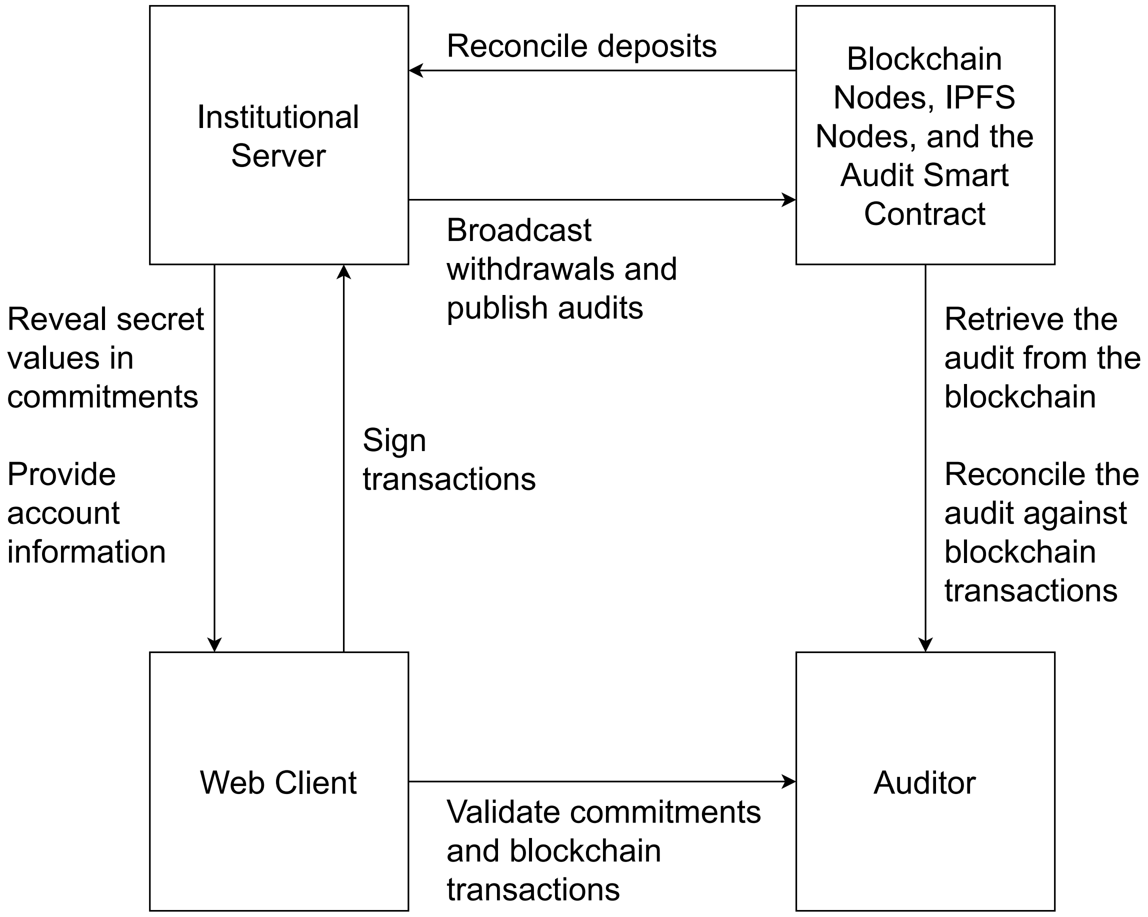

Institutional Server
Reconcile deposits
Blockchain Nodes, IPFS Nodes, and the Audit Smart Contract
Broadcast withdrawals and publish audits
Reveal secret values in commitments
Provide account information
Sign transactions
Retrieve the audit from the blockchain
Reconcile the audit against blockchain transactions
Web Client
Validate commitments and blockchain transactions
Auditor

Figure 4-1: System Architecture.

values in clear-text, Pedersen commitments of secret values, and accompanying zero knowledge proofs. Auditors “listen” to this smart contract for audits.

4.4.2 Institutional Web Server

The institution runs a web server. This web server exposes remote procedure calls (RPCs) to the web client that customers can invoke, communicates with the Bitcoin and Ethereum nodes to process and broadcast transactions, computes audits, stores finished audits on IPFS, and broadcasts audits via the audit smart contract. All web server / client communication is encrypted via Transport Layer Security (TLS).

4.4.3 Web Client

The web client runs on customers’ devices and allows them to use Sancus. It communicates with both the institutional web server to perform transactions and retrieve account information and one or more auditors who can help validate responses provided by the institutional web server. As the web client manages all customer signatures (via WebAuthn), customers are responsible for ensuring the integrity of the web client they are running and that they are communicating with honest auditors. To minimize the attack surface of a malicious web client (e.g. that sends different values to the server than are presented to the user) and auditors (e.g. that return incorrect answers), one can audit the web client source code, build it oneself, run one’s own auditor, and configure the web client to communicate with one’s own auditor. Customers should not blindly use a web client or auditor provided by the institution.

4.4.4 Auditor

The auditor validates audits published by the institution on the blockchain and exposes a limited API for web clients. As the audit data is self-contained and hosted on IPFS, the auditor does not communicate with the institution, customers, or other auditors to retrieve or validate the audit. (However, it will need to retrieve the audit data from another IPFS node, which may be run by the institution, another auditor, or some other node with the content for the audit's IPFS address.)

4.5 System Parameters

4.5.1 Base Currency

The institution must pick a base currency into which commitments in other currencies are converted when computing solvency.

4.5.2 Alternative Generator Point

In a Pedersen commitment $C = xG + rH$ where $x \in \mathbb{Z}_q$ is the secret value being committed and $r \xleftarrow{\$} \mathbb{Z}_q$ is the binding random value, $H \in \mathbb{G}$ must be well-selected as to have a unknown discrete logarithm with G, the generator point of the SECP256K1 elliptic curve.

4.5.3 Currency Precision

Each currency in Sancus can have its own precision (number of supported decimal places). While new currencies can be added after initialization, the number of decimal places for currencies must never change.

4.5.4 Maximum Balance per User and Maximum Number of Users

The product of the maximum balance per user and maximum number of users must be less than 2^{254} due to the input constraints of algorithm 5. For efficient zero knowledge proofs, Sancus requires that the maximum balance per user is on $[0, 2^n) \wedge (n \in \mathbb{Z}) \wedge (n \in [0, 254])$. It follows that the maximum number of users is the integer quotient of $\frac{2^{254}-1}{2^n-1}$. We selected $n = 127$ which allows for $\log_{10}(2^{127}) \approx 38$ decimal digits of currency precision and $\frac{2^{254}-1}{2^{127}-1} \approx 10^{38}$ users.

4.5.5 Exchange Rate Tolerance

When verifying an audit, auditors will validate exchange rates. As there is no centralized virtual currency market, the tolerance specifies the allowable difference between the rates used by the institution and found by the auditor. A smaller tolerance would prevent the institution from cheating by using exchange rates that make it appear to be more solvent; however, too small a tolerance could cause audits to fail due to fluctuations across the world's virtual currency markets.

4.5.6 Audit Smart Contract

The audit smart contract must be at a well known address on the blockchain.

4.6 Data Models

Sancus introduces data models which are shared across the institutional web server, customer-facing web client, and the auditor. An audit is identified by a monotonically-increasing version number and contains all information that any auditor needs from the institution to verify its solvency at a point in time. Audits contain instances of other data models produced by operations as described in section 4.7. As audits are cumulative, they do not include records that have been included in a previous audit. Each audit contains the following metadata: timestamp when the audit was generated, the block numbers on each supported blockchain for the block at or after the timestamp, and the exchange rates between supported currencies at that timestamp.

A `User` represents a customer using Sancus and has a unique identifier and an associated username.

A `UserKey` has a unique identifier and represents a WebAuthn credential associated with a specific `User`. Each `User` may have only one `UserKey`.

An `Account` has a unique identifier, an associated currency, an `AccountType` enum, and an associated `User`. The only supported `AccountTypes` are deposit accounts; however, this field is provided for extensions of Sancus. Each `User` may have multiple

`Accounts`.

A `Key` has a unique identifier and represents a SECP256K1 key pair. From the public key, it is possible to derive the corresponding Bitcoin and Ethereum addresses. Each `Key` has an associated key permutation NIZK from algorithm 1 and a Pedersen commitment to an “ownership” bit (and a corresponding key bit NIZK from algorithm 2) representing whether the institution knows the private key for this key pair. Private keys and the ownership bit are never published in the audit; however, the accompanying Pedersen commitments and NIZKs are included.

A `KeyAccount` associates a `Key` with an `Account`. It contains a block number at which this `KeyAccount` becomes active and a “credit” bit representing whether future deposits made to the `Key`’s address in the `Account`’s currency will be deposited into the `Account`. Each `KeyAccount` has a key bit NIZK (from algorithm 2) for this bit. Each `Account` may have multiple `KeyAccounts`, and each `Key` may also have multiple `KeyAccounts`. However, each `Key` cannot have more than one `KeyAccount` per currency (determined by the associated `Account`’s currency) where this “credit” bit is set to 1. Moreover, should the `Key`’s “ownership” bit be set to 0, then all “credit” bits in `KeyAccounts` associated with this `Key` must also be set to 0. It is impossible for the institution to give “credit” for `KeyAccounts` where it does not “own” the underlying `Keys`. The “credit” bit is never published in the audit; however, the accompanying Pedersen commitments and NIZKs are included.

An `AccountDelta` represents applying a debit or credit on an `Account`. For example, an `AccountDelta` for $-x$ represents a withdrawal for x. While an `AccountDelta` applies to a specific `Account`, each `Account` can have many `AccountDeltas`. The audit includes a Pedersen commitment to the amount, never the amount itself.

An `UnsignedBlockchainTransaction` identifies no more than one blockchain transaction. During a withdrawal, the institution provides these identifiers to the user to demonstrate which transactions will be performed. It includes all fields needed to process a transaction, except for the signature. It is unsafe to share a signed transaction with a customer, as one could broadcast it preemptively.

An `AccountDeltaGroup` represents an exchange or withdrawal of virtual currency.

`AccountDeltaGroup`s contain `AccountDelta`s which a customer requested be applied on their accounts, and for withdrawals, `UnsignedBlockchainTransaction`s. All `AccountDelta`s must be applied atomically on the `Accounts`, all `Accounts` must belong to the same `User`, and if present, all `UnsignedBlockchainTransaction`s must eventually be included in the blockchain. Customers sign `AccountDeltaGroup`s with their WebAuthn credentials to indicate their approval.

4.7 Client-Server Operations

Sancus introduces web server and client operations for customer registration, account creation, deposits, exchanges, and withdrawals.

4.7.1 Customer Registration

Customer registration occurs when a new customer signs up for Sancus. Sancus follows the WebAuthn registration protocol [37]. Upon successful registration, the server creates a `User` and `UserKey`, which are included in the next audit.

4.7.2 Account Creation

Customers can request new `Account`s from the server. When the server creates a new `Account`, it includes the `Account` in the next audit.

4.7.3 Deposits

Deposits are transfers of virtual currency from external blockchain addresses into the addresses maintained by Sancus on behalf of customers. The audit proves that all funds deposited into customer addresses are credited to the proper `Accounts`. This guarantee prevents a malicious institution from reducing their liabilities by undercounting customer deposits.

To create a new deposit address for an `Account`, the server creates a `KeyAccount` for this `Account` where the "credit" bit is 1. For this `KeyAccount`, the server may use

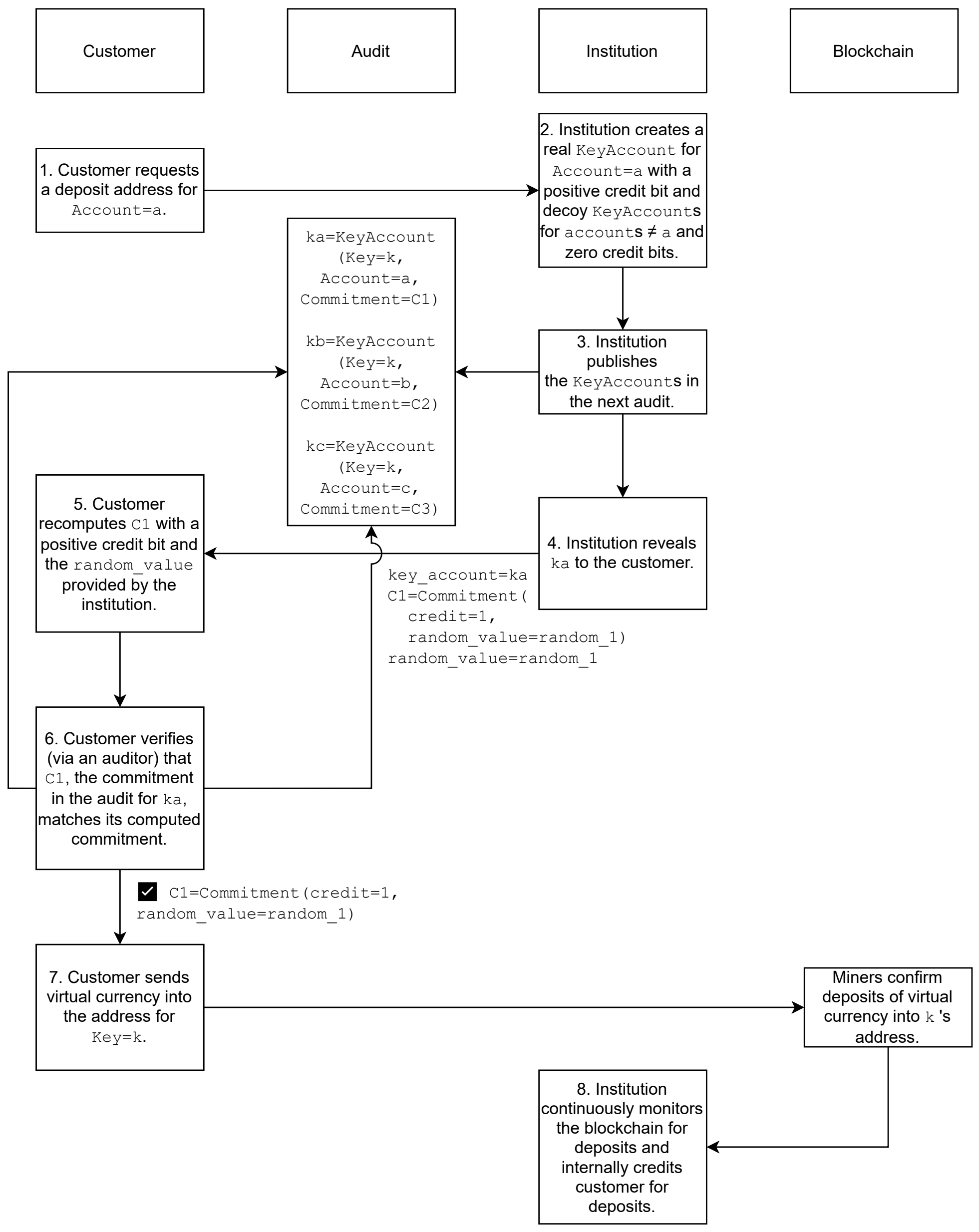

Customer
Audit
Institution
Blockchain
1. Customer requests a deposit address for Account=a.
2. Institution creates a real KeyAccount for Account=a with a positive credit bit and decoy KeyAccounts for accounts ≠ a and zero credit bits.
ka=KeyAccount (Key=k, Account=a, Commitment=C1)
kb=KeyAccount (Key=k, Account=b, Commitment=C2)
kc=KeyAccount (Key=k, Account=c, Commitment=C3)
3. Institution publishes the KeyAccounts in the next audit.
4. Institution reveals ka to the customer.
key_account=ka
C1=Commitment(credit=1, random_value=random_1)
random_value=random_1
5. Customer recomputes C1 with a positive credit bit and the random_value provided by the institution.
6. Customer verifies (via an auditor) that C1, the commitment in the audit for ka, matches its computed commitment.
✓ C1=Commitment(credit=1, random_value=random_1)
7. Customer sends virtual currency into the address for Key=k.
Miners confirm deposits of virtual currency into k 's address.
8. Institution continuously monitors the blockchain for deposits and internally credits customer for deposits.

Figure 4-2: Deposits Procedure. This diagram illustrates the process for creating new deposit addresses and crediting deposits, as discussed in section 4.7.3.

an existing `Key` or create a new one. The server also creates additional `KeyAccount`s for the same `Key` as the first `KeyAccount`. These additional `KeyAccount`s have "credit" bits set to 0 and are used to preserve privacy (see section 4.12.1). The server publishes any new `Key`s and the new `KeyAccount`s in the next audit.

The Pedersen commitment for the first `KeyAccount` is revealed to the client. The client uses this revealed secret to ensure that the Pedersen commitment for the "credit" bit for this `KeyAccount`, as it appears in the audit, has a value of 1. An auditor's Get Key Account RPC (see section 4.10.1) provides this functionality.

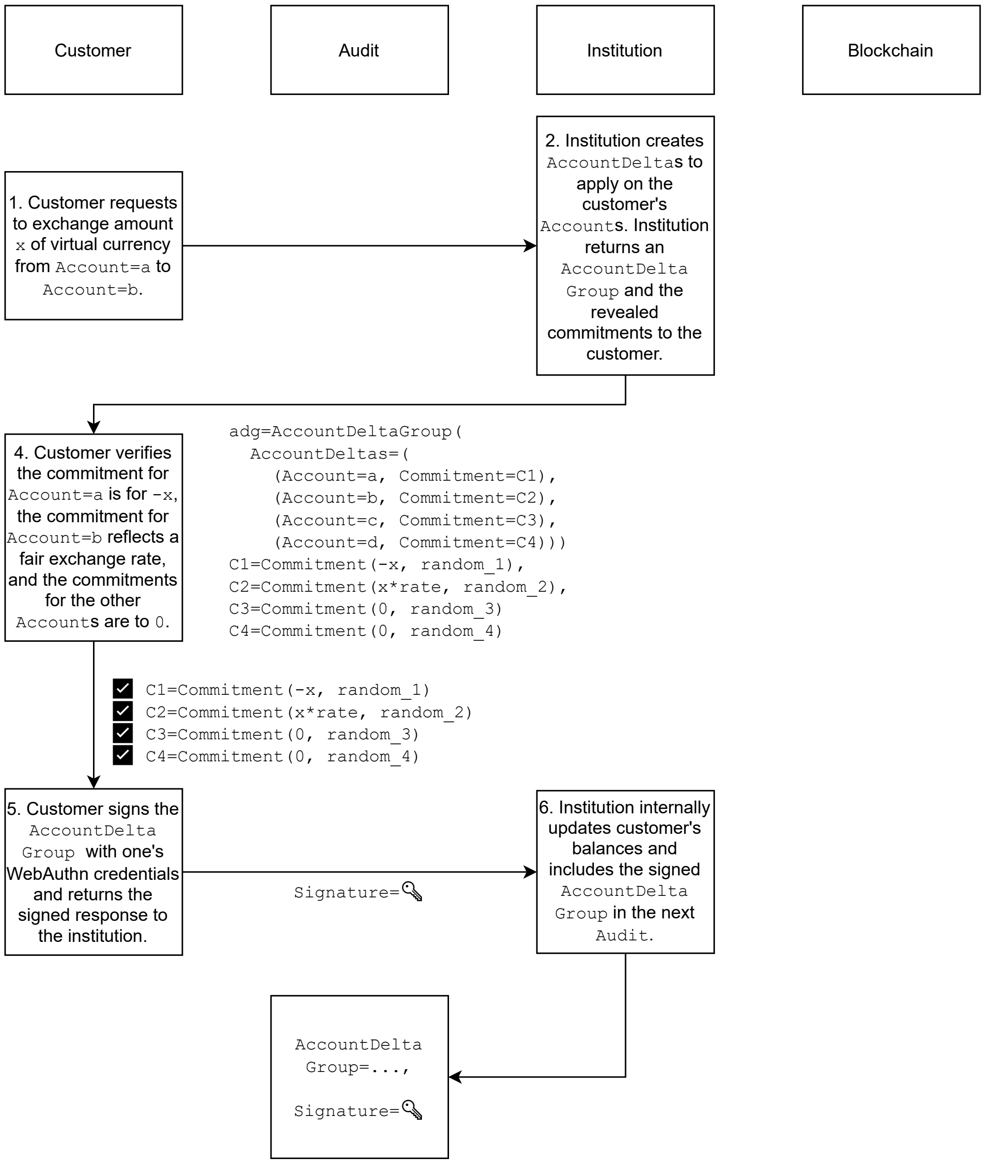
After the client validates that the audit contains a properly constructed `KeyAccount`, the client can deposit virtual currency into the address associated with its `Key`. It is crucial that the client waits until *after* the audit has been published before making a deposit. Only deposits made to addresses covered by `KeyAccount`s with "credit" bits set to 1 will be accumulated. The institution could attempt to under-count deposits and hide assets by tricking customers into sending virtual currency to addresses not covered by such `KeyAccount`s. For example, it could reveal one Pedersen commitment to the client but publish another one in the audit. However, because audits are cumulative and `KeyAccount`s are immutable, the institution cannot later change the commitments or NIZKs in published audits.

Figure 4-2 illustrates the deposit procedure.

4.7.4 Exchanges

Customers can exchange one virtual currency for another. Unlike deposits, exchanges do not take place on the blockchain; they are journal transactions within Sancus. However, because exchanges decrease customer liabilities in one currency, customers must demonstrate their approval of such transactions via WebAuthn signatures.

To perform an exchange, the client requests to exchange amount x of virtual currency from `Account` A into the currency of `Account` B. Assuming the customer has sufficient funds, the server responds with an `AccountDeltaGroup` that contains `AccountDelta`s of $-x$ for `Account` A, of an equivalent amount of x in B's currency for `Account` B, and of 0 for additional `Account`s, which are used to preserve privacy



Customer
Audit
Institution
Blockchain
1. Customer requests to exchange amount x of virtual currency from Account=a to Account=b.
2. Institution creates AccountDeltas to apply on the customer's Accounts. Institution returns an AccountDelta Group and the revealed commitments to the customer.
4. Customer verifies the commitment for Account=a is for -x, the commitment for Account=b reflects a fair exchange rate, and the commitments for the other Accounts are to 0.
adg=AccountDeltaGroup(
AccountDeltas=(
(Account=a, Commitment=C1),
(Account=b, Commitment=C2),
(Account=c, Commitment=C3),
(Account=d, Commitment=C4)))
C1=Commitment(-x, random_1),
C2=Commitment(x*rate, random_2),
C3=Commitment(0, random_3)
C4=Commitment(0, random_4)
✓ C1=Commitment(-x, random_1)
✓ C2=Commitment(x*rate, random_2)
✓ C3=Commitment(0, random_3)
✓ C4=Commitment(0, random_4)
5. Customer signs the AccountDelta Group with one's WebAuthn credentials and returns the signed response to the institution.
Signature=
6. Institution internally updates customer's balances and includes the signed AccountDelta Group in the next Audit.
AccountDelta Group=...,
Signature=

Figure 4-3: Exchanges Procedure. This diagram illustrates the process for exchanging virtual currencies, as discussed in section 4.7.4.

(see section 4.12.3). The server reveals all Pedersen commitments privately to the client. The client, using the revealed commitments from the server, verifies that the institution constructed an `AccountDeltaGroup` with proper Pedersen commitments. Specifically, the Pedersen commitment for `Account` A should be to $-x$, `Account` B should be for an equivalent amount of x at a fair exchange rate, and for all other `Accounts`, should be to 0. If the client wishes to proceed with the exchange, it signs the `AccountDeltaGroup` with its WebAuthn credentials and returns the signature to the server. The server verifies the signature, updates customer balances by the amounts of the `AccountDeltas`, and includes the `AccountDeltaGroup` in the next audit.

Figure 4-3 illustrates this procedure for performing exchanges.

4.7.5 Withdrawals

Withdrawals are transfers of virtual currency from institutional addresses maintained by Sancus to external addresses specified by the customer. To maintain customer privacy, withdrawals need not come from the same virtual currency addresses into which customers originally deposited their funds. As withdrawals affect customer balances, Sancus requires customers to sign withdrawal requests to reflect their legitimacy. These withdrawal requests reveal neither customer balances, firm balances, nor transaction amounts, and provide plausible deniability for blockchain addresses.

To perform a withdrawal, the client requests to withdraw amount x of virtual currency from `Account` a to a destination address w. If the user has sufficient funds, the server responds with an `AccountDeltaGroup`. In the group, the `AccountDeltas` must include a Pedersen commitment to $-x$ for `Account` a, and for privacy, additional "decoy" `Accounts` with Pedersen commitments to 0. The server reveals all Pedersen commitments privately to the client. The `UnsignedBlockchainTransactions` must contain a transaction which sends x to w. For privacy, the server should also include "decoy" transactions that move funds within the institution's own accounts or are recent blockchain transactions. The client verifies that the institution constructed a proper `AccountDeltaGroup`. Specifically:

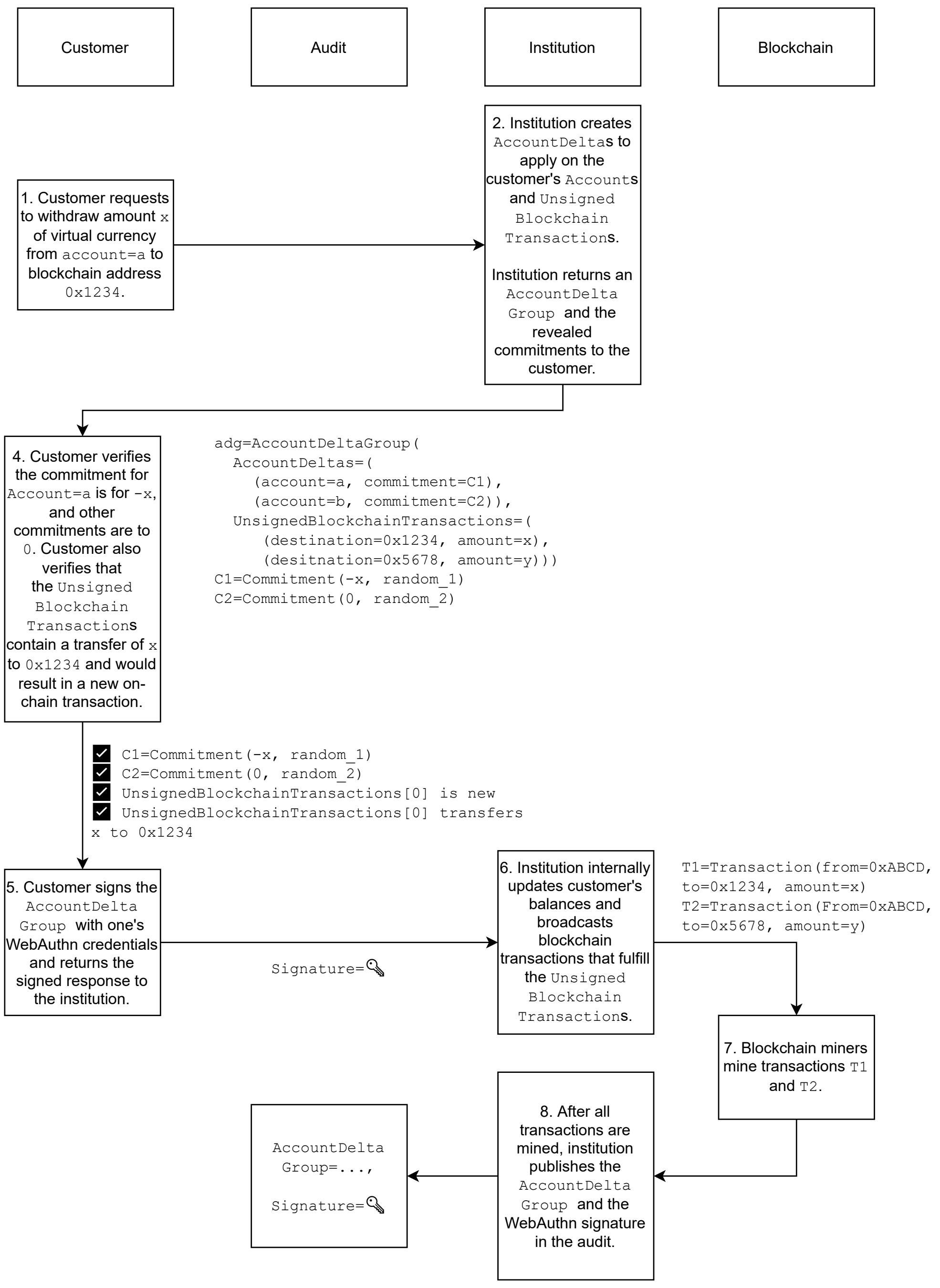

Customer
Audit
Institution
Blockchain
2. Institution creates AccountDeltas to apply on the customer's Accounts and Unsigned Blockchain Transactions.
Institution returns an AccountDelta Group and the revealed commitments to the customer.
1. Customer requests to withdraw amount x of virtual currency from account=a to blockchain address 0x1234.
4. Customer verifies the commitment for Account=a is for -x, and other commitments are to 0. Customer also verifies that the Unsigned Blockchain Transactions contain a transfer of x to 0x1234 and would result in a new on-chain transaction.
adg=AccountDeltaGroup(
AccountDeltas=(
(account=a, commitment=C1),
(account=b, commitment=C2)),
UnsignedBlockchainTransactions=(
(destination=0x1234, amount=x),
(desitnation=0x5678, amount=y)))
C1=Commitment(-x, random_1)
C2=Commitment(0, random_2)
✓ C1=Commitment(-x, random_1)
✓ C2=Commitment(0, random_2)
✓ UnsignedBlockchainTransactions[0] is new
✓ UnsignedBlockchainTransactions[0] transfers x to 0x1234
5. Customer signs the AccountDelta Group with one's WebAuthn credentials and returns the signed response to the institution.
Signature=
6. Institution internally updates customer's balances and broadcasts blockchain transactions that fulfill the Unsigned Blockchain Transactions.
T1=Transaction(from=0xABCD, to=0x1234, amount=x)
T2=Transaction(From=0xABCD, to=0x5678, amount=y)
7. Blockchain miners mine transactions T1 and T2.
8. After all transactions are mined, institution publishes the AccountDelta Group and the WebAuthn signature in the audit.
AccountDelta Group=...,
Signature=

Figure 4-4: Withdrawals Procedure. This diagram illustrates the process for withdrawing virtual currency, as discussed in section 4.7.5.

1. Using the revealed secret values from the server, the client verifies that the `AccountDelta` for `Account` a debits it by $-x$ and has commitments to 0 for all other `Accounts`.

2. The client ensures that the `UnsignedBlockchainTransactions` contain a transaction for sending x to w.

3. The client ensures that this `UnsignedBlockchainTransaction` which sends x to w would result in a new blockchain transaction. An auditor's Is Transaction New RPC (see section 4.10.2) provides this functionality.

If the client wishes to proceed with the transaction, it signs the `AccountDeltaGroup` with its WebAuthn credentials and shares the signature with the server. The server verifies the signature, updates customer balances to reflect the `AccountDeltas`, and broadcasts blockchain transactions consistent with the `UnsignedBlockchainTransactions` *in a random order* and *with random delays*. Randomness helps preserve privacy (see section 4.12.2). Finally, after all transactions are in the blockchain, the server includes the `AccountDeltaGroup` in the next audit.

It is critical that the client performs step 3. Without this check, if a user attempted to withdraw the same amount of virtual currency to the same address as was done in a previous withdrawal, then the institution could reuse the `UnsignedBlockchainTransaction` from this previous withdrawal. As this transaction was already in the blockchain, the customer would not receive new virtual currency in the destination address even though account balances would still be adjusted by the `AccountDeltas`. To prevent this attack, users must validate that the non-"decoy" transaction is new.

Figure 4-4 illustrates this procedure for performing withdrawals.

4.8 Audit Generation

The institution generates audits periodically. The audit demonstrates that the institution accounted for all deposits, applies only properly executed exchanges and withdrawals, and has at least as much in assets as in liabilities.

4.8.1 Audit Metadata

When generating an audit, the institution must pick a timestamp after the previous audit's timestamp, and use the block for each blockchain at or after this timestamp. The institution must also determine acceptable exchange rates between the base currency parameter and each supported currency at this timestamp. While Sancus does not specify a specific auditing interval or source for exchange rates, it is trivial for auditors to validate these conditions. Block timestamps are embedded in the blockchain, and protocols such as Compound [25] and Uniswap [1] freely publish historical exchange rates.

4.8.2 Proof of Liabilities

To create a commitment for the total liabilities, the institution must sum together all customer deposits, exchanges, and withdrawals. For every `KeyAccount`, the institution uses the blockchain balance algorithm (algorithm 4) to create a commitment and NIZK to the cumulative deposits. Cumulative deposits, which are measured from the block number of the `KeyAccount` through the block number of the audit, are defined as deposits made in the corresponding `Account`'s currency to the address spendable by the corresponding `Key`. The commitment and NIZK are included in the audit. The institution sums together these commitments by currency for each user. To account for exchanges and withdrawals, the institution sums in all `AccountDelta`s generated by the exchange and withdrawal procedures, respectively. To get the current balance, all `AccountDelta`s, including those that appeared in a previous audit, must be included. The currency conversion algorithm (algorithm 3) is used to convert these cumulative commitments (one per currency for each user) into the institution's base currency. The NIZKs and resulting commitments from currency conversion are included in the audit. These resulting commitments are summed together to produce one commitment per user. This commitment represents the user's holdings in the institution's base currency.

Users with negative balances must be excluded from the total liabilities commit-

Algorithm 6: User Deposits. This sub-procedure, invoked by algorithm 7, computes the cumulative deposits as part of the proof of liabilities (see section 4.8.2).

```
Input: auditBlockNumber[$c$] : $c \to n_c$, a mapping from currency $c$ to the
       audit block number $n_c$ for $c$
Input: User $u$
Output: $d[c] : c \to (x_c, r_c)$, a mapping from currency $c$ to the deposit
        amount $x_c$ and random secret $r_c$
begin
    foreach Currency $c$ do
        $d[c] \leftarrow (0, 0)$ ;  // Initialize deposit value and random secret
         by currency to 0
    end
    foreach Account $a$ belonging to $u$ do
        $c \leftarrow a$'s currency;
        foreach KeyAccount $ka$ with Key $k$ and Account same as $a$ do
            $startBlock \leftarrow$ block number of $ka$;
            $Y^* \leftarrow$ permuted public key for $k$;
            $b \leftarrow$ total deposits made into the blockchain address of $k$ in
             currency $c$ on [$startBlock$, auditBlockNumber[$c$]];
            $(s, L, r_L) \leftarrow$ "credit" bit and corresponding key bit commitment
             and random value from $ka$;
            if $s = 0$ then
                $\hat{x^*} \leftarrow 0$;
            end
            else
                $\hat{x^*} \leftarrow$ permuted private key $x^*$ for $k$;
            end
            $(r_P, NIZK_P) \leftarrow$ blockchain balance algorithm (algorithm 4) for
             balance $b$, permuted public key $Y^*$, secret bit $s$, permuted private
             key or zero value $\hat{x^*}$, key bit commitment $L$ and random value $r_L$;
            $(x_c, r_c) \leftarrow d[c]$;
            $d[c] \leftarrow (x_c + sb, r_c + r_P)$;
            Add $NIZK_P$ to the audit;
        end
    end
end
```

Algorithm 7: Proof of Liabilities. This procedure illustrates how to compute the proof of liabilities (see section 4.8.2).

```
Input: $bc$, the base currency parameter (see section 4.5.1)
Input: $maxBal$, the maximum balance per user parameter (see section 4.5.4)
Input: $e[c] : c \rightarrow e_c$, a mapping from currency $c$ to the audit's exchange rate
       $e_c$ for converting $c$ into $bc$
Output: $(x_l, r_l)$, the total liabilities and random secret
begin
    $(x_l, r_l) \leftarrow (0, 0)$ ;                  // Initialize total liabilities to 0
    foreach User $u$ do
        $d[c] \leftarrow$ user deposits algorithm (algorithm 6) for User $u$;
        foreach AccountDeltaGroup $ag$ belonging to $u$ not in any audit do
            if all UnsignedBlockchainTransactions in $ag$ have been included
             in the blockchain then
                foreach AccountDelta $ad \in ag$ with Account $a$ do
                    $(x_d, r_d) \leftarrow$ value and random secret of $ad$;
                    $c \leftarrow$ currency of $a$;
                    $(x_c, r_c) \leftarrow d[c]$;
                    $d[c] \leftarrow (x_c + x_d, r_c + r_d)$;
                end
            end
        end
        $(x_{bc}, r_{bc}) \leftarrow (0, 0)$ ;    // Initialize user liabilities in $bc$ to 0
        foreach Currency $c$ do
            $(C', x', r'_C, NIZK_{C'}) \leftarrow$ currency conversion algorithm (algorithm
             3) for exchange rate $e[c]$ and from currency value and random
             secret $d[c]$;
            $(x_{bc}, r_{bc}) \leftarrow (x_{bc} + x', r_{bc} + r'_C)$;
            Add $NIZK_{C'}$ to the audit;
        end
        if $x_{bc} < 0$ then                      // Check for negative balances
            $userNeg \leftarrow 1$;
            $x_{bc} \leftarrow x_{bc} + maxBal + 1$;
        end
        else
            $userNeg \leftarrow 0$;
            $(x_l, r_l) \leftarrow (x_l + x_{bc}, r_l + r_{bc})$;
        end
        $NIZK_{bc} \leftarrow$ less than equal algorithm (algorithm 5) proving
         $x_{bc}$ (with random secret $r_{bc}$) $\leq maxBal$ (with random secret 0);
        Add $userNeg, NIZK_{bc}$ to the audit;
    end
end
```

ment. Should negative user balances be included, an institution could under-count how much it owes to customers. To execute this attack, the institution could create its own `Users` where it controls the `UserKeys`. It could then run the exchange protocol (see section 4.7.4) to create properly-signed negative balance commitments. Summing these negative commitments into the institution's total liabilities would reduce this commitment and allow the institution to circumvent the audit. As such, no individual user is allowed to reduce the institution's total liabilities.

For each user, the institution performs the less than equal algorithm (algorithm 5) to prove that the user's balance commitment is either negative (i.e. greater than or equal to the maximum allowed user balance system parameter) or positive (i.e. less than this parameter). It includes in the audit whether the balance is negative and the corresponding NIZK from the less than equal algorithm. Non-negative balances are checked per-user, rather than per-account, both to reduce the number of required zero-knowledge proofs and to support future use cases, such as fully collateralized loans.

The sum of all users' balance commitments in the base currency represent the institution's cumulative liabilities. Algorithm 7 describes this proof of liabilities procedure.

4.8.3 Proof of Assets

The audit must include a commitment to the total assets for the institution. Sancus extends the proof of reserves described in Provisions [12] to support offline wallets with algorithm 1 and multiple currencies with algorithm 3 .

For the proof of assets, the institution runs the blockchain balance algorithm (algorithm 4) for each `Key` k in each supported currency c. When running this algorithm, s is set to the "ownership" bit of k. The balance b is the minimum of the current balance at the audit's block number for c and at the previous block.

Using the minimum of these balances prevents the institution from double-counting funds by using a third-party exchange. For example, suppose there were two currencies c_1 and c_2 on different blockchains (e.g. Bitcoin and Ethereum) with audit block

Algorithm 8: Proof of Assets. This procedure illustrates how to compute the proof of assets (see section 4.8.3).

```
Input: $bc$, the base currency parameter
Input: auditBlockNumber[$c$] : $c \rightarrow n_c$, a mapping from currency $c$ to the
       audit block number $n_c$ for $c$
Input: $e[c] : c \rightarrow e_c$, a mapping from currency $c$ to the audit's exchange rate
       $e_c$ for converting $c$ into $bc$
Result: $(x_a, r_a)$, total assets and random secret
begin
    $(x_a, r_a) \leftarrow (0, 0)$ ;                      // Initialize total assets to 0
    foreach Currency $c$ do
        $(x_c, r_c) \leftarrow (0, 0)$ ;            // Initialize assets by currency to 0
        foreach Key $k$ do
            $Y^* \leftarrow$ permuted public key for $k$;
            $b \leftarrow$ minimum balance of the blockchain address for $k$ in currency
              $c$ on blocks [auditBlockNumber[$c$] $- 1$, auditBlockNumber[$c$]];
            $(s, L, r_L) \leftarrow$ "ownership" bit and corresponding key bit
              commitment and random value from $k$;
            if $s = 0$ then
                $\hat{x^*} \leftarrow 0$;
            end
            else
                $\hat{x^*} \leftarrow$ permuted private key $x^*$ for $k$;
            end
            $(r_P, NIZK_P) \leftarrow$ blockchain balance algorithm (algorithm 4) for
              balance $b$, permuted public key $Y^*$, permuted private key (or zero
              value) $\hat{x^*}$, and secret bit $s$ with key bit commitment $L$ and
              random value $r_L$;
            $(x_c, r_c) \leftarrow (x_c + sb, r_c + r_P)$;
            Add $NIZK_P$ to the Audit;
        end
        /* Convert to base currency */
        $(C', x', r'_C, NIZK_{C'}) \leftarrow$ currency conversion algorithm (algorithm 3)
          with exchange rate $e[c]$ and from currency value $x_c$ and random
          secret $r_c$;
        $(x_a, r_a) \leftarrow (x_a + x', r_a + r'_C)$;
        Add $NIZK_{C'}$ to the audit;
    end
end
```

numbers b_1 and b_2, respectively. After b_1, the institution transfers funds from its addresses in c_1 to a third-party exchange, converts these funds from c_1 to c_2, and withdraws funds in c_2. Because blocks are not synchronized across blockchains, it is possible that such a withdrawal would be included in b_2. Should blockchain balances be computed at b_1 and b_2, the institution would double-count the funds it had in c_1 – once in c_1, and again in an equivalent amount of c_2. However, using the minimum balances on $[b_1 - 1, b_1]$ and $[b_2 - 1, b_2]$ prevents this attack. At the timestamp of the audit, which is guaranteed to be in these intervals by definition of how audit block numbers are selected, it is impossible for the same funds to simultaneously be in two currencies at once.

The institution accumulates, by currency, the resulting Pedersen commitments from running the blockchain balance algorithm. It uses the currency conversion algorithm (algorithm 3) to convert these commitments into the base currency, which are summed into one Pedersen commitment for the total assets.

Algorithm 8 describes this proof of assets procedure.

4.8.4 Proof of Solvency

To demonstrate solvency, the institution runs the less than equal algorithm (algorithm 5) to prove that the commitment from the proof of liabilities procedure (see section 4.8.2) is less than or equal to the commitment from the proof of assets procedure (see section 4.8.3). The resulting proof from this call to the less than equal algorithm is included in the audit.

4.9 Auditor Verification

The novelty of Sancus is that it allows anyone to independently verify the validity of an audit and validate the solvency of an institution. Auditors do not directly communicate with the institution or any customers; all an auditor needs is the audit itself (and all previous audits), access to the blockchains, and a trusted source for virtual currency exchange rates (which could be on the blockchain, such as via Compound

[25] or Uniswap [1]).

As audits are cumulative, auditors must have already processed and verified all previous audits. Should any step of audit verification fail for any reason, the auditor reports that the institution failed.

The auditor monitors the audit smart contract for new audits. When one is available, the auditor retrieves it from IPFS using the IPFS address contained within the smart contract. As all blockchain transactions are signed, it is trivial for the auditor to verify that the institution published the audit. IPFS addresses, which are cryptographic hashes of the underlying content, ensure the integrity of the data. As audits are cumulative, uniqueness and existence checks must be run against the current audit and all previous audits. Each audit contains `AuditMetadata`, new entries, a proof of liabilities, a proof of assets, and a proof of solvency.

To verify the `AuditMetadata`, the auditor checks that the version numbers are monotonically increasing and that the timestamp is increasing relative to the previous audit. It also ensures that all block numbers correspond to a block with a timestamp at or after the `AuditMetadata`'s timestamp. The auditor compares the metadata exchange rates to the historical rates (as provided by a trusted source) at the `AuditMetadata`'s timestamp. It ensures that these rates are within the exchange rate tolerance.

To verify the `Users`, `UserKeys`, `Accounts`, `Keys`, `KeyAccounts`, and `AccountDeltaGroups` in the audit, the auditor validates that they are unique, that referenced components exist (e.g. for an `Account`, that the associated `User` exists), and that all zero knowledge proofs hold. In the case of `AccountDeltaGroups`, the auditor also verifies the WebAuthn signature and ensures that each `UnsignedBlockchainTransaction` has a "matching" transaction in the blockchain by the block number of the audit. For Ethereum, a "matching" transaction is one with the same `from`, `nonce`, `to`, `value`, `gas`, and `data` attributes. For Bitcoin, a "matching" transaction is one where the blockchain transaction's sources and destinations are both a super-set of those in the audit. These checks ensure that institution broadcasts transactions as promised while providing flexibility for transaction fees.

4.9.1 Proof of Liabilities

The auditor verifies the institution's proof of liabilities by checking each step described in section 4.8.2. Specifically, it must check that the institution includes a blockchain balance commitment and NIZK for every known `KeyAccount`, that all proofs hold, and that all commitments are consistent with each-other, blockchain data, and other components of the audit. Skipping `KeyAccount`s would allow the institution to undercount deposits.

4.9.2 Proof of Assets

The auditor verifies the institution's proof of assets by checking each step described in section 4.8.3. Specifically, for every blockchain balance NIZK generated by the institution, the auditor ensures that the NIZKs use correctly selected blockchain balances. The auditor also verifies the summation and conversion of these commitments into the institution's base currency. The resulting commitment is to the institution's total assets.

4.9.3 Proof of Solvency

The auditor verifies the institution's proof of solvency using the resulting commitments from the proof of liabilities and proof of assets procedures. Specifically, given these commitments, it verifies that the less than equal NIZK published by the institution for the proof of solvency (as described in section 4.8.4) holds.

4.10 Auditor Remote Procedure Calls

Auditors offer remote procedure calls (RPCs) which web clients query to validate values provided by the institution. The web clients cannot directly perform validations that require previous audits or indexed blockchain transactions.

4.10.1 Get Key Account

This RPC allows the web client to get the `KeyAccount` commitment and deposit address for a given `Key` ID and `Account` ID. The auditor returns the “credit” Pedersen commitment for the `KeyAccount` that was included in any processed audit, or an exception if it was not found. The client can then validate that the “credit” commitment privately revealed by the institution during the deposit operation (see section 4.7.3) is consistent with the commitment in the audit. The client must perform this check to ensure it gets credit for deposits sent to the corresponding address.

4.10.2 Is Transaction New

This RPC allows the client to check that an `UnsignedBlockchainTransaction` would result in a new on-chain transfer. The auditor may return true only if there is not any transaction in the blockchain that matches the `UnsignedBlockchainTransaction`. This check allows the client to ensure that a withdrawal (see section 4.7.5) would result in a new blockchain transaction. Otherwise, should a customer request a withdrawal for an amount and destination address that matches a previous blockchain transaction, the institution could cheat by presenting this previous transaction, with which the customer may not receive new funds.

4.11 Security Analysis

The Sancus protocols are designed to provide a secure means for customers to transact with the institution while also preventing the institution from passing audit verification while cheating. Here, we analyze possible attack vectors and their defenses.

4.11.1 Delay or Refusal to Process Transactions

Sancus does not require the institution to process transactions. While the protocol for counting deposits will ensure that all deposits made into addresses with proper `KeyAccounts` will be counted on the next audit, the institution is allowed to refuse to perform exchanges or withdrawals at any stage in the process. For example, the institution could refuse to offer an exchange between virtual currencies, offer an exchange that reflects an unattractive exchange rate, or never include an `AccountDeltaGroup` in an audit. Section 7.3 offers mitigation strategies.

4.11.2 Loss of Account Access

Should users lose their WebAuthn credentials, Sancus does not define a procedure for replacing these lost credentials so users can regain access to their accounts. A traditional password reset system, where users click links sent to their email, would not be verifiable by the auditors. To mitigate this concern, Sancus can be extended to allow users to have multiple `UserKey`s, where new `UserKey`s are signed by already-authorized `UserKey`s or by recovery codes (such as secret questions and answers) committed to during customer registration.

4.11.3 Loss of Keys

Sancus uses permuted key pairs to support cold wallets, a security best practice. However, permuted key pairs are not foolproof. Auditors only know that the institution knew the original key pair when the permuted key pair was generated; audits do not prove that the institution *still* knows the original private key needed to spend virtual currency secured by it. While possible, it is unlikely for an institution to lose its private keys. Nonetheless, to mitigate this concern, auditors could require that the permuted key pairs be rotated periodically (e.g. every n audits).

Hacks of internet connected systems, however, are more likely. Specifically, consider the scenario where adversaries acquired permuted private keys. They would learn whether an institution controls ownership of the corresponding blockchain addresses. This leak would compromise privacy. However, they would be unable to steal any virtual currency, since it is impossible to recover the original private key from the permuted private key.

4.11.4 Collusion Across Institutions

Multiple institutions implementing Sancus could attempt to collude to make each appear more solvent than they are individually. One such attack would be generating permuted key pairs for each other and sharing the permuted private keys. With permuted private keys, institutions could set the "ownership" bit of a `Key` to 1 and pass the audit, even though funds stored in such `Key`s are inaccessible. This collusion would allow institutions to inflate their proof of assets. To mitigate this attack, the proof of non-collusion (section 7) from Provisions [12] can be adapted and applied on the non-permuted keys. Since this proof would need to be computed only once per key, it would be compatible with Sancus's support for cold wallets.

Another such attack would involve institutions borrowing funds from each other. Suppose there are two institutions, and neither could pass an audit independently due to a lack of funds, but combined, they both could pass. These institutions could collude by first transferring all assets into one institution and then having that institution generate an audit. After the audit, that institution transfers all its funds into the other institution, and this institution now generates an audit. This cycle can repeat indefinitely, and both institutions would appear solvent. To prevent this attack, Sancus could dictate a universal schedule specifying when audits should be generated. Blockchain funds can only belong to one address at any timestamp.

4.12 Privacy Analysis

Sancus never reveals account balances and transaction amounts through the deposit, exchange, and withdrawal protocols. However, honest-but-curious auditors could attempt to infer such information from blockchain transactions and the blockchain transaction graph. Here, we describe what information is leaked and suggest general techniques to mitigate such leakage.

4.12.1 Deposit Addresses

Through the `Key` and `KeyAccount` audit messages, the institution publishes a superset of the keys it controls and the keys assigned to accounts. `Keys` with a "ownership" bit of 0 and `KeyAccounts` with "credit" bit of 0 are decoys and do not affect balances; instead, they provide plausible deniability for the public keys that are associated with the institution or particular accounts. The institution must add a sufficient number of decoy `Keys` and `KeyAccounts` to prevent a curious auditor from uncovering with high probability which ones are not decoys.

4.12.2 Withdrawal Anonymity Set

The `AccountDeltaGroups` for withdrawals contain multiple `UnsignedBlockchainTransactions`. Including internal transfers between the institution's own addresses or unrelated, but recent, blockchain transactions hides the customer's destination address and the amount being withdrawn. A curious auditor cannot immediately identify the customer's withdrawals from the decoys; however, analysis of the blockchain transaction graph may leak such information. Leakage would reveal how much and to where a customer withdrew virtual currency.

4.12.3 Transaction History

The audit reveals whenever a customer made an exchange or withdrawal. By including `AccountDeltas` that have commitments to 0 in each `AccountDeltaGroup`, the institution hides which accounts, and thereby currencies, were involved in the transaction. However, auditors would be able to identify more active `Users` by the frequency of their `AccountDeltaGroups`.

4.12.4 Insolvent Users

During the proof of liabilities, the audit reveals which customers have negative balances. However, the actual account balances remain private.

4.12.5 Auditor RPCs

When clients query auditors, they should invoke additional "decoy" RPCs. These extra requests will mask the `KeyAccount`s or `UnsignedBlockchainTransaction`s of interest from a curious auditor who logs RPC calls.

Chapter 5

Implementation

We implemented Sancus following the design described in chapter 4. Our implementation uses a Python server for the institution, a Python server for the auditor, and a React-based JavaScript web client.

5.1 Blockchains and Currencies Supported

Sancus provides a modular interface for adding blockchains. We implemented support for Bitcoin and Ethereum. For Ethereum, we support both ETH and ERC-20 tokens [36]. Each currency can have a customizable number of decimal places: we selected 8 for Bitcoin (smallest unit = 1 Satoshi); 18 for Ethereum (smallest unit = 1 Wei), and for our mock ERC-20 smart contract representing a USD stablecoin, 2 (smallest unit = 1 cent).

5.2 System Components

5.2.1 Institutional Server

Our implementation of the institutional server uses the Python library of gRPC, Google's Remote Procedure Call (RPC) framework [18]. It is a high performance RPC framework that offers strong typing for requests and responses via Protobufs [20], simplified error handling, and built-in TLS encryption. The server requires a SQLite or MySQL database, which is managed via the SQLAlchemy [2] object-relational mapper. All elliptic curve operations and zero-knowledge proofs use the Petlib [13] and Zksk [26] libraries, respectively. The server communicates with the Bitcoin, Ethereum, and IPFS nodes over HTTP. To track deposits, it uses a background loop to query the blockchain nodes for new blocks. It is able to generate components of the audit in parallel.

5.2.2 Audit

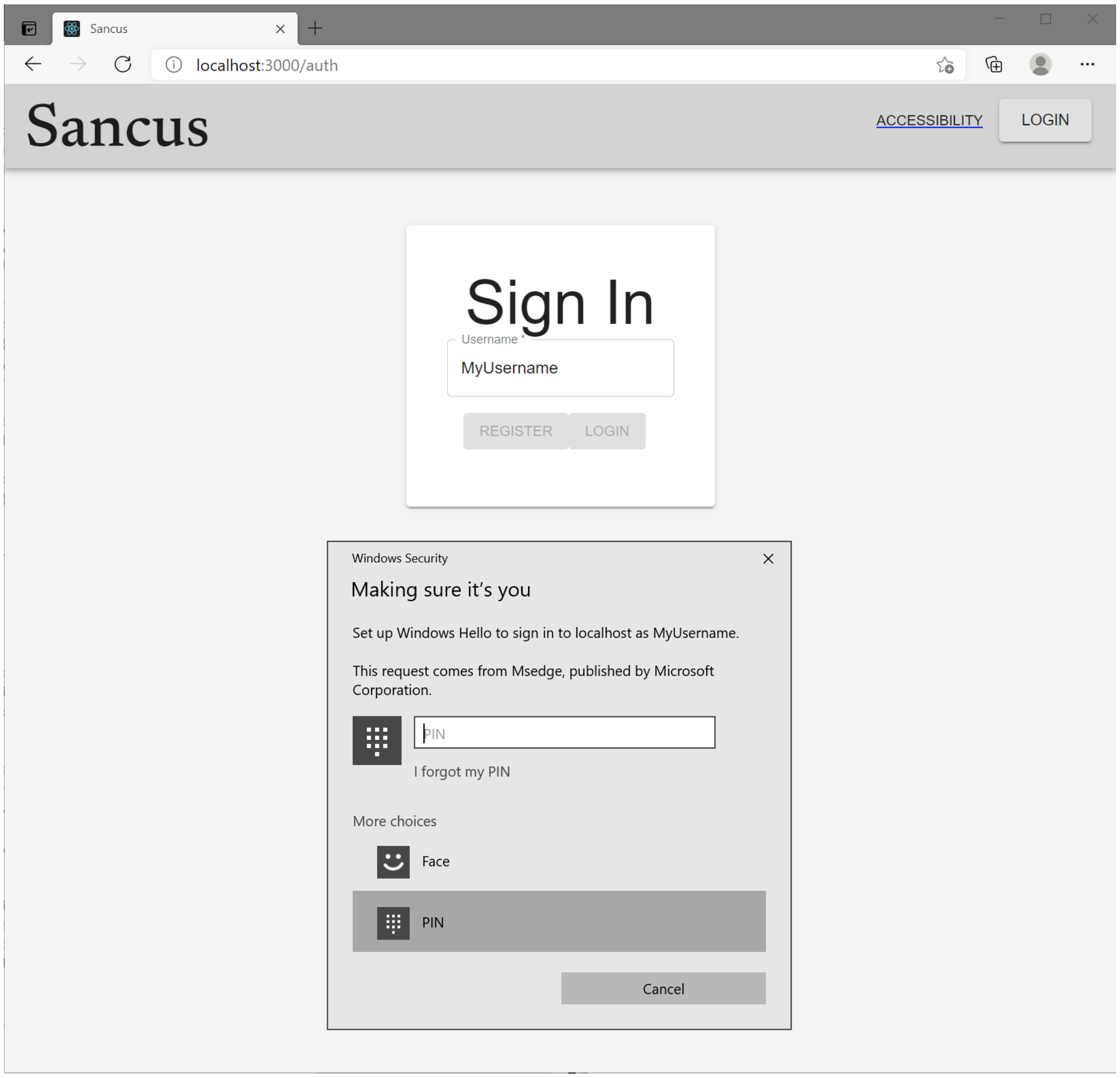
The audit is structured as a collection of Protobuf messages, with one message for each individual data model. Protobuf messages are serialized to individual binary files and compressed into a tarball. Using a binary encoding with a schema over a text-based, schemaless encoding (i.e. JSON) reduces the size of the audit.

5.2.3 Auditor

The auditor follows a similar design and uses the same technologies as the institutional server. Unlike the server, however, it processes each component of the audit serially.

5.3 Web Client

The web client is built on the React web framework [14]. It uses the JavaScript gRPC library with Grpcwebproxy [23] to issue RPC calls to the institutional server and auditors. It connects to the WebAuthn API [37] provided by the browser to handle user signature requests. Figures 5-1, 5-2, 5-3, and 5-4 respectively demonstrate the customer registration, deposit, exchange, and withdrawal functionality.



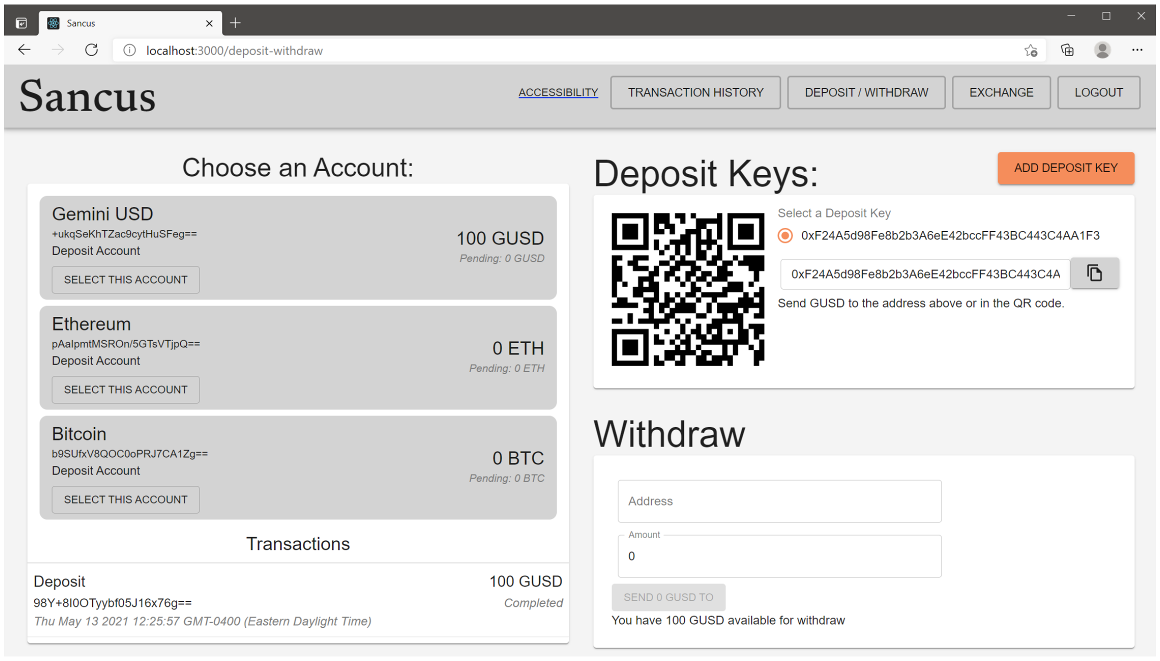
Windows Security

Making sure it's you

Set up Windows Hello to sign in to localhost as MyUsername.

This request comes from Msedge, published by Microsoft Corporation.

Figure 5-1: Customer Registration. Customers register with Sancus using their W̃ebAuthn credentials.



Choose an Account:

Gemini USD
+ukqSeKhTZac9cytHuSFeg==
Deposit Account
SELECT THIS ACCOUNT
100 GUSD
Pending: 0 GUSD

Ethereum
pAaIpmtMSROn/5GTsVTjpQ==
Deposit Account
SELECT THIS ACCOUNT
0 ETH
Pending: 0 ETH

Bitcoin
b9SUfxV8QOC0oPRJ7CA1Zg==
Deposit Account
SELECT THIS ACCOUNT
0 BTC
Pending: 0 BTC

Transactions

Deposit 100 GUSD
98Y+8I0OTyybf05J16x76g==
Completed
Thu May 13 2021 12:25:57 GMT-0400 (Eastern Daylight Time)

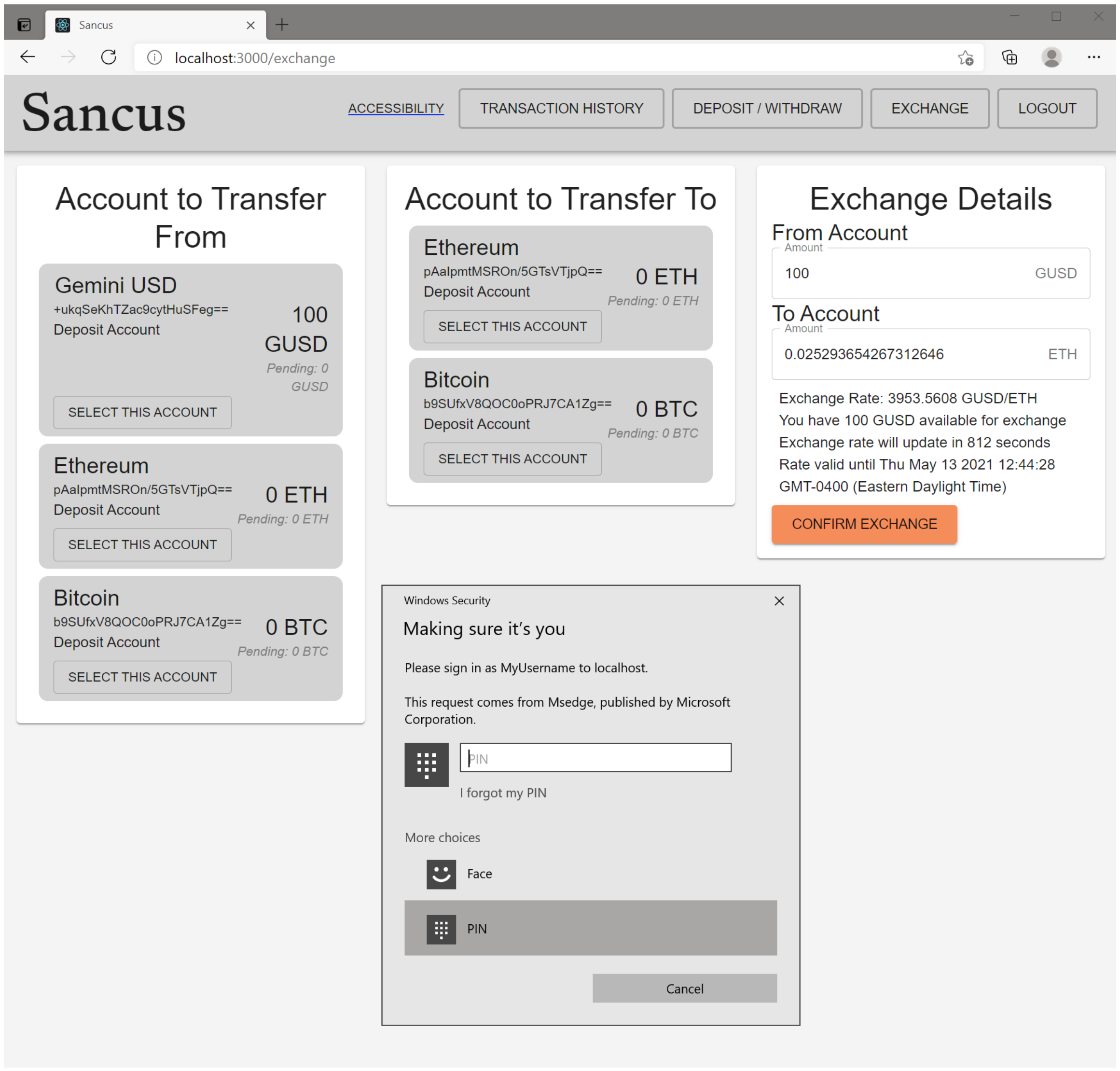
Deposit Keys:
ADD DEPOSIT KEY

Select a Deposit Key
● 0xF24A5d98Fe8b2b3A6eE42bccFF43BC443C4AA1F3
0xF24A5d98Fe8b2b3A6eE42bccFF43BC443C4A
Send GUSD to the address above or in the QR code.

Withdraw

Address
Amount
0
SEND 0 GUSD TO
You have 100 GUSD available for withdraw

Figure 5-2: Deposits. Customers can create new deposit addresses into which they can send virtual currency. Deposits made to these addresses are credited to their accounts.



Account to Transfer From

Gemini USD
+ukqSeKhTZac9cytHuSFeg==
Deposit Account
100 GUSD
Pending: 0 GUSD
SELECT THIS ACCOUNT

Ethereum
pAalpmtMSROn/5GTsVTjpQ==
Deposit Account
0 ETH
Pending: 0 ETH
SELECT THIS ACCOUNT

Bitcoin
b9SUfxV8QOC0oPRJ7CA1Zg==
Deposit Account
0 BTC
Pending: 0 BTC
SELECT THIS ACCOUNT

Account to Transfer To

Ethereum
pAalpmtMSROn/5GTsVTjpQ==
Deposit Account
0 ETH
Pending: 0 ETH
SELECT THIS ACCOUNT

Bitcoin
b9SUfxV8QOC0oPRJ7CA1Zg==
Deposit Account
0 BTC
Pending: 0 BTC
SELECT THIS ACCOUNT

Exchange Details

From Account
Amount: 100 GUSD

To Account
Amount: 0.025293654267312646 ETH

Exchange Rate: 3953.5608 GUSD/ETH
You have 100 GUSD available for exchange
Exchange rate will update in 812 seconds
Rate valid until Thu May 13 2021 12:44:28 GMT-0400 (Eastern Daylight Time)

CONFIRM EXCHANGE

Windows Security

Making sure it's you

Please sign in as MyUsername to localhost.

This request comes from Msedge, published by Microsoft Corporation.

PIN

I forgot my PIN

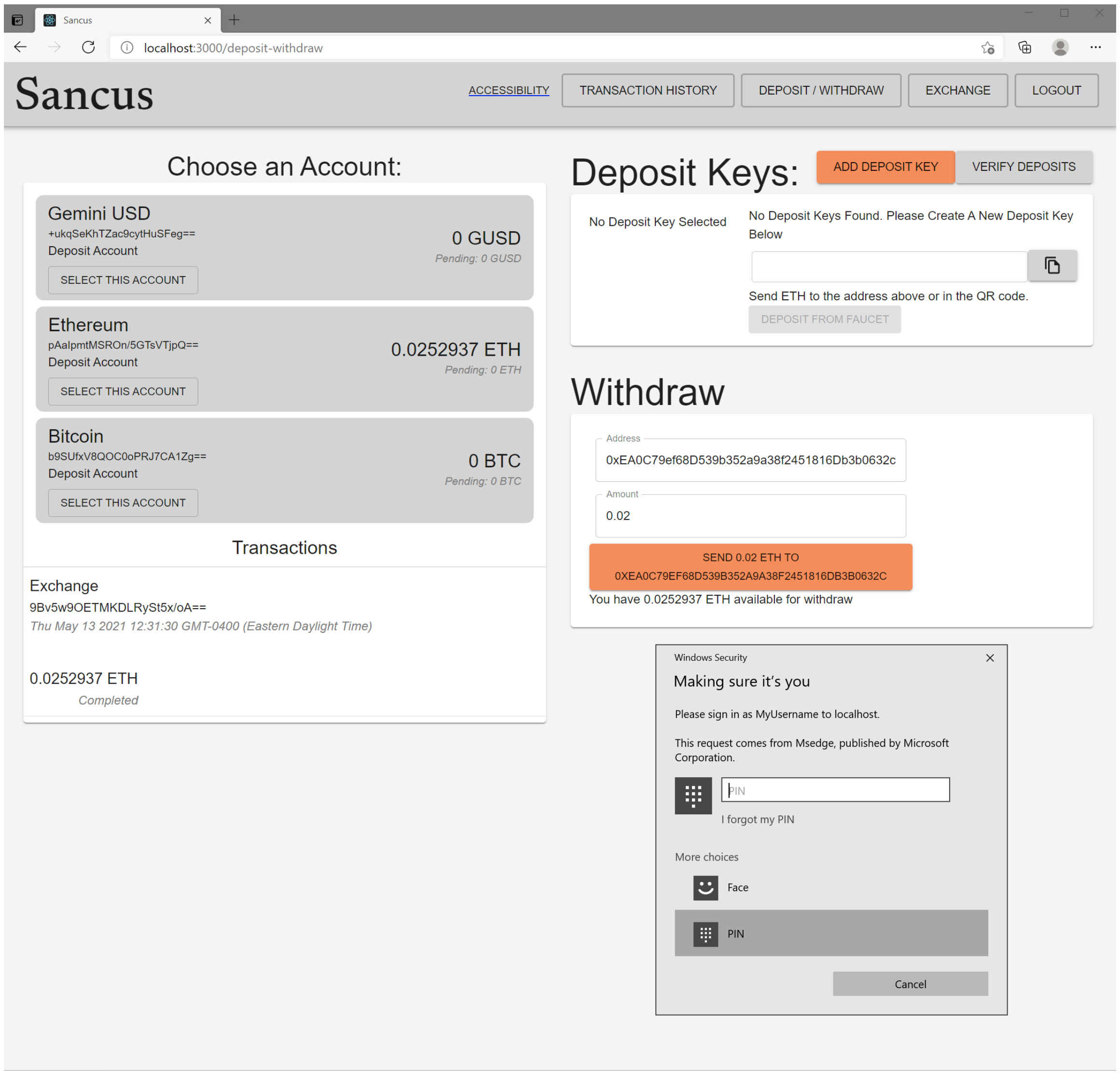
More choices

Face

PIN

Cancel

Figure 5-3: Exchanges. Customers can exchange virtual currency between their accounts. Customers authorize exchanges using the same WebAuthn credentials created during registration.


Sancus

localhost:3000/deposit-withdraw

Sancus

ACCESSIBILITY | TRANSACTION HISTORY | DEPOSIT / WITHDRAW | EXCHANGE | LOGOUT

Choose an Account:

Gemini USD
+ukqSeKhTZac9cytHuSFeg==
Deposit Account
0 GUSD
Pending: 0 GUSD
SELECT THIS ACCOUNT

Ethereum
pAaIpmtMSROn/5GTsVTjpQ==
Deposit Account
0.0252937 ETH
Pending: 0 ETH
SELECT THIS ACCOUNT

Bitcoin
b9SUfxV8QOC0oPRJ7CA1Zg==
Deposit Account
0 BTC
Pending: 0 BTC
SELECT THIS ACCOUNT

Transactions

Exchange
9Bv5w9OETMKDLRySt5x/oA==
Thu May 13 2021 12:31:30 GMT-0400 (Eastern Daylight Time)

0.0252937 ETH
Completed

Deposit Keys: ADD DEPOSIT KEY | VERIFY DEPOSITS

No Deposit Key Selected

No Deposit Keys Found. Please Create A New Deposit Key Below

Send ETH to the address above or in the QR code.

DEPOSIT FROM FAUCET

Withdraw

Address: 0xEA0C79ef68D539b352a9a38f2451816Db3b0632c

Amount: 0.02

SEND 0.02 ETH TO 0XEA0C79EF68D539B352A9A38F2451816DB3B0632C

You have 0.0252937 ETH available for withdraw

Windows Security

Making sure it's you

Please sign in as MyUsername to localhost.

This request comes from Msedge, published by Microsoft Corporation.

PIN

I forgot my PIN

More choices

Face

PIN

Cancel

Figure 5-4: Withdrawals. Customers can withdraw virtual currency from Sancus to external addresses. Customers authorize withdrawals using the same WebAuthn credentials created during registration.

Chapter 6

Evaluation

We evaluated the performance and space requirements for our implementation of Sancus to generate, store, and validate audits. We measured how Sancus scales with respect to the number of users, number of transactions, number of deposit keys, and sizes of the anonymity sets.

Our evaluation setup consisted of a quad-core, 16 GB virtual machine running Ubuntu 20.04 and CPython 3.8. We ran Go Ethereum (GETH), Go IPFS, Bitcoin Core, and MySQL databases via Docker on the same machine to eliminate network latency. Blocks were mined every 30 seconds. The institutional server and auditor communicated with these services via normal HTTP sockets on `localhost`. We used a headless WebAuthn client to simulate customer signing of transactions.

For all experiments, we measured the end-to-end duration to generate the audit as well as the cumulative sum of latencies from generating individual audit components. The end-to-end duration is generally less than the cumulative sum, as components were generated in parallel. For each audit, we measured the total size by summing together the sizes for each component of the audit. Finally, the audit validation time measures how long it took our implementation of the auditor to verify an audit after downloading it. Unlike audit generation, our implementation of the audit validator is single-threaded.

Figure 6-1 illustrates how the number of `User`s affects the audit. Each additional `User` requires a constant number of additional audit components, including

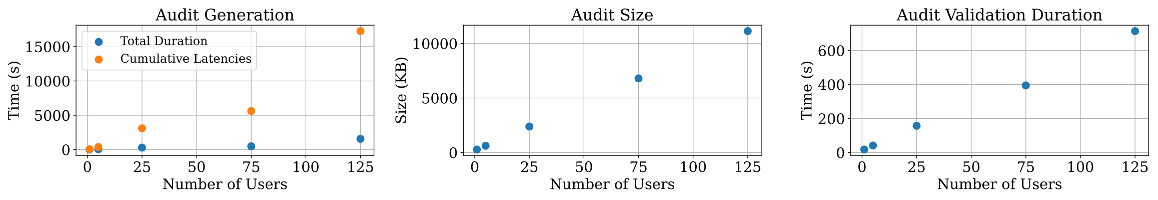


Figure 6-1: Number of Users. We varied the number of users and performed 625 transactions, split equally among the users. With a 3 : 1 : 1 split of deposits, exchanges, and withdrawal transactions, respectively, each additional `User` resulted in 11.5 additional seconds for audit generation, an additional 87.7 KB in the audit, and 5.5 additional seconds for audit validation.

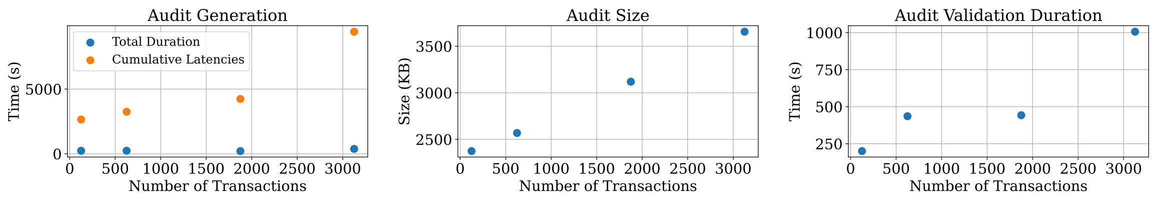


Figure 6-2: Number of Transactions. We varied the number of transactions, split equally among 25 users. With a 3 : 1 : 1 split of deposits, exchanges, and withdrawal transactions, respectively, each additional transaction, on average, increased the audit size by 0.43 KB and lengthened the audit validation duration by 0.235 seconds. The total duration for audit generation remained flat between 210.7 and 237.7 seconds for up to 1875 transactions, after which thread contention limited performance.

`UserKeys`, `Accounts`, `Keys`, and `KeyAccounts`. These requirements for each additional user resulted in linear scaling of all components up to 75 users. At 125 `Users`, the average latency to generate components for an audit increased by 11.5% compared to 25 `Users`; this increase suggests that thread contention limited performance.

Figure 6-2 illustrates how the number of transactions affects the audit. We achieved linear scaling with respect to the audit size and audit validation time, which is expected as one `AccountDeltaGroup` is required for each withdrawal and exchange transaction. Audit generation had a flat total duration between 210.7 and and 237.7 seconds for up to 1875 transactions, after which thread contention limited performance.

Figure 6-3 illustrates how the number of deposit `Keys` affects the audit. We created one `User` with one `Account` in each of the three currencies, and varied the number of deposit `Keys` for these three `Accounts`. No decoy `KeyAccounts` (i.e. those with a

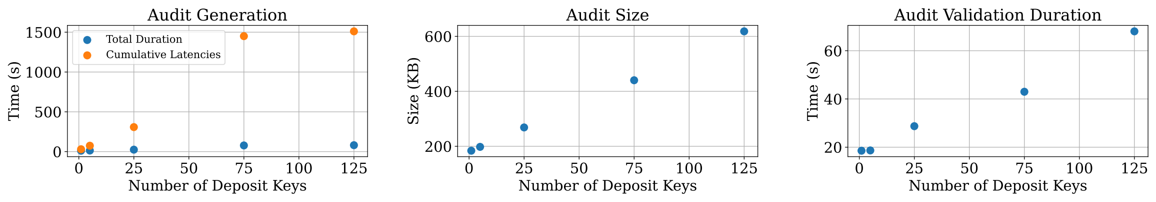


Figure 6-3: Number of Deposit Keys: We varied the number of deposit keys and experienced linear scaling with respect to the audit generation total duration (0.6 seconds), audit size (3.5 KB) and audit validation duration (0.4 seconds) per `Key`.

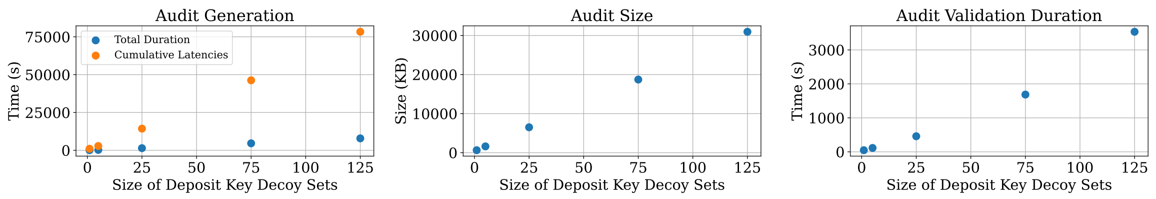


Figure 6-4: Size of Deposit Key Decoy Sets. We created one `User` with 125 `Accounts` for each of the three currencies. We created 125 `Keys`, and varied the deposit key decoy set size (i.e. the number of `KeyAccounts` per `Key`). All components experienced linear scaling, with each additional `KeyAccount` resulting in 0.51 seconds for the total duration of audit generation, 1.6 KB in the audit, and 0.22 seconds for audit validation, when normalized on per-account basis.

"credit" bit set to 0) were used. The proof of liabilities protocol (see section 4.8.2) requires each `KeyAccount` to appear in every audit.

Figure 6-4 illustrates how the size of deposit key decoy sets (i.e. the number of `KeyAccounts` per `Key`) affects the audit. Unlike additional `Keys`, which would require the auditor to query the balances for additional blockchain addresses, additional `KeyAccounts` only require verification of blockchain balance NIZKs from algorithm 4.

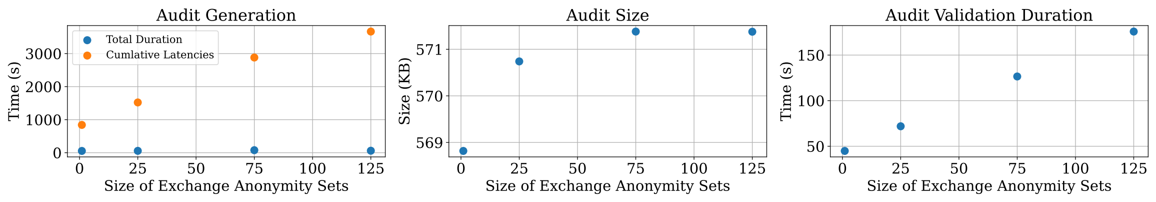


Figure 6-5: Size of Account Delta Group Anonymity Sets. We created one user with 125 Bitcoin and 125 Ethereum `Accounts` and performed one exchange per `Account`. The total duration for audit generation remained flat with an average of 61.8 seconds, and the audit size marginally increased. Each additional account in the anonymity set added 1.1 seconds to the audit validation duration.

This data illustrates how the marginal computing costs scale linearly with decoy set size. The privacy benefits from larger decoy sets also increases proportionally to the decoy set size, indicating a trade-off between privacy and cost.

Figure 6-5 illustrates how the size of the account delta group anonymity sets (i.e. the number of `AccountDelta`s in the group) affects the audit. Unlike other components of the audit, larger set sizes only result in additional Pedersen commitments, not additional NIZKs or blockchain queries. As such, there is no impact on audit generation time and a minimal impact on audit size. As the privacy benefits scale with set size, larger set sizes should be used as the costs are minimal.

Chapter 7

Future Work and Conclusion

Here we discuss possible improvements for Sancus to extend the protocol and increase its security.

7.1 Loans, Interest, and Margin Calls

While the Sancus system design (as described in chapter 4) only supports deposits, exchanges, and withdrawals, it can be extended to support loans, interest, and margin calls. Unlike exchanges and withdrawals, these transaction types present challenges as they cannot use WebAuthn signatures for authorization at the time of each transaction. For example, when a customer agrees to the loan terms, future transactions involving this loan (such as interest charges or margin calls) must be validated exclusively through zero-knowledge proofs.

Here, we present a proposal for fully collateralized, fixed rate loan support in Sancus. Let there be two additional `AccountTypes`: loan accounts and collateral accounts. When customers wish to borrow funds, they must transfer sufficient collateral from their deposit accounts into collateral accounts. They then request to initiate a loan from the institution, specifying the currency and amount they wish to borrow, the blockchain address to send funds into, and collateral accounts used to secure the loan. After checking if the collateral is sufficient, the institution responds with a loan contract. The loan contract, similar to that of an `AccountDeltaGroup`, contains

`AccountDelta`s for the loan and collateral accounts, unsigned blockchain transactions for the initial funds disbursement, the IDs of the loan accounts and collateral accounts, the interest rate of the loan, the margin call threshold, and the margin maintenance requirement. If the customer agrees with the loan parameters, one would sign it using one's WebAuthn credentials, and returned the signed loan request back to the institution. This WebAuthn signature, along with the loan parameters, would be included in the next audit.

The institution is responsible for charging interest and triggering margin calls. To charge interest, it would create a Pedersen commitment for the interest rate times the outstanding balance on the loan. Through a zero knowledge proof, it can demonstrate that interest was correctly calculated. For a margin call, the institution would prove two statements. First, it would use the currency conversion algorithm and demonstrate that the collateral, when discounted by the margin call loan-to-value ratio, is less than the outstanding balance of the loan. Second, after applying the margin call on both the loan account and collateral account, the institution proves that the new collateral balance, when discounted by the maintenance loan-to-value ratio, is greater than the remaining balance of the loan. Collectively, these proofs ensure that the institution performed a properly-sized margin call, only when permitted by the loan parameters. This protocol can be adapted to allow for exchange rate markups during margin calls or for margin calls that take place between audits.

To validate an audit, the auditor would first validate that the loan parameters are properly signed by the customer's WebAuthn credentials. For each interest charge, the auditor would validate that the commitments are consistent with the interest rate of the loan and that the NIZKs hold. For each margin call, the auditor would use the exchange rates of the audit, as well as the cumulative commitments to the loan and margin accounts, to validate the margin call NIZKs.

7.2 Anonymity Set Construction

Sancus, similar to Provisions [12], relies on anonymity sets to preserve privacy. Anonymity sets provide plausible deniability by publicly specifying (in the audit) a super-set of blockchain addresses and creating `BlockchainBalance` proofs and commitments, which hide the underlying addresses that the institution controls or has assigned to their customers. A curious auditor would not directly be able to uncover which addresses are decoys; however, by analyzing the blockchain transaction graph, one might be able to create such an inference with high probability. As such, the institution must carefully construct anonymity sets and manage transfers between institutional accounts so as not to inadvertently leak sensitive information.

While the methods to construct anonymity sets and internal transactions are beyond the scope of Sancus, one such approach is to reverse engineer techniques used for anti-money laundering (AML) on public blockchains. Weber, Domeniconi, et al. [38] describe how they trained machine learning models on 166 features extracted from the public transaction graph to identify nefarious activity. Sancus could attempt to defeat such techniques by constructing anonymous transactions such that the distribution of its 'decoy" transactions, across these 166 features, approaches that of the non-"decoy" transactions.

7.3 Transaction Expiration and Guaranteed Execution

As discussed in section 4.11.1, Sancus permits the institution to choose which exchange and withdrawal transactions are included in the audit. It may decide to delay inclusion, or never include, such transactions. To prevent such discretion, transactions can be adapted to have sequence numbers and expiration timestamps which require that they be included in order in an audit with a timestamp before the expiration; otherwise, they are void. In addition, the audit protocol can be extended to allow the customer to submit institution-signed exchange and withdrawal transactions directly to the audit smart contract instead of first remitting them to the institution. Both the institution and auditors would listen to the smart contract to know which transactions to perform. While this modification would guarantee that the next passing audit would include the transactions, it would also increase blockchain transaction fees, as each transaction would have its own entry in the audit smart contract (rather than be aggregated together into an audit).

7.4 Conclusion

Sancus introduces a novel protocol and reference implementation for generating irrefutable audits for virtual currency institutions. Unlike previous works, Sancus supports multiple currencies across existing blockchains, follows security best practices and uses offline wallets, preserves privacy for the institution and its customers, and does not rely on customers to validate their own balances in the audit. Our evaluation demonstrates the feasibility of this approach and the usability of the system. The modular reference implementation can be extended to support new blockchains and transaction types. As the virtual currency economy continues to evolve, it will become increasingly important for institutions to adopt trustworthy auditing solutions.

Bibliography

[1] Hayden Adams, Noah Zinsmeister, Moody Salem, River Keefer, and Dan Robinson. Uniswap v3 core, 2021.

[2] Mike Bayer. Sqlalchemy - the database toolkit for python, 2005.

[3] Juan Benet. Ipfs - content addressed, versioned, p2p file system, 2014.

[4] Blockchair. Bitcoin transactions per second, November 2020.

[5] Blockchair. Ethereum transactions per second, November 2020.

[6] Fabrice Boudot. Efficient proofs that a committed number lies in an interval. In Bart Preneel, editor, *Advances in Cryptology — EUROCRYPT 2000*, pages 431–444, Berlin, Heidelberg, 2000. Springer Berlin Heidelberg.

[7] Jan Camenisch, Rafik Chaabouni, and Abhi Shelat. Efficient protocols for set membership and range proofs. In Josef Pieprzyk, editor, *Advances in Cryptology - ASIACRYPT 2008*, pages 234–252, Berlin, Heidelberg, 2008. Springer Berlin Heidelberg.

[8] Jan Camenisch and Markus Stadler. Efficient group signature schemes for large groups. In Burton S. Kaliski, editor, *Advances in Cryptology — CRYPTO '97*, pages 410–424, Berlin, Heidelberg, 1997. Springer Berlin Heidelberg.

[9] Coinbase. Secure bitcoin storage - coinbase, November 2020.

[10] CoinMarketCap. Coinmarketcap, November 2020.

[11] Ronald Cramer, Ivan Damgård, and Berry Schoenmakers. Proofs of partial knowledge and simplified design of witness hiding protocols. In Yvo G. Desmedt, editor, *Advances in Cryptology — CRYPTO '94*, pages 174–187, Berlin, Heidelberg, 1994. Springer Berlin Heidelberg.

[12] Gaby G. Dagher, Benedikt Bünz, Joseph Bonneau, Jeremy Clark, and Dan Boneh. Provisions: Privacy-preserving proofs of solvency for bitcoin exchanges. In *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*, CCS '15, page 720–731, New York, NY, USA, 2015. Association for Computing Machinery.

[13] George Danezis. Petlib: A python library that implements a number of privacy enhancing technologies (pets), 2014.

[14] Facebook. React: A javascript library for building user interfaces, 2021.

[15] FDIC. Fdic law, regulations, related acts - rule 2000 part 363, 2009.

[16] FDIC. Fdic: Insured or not insured?, May 2020.

[17] U. Feige and A. Shamir. Witness indistinguishable and witness hiding protocols. In *Proceedings of the Twenty-Second Annual ACM Symposium on Theory of Computing*, STOC '90, page 416–426, New York, NY, USA, 1990. Association for Computing Machinery.

[18] Linux Foundation. grpc - a high-performance open-source universal rpc framework, 2021.

[19] Gemini. Gemini user agreement, October 2020.

[20] Google. Protobufs, 2008.

[21] Nermin Hajdarbegovic. Kraken bitcoin exchange passes 'proof of reserves' cryptographic audit, March 2014.

[22] Darrel Hankerson, Alfred J Menezes, and Scott Vanstone. *Guide to elliptic curve cryptography*. Springer Science & Business Media, New York, New York, USA, 2006.

[23] Improbable. Grpc web proxy, 2021.

[24] Kraken.

[25] Robert Leshner and Geoffrey Hayes. Compound: The money market protocol, 2019.

[26] Wouter Lueks, Bogdan Kulynych, Jules Fasquelle, Simon Le Bail-Collet, and Carmela Troncoso. Zksk: A library for composable zero-knowledge proofs. In *Proceedings of the 18th ACM Workshop on Privacy in the Electronic Society*, WPES'19, page 50–54, New York, NY, USA, 2019. Association for Computing Machinery.

[27] Wenbo Mao. Guaranteed correct sharing of integer factorization with off-line shareholders. In Hideki Imai and Yuliang Zheng, editors, *Public Key Cryptography*, pages 60–71, Berlin, Heidelberg, 1998. Springer Berlin Heidelberg.

[28] Nacha. Ach network moves 23 billion payments and $51 trillion in 2018, February 2019.

[29] Satoshi Nakamoto. Bitcoin: A peer-to-peer electronic cash system. Technical report, Manubot, 2009.

[30] Neha Narula, Willy Vasquez, and Madars Virza. zkledger: Privacy-preserving auditing for distributed ledgers. In *15th USENIX Symposium on Networked Systems Design and Implementation (NSDI 18)*, pages 65–80, Renton, WA, April 2018. USENIX Association.

[31] New York Department of Financial Services. Virtual currency businesses: Regulation and history, 2015.

[32] Torben Pryds Pedersen. Non-interactive and information-theoretic secure verifiable secret sharing. In Joan Feigenbaum, editor, *Advances in Cryptology — CRYPTO '91*, pages 129–140, Berlin, Heidelberg, 1992. Springer Berlin Heidelberg.

[33] Joseph Poon and Thaddeus Dryja. The bitcoin lightning network: Scalable off-chain instant payments, 2016.

[34] Claus-Peter Schnorr. Efficient signature generation by smart cards. *Journal of cryptology*, 4(3):161–174, 1991.

[35] Pete Schroeder. U.s. crypto, fintech firms to benefit from slimmed down regulatory process, September 2020.

[36] Vitalik Vogelsteller, Fabian; Buterin. Eip-20: Erc-20 token standard. Technical Report 20, Ethereum Improvement Proposals, November 2015.

[37] W3C. *Web Authentication: An API for accessing Public Key Credentials*. W3C, March 2019.

[38] Mark Weber, Giacomo Domeniconi, Jie Chen, Daniel Karl I Weidele, Claudio Bellei, Tom Robinson, and Charles E Leiserson. Anti-money laundering in bitcoin: Experimenting with graph convolutional networks for financial forensics. *arXiv preprint arXiv:1908.02591*, 2019.

[39] Bitcoin Wiki. *Hashcash*, April 2019.

[40] Bitcoin Wiki. *Bitcoin Protocol*, September 2020.

[41] Gavin Wood. Ethereum: A secure decentralized generalized transaction ledger. *Ethereum project yellow paper*, 151:1–32, 2014.