Analyzing Issues of Privacy and Offline Transactions In Central Bank Digital Currencies

by

Michael Lee

A thesis
presented to the University of Waterloo
in fulfillment of the
thesis requirement for the degree of
Master of Applied Science
in
Electrical and Computer Engineering

Waterloo, Ontario, Canada, 2023

Author's Declaration

I hereby declare that I am the sole author of this thesis. This is a true copy of the thesis, including any required final revisions, as accepted by my examiners.

I understand that my thesis may be made electronically available to the public.

Abstract

With the popularity of cryptocurrencies like bitcoin and Ethereum, many central banks have begun to look into issuing their own digital currency. For many central banks, the goal of a central bank digital currency (CBDC) is to provide a user experience similar to paper money, but fully digital. The central bank also plays an important role in the system, namely acting as a source of trust. This source of trust is an important differentiator, as it incentivizes the use of alternative technologies to confirm transactions, rather than using inefficient consensus protocols such as a proof-of-work blockchain.

In order to act as a true paper money alternative, two of the biggest hurdles that need to be overcome are privacy and offline transactions. In this thesis, we will examine these issues in more detail, discussing what problems they pose and what (if any) solutions have been presented in the existing literature. Additionally, we will be offering our own solutions, using hash chains to provide user privacy, and presenting a prototype CBDC system for offline transactions.

Acknowledgements

I would like to thank my supervisor, Dr. Anwar Hasan, for his help and support, not only in writing this thesis, but also throughout my degree as a whole. Without his guiding aid, I could not have completed this journey.

I would like to also thank the professors who taught me so much during my time at the University of Waterloo: Dr. David Jao, Dr. Mahesh Tripunitara, Dr. Wojciech Golab, Dr. Mark Crowley, and Dr. Guang Gong. Thank you all.

Additionally, I would like to thank Brett Mollin from Ripple and the kind folks in the XRP Ledger Developers discord server for their help aid in helping me configure rippled and XRPL. Thank you also to William Park for creating the UI.

Lastly, I would like to thank my parents and family for their love and support. A special thank you to my dad, for sparking a curiosity and fascination with computers that has lasted a lifetime, and without whom I would not have taken on this endeavour.

Dedication

This is dedicated to my parents, Allen and Victoria.

Table of Contents

List of Figures

List of Tables

List of Abbreviations

AML Anti-Money Laundering 24, 31

BIS Bank for International Settlements 16

CBDC Central Bank Digital Currency 1, 10, 16, 31, 37, 45

CTF Combating the Financing of Terrorism 24, 31

ECDSA Elliptic Curve Digital Signature Algorithm 4

GCD Greatest Common Divisor 5, 6

RSA Rivest–Shamir–Adleman 4, 6

SHA Secure Hash Algorithm 7

UHS Unspent Funds Hash Set 26, 32

UTXO Unspent Transaction Output 26

XRP The native currency of Ripple Labs' XRP Ledger 13, 37

XRPL XRP Ledger 2, 10, 12, 37, 45

zk-SNARK Zero-Knowledge Succinct Non-Interactive ARgument of Knowledge 15, 22

ZKP Zero-Knowledge Proof 14, 31, 40

Chapter 1

Introduction

A Central Bank Digital Currency (CBDC) is defined by the *Bank for International Settlements* as "a digital payment instrument, denominated in the national unit of account, that is a direct liability of the central bank" [17]. Put more simply, it is a digital currency issued by a central bank, as opposed to digital currencies such as Bitcoin or Ethereum which are run by private entities. With the rise in popularity of these privately issued digital currencies, and a decline in the usage of paper money, central banks have begun to investigate the feasibility of issuing their own digital currency. Though many, such as the Bank of Canada, do not have any plans to issue a CBDC in the near future, it is important to investigate technologies now so that they can be implemented quickly and efficiently in the future, should a central bank choose to do so. By issuing a CBDC, central banks hope to reduce the risk of an alternative currency, one which the central bank would have no control over, dominating domestically. Importantly, if an alternate currency were to dominate, it would greatly hinder the central bank's ability to direct monetary policy. This means that how a central bank issues a digital currency, as well as what features said currency would have, is of great importance. Should the roll-out of a CBDC go poorly, whether because of technology issues, because it has less features than existing digital currencies, or due to any other number of reasons, it will have a difficult time reducing the usage and popularity of competing digital currencies. In the worst case it may have the opposite effect of what was intended and drive users towards alternate currencies. This is why coming up with solutions to the existing problems now facing CBDCs is so important.

Despite the similarities to well known cryptocurrencies like Bitcoin and Ethereum, CBDCs have a number of differences that make them wholly unique, the biggest being that CBDCs have the central bank as a source of trust. This makes technologies such as *permissioned blockchains* (as opposed to the permissionless blockchains used by most other

cryptocurrencies) more appealing, while also reducing the amount of resources required by protocols such as *proof-of-work*.

With the goal of many central banks being to have their CBDC act as a digital form of paper money, privacy is another key feature that is shared with cryptocurrencies. In other words, not only is it a feature that many users would expect and want from a digital currency, but it is a feature that central banks would like to include as well. However, unlike with most cryptocurrencies, how user data is protected and secured becomes much more complicated due to the need to strictly enforce fiscal compliance. Compared to paper money, the ability to more easily enforce fiscal compliance within a digital currency ecosystem is a major draw for central banks. However, the discussion on how best to balance compliance with privacy is still ongoing.

Another cash-like feature that is seen as a key requirement by central banks is the ability to perform transactions without a connection to the internet. Yet, as we will see, this is a feature that goes entirely unmentioned in many of the proposed systems.

1.1 Contributions

It is the two issues of user privacy and offline transactions that will be the main focus of this thesis. On the privacy side, we will examine the current literature, taking a look at privacy goals, stakeholders, and ideas surrounding privacy. We will also take a look at how different systems handle user data, whom they protect user data from, and their overall privacy measures. Lastly, we will take a look at how hash chains could be used in order to protect user privacy.

For offline transactions, we have created a prototype CBDC system that emulates offline transactions for the purposes of studying and aiding further development. The system, built off Ripple Labs' XRP Ledger (XRPL) blockchain technology, simulates a CBDC application, allowing users to deposit, withdraw, and initiate transactions to exchange money between users. These transactions can happen in real time or offline, in which case the users' balances are updated locally, then updated on the online ledger once one of them has reconnected to the system.

1.2 Outline

The remainder of the thesis is organized as follows: Chapter 2 provides an overview of cryptographic concepts, how they work and their relevance, to hopefully allow those with

little prior knowledge to understand the ideas presented in this thesis. Specifically, public-key cryptography, cryptographic hashing, Merkle trees, blockchain systems, homomorphic encryption, zero-knowledge proofs, and threshold cryptography will be covered. Chapter 3 discusses the existing related literature, looking at key features and requirements for CBDCs, what central banks are looking for, and the current issues facing CBDCs. In addition, we will be examining proposed CBDC systems, specifically *Platypus CBDC*, a proposed system using Chaum-style blind-signatures, MIT's *Project Hamilton*, and *KAIME*. In Chapter 4, we will go into more details on our strategy for offline transactions, as well as discussing privacy in CBDC systems, looking at who has access to user data, how proposed systems handle user data, and how hash chains could be used to provide privacy. Chapter 5 details the XRPL-based prototype CBDC system developed for this thesis, how it works, and what benefits it provides. Lastly, Chapter 6 concludes the thesis and discusses possible future work.

Chapter 2

Background

While the layperson may think of cryptography simply as a way to send messages secretly, the field of cryptography covers much more, and allows us to do things such as hiding and confirming identities, or proving knowledge of a secret without giving the secret away. In this section, we will cover topics to familiarize readers with cryptographic techniques discussed and used in this thesis, and in digital currency systems in general.

2.1 Public-key Cryptography

Public-key cryptography, also called asymmetric key cryptography, allows two parties to communicate securely, without having to share a common key beforehand (as would need to be done in traditional symmetric key cryptosystems). To do this, these systems use key pairs, consisting of a private key and a public key (hence public-key cryptography). Public keys are shared with everyone, while the private keys are kept hidden. In addition to allowing for secure communication, these key pairs can also be used to confirm the identity of a sender using digital signatures. To take a look at public-key cryptography in more detail, we will use RSA, as it is one of the oldest public-key cryptosystems, it is still widely used, and it is simpler/easier to understand compared to other commonly used public-key cryptosystems, such as those based on elliptic curves defined over finite fields [2]. With that being said, elliptic curve cryptography does generate smaller keys sizes and requires less computation when compared to RSA for the same security level, and hence is better suited for more constrained environments. This is also true for digital signatures, which we will cover shortly, and these advantages are why Bitcoin and Ethereum both use Elliptic Curve Digital Signature Algorithm (ECDSA) as their signature schemes [4][14].

2.1.1 RSA Encryption

To encrypt a message using RSA, two large prime numbers p and q are chosen, and the product, N, is calculated. We then compute the public key (e, N) and private key d by performing the following operations:

1. Calculate $\phi(N) = (p-1)(q-1)$.
2. Choose a value e that is coprime to $\phi(N)$. That is, e is between 1 and $\phi(N)$ and GCD$(e, \phi(N)) = 1$
3. Calculate $d = e^{-1} \bmod \phi(N)$ (e.g., by using the extended Euclidean algorithm)

Notably, e and d now have the property that for any number m

$$(m^e)^d \equiv m \bmod N.$$

If two parties, Alice and Bob, want to communicate using RSA, they will both compute private and public keys individually, and share the public keys with one another. When Alice sends a message to Bob, she will first convert her message m into an integer (e.g. by taking the binary representation of the message). Next, using Bob's public key (e, N), she will compute

$$c \equiv m^e \bmod N.$$

The resulting ciphertext, c, is then sent to Bob, who can decrypt it using his private key d by computing

$$c^d \equiv (m^e)^d \equiv m \bmod N.$$

If Bob wants to send a message back to Alice, the procedure would be repeated, but using Alice's key pair.

2.1.2 RSA Digital Signatures

Digital signatures are a feature of public-key cryptosystems that allow for authentication, and can function much like a real life signature. The most common use case is to confirm the identity of a message's sender (for this to be completely effective, the recipient must already know the public key of the sender), though they can be used for other, related, tasks as well.

Once again using RSA, supposing Alice already knows Bob's public key, if Bob wants to prove that a message he is sending is from himself, he can sign it by first calculating the message's hash $h(m)$ (Section 2.2). Next, Bob can sign the message using his private key d by calculating

$$s(m) = (h(m))^d \bmod N.$$

When Alice receives Bob's message and signature, she can calculate the message's hash herself, then use Bob's public key to compute

$$s(m)^e \bmod N = h(m).$$

If the two hashes match, Alice knows the message she received was indeed from Bob.

In terms of digital currencies, signatures are used in a number of ways. For example, they are often used for transaction authorization. By having the individual sending funds sign the transaction, the transaction processor can verify the origin of the transaction, and thereby verify that the transaction is not from a malicious actor trying to spend someone else's funds.

Blind Signatures

Blind signatures, introduced by David Chaum in *Blind Signatures for Untraceable Payments*, are a variation of digital signatures that are of particular note for digital currencies. They allow users to preserve privacy when providing a message to be signed by a server (central bank, ledger, etc.), while hiding the contents of the message. The process of hiding the message's contents is called blinding.

Still using RSA, obtaining a blind signature is done by performing the normal signature algorithm on a blinded message. To blind the message, after obtaining the server's public signing key (e, N), we follow the steps below:

1. Choose a number r at random where the GCD of r and N is 1 (r and N are co-prime)
2. Calculate the blinding factor $r^e \bmod N$
3. Using the blinding factor, obtain the blinded message $m' = mr^e \bmod N$

We can then send m' to the server, which will sign the message as normal by computing

$$s' = (m')^d \bmod N.$$

and return s'. We can then obtain the signature s for the original message m by calculating

$$s = s'r^{-1} \bmod N.$$

Using this, we are able to obtain a signature for m, despite the server never knowing the contents of m.

2.2 Cryptographic Hashing

Hashing takes an arbitrary amount of data as input, and maps it to an output of fixed size. This output is called a "digest," "hash value," or simply "hash." When used in cryptographic applications, it is important that the hashing algorithm used is secure: That is, the hashing algorithm is collision resistant, second pre-image resistant, and pre-image resistant. Collision resistance means that the probability that two different inputs output the same hash is negligible. We say that the algorithm has second pre-image resistance if, given an input, there's a negligible probability of finding a second input that gives the same hash value as the first input. Lastly, the algorithm is pre-image resistant if, given a hash value, there is a negligible probability of finding an input that will result in that hash value.

2.2.1 SHA-256

To better understand how hashing works, we can take a look at SHA-256. Developed by NIST and the NSA SHA-256 [21] and published in 2001, it is widely used for internet communication, as well as in many cryptocurrencies, including Bitcoin, due to its security, compact size (256 bits), and computationally simple operations (logical shifts, rotations, and XOR). This makes SHA-256 not only efficient in terms of memory size, but also fast [3].

To obtain a hash value, we start by creating an array, h, of size 8, with the elements being the first 32 bits of the fractional parts of the square roots of 2, 3, 5, 7, 11, 13, 17, and 19 (the first 8 prime numbers). We also initialize a second array, k, of size 64, as a set of constants consisting of the first 32 bits of the fractional parts of the cube roots of the first 64 prime numbers. Each of these array elements are 32 bit unsigned integers, represented in big-endian. We then take the value we want to hash, x, and a 64 bit integer, l, which is the number of bits in x. Next, we obtain $x' = (x \parallel 1 \parallel z \parallel l)$, where $\parallel$ is the concatenation operation, and z is a number of zeros such that the number of bits in x' is divisible by

512. x' is then broken down into 512 bit *chunks*. For each chunk, an array, w, of size 64, is initialized, with each element consisting of 32 bits. The chunk is then copied into the first 16 elements of the array. To calculate the remaining 48 indices, the following loop is executed:

```
for i from 16 to 64:
    s0 = (w[i-15] >>> 7) ⊕ (w[i-15] >>> 18) ⊕ (w[i-15] >> 3)
    s1 = (w[i-2] >>> 17) ⊕ (w[i-2] >>> 19) ⊕ (w[i-2] >> 10)
    w[i] = w[i-16] + s0 + w[i-7] + s1
```

Here, $>>>$ represents the bit rotation right operation, and $>>$ represents the bit shift right operation. Once we have filled w, we copy the values of h to a temporary array, h'. The *compression function* is then run, which looks like the following:

```
for i from 0 to 64
    S1 = (h'[4] >>> 6) ⊕ (h'[4] >>> 11) ⊕ (h'[4] >>> 25)
    ch = (h'[4] ∧ h'[5]) ⊕ ((¬ h'[4]) ∧ h'[6])
    temp1 = h'[7] + S1 + ch + k[i] + w[i]
    S0 = (h'[0] >>> 2) ⊕ (h'[0] >>> 13) ⊕ (h'[0] >>> 22)
    maj = (h'[0] ∧ h'[1]) ⊕ (h'[0] ∧ h'[2]) ⊕ (h'[1] ∧ h'[2])
    temp2 = S0 + maj

    for j from 7 to 0
        h'[j] = h[j-1]
    h'[4] += temp1
    h'[0] = temp1 + temp2
```

The final step for each of the chunks is to add the result of each element in h' to its corresponding element in h (i.e., $h[i] = h[i] + h'[i]$). Once each chunk is processed, we can obtain the hash value by concatenating each element in h, giving us:

$$v = h[0] \parallel h[1] \parallel h[2] \parallel h[3] \parallel h[4] \parallel h[5] \parallel h[6] \parallel h[7].$$

2.3 Merkle Tree

Merkle trees, also known as hash trees, are a data structure consisting of hash values, that can be used for data verification. In a Merkle tree, leaf nodes consist of the hash value

for some piece of data, and each parent node contains the hash of the concatenation of its child nodes. That is, node $n_i = h(n_{i-1,0} \parallel n_{i-1,1})$, where $h(x)$ is a hash function, $\parallel$ is the concatenation operation, and $n_{i-1,0}$ and $n_{i-1,1}$ are the two child nodes of n_i. This results in a binary tree structure, which can be seen in Figure 2.1. Given a piece of data, x_i, that is claimed to be contained in the tree, to verify this claim, the user only needs the root and the structure of the tree initially. The user can verify x_i by finding a path from the root to x_i's hash. Then, by querying the tree's owner for the values of sibling nodes along our path, they can verify if the resulting hash chain gives the same value as the root.

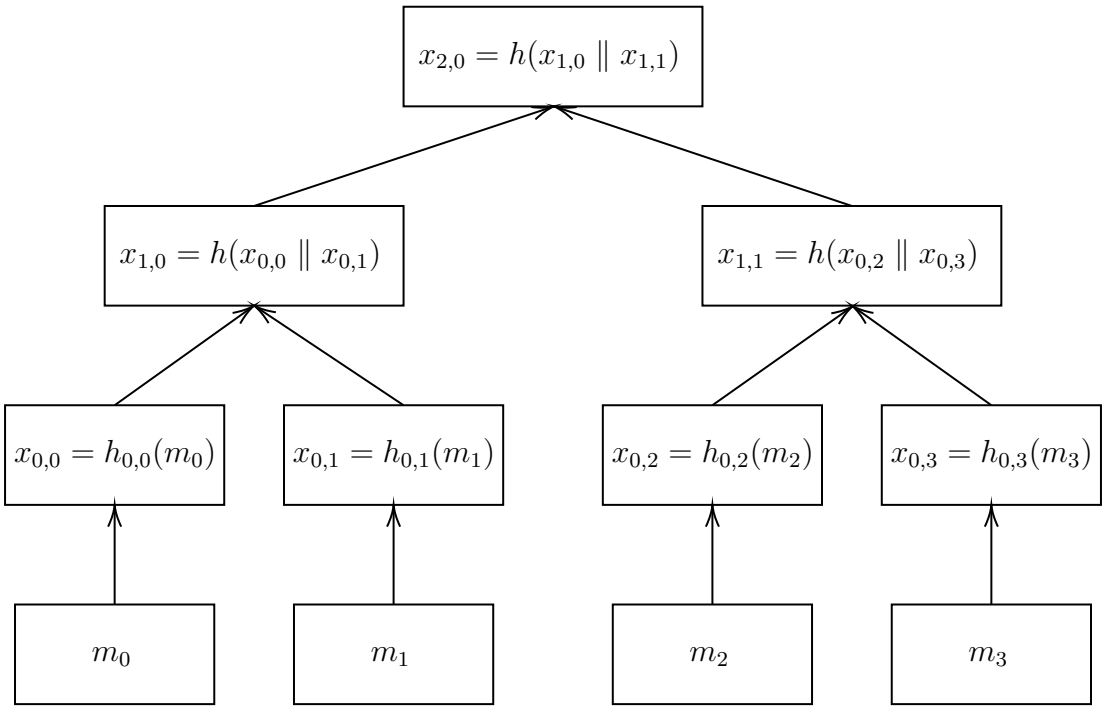

Figure 2.1: Structure of a Merkle Tree

2.4 Blockchain

A blockchain is the backbone for many digital and cryptocurrencies. Introduced by Satoshi Nakamoto in their Bitcoin whitepaper [14], a blockchain is a distributed, peer-to-peer, ledger which stores records and information (usually transactions in the case of digital currencies). At a high level, users will broadcast any transactions they perform to the

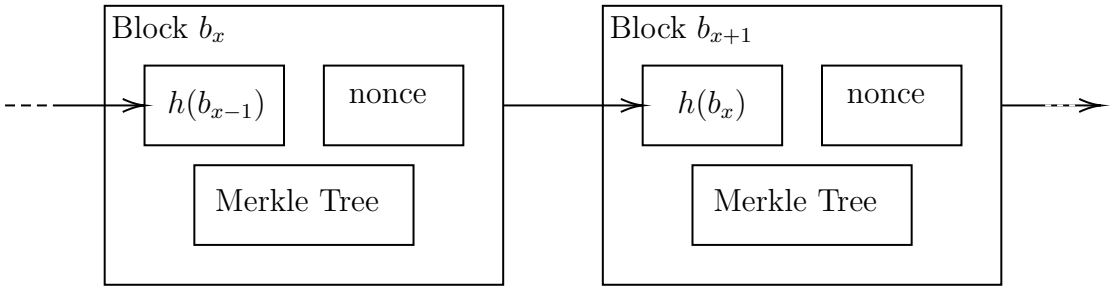

Figure 2.2: Bitcoin Blockchain Structure

network. Systems on the network will then collect these transactions into *blocks*. These systems store a record of previous blocks, and using the most recent one, will add its hash to the block being created. This chain of block hashes is what gives the ledger its name, blockchain. Once a system has computed a satisfactory block, it will broadcast the block to the network. If other systems agree that the block is valid, they will accept the new block and add it to their blockchain, and use its hash when creating the next block. Who and how transactions and blocks are validated is an important differentiator between different blockchains, and is what we will focus on below.

2.4.1 Permissioned vs Permissionless Blockchains

One differentiator of note is how we choose who can perform validation. Most of the largest blockchains, such as Bitcoin and Ethereum, are *permissionless* blockchains. In permissionless blockchains, anyone can join and participate in the consensus protocol. In other words, participants are self-selecting, though they need to expend resources in order to participate. These systems rely on an *honest majority*, where the majority of actors behave according to the outlined protocol. To enforce this, these systems offer incentives in the form of their ledger's native cryptocurrency to miners. In contrast to this are permissioned blockchains, where participants are selected using an external selection process. Permissioned blockchains align well with CBDCs, as they can leverage the central bank being a source of trust. Hyperledger Fabric, Quorum, and Corda are all examples of permissioned blockchains. Additionally, although the XRPL is a permissionless blockchain, it has features that resemble a permissioned blockchain, as specific nodes can be specified as validators and there is no incentive involved with being a validator.

2.4.2 Bitcoin

To interact with the Bitcoin blockchain, all Bitcoin users must generate a key pair, with the public key being used as their address. In order to perform a transaction, the sender must supply their wallet's address, amount, a record of where the bitcoins being sent came from (usually one larger transaction or multiple smaller ones, compared to the amount being sent), and a destination address (or addresses if "change" needs to be returned to the sender). This transaction is then signed by the sender and broadcast to the network, who will verify the signature. Systems on the network collect these transactions, and store them in a Merkle tree. The resulting Merkle tree is then added to a new block, with the tree's root hash and the hash of the previous (i.e. most recent) block being contained in the new block's header. Lastly, in order to save disk space, branches of the Merkle tree already included in previous blocks are stubbed off from the new block.

In Bitcoin, and other proof-of-work cryptocurrencies, each block also includes a nonce, an additional value such that the resulting block's hash meets the requirement of having a specified number of leading zero bits. As the hashing algorithm being used is cryptographically secure, due to pre-image resistance, the only way to find such a hash is to try numerous values for the nonce until a satisfactory value is found. As a side note, the high energy concerns that have become associated with cryptocurrencies is due to this repeated hash calculation with difference nonce values.

The resulting block, illustrated in Figure 2.2, is then broadcast to the network. Other systems will verify its validity, and will then use it as the latest block in their chain. In the event that more than one block is found and broadcast at the same time, systems will continue to operate on whichever block they received and verified first, but will switch to any other chain of blocks that becomes longer than the one they are operating on. Summarily, the longest, valid, blockchain is always considered to be the correct version of the ledger. Transactions can be said to be timestamped into the block they were included in, as it would take a majority of the network's computing power to falsify new transactions, or to re-create previous blocks and subsequently to create a longer, divergent chain. The network operates under the premise that such a concentration of computing power is not feasible. This is how the double spending problem is dealt with.

2.4.3 Ethereum

Along with Bitcoin, Ethereum is one of the most popular blockchain systems. The largest difference between the two is transaction verification. When verifying a transaction using

Bitcoin, all that is being computed is that the user has sufficient funds for the transaction, and that the signature on the bitcoin(s) being spent is valid. Ethereum expands this model, and allows users to define what function must be satisfied in order for the transaction to be valid. In the Ethereum system, accounts contain *ether*, and are either externally owned and controlled by a key-pair (the account is owned and controlled by an individual), or are contract accounts controlled by a *contract code*. It is these contract codes that give Ethereum its programmability. Ethereum features its own low-level, Turing-complete programming language for use in these contract codes, and when a contract account receives a transaction/message (defined by whether the sender is an externally owned account or a contract account), the contract code is executed as a part of the verification process. These transactions/messages not only contain the amount of ether being sent, but also any data the contract code may require, as well as how much *gas* (ether) is allowed to be used in processing the contract code as an additional fee. This fee is based off how many bytes of additional data the transaction/message contains, combined with how many computational steps it takes for the execution of the contract code. These fees ensure that code terminates eventually, and act as an anti-denial of service method. The sender is only charged for the amount of gas "used" in the computation, though if the transaction fails (for example, if the specified maximum amount of gas is consumed before reaching the end of the code), then none of the consumed gas will be refunded.

In addition to this programmability, the structure of the blocks in the blockchain is also slightly different. Unlike a block on the Bitcoin block chain, an Ethereum block contains a copy of the transaction list and the most recent state, along with a block number and the proof-of-work difficulty[1]. As the entire state is stored in each block, the tree containing transaction records is also different compared to Bitcoin, using a *Patricia tree* instead of the standard Merkle tree.

2.4.4 XRP Ledger

The XRP Ledger (XRPL) is an open source blockchain developed by Ripple Labs, which uses the "XRP Ledger Consensus Protocol," a consensus protocol for transaction validation, rather than proof-of-work or proof-of-stake. In the consensus protocol, validator servers are responsible for forming this consensus. Servers connect to multiple of these validators, selecting ones not expected to collude with others. During each round, when forming a consensus, validators propose a set of transactions to include in the next version of the ledger. A consensus is reached if a large enough percentage (a super majority) of

[1]In September 2022, Ethereum switched from proof-of-work to proof-of-stake.

proposals agree on a set of transactions and the resulting ledger. If a consensus cannot be reached, validators modify their proposals by adding transactions contained in proposals from most other validators that they trust, and by temporarily removing transactions not included by those other validators. Ripple states that the ledger can progress without issues as long as fewer than 20% of validators are faulty. However, if between 20% and 80% of validators are faulty, the network will be unable to make progress, and if greater than 80% of trusted validators collude, then they could confirm invalid transactions. Overall, it takes about 3-5 seconds for the XRPL to settle a transaction.

Each ledger, more accurately, "ledger version," contains the settings, balances, and objects being stored on the ledger, the transactions applied to the previous ledger to make the current ledger, and metadata about the current ledger version, including the ledger's index, hash, and parent ledger. In addition, each ledger contains the entirety of the current state, meaning that the current state does not need to be calculated from previous ledger versions. These ledgers can be accessed and read by any individual, and transactions are pseudonymous (that is, they include an identity, but the identity does not reveal who the person actually is), which can present privacy concerns.

The native cryptocurrency of the XRPL is XRP. XRP is not only traded on the ledger, but is also required to be held in a wallet as a reserve, and when executing a transaction, an additional transaction cost must also be paid. Also tradeable on the XRPL are tokens. Tokens can be used to represent any asset, though in order to receive tokens, the receiving account must first create a "line of trust" between itself and the account that created the token. Both tokens and account reserves (a minimum amount of XRP that an account must hold in order to initiate transactions) are used to construct the CBDC prototype. Tokens are used to represent our CBDC currency, while holding XRP and meeting the account reserve can be used to confirm that a user has been validated and onboarded properly, meeting any regulatory requirements.

In addition to XRP balance and any tokens possessed, a user's wallet also consists of an address, sequence number (to maintain transaction order), account transaction history, and transaction authorization method. Each time an account submits a transaction, a sequence number must be included in the transaction. The sequence number increases by one for each validated transaction, and sequence numbers cannot be skipped. That means that the provided user's sequence number in a transaction must be exactly one greater than the previous one, or the transaction will be rejected. The transaction authorization is usually a master public/private key pair that is intrinsic to the wallet, though alternatives include a different key pair that can be rotated, or a signer list used for multi-signing.

As the XRPL is open source, it is also possible to run private networks that do not

interact with the main XRP ledger. These private networks must consist of at least three validator nodes, though they can be running on the same machine, with each validator being bound to a different port (this is the set up used to create the CBDC prototype system presented later in this thesis).

2.5 Homomorphic Encryption

Homomorphic encryption is a form of encryption in which computational operations performed on encrypted ciphertexts will be reflected in the plaintext when decrypted. For example, given homomorphic encryption and decryption functions ($E(x)$ and $D(x)$), if we perform an addition operation on two homomorphically encrypted values, $c_1 = E(m_1)$ and $c_2 = E(m_2)$, the resulting cipher text when decrypted will the sum of the two plaintext values ($D(c_1 + c_2) = m_1 + m_2$). A somewhat trivial example for a homomorphic encryption scheme is given in [22]. If we want to encrypt the plaintext value m, we can select a natural number, g, as a base and obtain ciphertext c by taking g to the power of m, or $c = g^m$. Let x be some value we want to use to perform a mathematical operation on m. Multiplication can now be performed without decrypting c through exponentiation

$$c^x = (g^m)^x = g^{mx}$$

If we encrypt x using the same base, addition can be performed by multiplying the two encrypted values, and subtraction by dividing the values

$$c \cdot g^x = g^m \cdot g^x = g^{m+x}$$

$$\frac{c}{g^x} = \frac{g^m}{g^x} = g^{m-x}$$

2.6 Zero-Knowledge Proofs

A Zero-Knowledge Proof (ZKP), or more specifically a non-interactive zero knowledge proof, is a cryptographic primitive that allows a user (*prover*) to prove the validity of a statement to a *verifier*, without providing any additional information to the verifier beyond the statement and its validity. ZKPs are highly applicable in digital currency systems as a method of preserving privacy, for example proving that a user has sufficient funds for a transaction without revealing how much they are holding in their account. One of the

most common ZKP systems for cryptocurrencies is the Zero-Knowledge Succinct Non-Interactive ARgument of Knowledge (zk-SNARK), as the generated proofs are small, only a few hundred bytes in size, and can often be verified in less than 10ms [20]. The mathematics behind zk-SNARKs are complex, and use polynomial arithmetic as the basis for its proofs. As a high level overview, a prover generates a proof by translating their statement into an arithmetic circuit that uses publicly available, homomorphically encrypted values. These values, called the "Common Reference String" (CRS), are generated in a one-time trusted setup, in which the required parameters are generated and encrypted before being discarded. As the values are homomorphically encrypted, the polynomial arithmetic of the arithmetic circuit can be performed without knowing the underlying values. This allows the verifier to evaluate the circuit, and verify the authenticity of the statement. The need for a trusted setup is often cited as a weakness of zk-SNARK, as fake proofs could be generated if a party were to know the unencrypted parameters, though this is much less of a concern for CBDCs, where the central bank is a source of trust and authority.

2.7 Threshold Cryptography

In threshold cryptography, a secret is encrypted with a public key, while a private key to decrypt the secret is distributed amongst a number of parties. The parties must work together for the secret to be decrypted. Specifically, a number of parties greater than a specified *threshold* must combine their keys in order to decrypt the secret. For example, in Shamir's secret sharing [25] a polynomial $f(x)$ with degree $t-1$ that uses the secret as the constant term, is randomly selected, where t is the threshold. For each party, i, (i, y_i) is shared, where $y_i = f(i)$. The secret can be reconstructed if at least t individuals share their given coordinate pair with one another by using the Lagrange interpolation formula.

Chapter 3

Analysis of CBDC Requirements and Existing Systems

In this chapter, we will take a look at some of the existing literature on CBDCs. Specifically, we will first examine what is wanted from a CBDC and what the requirements are from both a consumer and central bank perspective. We will then discuss the design decisions and trade-offs associated with these requirements, and lastly we will take a look at a sample of recently proposed CBDC systems, and how well they align with these requirements.

3.1 CBDC Criteria and Features

The Bank for International Settlements (BIS) compiled a report from a number of central banks detailing what they believe to be the key concepts and characteristics of a CBDC. This includes outlining the motivations for a CBDC, what hurdles a CBDC would need to overcome, and prominent features of importance, as well as design decisions that would need to be made and their foreseen associated trade-offs [17][18]. We can use their paper as a template for examining these features and requirements.

To discuss these features and requirements, as well as CBDC systems as a whole, we can break these criteria down into four categories. First, we can look at tradition requirements. By this, we want to examine requirements related to the client side or user experience (how the payment system should function from a user perspective) and requirements from the server side (how the system should run and be built by a central bank). Next are design decisions of the system, which include the underlying technology and how the system

is configured. Last are privacy and offline transactions. While both of these could be discussed in the previous sections, due to their importance to CBDCs and to this thesis as a whole, we will discuss them on their own.

3.1.1 Traditional Requirements

Starting from the user or client side, the user experience is an important factor in a payment system, as a poor user experience will greatly hamper growth and adoption rates, in spite of whatever technology and features the system touts. Following this, there are three main criteria that need to be examined. The first aspect to consider is the convenience of making transactions in the CBDC system. As the BIS puts it, payments should ideally, "be as easy as using cash, tapping with a card, or scanning a mobile phone." Second is the speed of transactions, or how long a transaction takes from end-to-end. Ideally, transactions should be settled nearly instantaneously (a matter of seconds at most). Lastly is the acceptance and availability of the CBDC, as it is important that the digital currency be accepted and usable in the same way paper money is. Importantly, the BIS puts offline transactions, in addition to point of sale and person-to-person transactions, in this category, though offline transactions will be discussed later as previously mentioned. All three of these measures can be considered a measurable and benchmarkable aspect of the user experience, and it is important that they be taken into consideration when designing any kind of payment system.

Server side requirements look at the requirements a central bank or other institution should adhere to when running a CBDC. These requirements, again coming from the BIS report, include the resilience of the system, its availability, its throughput, how scalable the system is, and lastly, security. While availability and resilience both relate to uptime, availability refers mainly to system uptime, while resilience refers to the ability to make payments during operational failures and disruptions such as natural disasters or electrical outages (i.e., when the system is unavailable), and also includes the ability to make offline transactions. Next, throughput looks at a system's ability to quickly process transactions, while scalability refers to both the ability to handle large numbers of transactions, as well as being able to expand this capacity in the future. Lastly, extra care must be taken with the security of the system, as there is potential for serious financial harm should an exploit be found. This goes beyond just the design of the CBDC itself, and extends to how the systems and servers are set up, both in reference to the physical setup and to the software. These requirements place importance on considering how a CBDC system will be deployed in the real world when designing a system, with the hope that methods used in traditional data centers can be replicated for any potential CBDC system.

Instrument features	
Convenient	CBDC payments should be as **easy as using cash**, tapping with a card or scanning a mobile phone to encourage adoption and accessibility.
Accepted and available	A CBDC should be usable in many of the same types of transactions as cash, including point of sale and person-to-person. This will include some **ability to make offline transactions** (possibly for limited periods and up to predetermined thresholds).
System features	
Secure	Both the infrastructure and participants of a CBDC system should be extremely **resistant to cyber attacks** and other threats. This should also include ensuring effective protection from counterfeiting.
Instant	Instant or near-**instant final settlement** should be available to end users of the system. *(No time scale specified, but likely no greater than a few seconds.)*
Resilient	A CBDC system should be **extremely resilient to operational failure and disruptions**, natural disasters, electrical outages and other issues. There should be some ability for end users to make offline payments if network connections are unavailable.
Available	End users of the system should be able to make payments **24/7/365**. *(100% uptime is not feasible in the real world, but uptime as good as or greater than existing electronic payment methods is likely the goal)*
Throughput	The system should be able to process **a very high number of transactions**. *(Specifics on what constitutes a high number of transactions not given, though it would likely vary from country to country depending on population size.)*
Scalable	To **accommodate the potential for large future volumes**, a CBDC system should be able to expand.
Flexible and adaptable	A CBDC system should be **flexible and adaptable to changing conditions** and policy imperatives.

Table 3.1: Excerpt from *Table 1: Core CBDC features* in [17], listing and detailing CBDC features. *(Commentary italicized in brackets)*

Design and technology trade-offs	
Security/offline transactions	There may be a desire for a CBDC to enable users to settle transactions peer-to-peer, similarly to banknotes. This heightens the need for fraud protection and other security features. Depending on the features, the number or value of transactions permitted offline could be capped (before being reset by a verified online transaction).
Cost of service provision/ universal access	Banknotes create the same user experience for all users. CBDC, assuming multiple devices are available, can create differing experiences. For example, smartphone users will have greater functionality than those with stored value cards. Active dedicated devices can close that gap, albeit at a higher cost.
Privacy/ compliance	Privacy is designed to hide information and compliance to reveal it as required. A combination of cryptography and operational or institutional arrangements may enable both features and satisfy users that privacy is well preserved. As an example, multiple agencies could hold fragments of decryption keys that are only brought together after due process to reveal information.
Privacy/ capacity and scalability	Privacy techniques that are computationally demanding may be costly and impose limits on a system's capacity and scalability.
Programmability/ performance	Heavy use of programmable functions will require a higher level of technical performance from the system, adding costs or reducing operational resilience.

Table 3.2: Excerpt from *Table 2: Summary - key design and technology decisions* in [18], listing and explaining design and technology trade-offs.

3.1.2 Design Considerations

Beyond the traditional requirements, we also have design decisions more specific to CBDCs. While the underlying system may dictate some of these decisions, it is important for central banks to determine the features they require, and use a technology that supports these decisions. Here, we will discuss offline transactions and privacy, as well as two other areas brought up by the BIS: ledger design and the design of financial policy instruments.

Ledger design is broken down by the BIS into the following five categories: structure, authenticated scheme, additional functionality, access, and governance. Structure mainly deals with whether a ledger is centralized, decentralized, or somewhere in between. A centralized ledger would force the usage of intermediates for the transferring of liabilities, which in turn would make anti-fraud and security measures easier to incorporate. A decentralized ledger, on the other hand, could have other advantages such as making peer-to-peer and offline transactions easier to implement. The authentication scheme refers to the way in which transactions will be authenticated, such as using an identity or token-based scheme, some form of multi-factor authentication, etc. Additional functionality examines what, if any, functionality the ledger should have beyond keeping records, with payment synchronization being given as an example. Ledger access looks at who should have read or write permissions to the ledger, and is mostly concerned with balancing privacy and security versus potential competition and diversity within a CBDC ecosystem. Lastly, governance refers to how to manage and set the roles of the operator, participants, and any other stakeholders.

As a CBDC is purely digital, there is potential for adding features to the currency that would not be possible with paper money. These financial policy instruments, specifically whether or not to make the currency interest-bearing or whether to impose a cap/limit on individual holdings, can have large consequences. Imposing a limit could impede adoption and limit the effectiveness of any interest-bearing decisions, yet could reduce risks to the financial system, such as any type of bank run. If a CBDC is interest-bearing, the interest rate would need to be competitive, which induces financial stability risks, namely due to the disintermediating of banks. Imposing negative interest rates has also been proposed, but this would once again impede the adoption and usage of a CBDC.

3.1.3 Privacy

One of the biggest concerns with CBDC is the need to enforce financial compliance while balancing user privacy. The paper titled *Mapping the Privacy Landscape for CBDC* [1], by

Auer et al, identifies who they believe to be the three main stakeholders when it comes to privacy, viz. privacy enthusiasts, law enforcement, and data holders. Privacy enthusiasts are concerned with maintaining their privacy. Law enforcement needs access to records to investigate financial crime. Data holders may record and/or monetize financial data. Data holders include merchants, banks, and payment processors. We will use these stakeholders to discuss privacy in this section.

Also in their paper, Auer et al. use the terms *soft privacy* to describe CBDC systems in which, "judicial oversight [is used] to allow human discretion in balancing exceptional access to payment data with privacy," and *hard privacy* to describe systems that "eliminate[s] human intervention by relying solely on cryptography and, perhaps, tamper-resistant hardware."

While hard privacy is preferred by privacy enthusiasts, the authors argue that hard privacy is too strict and rigid, and cannot adequately cover every scenario in which law enforcement may need access to financial records. Using transaction value thresholds as an example for hard privacy, where access is allowed to records of transactions above a certain value, they point out that there are scenarios where access to records of a number of low-value transactions may aid in solving time-sensitive crimes by tracking a criminal's spending. In addition, any bad actors aware of these hard privacy rules could adapt their methods to avoid law detection. There are also the concerns that malicious data holders would end up with more data than law enforcement through data hoarding or through malicious means. At the same time, the paper argues that soft privacy is also insufficient. While soft privacy would appease law enforcement and data holders, its similarity to contemporary payment systems would do little to incentivize the on-boarding of privacy enthusiasts. Auer et al. claim that a hybrid approach is necessary, and that an ideal system would protect bulk data records, but offer plaintext access in, "justified cases... rooted in appropriate law."

Auer et al. also offer other ideas and solutions, such as having the central bank process and record all transactions in order to prevent the data from abuse by law enforcement and politicians. This would also minimize the amount that data holders have access to, as a central bank, itself, should have no interest in monetizing the data, though commercial banks could still serve a purpose, such as ensuring users are compliant during the on-boarding procedures, and by continuing to offer traditional payment services. Lastly, Auer et al. dismiss the idea of using cryptographically protected identities when on-boarding users. Though these could be used for selective traceability by law-enforcement, the authors believe that this is outweighed by the, "greater costs [imposed] on commercial banks with additional computation, procedures, and internal controls relating to the involved cryptography."

3.1.4 Offline Transactions

Central banks place a large importance on the ability of CBDC systems to offer offline transaction capabilities. In their report titled *A central bank digital currency for offline payments* [13], Minwalla et al. provide insight into, and categorization of, offline transactions for a CBDC, notably, the differentiation between *intermittent offline transactions* and *extended offline transactions*. A CBDC that supports intermittent offline transactions allows for users to perform transactions during temporary internet outages. They require at least one of the users to connect back to the online system to complete and settle the transaction. CBDCs that support extended offline transactions allow transfer and settlement of funds over a local network, without needing a connection to the internet or to a wider system. This allows peer-to-peer transactions in areas without regular internet access, and also implies the transitivity of funds – that is, that funds received can be re-spent immediately. In an intermittent offline transaction system, such transitivity is not present, as re-synchronizing with the central system is required to finalize and settle funds before they can be spent again.

The paper also raises considerations and concerns related to offline transactions in a CBDC system. It mentions security concerns, for both hardware and software, and acknowledges the elevated risks associated with offline transactions. The accessibility benefits that offline transactions facilitate are also discussed, especially with an extended offline transaction system. Such a system not only gives an alternate form of payment to those in remote communities, but may also have built-in privacy preserving features.

3.2 Recently Proposed CBDC Systems

Here, we will discuss a number of recently proposed CBDC systems, and compare them to the requirements and design decisions outlined above. These systems are diverse, each using different cryptographic techniques. Each system also has a different focus in terms of the requirements and design decisions we just mentioned. Four different CBDC systems are discussed: Platypus [27], a Chaum-style blind-signature based CBDC system [7], Project Hamilton [11], and KAIME [10].

3.2.1 Platypus CBDC

The Platypus CBDC, proposed by Wüst et al., uses zero-knowledge proofs, specifically zk-SNARK, in order to facilitate secure transactions. The use of zero-knowledge proofs not

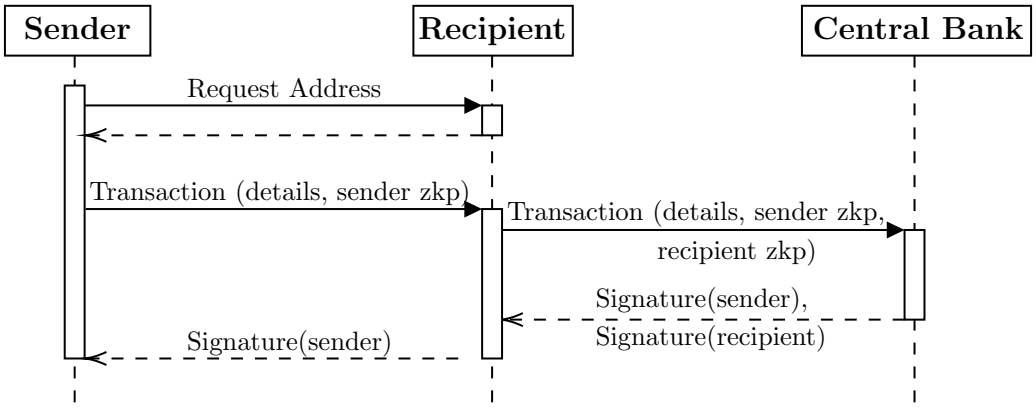

Figure 3.1: Platypus Transaction Sequence Diagram

only allows for secure transactions, but also allows for additional parameters to be added as additional information in the transaction. This allows for hard privacy measures to be implemented, such as disclosing identity when transactions are over a specific threshold, which is otherwise protected. While this method provides flexibility to how transactions operate, and would not significantly add to the zero-knowledge proof complexity, the adaptability is limited, as a new trusted setup for the zero-knowledge proof would need to be created if additional parameters were ever added.

In this system, users must first register with the central bank to establish their identity. This identity would then be associated with a key pair. This is not only used for regulatory purposes, but the private key is the sole means of authentication as well. In the transaction process, as illustrated in Figure 3.1, the recipient first shares their public key with the sender (e.g. using a QR code). Both then create zero-knowledge proofs that they have updated their wallet correctly. These are then sent to the central bank by the recipient. The proofs are verified by the central bank, which signs the transaction before sending it back to the recipient. The recipient is then responsible for forwarding the signature to the sender, though the sender can also check the publicly readable transaction log to verify the transaction in the case they do not receive the signed transaction from the recipient (note that only the central bank servers can write to this log).

Examining the system in terms of server side requirements, the system should have good scalability, reliability, and availability, as verification of zero-knowledge proofs is quick, and the system's only other responsibilities are to sign and record transactions. The provided benchmarks also show that the system is capable of hundreds of transactions per second on a single computer using consumer-grade hardware, with a server throughput of 600-950

transactions/second. For the user, generating the zero-knowledge proofs is also relatively quick, though requires at least moderately powerful hardware, with the slowest provided benchmark taking 1.5 seconds to generate a zero-knowledge proof using an iPhone 13. The system requires a connection to the central bank system, however, raising issues with regards to offline transactions.

3.2.2 Chaum-Style Blind-Signature CBDC

Chaum et al. propose a CBDC based on Chaum blind signatures. This scheme mimics paper money, in that it uses tokens with specific, fixed, discrete denominations. The tokens are key pairs, in which only the token's owner knows the private key. The token is signed by the central bank, which has a different signature for each denomination. In other words, the denomination of a given token is determined by which of the central bank's signatures was used to sign the token. These central bank key pairs would have an expiry date, which would also impose a time limit on the tokens, meaning users would have to exchange their existing tokens with those that have a newer, equivalent signature. The authors state that this has the advantage of improving efficiency (as it would reduce the number of transactions the central bank would need check for double spending), of improving security (bad actors would have less time and incentive to try and attack/discover the bank's private keys), and of allowing for the implementation of limited fiscal policies. Specifically, a sort of negative interest could be imposed by charging a fee for this sort of tokens exchange, and/or a conversion limit could be implemented to reduce the risk of a bank run (or other financial stability factors) or to ensure financial compliance (Anti-Money Laundering (AML)/Combating the Financing of Terrorism (CTF)). The expiration of tokens, however, would likely harm the adoption rate for the CBDC.

As tokens represent specific, discrete values, when performing a transaction, multiple tokens may be required, and it may be necessary to convert a token to an equivalent value of multiple tokens of smaller value (obtain change), in much the same manner as paper money. Unlike with paper money, the customer would need to obtain any change before the transaction, going through the central bank to do so. Once they have exact change for the transaction, the customer signs the bill of sale using the necessary tokens, and transmits the signature to the merchant. The merchant validates the signatures and forwards the associated signed tokens to their commercial bank, who then forwards them again to the central bank. The central bank verifies the signatures on the tokens and checks for double spending. Once everything has been validated and the tokens have been recorded as spent, the commercial bank's account is updated, who in turn updates the merchant's account.

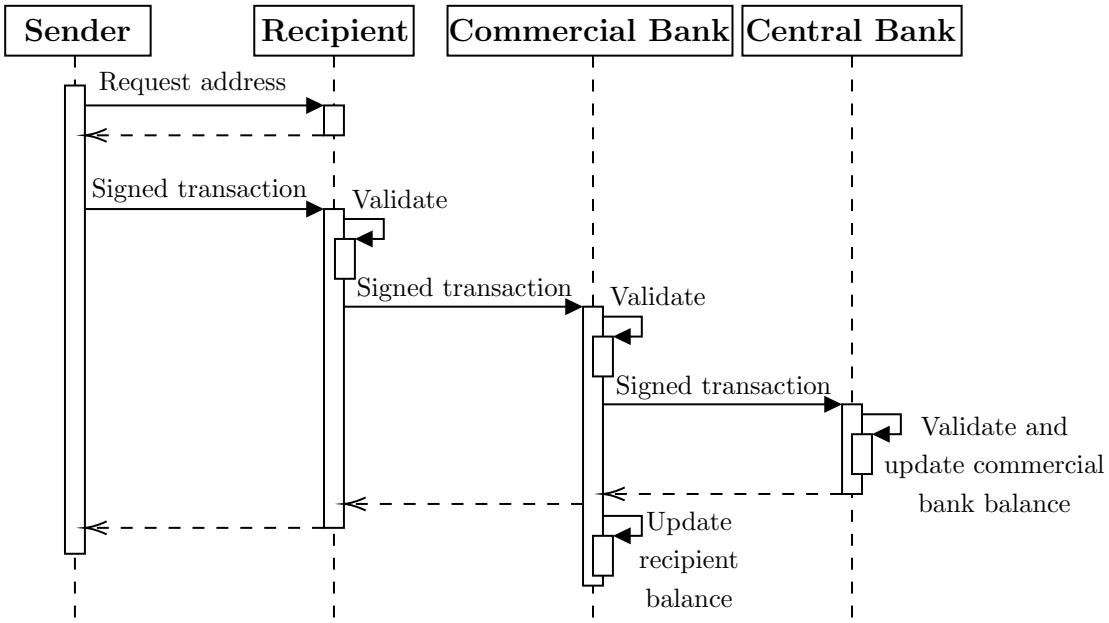

Figure 3.2: Chaum Blind Signature Transaction Sequence Diagram

This process (summarized as a sequence diagram in Figure 3.2), would only take a few hundred milliseconds according to the paper's authors.

As can be seen when detailing a transaction, in their proposal, the authors outline their system as having a two-tier architecture. The role of the central bank is to create the tokens and to verify transactions. Consumers and merchants, on the other hand, interact with commercial banking institutions, and need to go through them to withdraw and/or spend tokens.

The use of blind signatures in this system has a number of implications. First, it hides the identity of users. This puts compliance responsibilities on commercial banks, much like with existing money, as transactions must go through these commercial banks. One advantage the system has over paper money, though, is income transparency, which is a consequence of the way transactions are performed. This also has the additional benefit of making tax evasion more difficult. Second, as the system only really uses key-exchange protocols, the complexity of the hardware required is very low, as the process is not very computationally demanding.

The responsibilities of the central bank in this system are fairly simple. Blind signatures are no different than regular digital signatures, from the signer's perspective. This is to say

that normal data center availability, reliability, and scalability methods can easily be used for this system, which should be capable of very high throughput. This means that the cost of the system for the central bank, and commercial banks as well, should be comparable to those of modern real time gross settlement systems currently used in the banking industry. The simplicity of the system does mean that there is very little in the way of additional features that could be added, aside from those already discussed. Lastly, due to the ledger (which is simply a database in this system) only being accessible by the central bank, it would make offline transactions infeasible.

3.2.3 Project Hamilton

Project Hamilton is a "hypothetical CBDC" created by the Massachusetts Institute of Technology, in collaboration with the Federal Reserve Bank of Boston. This CBDC is similar in function to Bitcoin and blockchains, and uses Unspent Transaction Output (UTXO) tokens. The UTXOs consist of a monetary value, an encumbrance predicate, and a serial number. The encumbrance predicate is used to authorize the use of its UTXO in a transaction, and contains the public key of the user who is able to authorize its use. The serial number is a unique value derived by taking the hash of the transaction that created the UTXO and appending it with the UTXO's index in that transaction's output (i.e. the position number this UTXO had in the transaction's list of output UTXOs). In this way, UTXOs are deterministically generated and recursively incorporate the transaction history of the UTXO.

In project Hamilton, when performing a transaction, existing UTXOs are completely consumed, and new UTXOs will be created, where the sum of the new UTXOs' values equal that of the spent UTXOs'. To perform a transaction, the recipient shares their public key with the sender. The sender then creates the transaction, consisting of the UTXOs being spent, assigning the values each new UTXO should have, and specifying the encumbrances for each of the new UTXOs. The consumption of each UTXO being spent is authorized by signing the transaction with the corresponding private key. This list of signatures is added to the transaction to finalize the transaction's creation. It is then shared with all parties involved in the transaction before it is sent to the central bank. The central bank verifies that the syntax of the transaction is correct, that the value of the input UTXOs and requested output UTXOs are equal, and that each of the UTXOs being spent matches the signature provided for it. If the transaction is valid, the serial number of each output UTXO is generated, then all UTXOs are hashed. The central bank keeps a set of hashes for all unspent UTXOs (an Unspent Funds Hash Set (UHS)), rather than the UTXOs themselves, and checks that the UTXOs being spent are in the UHS. Once verified, the

old UTXOs are removed from the UHS, and the output UTXOs are added. The UHS can only be accessed by the central bank, but users are able to query the existence of a UTXO's hash in the UHS. Returning to the users, they are also able to compute the serial numbers of the output UTXOs, and can query the central bank for the existence of the output UTXOs. Once the users see that the output UTXOs exist in the central bank's UHS, they know that the transaction was successfully computed.

A limitation of this system is that the user must be aware of any UTXOs they receive, meaning that if they are not made aware that funds have been sent to them, those funds are essentially lost. In addition, the need to query the central bank for the existence of UTXOs greatly hampers any future ability for offline transactions. As only a UTXO's hash is being stored, it does allow for some flexibility in design of the digital currency, as the specific data structure of the UTXO is malleable. Also worth mentioning in regard to the UTXOs and UHS is privacy. As only hashes are stored, the UHS provides some privacy to the user, though user and spending data could still be tracked by the government/central bank before the UTXOs are hashed, as identifying information is present in the transaction in the form of public keys.

In terms of operation, a central bank could run project Hamilton similarly to a traditional data center. MIT offers two implementations for transaction verification, one that uses an atomizer (in which an "ordering server [is used] to create a linear history of all transactions" [11]), and another that uses two-phase commits, with the atomizer being the slower of the two. Both can be implemented with Raft fault tolerance, and the two-phase commit implementation also allows for sharding to improve scalability. Transactions can be processed fairly quickly, with the benchmarks averaging from 0.5 to 0.7s, depending on implementations, and can be submitted by fairly low-powered devices, as generating hashes and signing transactions are the most computationally complex tasks asked of a user's device. System throughput benchmarks are also very high, with the two-phase commit implementation processing 1.7 million transactions/second, and the atomizer processing 170 thousand per second.

3.2.4 KAIME

KAIME aims to better address privacy between users, the central bank, and commercial banks. Proposed by Dogan and Bicakci, it utilizes public key signatures and encryption, as well as zero-knowledge proofs, homomorphic encryption, and threshold cryptography. In the system, hard privacy techniques are implemented to encrypt user data, with soft privacy used to allow regulatory agencies access to this data in justified cases, though

the criteria for this are not specified. This aligns with the strategy laid out in [1]. In this system, the central bank does not have any control over the status of user accounts, and their main responsibility is issuing the digital currency. Commercial banks have the responsibility of registering users, performing know-your-customer procedures and setting up customer accounts, though users generate their own keys. In KAIME, commercial banks have no ability to see a user's balance or transaction details without the said user's permission. To ensure compliance, a group of authorized institutions, named "regulatory agencies" in the paper, are responsible for conducting audit procedures and other related tasks. They are able to access user data only by joint decision, through the use of threshold encryption.

In KAIME, a user's balance is stored, homomorphically encrypted, on a distributed ledger. Each user also has a wallet containing one key pair used for encryption, a second key pair used for signing, the regulatory agencies' public key, and the user's balance. To perform a transaction, after receiving the recipient's public key (e.g., with a QR code), the sender encrypts the value they are sending thrice, once with their public key, a second time with the receiver's public key, and a third time using the regulatory agencies' key. The sender then generates two zero-knowledge proofs that show the plaintext of the three encrypted values are equal. Finally, two more zero-knowledge proofs are generated, this time being "range proofs," showing that the user has sufficient funds for the transaction, and that the encrypted value is within the range of 0 and 2^{31}-1. This transaction is then signed with the sender's public signing key, before being submitted to the blockchain. After the transaction and proofs have been verified, the ledger is updated homomorphically.

Analyzing the privacy of this transaction data with regards to regulatory agencies, this method conceals the details of the transaction by default. However, as the value is encrypted using the regulatory agencies' public key, if enough of the agencies agree it is necessary, the transaction history and balance of a user can be decrypted and accessed by these agencies through threshold encryption. Contrasting this, as mentioned previously, commercial banks and financial institutions can also be given access to a user's transaction history and balance, though the method through which banks can access this data is very different. To facilitate this, the bank gives the user a one-time public key, which the user uses to encrypt their transaction history. The user creates equality proofs, and sends them with the transaction history to the bank. This information is needed for traditional financial system functions, such as issuing a credit score.

There is also the ability to add additional privacy measures to transactions, hiding the recipient's identity. To do so, the sender selects any number of additional users. The amount being sent is partitioned into three separate transactions, and three transactions are sent to the recipient and to each of the additional users selected, with only the transactions

being sent to the recipient having any value (the other transactions have a plaintext value of 0). In this way, it is impossible to tell who the actual recipient is, as all accounts are homomorphically updated, with only one account's balance actually increasing in value. The authors point out that ring signatures and zero-knowledge proofs could also be used to hide the identity of the sender, though it would undermine the effectiveness of the regulatory authorities.

There are a few areas not discussed in the KAIME paper that we can make estimates on, based in part on the previously discussed CBDC systems. First is that the speed of transactions is likely to be fairly slow, in the order of multiple seconds. Creating a single zero-knowledge proof in [27] took up to 1.5 seconds, so if extrapolating for KAIME, we can expect it to take up to 6 seconds to send the transaction, which is fairly slow. On the server side, verification and updating the ledger should be relatively quick, though the authors do not go into detail on how the server would operate beyond its basic functions. Lastly, there does not appear to be any thought given to implementing additional features such as interest-bearing, nor to offline transactions.

3.2.5 Conclusion and Summary of the Different Systems

As we can see, there are a number of different technologies that could be used when designing a CBDC, all with different advantages and disadvantages, giving central banks a number of different options depending on where their priorities lie. Of course, this is not an exhaustive overview, but does give an idea of the diversity available, and shows that a blockchain is just one of many possible technologies for a digital currency. The four systems we have discussed have been summarized into the table below:

Platypus	• ZKPs used to prove transactions are non-fraudulent • ZKPs provide user privacy; identity only disclosed when exceeding thresholds • Transaction creation can take time
Chaum-Style Blind-Signature System	• Tokens have discrete, specific units of value (like paper money) based on signature from the central bank • Blind signatures provide privacy to users from the central bank when tokens are being signed/redeemed • Signatures expire, requiring users to exchange tokens • Has two-tier architecture, with central bank responsible for minting and verifying tokens, while commercial banks handle transactions and ensure financial compliance
Project Hamilton	• UTXOs used to prevent double spending • Most mature architecture of the four, with the implementation using a distributed system and testing including fault tolerance.
KAIME	• Largest focus on privacy out of the four systems • Has two-tier architecture, similar to the Chaum blind signature system • Most complex of the four systems, using multiple ZKPs, threshold cryptography, and homomorphic encryption

Table 3.3: Summary of four CBDC systems

Chapter 4

Privacy Analysis and Offline Transaction Solution

One of the big issues facing CBDCs is the need to ensure compliance (i.e., AML/CTF) while protecting user privacy. This privacy issue is not only concerned with who can access user data, but also how user data is being protected, and what the consequences of such protection measures are. In this chapter, we will discuss and compare the privacy measures of some of the proposed CBDC, how they are providing users privacy, and who user data is being protected from, whether it be *public sector entities* (i.e., government, central bank, law enforcement, etc.), *private sector entities* (merchants, retailers, payment providers, etc.), or both.

4.1 Privacy implementation

As discussed in the previous chapter, when examining the implementation of privacy measures in CBDC systems, it is useful to classify them on a spectrum between "hard privacy" and "soft privacy" (as proposed by Auer et al.). For example, on either end of the spectrum we can look at *Platypus* for hard privacy, and *Project Hamilton* for soft privacy.

Summarizing the relevant details from the previous chapter, the Platypus CBDC system is reliant on ZKPs, and adds additional parameters to these proofs in order to protect user privacy. Users are required to provide their identity when creating a wallet, and these additional parameters specify under what conditions public sector entities can gain access to this data, which is otherwise protected. For example, a transaction may require the user

to disclose their identity if the transaction would put their balance above a specified threshold (holding limit), or if the value of the transaction itself is above a specified threshold (receiving limit). This use of cryptographic systems for its privacy measures firmly cements Platypus as using hard privacy measures. One limitation of the hard privacy implementation in this case is its rigidness due to the use of ZKPs. If any of the parameters needs to be changed, due to inflation for example, a new trusted setup would need to be generated, and any CBDC devices or applications would need to be updated to use this new trusted setup. Also, like other systems that rely on hard privacy, bad actors could avoid detection with prior knowledge of the thresholds and other hard privacy triggers.

Conversely, MIT's *Project Hamilton* relies almost entirely on soft privacy measures to protect user data. Like Bitcoin, *Project Hamilton* only provides users with pseudonymity, though it is an improvement over Bitcoin as the ledger (UHS) does not store identifiable information such as wallet addresses. As Project Hamilton does not provide means of ensuring compliance, users would likely need to link their wallet to their identity, much like with Platypus. Similar to current electronic payment systems, no guarantees are made by the system to protect user data. Instead the user must trust that public sector entities only access this data when issued a warrant or similar permission. In addition, wallet addresses and other identifiable information is supposed to be discarded when the transaction is stored in the UHS, though there is no mechanism preventing this information from being stored by public sector entities. Similarly, recipients (i.e., private sector entities) would still be able to collect information on user spending when receiving a transaction.

Looking at these examples, we get a better understanding of why Auer et al. believe a hybrid approach is necessary, where both hard and soft privacy measures are used to protect user privacy. This approach does have its own trade-offs, however. Namely, it greatly increases the complexity of the system, as can be seen in the CBDC system KAIME. In KAIME, transaction amounts are encrypted and hidden from public sector entities, but are guaranteed using ZKPs, and account balances are updated using homomorphic encryption. However, the contents of the transaction can be decrypted using threshold cryptography if enough parties (regulatory agencies) agree that the transaction/parties involved in the transaction needs to be investigated. Here, while there are many hard privacy measures being utilized, there are also soft privacy measures in the form of the regulatory agencies who have the power to decrypt transactions. As account balances and transaction amounts are hidden from public sector entities, transactions require four ZKPs. This complexity has a real world cost in the form of the time required to complete a transaction. Extrapolating from the benchmarks provided for Platypus (as KAIME does not provide any), proof generation alone could be expected to take around 6 seconds, a non-negligible amount of time. Though this does assume single threaded performance, the benchmarks in Platypus

were generated using a powerful and expensive phone (an iPhone 13). This presents a serious detriment to adoption, both due to the time that transactions will take, and also due to the lack of accessibility (requiring relatively powerful hardware, lest transactions take even more time). In order for a system like KAIME to see widespread adoption, serious changes would need to be made in order to reduce the complexity, and therefore the amount of time it takes to perform transactions.

4.2 Data Collection and Access

Broadly, when we examine privacy in CBDC systems, the two major parties we are concerned with hiding user information from are *public sector entities* (i.e., government, central bank, etc.) and *private sector entities* (merchants, retailers, payment providers, etc.). Taking a closer look at the CBDC systems discussed so far, which entities that user data is protected from varies greatly. From the previous section we know Project Hamilton only provides pseudonymity to the user from both private and public sector entities, and not true privacy. KAIME provides privacy from both entities, but at the cost of complexity and slow transaction generation (also discussed in the previous section). Platypus also provides user data protection from both private sector and public sector entities, though relies on hard privacy measures. Lastly, the Chaum blind signature system provides privacy from public sector entities but leaks data to some private sector entities.

Discussing these systems more in depth, Project Hamilton only provides pseudonymity, since transactions share information from the sender with the recipient, information that can be used to track the sender. Though this is limited to transactions that involve the same retailer or payment provider, the level of privacy is similar to that of credit cards. KAIME does not share data thanks to its usage of ZKPs and homomorphic encryption. Similarly, Platypus also uses ZKPs to hide information from both public and private sector entities. Finally, the Chaum blind signature system provides user data protection from the public sector through the use of blind signatures, which hides transaction details from the central bank. Though for private sector entities, while the transaction process does not provide the recipient with information that could be tied to the sender, the commercial banks (which the system relies on for the facilitation of transactions) can track transaction details. This puts the level of privacy provided by the system about on par with traditional electronic payment methods, where the payment processing company can still collect details on any transaction performed.

Platypus	• Hides data from public and private sector entities • Relies on hard privacy
Chaum-Style Blind-Signature System	• Privacy from public sector and *some* private sector entities • Commercial banks still have access to transaction details
Project Hamilton	• Pseudonymity from both public and private sectors • Soft privacy only
KAIME	• Privacy from both private and public sectors • Combination of hard and soft privacy • Considerably more complex and slower than other systems

Table 4.1: Comparison of privacy provided by the four CBDC systems

4.3 Hash Chains and User Privacy

While privacy protecting measures against public sector entities have been proposed, such as those found in KAIME (Section 3.2.4), this section will focus on providing privacy from non-governmental (i.e. private sector) entities. Specifically, in systems where the recipient is able to view the details of a submitted transaction, we can employ *hash chains* to provide authentication while hiding the sender's identity.

4.3.1 What is a Hash Chain?

Hash chains are created by repeatedly using the output of a hash function as input for the same function. Hash chains are a more specialized form of a Merkle tree (Section 2.3), where each node only has a single child. In this way, we are creating a line by "chaining" hash values, which is then often used for authentication. For example, given a seed x, we can create a hash chain of length 5 by calculating $h(h(h(h(h(x)))))$. A server can be supplied with the final hash value in the hash chain, denoted as $h^5(x)$ using the previous example, and when a user needs to be authenticated, the user only needs to supply the second to last value in the hash chain ($h^4(x)$). Each successive authentication uses the previous value in the chain relative to the most recently given value ($h^3(x)$, followed by $h^2(x)$, etc.). The server authenticates the user by hashing the value supplied by the user, and verifying that it matches the value it has stored. This method of authentication gains its security from the security properties of cryptographic hashing algorithms (Section 2.2), namely the collision resistance, second pre-image resistance, and pre-image resistance.

4.3.2 Using Hash Chains to Protect User Privacy

With traditional electronic payment methods (namely credit and debit cards), there is very little in the way of preventing parties from tracking an individual's spending, as credit card details[1] are given as part of the transaction. This is especially true for payment providers, as they are able to record transactions from a multitude of retailers that a card is used at. This gives payment providers access to a tremendous amount of personal information, which they can use to estimate other details, such as an individual's socioeconomic status. There do exist services, such as Apple Pay, that hide payment details from merchants and payment providers, though this simply shifts the burden of trust from one system to another. One of the goals of CBDCs is to provide privacy at a level similar to paper money, but with the convenience of traditional digital payments. Similar to how Apple Pay works, where merchants and retailers (along with their payment providers) are not given your true credit card number, we can obscure a user's digital wallet details by using a hash chain during a transaction to deny these private sector entities the ability to collect information about the customer.

Like normal hash chain identification, when creating a transaction, the sender can provide the $(n-1)^{\text{th}}$ hash in the chain, where n is the most recently used hash, as the sender's identifier. Then, when the central bank performs verification, they can identify the user by calculating $h(h^{n-1}(x))$, and verify that it matches the hash value stored for that user. After obtaining a user's account and performing the normal transaction verification and commitment, the account is also updated with the provided hash value $h^{n-1}(x)$. With this, the source of the funds can be hidden from private sector entities, while the central bank is able to identify the user. Finally, when providing the hash, we would want to encrypt it using the central bank's public key in order to prevent private sector entities from following a hash chain. As an aside, encrypting an identifier without the hash chain is not enough, as without the hash chain, the encrypted identifier would be the same in every transaction.

In systems where transaction details are visible to any entity, a hash chain of the recipient's address can also be used in place of their address in the transaction. While this would keep the recipient's identity obscured as well, it could add complications to sharing wallet addresses, especially for larger retailers due to concurrency issues (though privacy would likely be less of a concern for these entities). Sharing addresses using static media, such as a printed QR code, would also be much more difficult in this system, though not

[1] For the sake of simplicity, credit cards will be used as a stand-in for any other traditional electronic payment methods

impossible (e.g., having the QR code link to a URL which would provide an up-to-date address).

Lastly, if we would want to provide some level of protection from the central bank, we could take an approach similar to KAIME and use threshold cryptography to encrypt a user's identity, requiring a number of regulatory agencies to agree in order to discover the person associated with a hash chain. While this would only provide pseudonymity to the central bank, overall, it is much more simple than KAIME, and would provide stronger overall privacy protection when compared to Project Hamilton.

4.4 Offline Transaction Solution

One of the major concerns with offline transactions is how to prevent double spending. One possible solution to mitigate this risk is to incorporate heuristics into the payment systems used by merchants and retailers. The payment system could store a copy of the ledger locally, keeping it as up to date as possible. Then, in the event of an internet outage, the most recent copy of the ledger could be used to assign a risk factor for each potential offline transaction. Depending on how the ledger is set up, and specifically what information is publicly accessible, there are a number of factors that could be used in the heuristic analysis. For example, if accounts are only pseudonymous, the payment system could check the user's past transactions with the retailer, and use information such as how often and how long the user has shopped at the retailer, and if the transaction looks similar or dissimilar to past transactions. If account balances are publicly viewable, the retailer could even verify that the user had sufficient funds, and determine risk based on how large the difference is between the transaction amount and the account balance. Although not eliminating the risk, by employing heuristics, the risk of fraud could be reduced greatly, to the point that retailers could feel confident accepting these offline transactions. As a final measure, users could be penalized if they submitted fraudulent transactions. To comply with monetary policies, accounts in a CBDC system would need to be tied to a user's identity. As the transaction would be signed with the user's key private key, a retailer could provide proof that the user (or at least someone with access to the user's account) submitted a transaction for which they did not have the funds. Potential penalties could include various fines, or outright banning from the CBDC platform in the worst case.

Chapter 5

CBDC Prototype

For this thesis, a prototype CBDC system was created using XRPL. This system was built on top of Ripple's rippled service on Linux, in conjunction with their xrpl-py Python library. A web-based UI was created as well using React, with Flask for the backend. Notably, the system includes provisions for offline transactions, something that is sorrowfully lacking in the current literature.

The XRP ledger was chosen for a few reasons. First was its consensus protocol, as it aligns well with the requirements for a CBDC, especially compared to other blockchain protocols like proof-of-work. Compared to proof-of-work, the consensus protocol requires significantly less electricity [24], one of the main detriments of many blockchain systems. In general, XRPL's consensus protocol is well suited for CBDC, as CBDCs have a trusted party (i.e., the central bank). We are able to capitalize on that as the rippled setup allows designating specific machines as trusted validators. Another reason why the XRPL was chosen was its ability to easily exchange tokens between users, allowing for easy creation and exchange of CBDCs, and also allowing for the exchange of other CBDCs on the same XRPL network.

5.1 Overview

Within the system, both XRP and tokens, representing the CBDC currency, are utilized. In this case, an account's wallet containing XRP and meeting the XRP reserve signifies that the user it belongs to has been on-boarded and meets any compliance and regulatory requirements.

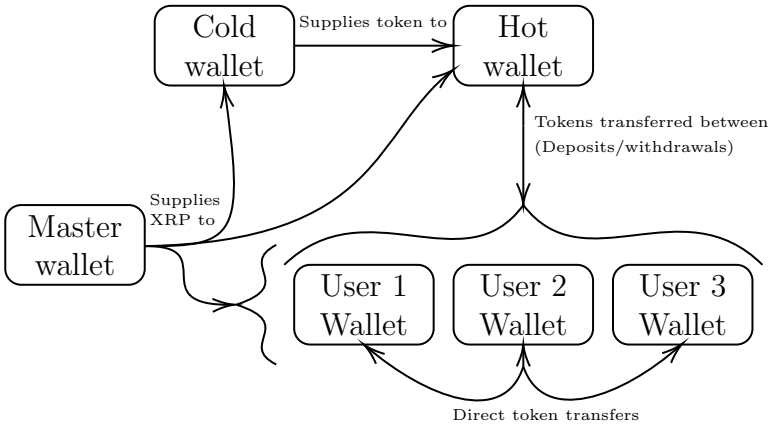

Figure 5.1: Wallet Setup and Interactions with Three Users

5.2 Setup

To start the system, three rippled processes are spun up to create the XRP ledger. A master wallet is created on initialization that is used as the source of all XRP. As stated earlier, wallets (accounts) must contain enough XRP to meet the account reserve, and when a wallet is created, the master wallet provides enough XRP to meet this threshold, signifying that a user has been on-boarded. Once the ledger is running, two wallets are created in order to handle the minting and distribution of tokens. A "cold wallet" is used to generate the CBDC tokens, which it transfers to a "hot wallet" responsible for the actual distribution of the tokens (this is in line with Ripple's recommended practices for issuing tokens). When creating a new account, a user first generates a new wallet. This includes a private/public key pair, as well as the user's wallet's address. The user will be given this address when being on-boarded, so that XRP can be transferred to their wallet, once again signifying that they have met any compliance requirements. The final step is to create a line of trust between the user's wallet and the token-issuing cold wallet so they can receive the CBDC tokens. The various wallet relationships and transactions are summarized in Figure 5.1.

5.3 Transactions Process

The transaction process follows the normal XRPL token transaction flow, as illustrated in Figure 5.2. When executing a transaction, a payment object is first created, which

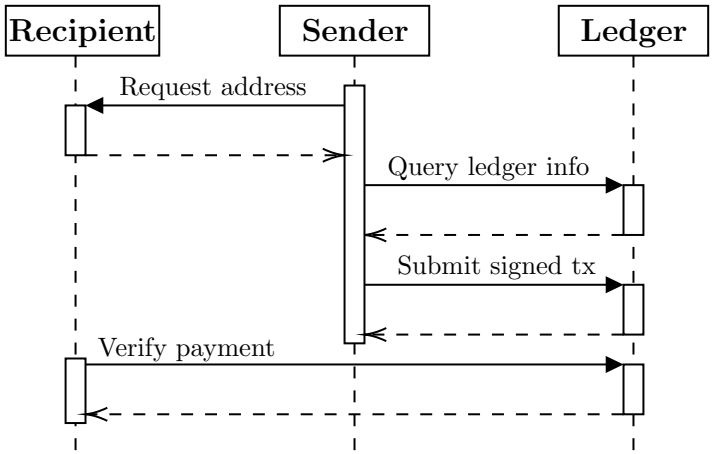

Figure 5.2: Transaction Sequence Diagram

includes the address of the sender's wallet, the address of the destination wallet, and the quantity and type of token being transferred. When connected to the network, some of these fields, such as specifying which ledger index the transaction must be submitted by, can be populated automatically during the signing process by querying the network. This payment object is then signed by the sender before being submitted to the ledger. The ledger then verifies the transaction before committing it to the ledger and notifying the sender that the transaction completed successfully. The recipient can then query the network and verify that their funds have been updated.

5.3.1 Offline Transactions

In this system, intermittent offline transactions can also be performed. To do so, a wallet address is shared with the sender, who specifies how many tokens to transfer. As the user is offline, their device must fill the payment object parameters by itself. The only real consequence of this is that the transaction fee cannot be obtained from the ledger (though this would likely always be zero when deploying XRPL in a CBDC system), and that the maximum ledger index for the transaction must be decided without knowing the current ledger index. The problem this presents is that the ledger index is incremented by 1 each time it updates, and transactions with a ledger index less than the current index will be rejected. This does not pose as large a problem as one might think, however, as an appropriate maximum ledger index can be easily estimated. For example, an arbitrarily large value may be selected if there is no expiry on the transaction, otherwise, an estimate

can be obtained by taking the value and time of generation for the last known index. From there, given a target expiry time, an estimate could be calculated and used as the maximum ledger index, as the ledger updates regularly every 3-5 seconds. It would likely be necessary to select an index on the larger side of the estimate, as the transaction will be rejected if the index selected is too small, while a larger estimate only provides a larger window during which the transaction can be submitted. Once the payment object has been created and signed, it can be serialized and shared with the receiver, so that either the sender or receiver could submit the transaction once they were back online. The prototype also includes functionality to automatically submit these transactions once the user comes back online, and also keeps track of their balance and account sequence number. This allows the system to keep track of the number of transactions that need submitting, the order they need to be submitted in, and also will not allow the user to create transactions that require more funds than they have available/have already promised to spend.

Here, we can employ the heuristics mentioned in Section 4.4. Specifically for our XRPL-based system, we can check a user's funds as proposed previously, as well as see how many transactions an individual has purported to have performed while offline (as an individual's account sequence number is provided as part of the transaction, and is also stored in the ledger). A large purported number of offline transactions could be deemed more risky, and in such a case, the merchant could also request to be provided with signed copies of these previous transactions. With this, the merchant would also not have to wait for the previous transactions to be submitted to the ledger before theirs could be, as all transactions could be submitted by the merchant. It would also allow the merchant to be more confident about the self-reported state of the user's account. This does raise privacy concerns, as it would give the merchant access to previous transactions. In future work, this could be expanded, encrypting the transactions with a public key from the central bank to hide the details of the transactions, and perhaps utilizing ZKPs to show that the sum of the transactions is as reported by the user.

Expanding on the penalties also mentioned in Section 4.4, should a user be found to have provided a fraudulent transaction (or other financial crimes) using the CBDC system, native XRPL functionality could be leveraged in our system. If XRP transactions by non-authorized parties are banned, for example, the minimum reserve could be increased. By increasing the reserve while simultaneously distributing XRP to all other accounts such that they met this new reserve, a user/users could effectively be banned from the system. Other limitations could also be put in place, such as limiting the number of transactions a user could perform, or requiring the user to pay a specified fine before allowing them access to the platform again.

5.4 Experimental Results

This section will discuss the results from testing of the CBDC prototype system. The testing was performed on a computer equipped with an Intel Core i5-11400, 80GB of RAM (two 8GB sticks and two 32GB sticks), and a 7200 RPM SATA HDD. As discussed in the last section, all three rippled instances were running on this machine, with the Python xrpl-py library being used to conduct the transactions. Although the hardware and rippled configuration is not ideal, it was used due to time and resource constraints, as well as due to the changes made to the API (discussed more in Chapter 6).

To perform the experiment, a total of 4500 transactions were executed, with the amount of time taken for the transactions to complete being recorded. These transactions consisted of unique wallet pairs being selected, and a single tokens being transferred from one wallet to another. The test completed in approximately 286.88 seconds, giving an average throughput of 15.7 transactions/second (Figure 5.3). While performance may have been lacking, it is not a true indication of the potential capacity, as real-world XRP ledgers are able to reach much higher throughputs (up to 1500 transactions/second [29]). There are a few likely causes for this low performance. First is the setup of the system, with three instances of the rippled service running on a single machine. The XRPL requires a minimum of three instances, and while these can all be run from a single machine, the intended deployment is to be across multiple machines. Deploying the instances on a single machine likely led to a significant degradation in performance, and resource monitoring during the benchmark indicated a CPU bottleneck. If the CBDC system were to be redeployed with remediation of these factors, we would expect to see throughput much closer to existing XRPL networks, as only minor configuration changes were made to rippled for the CBDC prototype.

Looking at the data more closely, each transaction took between 5 and 32 seconds (Figure 5.4). Mean transaction time was 20.80 seconds, with a 90^{th} percentile of 23.92 seconds and 99^{th} percentile of 27.12 seconds (Figure 5.5). This corresponded to a standard deviation of 2.60 seconds. We can also examine the number of ledger indices that passed between a transaction being submitted and being included in the ledger (i.e. ledger deltas). This tells us how many times the ledger was updated between when the transaction was submitted, and when it was verified. Overall, the ledger was updated 85 times during the 286.88 seconds it took to complete the 4500 transactions, and all transactions were completed in between 1 and 8 updates (Figure 5.6). The mean number of ledger deltas was 5.81, with a both a 90^{th} and 99^{th} percentile of 7 ledger updates. This, and the low standard deviation of 0.716, can be seen in (Figure 5.7).

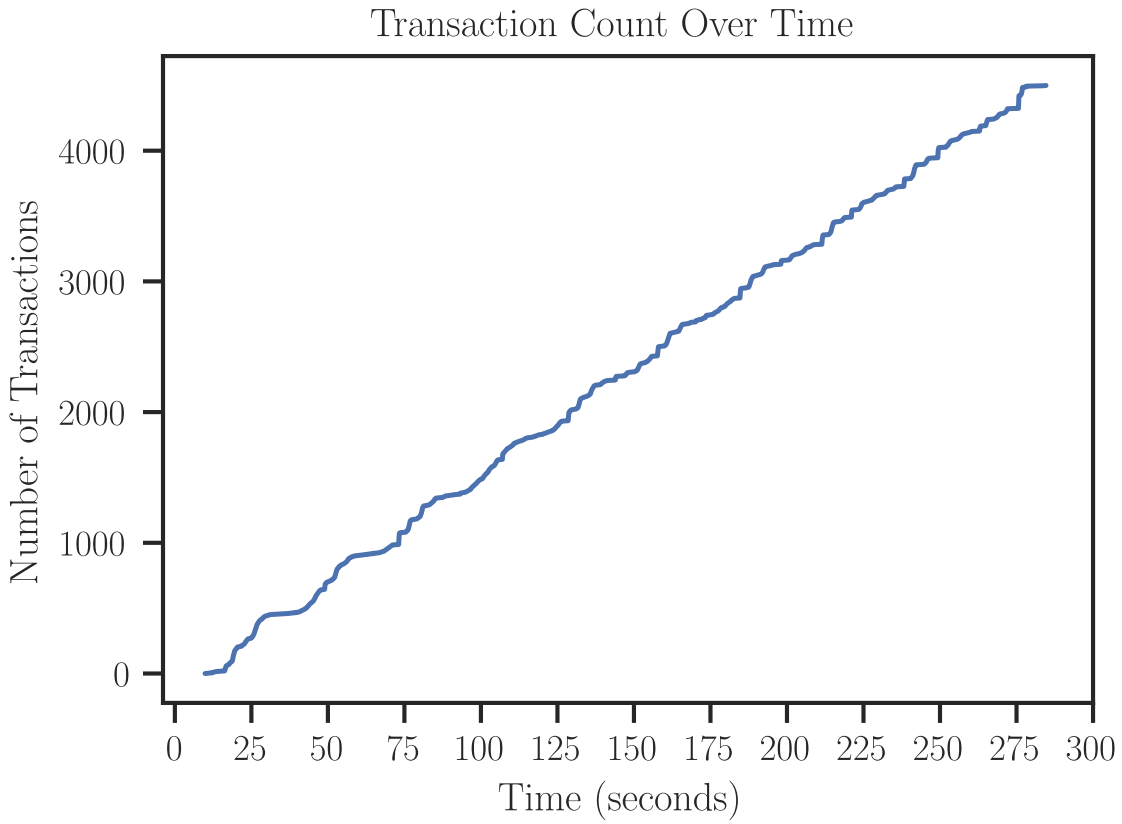

Figure 5.3: Number of Transactions Processed Over Time

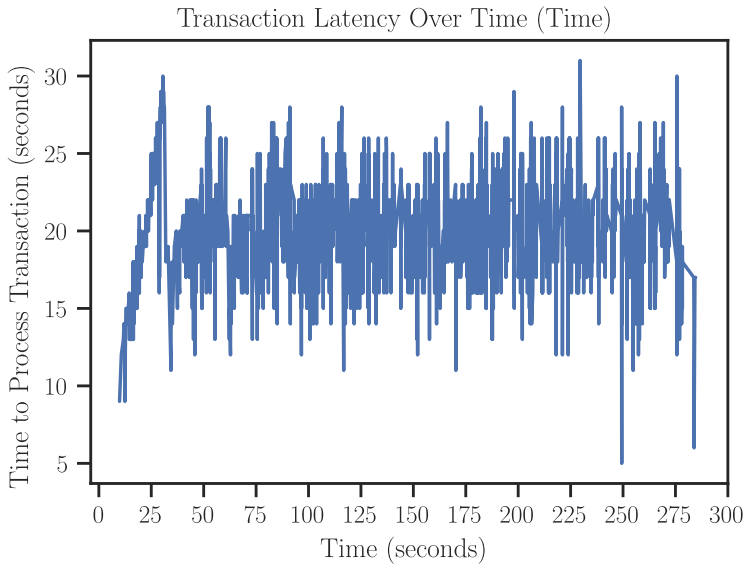

Figure 5.4: Number of Seconds It Took to Process a Transaction Over Time

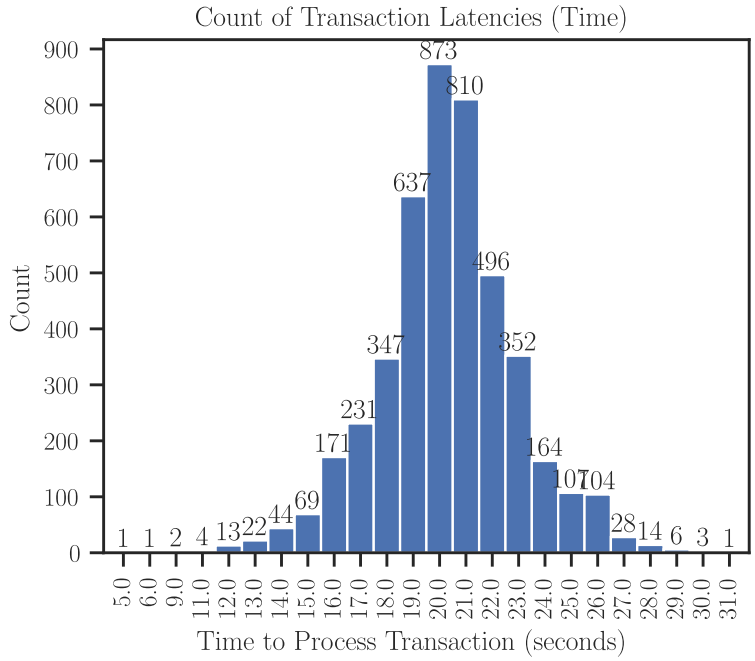

Figure 5.5: Distribution of Time Taken to Process a Transaction

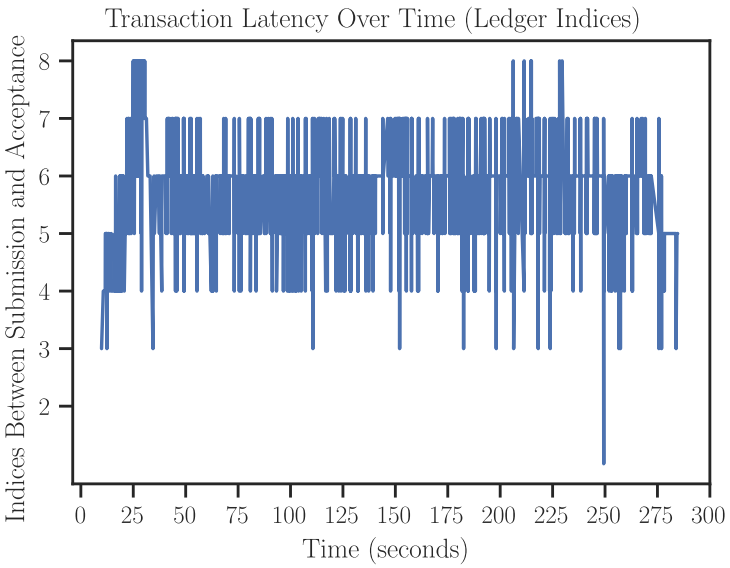

Figure 5.6: Number of Ledger Indices Elapsed During Transaction Processing Over Time

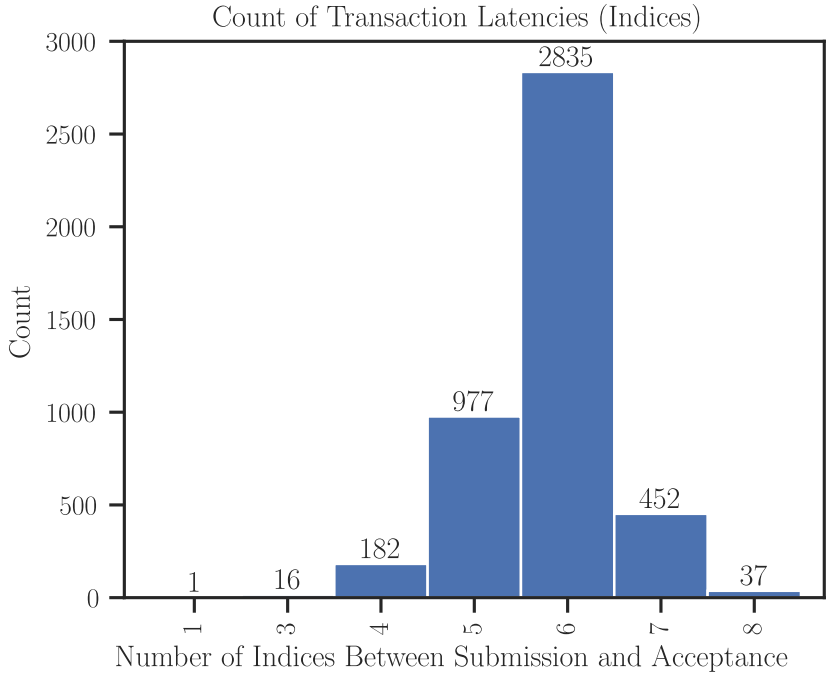

Figure 5.7: Distribution of Number of Ledger Indices Elapsed During Transaction Processing

Chapter 6

Conclusion and Future Work

6.1 Conclusion

Two of the large, outstanding issues still facing CBDCs are that of privacy and offline transactions. How to adequately handle privacy and user data is a topic of great importance for central banks looking to issue a digital currency, and there is still much room for discussion. CBDCs not only need to address how user data is being protected and under what circumstances it can be accessed, but also who the user data is being protected from (whether it be the government, merchants, or other third parties), and how user data could still be collected. Equally as important is the impact these privacy measures have on compliance measures, and how to balance the two. Using hash chains was one solution that was proposed in this thesis, although as can be seen from how different each of the examined systems handled privacy, it is far from the only solution.

As noted previously, the ability to perform transactions offline is an important requirement for many central banks, yet there seems to be a major disconnect in the perception of this importance between central banks and researchers. Presented in this thesis was a prototype CBDC system built on Ripple's XRPL, in which transactions can be serialized and submitted at a later date in order to mimic the behaviour of offline transactions, and the proposal that this system could be combined with heuristics in order to provide a greater degree of trust in the transaction, even if the transactions could not be guaranteed cryptographically.

6.2 Future Work

To build on what was presented in this thesis, the prototype CBDC system could be expanded further, deploying it to a larger network, and implementing some of the ideas presented here. An easy first step would be the implementation of heuristics in order to assign risk factors to presented offline transactions, and to use that risk factor to determine if the transaction should be accepted or rejected. This could be combined with a service to review fraudulent offline transactions, and to issue penalties accordingly. In addition, deploying the system as a proper XRPL network, rather than on a single machine, should allow for higher throughput and lower latencies. Unfortunately, between the development of the prototype and the time of writing for this thesis, the rippled service and xrpl-py libraries were updated. This update changed the API, making programs and scripts written for the older version (like the CBDC system) incompatible. To run the prototype, therefore, would require an update/rewrite, or for the user to manually install the older versions of the service and Python libraries (rippled version 1.10.1-1 and xrpl-py version 1.8.0).

An alternative way to expand in this work could be a CBDC implementation featuring the privacy measures presented here. There are factors that would need to be considered if hash chains were used to hide a user's identity, such as balancing the length of the chain with the time it would take to generate, and what to do when approaching the end of the hash chain. This raises additional questions, such as:

- Would a new chain be issued, and if so, how would it be generated?
- How would any changes be synchronized if the user had multiple devices?
- What are the security concerns that arise with the chosen method?

If a dataset is available, it would also be pertinent to determine what data a government entity could extract if users were pseudonymous. Also, what data would a government entity need to collect in order to determine a user's identity, and how could this data be hidden? These are important factors to consider, as they are likely the weakest part of the proposal.

Likely any system built with an emphasis on privacy would not be based on XRPL, due to the pseudonymous nature of its ledger, but seeing as the rippled service is open source, an ambition extension of this thesis could be combining the two approaches, providing greater levels of user privacy, while also offering the ability to submit transactions offline.

This would complicate determining the risk factor of a submitted transaction, as some information that could be extracted before, such as the number of transactions the user has performed while offline, could no longer be determined. Ultimately, a different set of heuristics would likely need to be used, and would probably be unable to offer the same degree of confidence.

References

[1] Raphael Auer, Rainer Böhme, Jeremy Clark, and Didem Demirag. Mapping the privacy landscape for central bank digital currencies: Now is the time to shape what future payment flows will reveal about you. *Queue*, 20(4):16–38, sep 2022.

[2] Elaine Barker, Lily Chen, Allen Roginsky, Apostol Vassilev, and Richard Davis. Recommendation for pair-wise key-establishment schemes using discrete logarithm cryptography. Technical report, National Institute of Standards and Technology, 2018. Last accessed 01 December 2023.

[3] Daniel J. Bernstein and Tanja Lange (editors). eBACS: ECRYPT Benchmarking of Cryptographic Systems. https://bench.cr.yp.to/results-nistlwc-hash.html, Last accessed 30 November 2023.

[4] Vitalik Buterin. Ethereum: A next-generation smart contract and decentralized application platform., May 2023. Last accessed 01 December 2023.

[5] George Calle and Daniel Eidan. Central bank digital currency: an innovation in payments, April 2020. Last accessed 01 December 2023.

[6] David Chaum. Blind signatures for untraceable payments. In David Chaum, Ronald L. Rivest, and Alan T. Sherman, editors, *Advances in Cryptology*, pages 199–203, Boston, MA, 1983. Springer US.

[7] David Chaum, Christian Grothoff, and Thomas Moser. How to issue a central bank digital currency, 2021. Last accessed 01 December 2023.

[8] Quynh H. Dang. Secure hash standard. Technical report, National Institute of Standards and Technology, 2015. Last accessed 01 December 2023.

[9] Sriram Darbha and Rakesh Arora. Privacy in CBDC technology. Report, Bank of Canada, 2020. Last accessed 01 December 2023.

[10] Ali Dogan and Kemal Bicakci. KAIME : Central bank digital currency with realistic and modular privacy. Cryptology ePrint Archive, Paper 2023/713, 2023. https://eprint.iacr.org/2023/713. Last accessed 01 December 2023.

[11] James Lovejoy, Cory Fields, Madars Virza, Tyler Frederick, David Urness, Kevin Karwaski, Anders Brownworth, and Neha Narula. A high performance payment processing system designed for central bank digital currencies. White paper, Digital Currency Initiative, Massachusetts Institute of Technology, February 2022. Last accessed 01 December 2023.

[12] Ralph C. Merkle. Method of providing digital signatures, 09 1979. US Patent 4309569.

[13] Cyrus Minwalla, John Miedema, Sebastian Hernandez, and Alexandra Sutton-Lalani. A central bank digital currency for offline payments. Report, Bank of Canada, 2023. Last accessed 01 December 2023.

[14] Satoshi Nakamoto. Bitcoin: A peer-to-peer electronic cash system, 2009. Last accessed 01 December 2023.

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

[16] Bank of Canada. Contingency planning for a central bank digital currency, 2020. Last accessed 25 September 2023.

[17] Bank of Canada, European Central Bank, Bank of Japan, Sveriges Riksbank, Swiss National Bank, Bank of England, Board of Governors Federal Reserve System, and Bank for International Settlements. Central bank digital currencies : Foundational principles and core features. Report, Bank for International Settlements, 2020. Last accessed 01 December 2023.

[18] Bank of Canada, European Central Bank, Bank of Japan, Sveriges Riksbank, Swiss National Bank, Bank of England, Board of Governors Federal Reserve System, and Bank for International Settlements. Executive paper - central bank digital currencies : Foundational principles and core features. Report, Bank for International Settlements, 2020. Last accessed 01 December 2023.

[19] Edwin Ayisi Opare and Kwangjo Kim. A compendium of practices for central bank digital currencies for multinational financial infrastructures. *IEEE Access*, 8:110810–110847, 2020.

[20] Bryan Parno, Craig Gentry, Jon Howell, and Mariana Raykova. Pinocchio: Nearly practical verifiable computation. Cryptology ePrint Archive, Paper 2013/279, 2013. https://eprint.iacr.org/2013/279. Last accessed 25 September 2023.

[21] Wouter Penard and Tim van Werkhoven. On the secure hash algorithm family. *Cryptography in context*, pages 1–18, 2008.

[22] Maksym Petkus. Why and how zk-SNARK works. *CoRR*, abs/1906.07221, 2019. Last accessed 25 September 2023.

[23] Ronald L. Rivest, Adi Shamir, and Leonard M. Adleman. Cryptographic communications system and method, 12 1977. US Patent 4405829.

[24] Crystal Andrea Roma and M. Anwar Hasan. Energy consumption analysis of XRP validator. In *2020 IEEE International Conference on Blockchain and Cryptocurrency (ICBC)*, pages 1–3, 2020.

[25] Adi Shamir. How to share a secret. *Commun. ACM*, 22:612–613, 1979.

[26] S. Shetty, C.A. Kamhoua, and L.L. Njilla. *Blockchain for Distributed Systems Security*. Wiley, 2019.

[27] Karl Wüst, Kari Kostiainen, Noah Delius, and Srdjan Capkun. Platypus: A central bank digital currency with unlinkable transactions and privacy preserving regulation. Cryptology ePrint Archive, Paper 2021/1443, 2021. https://eprint.iacr.org/2021/1443. Last accessed 01 December 2023.

[28] XRPL. Consensus protocol - xrpl.org, 2023. https://xrpl.org/consensus.html. Last accessed 25 September 2023.

[29] XRPL. XRP - xrpl.org, 2023. https://xrpl.org/xrp-overview.html. Last accessed 01 December 2023.