Decentralized Utilization Incentives in Electronic Cash

by

Jeremy Lloyd Rubin

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

Master of Engineering in Electrical Engineering and Computer Science

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

June 2016

Author . . . .
Department of Electrical Engineering and Computer Science
May 17, 2016

Certified by . . . .
Ronald Rivest
Institute Professor
Thesis Supervisor

Accepted by . . . .
Dr. Christopher J. Terman
Chairman, Master of Engineering Thesis Committee

Decentralized Utilization Incentives in Electronic Cash

by

Jeremy Lloyd Rubin

Submitted to the Department of Electrical Engineering and Computer Science
on May 17, 2016, in partial fulfillment of the
requirements for the degree of
Master of Engineering in Electrical Engineering and Computer Science

Abstract

Many mechanisms exist in centralized systems that incentivize resource utilization. For example, central governments use inflation for many reasons, but a common justification for inflation in practice is as a means to incentivize resource utilization. Incentives to utilize resource may stimulate economic growth. However, the asymmetry of economic control and potential abuses of power implicit in centralized systems may be undesirable. An electronic cash design may be able to create resource utlization incentives via decentralized mechanisms. Decentralized mechanisms may be economically sustainable without centralized and potentially coercive forces.

We propose Hourglass, a novel electronic cash design that provides a decentralized mechanism to encourage utilization via expiration dates. Constructed in this way, decentralized utilization incentives may have less potential for coercive abuses than more centralized methods, but may be similarly effective in their ability to incentivize utilization. We present the Hourglass system at multiple levels of detail: a design overview, a minimal kernel framework, a series of descriptive refinements, and a concrete implementation as a fork of Bitcoin (a popular electronic cash protocol in common use). We also present several potential applications of Hourglass, such as renewable resource markets, spectrum allocation, stock issuance, and currency.

Thesis Supervisor: Ronald Rivest
Title: Institute Professor

Acknowledgments

I am indebted to my advisor Ron Rivest for his constant feedback, guidance, and mentorship during the production of my thesis. Thank you, Ron, for all that you have taught me. I am incredibly grateful to have had the opportunity to study under you.

I also must express my gratitude for the guidance and support of my mentors Joi Ito, Ethan Zuckerman, Andrew Lippman, and Nickolai Zeldovich.

A special thanks to my friends and colleagues, especially Madars Virza, Chris Peterson, and Neha Narula, for all their support during the production of this thesis.

I also must express my gratitude toward Daniel Elitzer for his friendship and for always encouraging for me to reach further in my work.

Additionally, thanks are in order to the Electronic Frontier Foundation. The Electronic Frontier Foundation represented me when my first cryptocurrency project resulted in an outrageous subpoena from the state of New Jersey that threatened to quell my interest and passion for this space.

The acknowledgments would not be complete without mention of the loving support of my mother, brother, and sister; Rachel, Zachary, and Emily. I am lucky to have such an amazing family.

Lastly, I dedicate my thesis to my deceased father, Lawrence Rubin MIT S.B. 1974 M.Eng 1975. He continues to be an inspiration to me throughout my life.

Contents

List of Figures

List of Tables

List of Algorithms

Chapter 1

Introduction

An electronic cash scheme is a protocol for tracking ownership of a set of assets or tokens using primitives and concepts from computer science and engineering, primarily cryptography and the Internet. The promise of electronic cash is more accessible banking systems that are less prone to fraudulent corruption with better user privacy and security. These concerns are not minor. There are 2 billion adults who are unbanked in the world according to a recent Gallup poll [56], while smartphones – networked general purpose computing devices – are expected to be owned by almost all adults in 2020 according to the 2015 Ericsson report [22]. The discrepancy between access to banking and networked computing devices implies that electronic cash systems that only need a networked computing device to participate have a real opportunity to offer these critical financial services to much of the world. The need for increased security and privacy in all systems has become more clear in recent years with many high profile cases such as Edward Snowden's leaks uncovering mass surveillance, the Panama Papers' uncovering major financial fraud, the exposure of Sony's private information revealing salary information, and the Federal Bureau of

Investigation cracking the San Bernardino iPhone's security.

Electronic cash schemes first became a popular research topic in 1982 when David Chaum published *Blind Signatures for Untraceable Payments* [12]. After a significant decrease in interest in electronic cash schemes during the years following the failing of Chaum's company, Satoshi Nakamoto's release of Bitcoin in 2008 has driven a resurgence of interest in this area [42].

A typical academic treatment of an electronic cash scheme attempts to address three issues, listed below:

1. ***Decentralization, distribution, and fault tolerance****, the reliability of the electronic cash scheme.*

2. ***Privacy and fungibility****, the protections afforded to users of the electronic cash scheme.*

3. ***Custodianship****, possession and control of the tokens or assets in the electronic cash scheme.*

This thesis develops notions and offer novel designs for a fourth category: ***Economic and social robustness****, the functionality of the electronic cash scheme's asset issuance and management capabilities.* In this thesis we define economic and social robustness of the asset issuance protocol and show that its consideration as a first-class citizen in the design of an electronic cash scheme provides fertile ground for developing new applications of electronic cash technology[1].

[1]If a design principle is a first-class citizen, its considerations take priority, perhaps at the expense of other principles.

1.1 Goals

The goal of this thesis is to develop an economically and socially robust electronic cash scheme. We define economic and social robustness to be the fulfillment of economic and social goals described in this section. To fulfill these economic and social goals, we require a novel set of features. The main novel feature we implement is a decentralized utilization incentive. The design and implementation of these decentralized utilization incentives is discussed in Chapters 4, 5, 6, and 7. The discussion of economic and social goals in this section should help to make clear what lead to such a design.

While it might be said that economic goals and social goals are the same kind of goal, we use these categories to loosely differentiate between goals that are more quantitative or qualitative.

There are other important goals that are not the focus of this work, such as privacy. Other works, such as Zerocash [23], compose well with this design to meet desired privacy goals, but including such goals as a priority in this work adds needlessly complexity.

1.1.1 Economic Goals

We define an economy to be the business conducted by a set of actors involving a set of resources. Thus, in our analysis, we can either *universally quantify* the economy over all actors and all resources or restrict it to a smaller set such as diamond dealers and diamonds. This distinction is important as there may be some markets where our technology works and others where it does not. We set goals to be market growth and stability as described in the following subsections.

1.1.1.1 Market Growth

We take growth as a goal in the sense that we want to encourage an economy to become more efficient and to create more wealth. This goal is both long term and short term. We take the Keynsian viewpoint that this mechanism should encourage the utilization of available resources. Alternatively, from a Hayekian view, it can also be said that the goal is to eliminate factors that discourage utilization by making it more equitable to not utilize currency[2]. Rather, the goal is to strike a balance by evolutionarily converging towards the rate of utilization incentive that optimally distributes resources. The optimality resource distribution is not easy to quantify fully, but there are techniques such as Pareto efficiency which characterize it.

1.1.1.2 Market Stability

We take stability as a goal in the sense that we do not want an economy that has turbulent or catastrophic episodes. Thus we are taking as an axiom that the avoidance such periods is desirable. That the avoidance of dramatic episodes is desirable is not unequivocally well established. For example, the culmination of David Graeber's *Debt: The First 5000 Years* is a recommendation to synchronously destroy all of the currency or records in a Jubilee [28]. There may be a benefit to such instability, but we make it a non-goal of our work as Graeber's call for it is not well justified (as discussed in Chapter 2.1).

[2]Section 2.2 covers the teachings of Hayek and Keynes in more detail.

1.1.2 Social Goals

1.1.2.1 Non-Coercive

It is desirable for the system to be non-coercive insofar as we want to minimize the potential for any individual to be forced into a specific type of behavior because of the behavior of another individual. This coercion can arise as a consequence of a limited set of choices. Maximizing individual liberty, or preventing others' ability to cause such influence over choice is paramount. The chief mechanism we use to ensure non-coerciveness is competition. Sufficient competition gives an individual potentially many options to choose from, allowing them to avoid onerous or unfavorable conditions.

For example, a university that requires all of its students take a single specific class is an example of coercive behavior[3], whereas a requirement which may be fulfilled by many classes offers students more liberty to select a class that they believe is most beneficial to them[4]. As a concrete example of potentially bad coercion, MIT has a requirement that students take Electricity and Magnetism, which may not be useful if a student's major course of study is finance. As a concrete example of potentially good non-coerciveness, MIT offers several versions of Electricity and Magnetism, enabling students to take a version of the course which matches their desired level of rigor and learning style. This example demonstrates both how a multiplicity of choices is a proxy for non-coerciveness and how a multiplicity of choices can help improve outcomes.

[3]Noting that in this hypothetical, students are not required to attend university at all.

[4]Or the class that the hypothetical student's advisor believes, with their personal relationship with the student and years of experience guiding students to academic success, is most beneficial for the student to take.

1.1.2.2 Participatory

The system should be participatory in the sense that users should be able to experiment with new behavior or practice nonstandard behavior. The ability to try new behaviors is derived from Hayek's infatuation with competitive evolutionary processes. By allowing users to shape the way they use the system, social norms can evolve based on what is most efficient in practice by evidence, not theory.

1.1.2.3 Decentralized

The system should not cause institutions of power, such as governmental or banking agencies, to become disproportionately powerful nor should it heavily lean upon centralized infrastructures to accomplish its goals. If it were to be the case that either of these were likely to occur then it is equivalent to centrally setting parameters as done in Freicoin (discussed in Chapter 2), or dynamically determining parameters by a cabal of trusted parties. Centralized controls or design choices do not necessarily lead to bad conditions in the short term, but in practice centralized controls and design choices often do lead to undesirable effects so we choose to avoid centralization where possible.

1.2 Motivation

The economic and social robustness of an electronic cash scheme is typically not addressed as a first-class citizen in the literature. There are other aspects of electronic cash that are less often addressed as first class citizens in the literature as well, such as protocol scalability[5], but we do not make such issues the focus of this thesis.

Economic and social robustness is of incredible importance to the long term success of electronic cash schemes. Despite the importance of economic and social robustness, these factors seem to have been of secondary concern for early-days Bitcoin compared to the issues Bitcoin had to solve to gain initial adoption such as correctness of code, developer awareness, bad actors, and growth of the mining market[37]. As Bitcoin and other electronic cash schemes expand their influence and role in society, it is critical to take a closer look at these issues. Presently, Bitcoin has a relatively simple fixed-supply economic model, meaning there is a limited number of bitcoins that ever circulate. Other popular alternative designs have similarly simple economic models. Table 1.2 provides information on several designs. The demonstrable need for novel economic models for electronic cash schemes is the motivation for this thesis.

Scheme	**Economic Model**
Bitcoin	Fixed supply, distributed along a decreasing schedule [42].
Stellar	Fixed Inflation, democratically distributed to be more equitable [53].
Ethereum	To Be Determined [63]
Dogecoin	Fixed Circulation/Hyperinflation[a] [24]
"Private Blockchains"	Usually, centralized issuance

[a]The reader should rightly be confused and amused, Dogecoin begun as a joke but became a success because of its quirky charm.

Table 1.1: Economic models of various electronic cash schemes

[5]Scalability in terms of outright design of protocol. There are works designed to layer on top of Bitcoin to be more scalable such as the Lightning Network [49]. Earlier works such as Peppercoin focus on scalability [41].

1.3 Approach

In general, electronic cash schemes offer new mechanisms for enforcement of arbitrary rules, thus opening the possibility of decentralized utilization incentives. This thesis examines an alternative economic model for electronic cash schemes based on the notion that assets that expire can encourage utilization before expiration, which is a positive economic outcome.

A coin where a percentage of the coins are *taken out* of circulation each day (and potentially reissued) is simple example of asset expiration. In contrast, a coin where a percentage of coins is *added* to the circulating supply each day is a simple example of inflation. Both inflation and asset expiration provide a mechanism to discourage saving and encourage utilization. Inflation is not a strict opposite of expiration; both inflation and expiration are mechanisms by which the value of a held asset is reduced over time.

The Keynesian economic model uses inflation to encourage utilization. Inflation is a highly centralized means of control as it requires some knowledgeable party to enact rate-setting. In contrast, asset expiration has the potential to be done in a decentralized manner in an electronic cash if participants can individually set expiration rates on their expenditures.

In this thesis we explore the construction of Hourglass, an electronic cash scheme with decentralized utilization incentives constructed with expiration rates, and its potential applications. To construct and analyze this scheme, we begin by defining a kernel of functionality that implements the minimal set of behaviors needed given strong assumptions, refine the kernel implementation to lend itself better to implementation and understanding, and then implement it in a practical manner.

1.4 Limitations of Prior Work

Previous work in this space is limited in several key ways. Traditional assets do not get the benefits that electronic cash may confer. Prior electronic cash schemes are either completely centralized or provide no mechanisms to encourage economic growth. The limitations of prior work is detailed in Chapter 2.

1.5 Applications

There are many potential applications of Hourglass detailed in Chapter 3. We briefly describe one such application here so that the reader may better contextualize the arc of this thesis.

Many types of resources are renewable in some sense, that is, we may utilize the resource fully at some point but there is more available in the future. An example of this is solar energy, there is a steady supply of sunlight but it is impossible to extract it at a faster rate than the rate at which sunlight arrives. Sunlight is an interesting resource because there is also a phase delay in the times in which it is available across regions (due to the sunrise-sunset differential). It is also very difficult to store energy rather than utilize it immediately. Therefore, one cannot buy more solar energy than that which is immediately available, and because of the phase delay one cannot just use solar locally[6]. By accounting for solar energy usage rights with a decentralized utilization incentive, it is possible to develop a market for solar energy that explicitly accounts for the availability of solar energy and prevents serviceable demand exceeding available supply. As electricity is difficult to transport over large distances, derivative goods that rely on electricity such as computation can also be

[6]We exclude batteries as they have low energy densities, are expensive, and lossy.

used. This application is given in more detail in Chapter 3.

1.6 Contributions

The core contributions of this thesis are:

- New economic model with decentralized utilization incentives.
- Novel asset tracking system with primitives around expiration.
- Several real world problems demonstrating this model and solutions using the novel asset tracking system.

1.7 Organization of Thesis

The remainder of this thesis is organized as follows: Chapter 2 details the history of economics, money, and currency and explores works related to this thesis; Chapter 3 describes real world problems we may offer solutions for in this thesis, Chapter 4 introduces the main Hourglass analogy to a physical hourglass, Chapter 5 defines a kernel framework that expresses the behaviors needed by Hourglass; Chapter 6 clarifies the kernel laid out in Chapter 5 and refines terminology; Chapter 7 discusses implementing Hourglass on top of real world systems like Bitcoin; Chapter 8 describes potential applications and uses for Hourglass as solutions to the problems laid out in Chapter 3; finally, Chapter 9 evaluates Hourglass in comparison to other works and discusses areas for future work.

Chapter 2

History and Previous Work

This chapter reviews some of the major history and related economic theory as well as previous work.

2.1 History of Money

Money is not a new idea. However, much of the common narrative around money's development were ideas popularized by Adam Smith's *Wealth of Nations.* His narrative is flawed, as he thought that instruments of currency follow from a barter economy, and systems of debt and credit follow from currency. As anthropologist Caroline Humphrey argues, "No example of a barter economy, pure and simple, has ever been described, let alone the emergence from it of money; all available ethnography suggests that there never has been such a thing." [30].

An alternative history is given by anthropologist David Graeber in *Debt: The First 5000 Years*, where credit and debt precede currency. The cause of misconceptions about the origins of credit, debt, and currency is, according to Graeber, that currency[1]

[1] To be precise, the *coinage*, the physical artifacts of a currency and not their backing.

creates easy to verify evidence whereas credit and debt can be more informally established without leaving any physical artifact of bookkeeping in the anthropological record. Money, in Graeber's view, is a means of making obligations and responsibilities into quantified debts [28].

Graber makes the case that for most of history, exchange was based on gift giving, as a system of informal debts. A gift economy was sufficient condition for society to form, as individuals were able to separate their responsibilities and specialize. The invention of currency, allowed these gifts to be precisely quantified and fungible across much larger communities. Graeber contends that this is problematic because the exact quantified value of gifts turns into obligation, effectively becoming slavery [28].

One of Graeber's key theses is that because mechanisms of credit and debt precede monetary implementations, it is not critical to maintain long-held monetary systems. Graeber even goes as far as to suggest restoring the historical practice of economic jubilee[2] as a means of preventing the entrenchment of broken and inequality reinforcing (and hence under-utilizing of human resource) financial systems common in the present day [28].

As a counterpoint, economist J. Bradford DeLong has an series of articles critiquing Graeber's work, exposing many factual inaccuracies and other problems throughout the book [18][17][16]. DeLong's critiques of Graeber's work is damning, however, he finds that the general premise of the book has a nice anthropological foundation (discussed above), while the latter half veers off the rails [19].

[2]Graeber defines Jubilee as follows:

> *The most famous of these is the Law of Jubilee: a law that stipulated that all debts would be automatically cancelled "in the Sabbath year" (that is, after seven years had passed), and that all who languished in bondage owing to such debts would be released.*

2.2 Economic Theory

This section summarizes the philosophies of several influential economists. John Maynard Keynes and Friedrich Hayek are two of the most noteworthy voices in modern economics. Although both had similar goals, mainly prosperity, the two differed radically in terms of the policies and practices they advocated. Although much ink has already been spilled[3] on the differences in view of these economists, we provide a brief review of their major ideas. We also discuss the work of Silvio Gesell, an economist whose work is influential in the design considerations of this thesis. Gesell did not achieve the notoriety of his peers, but his ideas are nonetheless noteworthy.

2.2.0.4 Keynes

The core of Keynes' beliefs was that through governmental intervention, economies can create more wealth. Keynes advocated practices that encouraged spending, such as inflating currencies during times of duress to stimulate growth. Keynes' viewpoint was that the ideal strategy was to worry more about present concerns than the future effects of such policies [33]. An example of a Keynesian policy is the United States Social Security program, which ran at a deficit in the past and is now strained [46][11].

2.2.0.5 Hayek

The core of Hayek's beliefs was that competitive – or evolutionary – processes are the only way to forge an *optimal economy*[4]. Hayek advocated policies that opened up opportunities for evolutionary processes, and advocated against coercive anti-

[3]And verse, there is an excellent rap music video that covers the subject that the author recommends as a primer [47].

[4]An optimal economy allocates resources in the most efficient manner possible.

competitive policy decisions. Hayek thought that by promoting competition, optimal systems are able to emerge. Hayek believed centralized policies or decisions cause the economy to stagnate as policy makers make important decisions with theories rather than direct evidence. Thus Hayek was not supportive of Keynes' inflationary spending as he worried that any government is inherently incapable of deciding how to invest, unable to make the optimal choices compared to a market process [59].

A notable example that establishes the nuance of Hayek's views, was his advocacy for Universal Basic Income (UBI). Hayek believed that without UBI, companies could exert a coercive pressure to force workers into labor that in turn made the labor market less competitive [58].

2.2.0.6 Gesell

Gesell, like Keynes, believed that encouraging spending improved economic outcomes. However, he did not share Keynes' view that this should occur through active government intervention. Instead, Gesell proposed a monetary system and currency called Freigeld that *demurred*, or had a cost of ownership. The intention is to disincentivize hoarding behavior and incentivize participation in the economy [25]. However, unlike Keynes' policies, the supply of money did not inflate meaning that government's ability to intervene was limited. In such a system hyperinflation is impossible.

2.3 Previous Work

We make the distinction between "traditional assets" and electronic cash schemes arbitrarily, based on general principles (government backed, physical tokens, fiat, etc). This distinction is not binding, a traditional asset may have aspects of electronic cash and vice versa.

2.3.1 Traditional Assets

2.3.1.1 Physical Assets

Physical assets are rare materials with limited availability, or representatives of such (such as a gold-backed paper currency).

Gold Gold has stood the test of time as a valued commodity, despite having only a modest functional use, its scarcity has driven value. Because of gold's scarcity, it is not easy for a regulatory body to drive inflation. Thus, for many years, gold was used as the standard medium of exchange in both domestic and global finance. Gold is a useful standard not only because it is scarce, but also because it is fungible (an ingot of gold is identical and immutable with respect to another of equal purity and mass). However, gold is too inflexible for modern economies and has been mostly abandoned (although some holdouts still advocate a return to the gold standard) [7] [31].

The Roentgen Standard In his short fiction piece, *The Roentgen Standard* [44], Larry Niven advocates radioactive money:

> *Radioactive money has certain obvious advantages.*
> *A healthy economy depends on money circulating fast. Make it radioactive and it will certainly circulate.*

While it is probably a bad idea to needlessly expose society to radioactivity, Niven does make an astute observations: currency circulates when the consequence of not spending is dire. Niven's conception of radioactive money has two distinct factors that incentive spending, health risks and degradation (via radioactive decay) of assets.

Salt and Pepper Historically, salt and pepper (separately) were among the most lucrative of resources. Salt and pepper were even used routinely as currency. What differentiates salt and pepper from other types of currency is that they are meant to be used, utilized as seasoning for food or health. Roman soldiers received remuneration in the form of a salary – literally defined as a salt payment. Elaborate trade routes for both salt and pepper formed, at times driving a significant portion of the global economy. The prevalence of the salt and pepper economy make it clear that robust economies can be formed on assets that routinely come out of circulation [52][62]. There are numerous similar historical examples one can find, such as the use of tobacco, chocolate, and olive oil as currency that do not provide much more insight than salt and pepper other than to further establish the merit for currencies based on assets that are intended to be directly consumed.

2.3.1.2 Sovereign Fiat Currencies and Banking

A sovereign fiat currency is one that is backed by a powerful (or at least, de facto) government or banking institutions. This subsection examines a few orthogonal examples briefly, from well functioning, to troubled, and outright broken currencies.

United States Dollar In 1933 the United States Dollar left the gold standard and went onto a silver standard. Thirty years later, in 1963 the United States Dollar left the silver standard as well, adopting a purely fiat currency backed only by the government itself.

Japanese Yen The Bank of Japan[5] recently switched to a negative interest rate. This means that it costs money to hold assets in the Bank of Japan or in bonds issued by the Bank of Japan; usually, a bank pays the asset holder interest rather than charging them. The Bank of Japan switched to a negative interest rate in the hope that the cost of keeping money in the bank would incentivize economic growth and spending. The problem with switching to a negative interest rate is not necessarily that it doesn't incentivize spending – it can – but that it signals to investors that the Japanese economy is unhealthy [50][60].

Argentinian Dollar Argentina has one of the most dysfunctional economies of any modern country. While there are many reasons the Argentinian economy is so dysfunctional, a few reasons are prominent. The first is that the government socialized private debts while hyperinflating the currency supply. Another major reason is that following the assumption of this debt, bad dealings with the International Monetary Fund left Argentina in a cycle of debt. Overall, these problems can be summarized as a centralized government incapable of making economically and socially responsible decisions. Protest by the Argentinian citizens eventually ousted the corrupt government. [29].

2.3.1.3 Complementary Currencies

A complementary currency is one that establishes itself as a means of exchange in a smaller community or in a way that does not seek to replace the dominant sovereign fiat currency.

[5]The Bank of Japan is the central bank of Japan. Strictly speaking, it is a private bank, not a proper governmental body, but in effect it is a governmental agency.

Ithaca Hours Ithaca Hours is a *local currency* used exclusively in Ithaca, New York. Ithaca Hours is based on the idea that members of the Ithaca community who accept the currency earn a small amount of it per year, based on how long they have been accepting Ithaca Hours. Ithaca Hours was founded on the principal that by serving only the Ithaca locality, Ithaca Hours would not be subject to the same sorts of currency controls and unethical[6] usage that are inherent to the United States Dollar. Ithaca Hours has been in use since 1991 and has issued over $100,000 dollars. Ithaca Hours demonstrates that decentralized monetary policy could be effective in better serving the needs of communities [26].

2.3.2 Electronic Cash Schemes

2.3.2.1 DigiCash

DigiCash was the earliest digital currency design, which was also the name of the business building it. In 1982 Chaum described constructing untraceable payments from blind signatures [12]. DigiCash worked by a mechanism (also of Chaum's invention) called blind signatures, a process that a requesting party can use to get a signatory to cryptographically sign a piece of information without revealing to the signatory what it was they signed. Using a blind signature, it is possible to instruct a trusted party to destroy an asset you own in exchange for sign a blinded document issuing a new asset. The unblinded document has the trusted party's signature noting that they were the owner of a coin. As long as there are many transactions occurring, everyone has transactional anonymity from the issuer.

As an enterprise, DigiCash ultimately failed. An anonymous insider cites that it failed due to personality conflicts among the founding team [55]. However, a large

[6] At least, unethical in the eyes of the Ithaca Hours participants

part of the failing from a technical perspective was the centralization of the system. Relying on a trusted third party to execute transactions and prevent double spending meant that the trustworthiness of the third party was critical for the success of the project.

2.3.2.2 Bitcoin

Bitcoin was first described by Satoshi Nakamoto in 2008. Bitcoin eliminated DigiCash's dependence on a trusted centralized party by appending publicly verifiable transactions onto a distributed ledger called a blockchain. Because of the public ledger, unlike DigiCash, Bitcoin does not provide strong anonymity guarantees. The blockchain is protected by a decentralized set of validators, or miners, performing a Proof-of-Work algorithm. The miners perform Proof-of-Work by attempting to cement an append operation to the blockchain by solving a HashCash puzzle[7]. Over time, the old data is covered by a sequence of HashCash puzzle solutions and is therefore made effectively immutable with reference to the most recent solution [42].

Bitcoin is still an ongoing experiment, but it is a system that enjoys real world usage, with a market cap in the billions of US Dollars. Nakamoto made several dogmatic economic design decisions in the creation of Bitcoin. It is fixed supply, meaning the number of assets in existence is a constant. The initial coins are distributed via an early-adopter favoring mechanism as a reward for validating. After a certain point this distribution stops, so it remains to be seen how Bitcoin works economically long term. A major concern is that as a fixed supply asset with unlimited ability to hold, "hoarding" behavior may become rampant and cause

[7]To solve a HashCash puzzle one must find a nonce n such that the hash of a message m concatenated with the nonce ($H(m \,||\, n)$) satisfies a cost function (e.g., number of leading zeroes depending on difficulty) [5]

hyper-deflation [6][61].

Bitcoin is often called by the moniker Hayek-Money. This association is often misattributed to be due to Bitcoin's monetary fixed supply, which is free of central issuance control. However the implication of the term Hayek-Money is that cryptocurrencies offer a platform for competing monetary standards, and a decentralized community maintenance structure that limits state actor coercion [57]. Bitcoin alone is not Hayek-Money, but via mechanism such as Sidechains [4], Bitoin could serve as a backbone for competing technologies to evolve.

2.3.2.3 Freicoin

Freicoin is an electronic cash scheme that is modeled after Gesell's Freigeld. Freicoin is similar to Bitcoin, except it uses demurrage to drive spending as tokens held for long periods of time incur significant overhead costs. In Freicoin, if an amount of 100 Freicoins is held for one year, the amount remaining is 95 Freicoins. The 5 Freicoin difference is released as an incentive to maintain the network[20].

Freicoin's approach is highly centralized in nature as the demurrage rate is set to a fixed value network-wide by the creators of Freicoin. A central set rate is not satisfactory as it may not be the appropriate rate for different cohorts of users. It is not trivial to decentralize the rate setting in Freicoin. The naive approach to do this – allowing the miners of Freicoin to vote on this rate – does not work for the same reasons that the Bitcoin community has struggled with block size – e.g., the miners do not represent all of the stakeholders, miners have incentives to falsely vote, and the miners do not necessarily want to make these economic decisions.

2.3.2.4 Other Schemes

Here we offer a brief survey of other electronic cash schemes. These schemes are less relevant to this thesis, we provide details on them for additional context if the reader desires.

Stellar Stellar uses a novel consensus algorithm designed by David Maziéres. The consensus algorithm is a version of Byzantine Agreement that permits open membership [38]. Economically, Stellar has a large pool of tokens they intend to equitably distribute to the world and a fixed yearly inflation amount that is received by individuals selected through a electoral process with an electorate of Stellar asset owners[8]. Stellar is not intended to be a standalone currency, but rather a complementary currency useful for liquidity in currency exchange and remittance[9] [53].

Ethereum Ethereum's main innovation is extensions to the scripting system found in Bitcoin. Ethereum treats the scripts as executing in a single virtual machine with the ability for the scripts to "call" one another. Economically, Ethereum has a small fixed inflation percentage and released most of their coins through a pre-sale process before the currency launched. Ethereum's consensus algorithm is still under development [63].

Dogecoin Dogecoin was originally launched as a joke, but because of a very well intentioned community, remains beloved. Dogecoin is nearly identical feature wise to Bitcoin, but has an odd economic model. Every year, 5 Billion new Dogecoins

[8]The fixed inflation amount is an upper bound, a lower amount may be engendered by voting for a "null ballot".

[9]If this intention is realized, Stellar's total market cap is likely proportional to the trading volume in foreign currency exchange on the Stellar platform.

are mined. The intention is to guarantee nonstop circulation of 100 billion Dogecoin. This decision was made by Dogecoin's creator after the currency had already launched [24][21].

"Private Blockchains" "Private Blockchain" does not refer to a specific electronic cash, but more generally to electronic cash protocols with closed membership. As such, the members of a "Private Blockchain" typically agree ahead of time to some economic rules and initial distributions and then internally agree (perhaps with legally binding contracts) that valid entries on their blockchain represents binding transfers. While the economic models of may in theory be anything, in practice they are primarily considered in making existing systems (such as stock exchange) more efficient. There are many criticisms of "Private Blockchains"; Arvind Narayanan[10] aptly characterized these criticisms, declaring, "'Private Blockchain' is just a confusing name for a shared database" [43].

[10]Naryanan is an assistant professor at Princeton who is active in cryptocurrency research.

Chapter 3

Real World Problems

Before delving into the specifics of how one might construct decentralized utilization incentives specifically, which is covered in Chapters 5, 6, and 7, we discuss some real world problems that applications of the system we construct in this thesis may solve. We opt to restrict our focus to tightly-scoped problems rather than the universally quantified scenario described in Subsection 1.1.1.

3.1 Renewable Resources with Phase-Delays

Renewable-but-limited-availability goods that have a phase-delay restricting when they can be produced have unique challenges preventing the creation of a reliable market for them.

An example of a resource with this challenge is solar power and solar credits. Obviously, solar power can only be made during the daylight hours, and not at night. As there is a longitudinal phase-delay when the sun rises and sets, this is an utility class that has the sought property, as demonstrated in Figure 3-1. The issue with

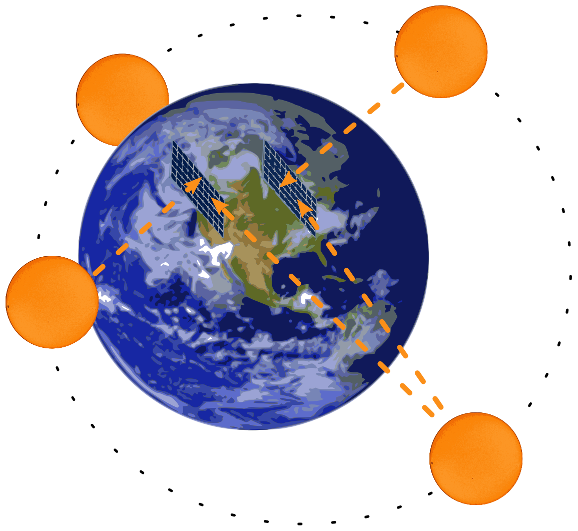


Figure 3-1: A diagram of the earth with solar panels placed far apart. The sun is pictured at several relative locations throughout the day; light from the sun hits each panel at different times.

solar energy is that while one might want to issue credit for energy created recently, one does not want to allow unlimited accumulation of credits because there is a limited global supply at any given time. To make the need for limiting possession of credits by time more clear, we can examine seasonal power usage which fluctuates greatly – power consumption during the summer is typically greater than during winter due to air conditioning. The higher demand for energy in the summer means that an energy token per-Joule should be worth less in the winter.

Solar energy is not the only type of asset that has the phase-delay property. Other examples include agricultural products, other seasonal goods, and tidal power.

Derivative phase-delayed assets – that is, assets that depend on the availability of another phase-delayed asset – can also be accounted for in this way. Datacenters operated on solar power is a motivating example of a derivative phase-delayed asset. By emphasizing basing the exchange of such derivative goods, the many pitfalls of both transmitting electricity over long distances and verifying production capacity can be avoided. Recent work on Zero Knowledge Contingent Payments can even be used to fairly trade computation now for computations later [3].

3.2 Spectrum Allocation

Spectrum allocation is the task of dividing radio spectrum among many different users or applications that need to broadcast using those frequencies. Appendix A includes a recent diagram of United States frequency allocations. Usually this is a task performed wholly by a central governing body, but spectrum is not re-allocated frequently. The United States Defense Advanced Research Projects Agency (DARPA) recently[1] announced the second iteration of their Spectrum Challenge (SC2).

> *"The primary goal of SC2 is to imbue radios with advanced machine-learning capabilities so they can collectively develop strategies that optimize use of the wireless spectrum in ways not possible with today's intrinsically inefficient approach of pre-allocating exclusive access to designated frequencies. The challenge is expected to both take advantage of recent significant progress in the fields of artificial intelligence and machine learning and also spur new developments in those research domains, with potential applications in other fields where collaborative decision-making is critical."*
>
> – New DARPA Grand Challenge to Focus on Spectrum Collaboration [45]

The DARPA Spectrum Challenge demonstrates the need for systems that can more smartly allocate and exchange spectrum.

[1]as of March 2016

3.3 Food Stamps

Food stamps are a government subsidized way to support low income individuals and families so that they can purchase food. Unlike communist dispensaries, food stamps still allow for consumer choice and works with existing commerce structures [32].

The purpose of food stamps is not to act as supplementary income, but to guarantee a standard of living for the poor. Thus it does not make much sense for food stamps to be a stable asset to hold, as the government intends to promote eating in the short term. Currently, food stamp issuance in the United States goes through the Supplemental Nutrition Assistance Program (SNAP) that costs approximately 74 Billion USD per year, of which approximately 4.5 Billion USD are spent on non-benefit costs [1]. SNAP stamps are typically valid for one year. In many food stamp programs, it is illegal to trade food stamps as the governments does not desire for the benefits to go to others[2]. Despite strict regulation, there is rampant fraud in government run benefit programs, as detailed by Katherine Meckel's report, "Is the Cure Worse than the Disease? Unintended Consequences of Fraud Reduction in Transfer Programs". Meckel shows that not only was there rampant fraud, such as price gouging[3], but measures that reduce the fraud also reduce the effectiveness of the programs [39].

[2]If SNAP participants redistribute their stamps it implies those stamps were over-allocated and another individual was under-allocated stamps.

[3]Price gouging is the practice of inappropriately raising prices on needy customers.

Chapter 4

Design Analogies

In this chapter we introduce Hourglass[1], a construction of an electronic cash that supports decentralized utilization incentives. We begin at an abstract analogical level to give the reader an intuition for the design of the system. Chapters 5, 6, and 7 delve into more concrete details of the design of Hourglass.

4.1 Analogy to a Physical Hourglass

Hourglass's design is described by analogy to a physical hourglass, pictured in Figure 4-1.

An hourglass is a mechanism that is able to measure the passage of time by waiting for an amount of sand to flow through a funnel. An hourglass is usually constructed as a hermetically sealed device and can be turned upside down to restart the clock. We refer to each side of the hourglass as a bulb. Sand is composed of discrete particles, therefore time is tracked in an hourglass in a discrete manner. However, because

[1] We refer to the Hourglass system as <u>H</u>ourglass, and a physical hourglass as <u>h</u>ourglass. In cases where context may be unclear to the reader we qualify the physical hourglass as a *physical* hourglass.

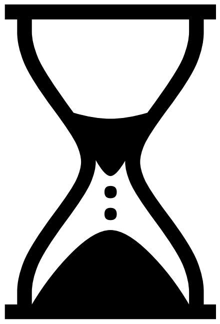


Figure 4-1: A representation of a physical hourglass.

these particles are very small and numerous, the discrete time provided is no less useful than a continuous version. Time in an hourglass is also discrete because one typically only has a calibration time for the entire volume of sand, and not of lesser subdivisions. Common modern uses of hourglasses are for egg timers, tooth brush timers, or board game timers. We refer to each particle of sand in an hourglass as a granule[2].

Hourglass is designed by analogy with the way a physical hourglass tracks time. One can model Hourglass (as we have done in this thesis) as describing the location and future locations of a granule of sand as it flows a single physical hourglass-like structure. Information about ownership of granules in Hourglass in encoded in the analogy to a physical hourglass as the locations and velocities of granules. These granules correspond to the assets issued in the electronic cash.

In Hourglass we say that a granule is owned by the custodian of the bulb through which it is currently flowing. A granule's positions are given by its chronotopes[3], or times that it is in a specific bulb. When a granule is flowing through a bulb for a

[2]A careful reader may be interested to know that we do not refer to these particles as grains in order to emphasize that the particles are countable. Grains may refer to an uncountable quantity, whereas granules may not.

[3]Chronotope comes from the Greek for time and place.

certain duration, the custodian of that bulb may further modify the remaining time that granule spends there. Thus, the hourglass could be modified for the times in that chronotope by the custodian, but not for times after or before (unless they were also owned by the same custodian).

The analogy of Hourglass to a physical hourglass breaks down slightly because the physical hourglass is potentially very complex. This is because Hourglass also has notions of operations that modify the mechanical structure of the network of bulbs that compose the physical hourglass – thereby changing the path that sand would take through the physical hourglass.

Starting from a standard physical hourglass, Figure 4-2 gives an pictorial example of such an operation. The hourglass on the left is modified to become the two tiered hourglass on the right. This kind of operation is referred to as a serial split because the newly created structures are one after another, sequential. A serial split expresses an operation where assets are divided among two different users by time.

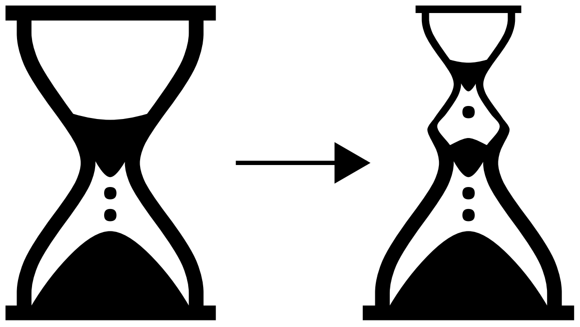


Figure 4-2: Serial modification of an hourglass's structure.

For example, suppose a two year lease on a car is being tracked via Hourglass. The car dealer would express via a serial split that the lessee is in custody of the car for two years, but then returns to the custody of the dealer after two years is up. Another example is a season baseball ticket tracked via Hourglass. The ticket holder

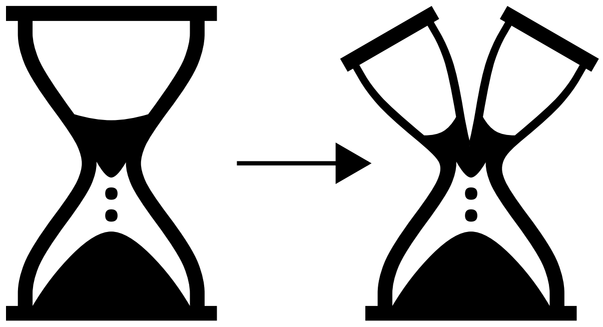


Figure 4-3: Parallel modification of an hourglass's structure.

could transfer the ticket for one game to a friend and ensure they regain access to the ticket for future games.

Figure 4-3 gives a pictorial example of another such operation. The hourglass on the left is modified into the two-headed hourglass on the right. This kind of operation is referred to as a parallel split because the newly created structures are side by side, simultaneous.

Suppose in our prior car example that instead of a lease, the car is purchased outright. Then a total transfer of the car would be done via a parallel split. In the ticket example, the parallel split would also be equivalent to transferring the ticket for all future games. Neither of these examples perfectly captures the notion of a parallel split. Instead, we introduce a third example from the preceding chapter. Suppose Hourglass is being used to track solar energy, and a particular user has 10 kilowatt hours available for the hour before sunset, they could transfer 3 kilowatt hours to another user for the same period of time and retain 7 kilowatt hours for themselves via a parallel split.

Note that in either case, serial or parallel, total flow over time through either network has not been affected by these operations. We also allow for batches of these operations to occur atomically[4].

4.2 Analogy to Existing Systems

To draw parallel to Bitcoin and sovereign currencies, a granule of sand in Hourglass is similar to a Satoshi in Bitcoin or a penny in the US Dollar; a Satoshi is the smallest indivisible token in Bitcoin, a penny is the smallest coin. Neither a granule of sand, a Satoshi, or a penny have an explicit owner, they simply exist. Nor can a granule be extracted from Hourglass much like a Satoshi can never be extracted from Bitcoin[5]. However, their current owner can be figured out by looking at the logs in Hourglass, the Blockchain in Bitcoin, or piggy banks in the US Dollar. There are many other nuanced differences and similarities between granules of sand, Satoshis, and pennies in certain facets of behavior, such as mixing.

Much like Bitcoin avoids Satoshi level transactions and pennies are uncommonly used for payments, the granule level view is too low level for typical operation in Hourglass. Instead, it is possible to also think of the a large quantity of sand in an hourglass which flows out along a schedule. This view of Hourglass allows a custodian to describe the granules they own by a discrete flux function, the amount coming in and going out of their bulb.

[4]Atomic operations are guaranteed to all-or-none occur.

[5]A flaw (or feature) of coins is that they can be extracted for their raw metals.

Chapter 5

Kernel Framework of Hourglass

In this chapter, we rigorously define Hourglass. We begin the chapter by admitting a set of strong assumptions and defining notation and syntax used to describe Hourglass. We then establish the kernel framework of Hourglass, a theoretical underpinning that can be used to describe decentralized utilization incentives as higher level behavior without directly describing these utilization incentives.

In Chapter 6 we describe higher level operations and define more terminology. The new operations do not require new rules or variations from the kernel behavior.

Further on, Chapter 7 discusses a concrete implementation on top of Bitcoin; and Chapter 8 discusses how applications of Hourglass solve the problems laid out in Chapter 3.

5.1 Notation

Throughout the remaining chapters, we adopt a particular style of pseduocode. The following examples serve to establish familiarity with this notation for the reader.

5.1.1 Types

Types represent a named tuple type structure. Types are used for organizing data and creating relationships between data. NewlyDefinedDataType is given below as an example.

```
immutable type : NewlyDefinedDataType<X> {field-one, field-two :
                 Integer, field-three : X}
```

Field one is given untyped, meaning it could store any data type, but has a particular name "field-one". Field two is given typed, meaning it could only ever be data of type Integer, and can be referred to by the name "field-two". The third field is also a template type, meaning that it could be defined in the context it is used. Template types are used for generic data structures such a List<X>. X may be any type, therefore it must be specified where a List<X> is used.

5.1.2 Elided Types

Types may have an *elided* construction in the pseudocode, meaning that their specific construction is elided. An example of this is:

```
immutable type : HardDisk {...}
```

The implementation is elided because the specifics of what a hard disk has are not immediately relevant.

We leave types implementations elided because their specifics are not highly relevant or can be defined at the application level.

5.1.3 Instances of Types

Types may be instantiated. Instances of types are not mutable by default, for clarity all types are marked by *immutable*. An example of instantiating a type is given below.

```
**immutable type**: BOX {a : INTEGER}
$b \leftarrow$ BOX{1};
```

Mutable data is only possible through pointers. Allocations for data structures are made via the **new** keyword. The style for reference and deference is similar to C-Style syntax. Below is example of a pointer to an integer, made via the **new** keyword:

```
$a \leftarrow$ * INTEGER;
$*a \leftarrow$ **new** 1;
```

By default, a pointer is **null** , it must be modified to point to data. We assume automatic lifetime safety of pointers, eliminating potential for use-after-free errors.

Pointers are only used where clarity is important (e.g., to demonstrate the layout of a data structure or global state).

5.1.4 Type Invariant

Occasionally, we follow a type definition with some invariant in square brackets. The presence of such an invariant means that the procedure CHECKINVARIANT can be called on instances of that type to check the invariant, but does not guarantee that the instance of the type is properly constructed. This is just syntactic sugar around individually defining CHECKINVARIANT, but is useful for highlighting desired properties of a type.

An example is given below:

```
immutable type: HasInvariant {a : Integer, b : Integer}[$a = 2b$]
CheckInvariant(HasInvariant{ 2, 1}) is True;
CheckInvariant(HasInvariant{ 10, 1}) is False;
```

5.1.5 Procedures

Procedures represent some sort of callable functional operation. ReplaceIfGreater is given as an example in Algorithm 5.1.

```
Procedure ReplaceIfGreater(a : * Integer, b : Integer) → Bool is
    if $*a < b$ then
        $*a \leftarrow b$;
        return True;
    else
        return False;
    end
end
```

Algorithm 5.1: ReplaceIfGreater(a : * Integer, b : Integer) is provided as an example procedure.

The name of the procedure is ReplaceIfGreater, and it takes two arguments a pointer to an integer $*a$ and an integer b, and replaces what a points to with b if $a < b$, returning True if the swap occurs and False otherwise. This syntax is fairly loose as it is pseudocode. Note the caption, which may provide important remarks on the procedure.

A function may not return a value, in which case there is no need for the return type annotation "→ Bool".

Lastly, procedures whose names begin with Do- typically denote a requirement of some sort of I/O or interaction with global state.

5.1.6 Basic Types

We assume some primitive types and operations on them, some of which have already been used in this section, such as INTEGER, BOOL, and DATA.

We also specify some more advanced default container types such as LIST and MAP.

A LIST<X>[1] is defined as:

```
immutable type: LIST<X> { next : * LIST<X>, data : * X }
```

A MAP<X→Y> is defined as:

```
immutable type: MAP<X→Y> {...}
```

Both MAP<X→Y> and LIST<X> have some special syntax for convenience (especially in giving examples). These are shown for completeness in Algorithm 5.2 but are mostly common sense.

```
a ← new LIST<INTEGER>[1,2,3,4,5];
a[0] is 1;
a[5] is null ;
a[−1] is null ;
a[1 :] is LIST<INTEGER>[2,3,4,5];
APPEND(a, 10) modifies a to be LIST<INTEGER>[2,3,4,5,10];
b ← new MAP<INTEGER→BOOL>[1→True, 2→False];
b[1] is True;
b[5] is null ;
INSERT(b, 3→False) modifies b to be MAP<INTEGER→BOOL>[1→True,
2→False, 3→False];
```

Algorithm 5.2: Examples of container type operations.

[1]The construction of LIST<X> provided is slightly atypical, with pointers in reverse order from standard lists. This is done to simplify indexing order logic.

Care should be taken, these lists and maps are distinct from Python-Style lists and maps (dicts) in two main ways: non present indexes or keys return a **null** pointer; they are homogeneous, containing elements of only one type.

5.2 Assumptions

The Hourglass kernel framework requires some baseline assumptions listed here:

1. Public Key Cryptography based signatures exist.
2. A perfect append-only log exists.
3. Time is discrete.
4. All participants can access a perfect time oracle capable of providing wall-clock timestamped signatures for arbitrary data.
5. A fair bootstrapping process exists (for fun, we have this run by a benevolent pirate "Captain Misson").

These baseline assumptions are detailed in this section.

5.2.1 Public Key Cryptographic Signatures

Public-key cryptographic signatures are needed to verify the authenticity of messages. There are many constructions in the literature, so we elide the implementation of the types SecretKey and PublicKey with specific constructions.

We denote this as

```
immutable type: SecretKey{...}
immutable type: PublicKey{...}
immutable type: KeyPair{pk : PublicKey, sk : SecretKey}
immutable type: Sig {...}
immutable type: Signature {pk : PublicKey, data : Data, sig : Sig }
```

And support several operations, DO-KEYGEN, SIGN, and VERIFY given in Algorithms 5.3, 5.4, and 5.5. These are given for completeness, but are mostly common sense.

```
Procedure DO-KEYGEN()→ KEYPAIR is
    $pk, sk \leftarrow \ldots$;
    return KEYPAIR{pk, sk};
end
```

Algorithm 5.3: DO-KEYGEN() creates a fresh KEYPAIR. The actual key generation algorithm is elided. Standard cryptographic assumptions about the returned KEYPAIR are assumed (such as uniqueness and randomness).

```
Procedure SIGN(kp : KEYPAIR, data : DATA) → SIGNATURE is
    $s \leftarrow$ signature generated by signing algorithm;
    return SIGNATURE {kp.pk, data, s};
end
```

Algorithm 5.4: SIGN(kp : KEYPAIR, data : DATA) creates a valid SIGNATURE, verifiable against a PUBLICKEY.

```
Procedure VERIFY(pk : PUBLICKEY, s : SIGNATURE) → BOOL is
    if s.pk matches pk and s.sig is valid for s.data
    for the corresponding SECRETKEY to pk then
        return True;
    else
        return False;
    end
end
```

Algorithm 5.5: VERIFY(pk : PUBLICKEY, s : SIG) checks a signature against a PUBLICKEY. Only returns true if created by SIGN with the appropriate KEYPAIR

5.2.2 Perfect Append-Only Log

The perfect append-only log is a globally writeable and readable LIST<DATA> where written entries are permanently immutable and ordered. Any participant may read or write the log at any time.

The perfect append-only log takes messages of the form APPENDLOGREQUEST to append information, and messages of the form READLOGREQUEST to read information at a certain index.

```
immutable type: APPENDLOGREQUEST {data : DATA}
immutable type: READLOGREQUEST {index : INTEGER}
```

The insertion order is stable and 0-indexed, that is, the first DATA to be appended can be read permanently by sending a READLOGREQUEST{0}, the second DATA to be appended can be read permanently by sending a READLOGREQUEST{1}, and so on.

We can model the perfect append-only log as a single computer maintaining a linked list of data entries by running DO-APPENDONLYLOG as defined in Algorithm 5.6. This implementation is given as a functional reference with simple data structures. As such, the run time of lookup and write operations are not optimized, much more efficient implementations could be given.

The two procedures that computer runs atomically are DO-APPENDLOG and DO-READLOG. These procedures are detailed in Algorithms 5.8 and 5.7.

For the kernel framework we do not consider the honesty, availability, consistentcy, fault tolerence, etc of the append-only log[2]. A perfect append-only log is still a viable assumption as there are many real world systems that approximate this under

[2]For discussion of these properties, see the CAP Theorem [51].

```
Procedure Do-AppendOnlyLog() is
    log ← new List<Data>[];
    while True do
        (m, sender) ← message from some client;
        switch m do
            case AppendLogRequest{d}
                Do-AppendLog(log, d)
            end
            case ReadLogRequest{i}
                v ← Do-ReadLog(log, i);
                reply to sender with v;
            end
        endsw
    end
end
```

Algorithm 5.6: Do-AppendOnlyLog() handles connections and processes the log.

```
Procedure Do-ReadLog(log : * List<Data>, index : Integer) → Data
is
    return log[index];
end
```

Algorithm 5.7: Do-ReadLog(log : List<Data>, index : Integer) reads an entry from the log at the requested index.

```
Procedure Do-AppendLog(log : * List<Data>, data : Data) is
    Append(log, data);
end
```

Algorithm 5.8: Do-AppendLog(data : Data) inserts the data to the end of the log.

other assumptions and address some of the aforementioned concerns, such as Bitcoin Blockchain [42] and Paxos Replicated State Machine [34][35].

Thus, we consider responses to ReadLogRequest as guaranteed to return the data (if any) at the requested index in the log, and any AppendLogRequests sent to the maintainer to be guaranteed to append an entry to the end of the log without race conditions.

5.2.3 Discrete Time

Time is considered to be discrete for the purpose of this construction. This is done mostly for convenience. Real numbers, which continuous time requires, are needlessly more difficult to work with. If continuous time is strongly desired, it can be approximated by allowing the minimum time step to be small (perhaps picoseconds).

We denote time as follows, with the implementation elided:

```
immutable type: Time {...}
```

5.2.4 Perfect Time Oracle

The perfect time oracle is an always available source of timestamped signatures.

```
immutable type: TimeStampRequest {data : Data }
immutable type: TimeStamp { time : Time, sig : Signature }
[ sig.pk is PublicKey of perfect time oracle ]
```

The perfect time oracle receives a message of type TimeStampRequest and returns a TimeStamp (a Signature structure signed by the perfect time oracle augmented with additional time information).

We can model the perfect time oracle as a server running Do-PerfectTimeOracle indefinitely as shown in Algorithm 5.9.

```
Procedure DO-PERFECTTIMEORACLE() is
    keypair ← DO-KEYGEN();
    publish keypair.pk to world;
    while True do
        (m, sender) ← receive TIMESTAMPREQUEST from some client;
        t ← true wall clock time;
        reply with TIMESTAMP { t, SIGN(keypair,m concatenated with t)} to
          sender;
    end
end
```
Algorithm 5.9: DO-PERFECTTIMEORACLE() Handles connections and returns timestamps.

The perfect time oracle has a wall clock that never repeats a time globally. TIMESTAMPs from the perfect time oracle are monotonically increasing.

5.2.5 Captain Misson

Captain Misson was a fabled pirate founder of Libertatia, a late 17th century socialist pirate colony in Madagascar [27].

Captain Misson is needed in Hourglass because this work does not solve the bootstrapping problems any currency faces. Hence we use the existence of Misson as proxy for saying there is some exogenous plan for bootstrapping the initial distributions and setting initial parameters in Hourglass.

5.3 Definitions and Concepts

In this section we introduce the core type definitions, constraints, and algorithms needed to construct Hourglass.

We define a set of operations that expresses when and how these operations occur.

Table 5.1 lists all the types and Table 5.2 procedures that are introduced in this section. Tables 5.1 and 5.2 are intended as an overview for the reader to familiarize themselves with or reference while reading the remaining chapters.

Type	**Purpose**
UUID	Universally unique identifiers
GRANULE	Smallest units tracked by Hourglass
CUSTODIAN	Access control mechanism for GRANULE ownership
BULB	Contains GRANULEs, wrapper around CUSTODIAN
TIMESPAN	Start and end time
CHRONOTOPE	TIMESPAN and a BULB, time and space
INTERVALMAP<X→Y>	MAP<X→Y> that operates on ranges of X
STATUS	CHRONOTOPE bound to a specific GRANULE
SPLITDATA<X>	Hourglass modifying operation
CUSTODIANAUTHORIZATION<X>	Authorized SPLITDATA
SPLIT<X>	TIMESTAMPed CUSTODIANAUTHORIZATION<X>
SERIALSPLIT	Template type for SPLIT<X>

Table 5.1: Overview of types introduced in Section 5.3.

Procedure	**Purpose**
LOCATIONS	Extract INTERVALMAP<TIME→BULB> from X in SPLIT<X>
VALIDSPLIT	Determine if an X in SPLIT<X> is valid w.r.t. an INTERVALMAP<TIME→BULB>
DO-LOCATEGRANULE	Get an INTERVALMAP<TIME→BULB> for a GRANULE by reading append-only log

Table 5.2: Overview of procedures introduced in Section 5.3.

5.3.1 Granules and Coinage

In Hourglass, we define granules as the base units of account, collectively called sand.

In the analogy to a physical hourglass, we think of each granule of sand as flowing past the neck between bulbs of an hourglass at an exact time that is based on the extrinsic characteristics of the granule (such as velocity, orientation, or position).

Therefore, we can define a granule of sand as just as a wrapper around a universally unique identifier (UUID), and define its extrinsic characteristics in the append-only log. The UUID is opaque to this framework, hence we elide its implementation. There are numerous potential implementations of UUID, and certain implementations may require specific types of identifier. An example implementation of UUID is just be the integers from 1 to the number of granules. The number of granules available in Hourglass is determined by Captain Misson and may depend on other mechanisms.

Notationally, we write a GRANULE and UUID as

```
    immutable type: UUID{...}
    immutable type: GRANULE { UUID }
```

5.3.2 Bulbs and Custodianship

In the analogy to a physical hourglass, we can think of each GRANULE being in a bulb of the hourglass during a certain span of times. We can also express some sort of permanent custody of a bulbs that confers temporary custody over any granule flowing through that bulb. Therefore, the bulbs of an hourglass are somewhat analogous to owners of a granule. This subsection rigorously defines that relationship.

5.3.2.1 Bulbs

We denote a bulb and custodian as follows:

$$\boxed{\begin{array}{l}\textbf{immutable type}: \textsc{Custodian}\{\ \dots\} \\ \textbf{immutable type}: \textsc{Bulb}\ \{\textsc{Custodian}\}\end{array}}$$

The ownership of a bulb is therefore explicit. If we know which BULB a granule is in, we know the custodianship.

We leave the CUSTODIAN type's implementation elided, as there might be a just a PUBLICKEY for them, but potentially much more (such as a contract system or multiple owners depending on application).

5.3.2.2 Chronotopes and Granule Status

We store in the append-only log information that allows us to extract the extrinsic properties of a granule – where it is when – as a STATUS. How this information is entered and evaluated is discussed in Subsection 5.3.3. A STATUS datum is composed on an identifier for the GRANULE in question and a CHRONOTOPE, or a BULB and duration of time in that BULB.

$$\boxed{\begin{array}{l}\textbf{immutable type}: \textsc{TimeSpan}\{\ \text{enter} : \textsc{Time}, \text{leave} : \textsc{Time}\}[\text{enter} < \text{leave}] \\ \textbf{immutable type}: \textsc{Chronotope}\ \{\text{bulb} : \textsc{Bulb}, \text{timespan} : \textsc{TimeSpan}\} \\ \textbf{immutable type}: \textsc{Status}\ \{\text{granule} : \textsc{Granule}, \text{chronotope} : \textsc{Chronotope}\}\end{array}}$$

As a reminder, in Subsection 5.2.3, we elided the definition of the type TIME that a TIMESPAN is composed of as it is an implementation detail.

A granule is considered in the custody of the CUSTODIAN of the BULB it currently occupies implicitly, as granule location is determined by information in the append only log as discussed in Subsection 5.3.1 and Subsubsection and 5.3.3.2. Therefore a granule does not have an explicit custodian, it just is in custody during the time and place specified in a Status's Chronotope.

A Granule occupies a Chronotope atomically – once the perfect time oracle's clock ticks past the exit time specified by the Chronotope, the Granule is no longer in the custody of the Custodian of the Bulb specified by that Chronotope.

5.3.3 Bulb Splits and Custodianship Modification

Custodianship modifications in Hourglass are done via *bulb splits*, or just splits. Custodianship modification is very similar to transactions in Bitcoin.

This is physically analogous to taking all the Granules out of one hourglass and putting those Granules into another hourglass with a different configuration of Bulbs without altering the amount of time the Granules take to flows through. Thus there is no way to increase the amount of time a Granule has remaining with any custodianship modification (or any operation for that matter).

We denote a split as follows:

```
immutable type: SplitData<X> {input : Granule, split : X}
[X has Locations Procedure]
immutable type: CustodianAuthorization<X>{splitdata :
                SplitData<X>, sig : Sign}[sig.data = splitdata]
immutable type: Split<X>{authorized :
                CustodianAuthorization<X>, timestamp :
                TimeStamp}[timestamp.sig.data without timestamp.time
                suffix is authorized.splitdata]
```

Any Split operation must be sent as an AppendLogRequest to the append-only log maintainer. This is the only type of data that needs to be stored in the log.

As expressed in the invariant, each type of X must have a Locations Procedure defined. LOCATIONS(x : X) extracts an INTERVALMAP<TIME→BULB>[3] with the new locations for a given GRANULE. The result of LOCATIONS is used in log processing as the result of applying a SPLIT.

In the kernel framework, there is only one kind of split, a SERIALSPLIT with restrictions on what kind of transfers are valid. Figure 5-1 demonstrates a SERIALSPLIT operation both pictorially and in pseudocode. In Chapter 6 we define some special cases of SERIALSPLIT for convenience, so we template our top-level data structure for SPLITs.

5.3.3.1 Split Validity

A SPLIT<X> s is considered valid with respect to an INTERVALMAP<TIME→ BULB> i as long as VALIDSPLIT(i, s), shown in Algorithm 5.10, returns True. VALIDSPLIT checks that: the owner marked in the SPLIT matches the owner of the bulb at the evaluation state for all times in the append-only log that the STATUS updates change; the perfect time oracle timestamp t is such that $t \leq$ the earliest STATUS (restricting the perfect time oracle timestamp ensures that the SPLIT does not occur retroactively); and all signatures are valid.

5.3.3.2 Determining Granule Location

The locations of all granules is determined by parsing and evaluating all the append-only log entries in order from index 0 to the last non-**null** entry. An entries can be acquired by sending the perfect append-only log maintainer a READLOGREQUEST. Critically, DO-LOCATEGRANULE's initial state of the universe is determined by Captain Misson arbitrarily.

[3]INTERVALMAP is a basic map data structure to track mappings of integer interval keys to values that supports standard types of lookup, insert, and update operations.

```
Procedure ValidSplit(I : IntervalMap<Time→Bulb>, split :
Split<X>) → Bool is
    L ← Locations(split.authorized.splitdata);
    split_time← earliest time in l2;
    if Verify(perfect time oraclePublicKey, split.timestamp);
    intervals used in L matches have one bulb b in I;
    Verify(b.owner's PublicKey, split.authorized);
    and split_time ≥ split.timestamp.time;
    then
        return True
    end
    return False;
end
```

Algorithm 5.10: ValidSplit(I : IntervalMap<Time→Bulb>, split : Split<X>) checks that a Split is valid compared to the future locations of a Granule.

For each Split<X> entry at index i in the append-only log, the changes made by that split are applied if the split is valid with respect to the application of all prior valid splits. If any part of a Split<X> is invalid, the entire Split<X> operation is discarded[4]. This method is consistent across many clients.

Algorithm 5.11 demonstrates the procedure Do-LocateGranule that determines the locations of a Granule.

5.3.3.3 Serial Split

Serial bulb splits can be thought of in the physical analogy as replacing the current hourglass with an hourglass that has smaller auxiliary bulbs in serial with multiple owners, as shown in Figure 5-1. A serial split is done to transfer custody of an asset over time, as described in Chapter 4, perhaps to create a lease on a car or give a

[4]Ignoring invalid split operations allows us to separate the concerns of the append-only log from any validation

```
Procedure DO-LOCATEGRANULE-RECURSIVE(index : INTEGER, l :
INTERVALMAP<TIME→BULB>, g : GRANULE) →
INTERVALMAP<TIME→BULB> is
    log_entry ← response from sending READLOGREQUEST { index };
    if log_entry is null then
        return l;
    else if log_entry.data is a SPLIT of g and VALIDSPLIT(l,log_entry.data)
    then
        l ← UPDATE(l, LOCATIONS(log_entry.data));
    return DO-LOCATEGRANULE-RECURSIVE(index+1, l, g);
end
Procedure DO-LOCATEGRANULE( g : GRANULE) →
INTERVALMAP<TIME→BULB> is
    l ← new INTERVALMAP<TIMESPAN→BULB>[initial distributions from
    Captain Misson];
    return DO-LOCATEGRANULE-RECURSIVE(0, l, g);
end
```

Algorithm 5.11: DO-LOCATEGRANULE(a : GRANULE) finds and processes the STATUSes of a given GRANULE.

friend a single game on a season ticket.

Formally, we allow modification of the STATUS of a GRANULE by creating a series of bulbs with a duration of time equivalent to that of the parent, but with each bulb having a different owner. We do not allow bulb splits to increase the latest valid time of a granule, however, we allow for increasing the earliest valid time[5]. Increasing the earliest time ensures we can meet the requirement for the TIMESTAMP to precede the earliest time in VALIDSPLIT.

No serial split may be of zero duration, that is set via the TIMESPAN invariant.

Following from that, we can allow for simultaneously splitting the bulbs of a granule with any set of non-overlapping otherwise valid TIMESPANs. Such operations are discussed in Chapter 6.

We denote a serial bulb split as Split extension data as follows

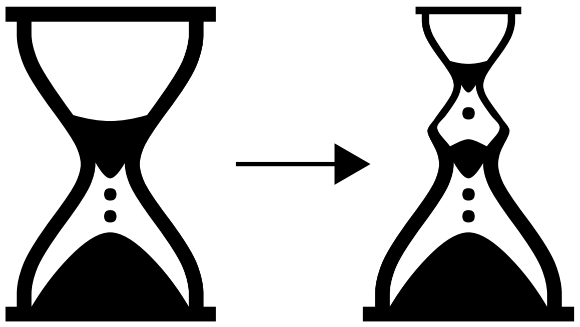
immutable type:	SERIALSPLIT {granule : GRANULE, top : CHRONOTOPE, bottom : CHRONOTOPE}[top.leave = bottom.enter,]

When a SPLIT<SERIALSPLIT> is evaluated from the log, if it is valid, the evaluation state is set to put the granule at the new location at that time, and the old location data for that granule is overwritten for the times between top.enter and bottom.leave. This is shown in detail in Subsubsection 5.3.3.2. The LOCATIONS function for a SPLIT<SERIALSPLIT> is shown in Algorithm 5.12.

[5]The direction of times can be a little confusing, so an example is provided. If the original TIMESPAN is TIMESPAN{6:00PM, 7:00PM} then: TIMESPAN{6:00PM, 8:00PM} is an example of increasing the latest time and is **not valid** as it causes an increase in duration a GRANULE is usable; and TIMESPAN{6:30PM, 7:00PM} is an example of increasing the earliest time and is **valid** as it does not increase when a GRANULE is usable.

```
Procedure LOCATIONS(split : SERIALSPLIT) →
INTERVALMAP<TIME→BULB> is
    l ← new INTERVALMAP<TIMESPAN→BULB>[];
    l ← INSERT(l, (split.top.bulb, split.top.timespan);
    l ← INSERT(l, (split.bottom.bulb, split.bottom.timespan);
    return l;
end
```

Algorithm 5.12: LOCATIONS(s : SERIALSPLIT)



```
g ← GRANULE { 1 };
l ← DO-LOCATEGRANULE(g);
Assert(LOOKUP(l, TIMESPAN {4:00PM, 11:59PM}) is BULB {Carol});
s ← SPLITDATA { g
, SERIALSPLIT
{ g , CHRONOTOPE { BULB { Alice }, TIMESPAN { 4:00PM, 7:00PM } }
, CHRONOTOPE { BULB { Bob }, TIMESPAN { 7:00PM, 11:59PM } }
} };
o ← CUSTODIANAUTHORIZATION { s, SIGN { Carol's SECRETKEY, s }};
p ← response from sending perfect time oracleTIMESTAMPREQUEST{o};
Assert(p.time is 3:00PM);
Send append-only log maintainer APPENDLOGREQUEST { SPLIT { o, p } };
```

Figure 5-1: The SERIALSPLIT operation from Figure 4-2 is pictured here, along with an example pseudocode operation where a granule is first located to be in Carol's BULB from times 4:00 PM to 11:59 PM and then split at 3:00PM to Alice's BULB from 4:00 PM to 6:59 PM and Bob's bulb from 7:00 PM to 11:59 PM.

5.3.4 Granule Value

In our analogy to a physical hourglass, the notion of granule value in our nautical metaphor is that all the sand is silicate, no portion of it is gold dust or diamond chips.

Formally, two GRANULEs held during the same TIMESPAN are fungible in Hourglass, that is, all granules held at the same time have the same worth[6]. In order to emulate a GRANULE with a greater value a user can atomically transfer many granule to another user. This is discussed in Chapter 6.

[6]This ignores that granules with different TIMESPANs may be worth more or less to different users.

Chapter 6

Refining Hourglass Kernel Framework

In Chapter 5, the definitions and concepts are given somewhat clumsily as they are designed to be a minimal framework for the behavior of Hourglass. In this chapter, we refine the terminology that better encapsulates the higher level properties of Hourglass, without changing any of the underlying properties.

6.1 New Terminology

6.1.1 New Split Types

This subsection we define special case of parallel splits and multi splits.

6.1.1.1 Parallel

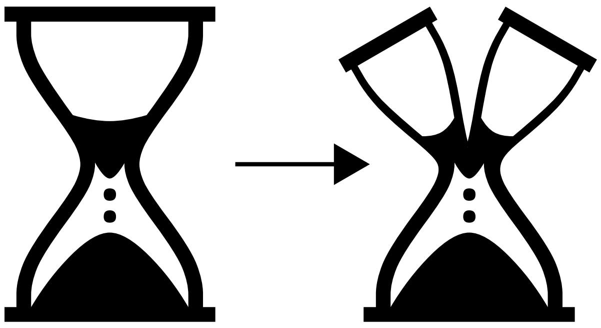
Conceptually a parallel bulb split can be thought of as moving a granule from one hourglass bulb to another with an identical timespan. Thus this preserves the amount of time the granule is contained by that bulb, only changing who controls the bulb. As described in Chapter 4, a parallel split is used to transfer ownership completely when used with a single asset such as a purchased car, or can be used to divide the amount transferred such as the division of a 10 kilowatt hours an hour before sunset power usage right.

Formally, a parallel bulb split is a message in the append-only log from the last owner of the granule, π, specifying a new owner π' with a timestamp from the perfect time oracle within the valid existence times of the granule. As this is an inductive definition, it requires a base case. We assume that all granules belong to the "Captain Misson" user initially, denoted $\pi_{captain}$, who somehow distributes them to other users. We denote a parallel bulb split as Split extension data as follows

immutable type: PARALLELSPLIT { g : GRANULE, ct : CHRONOTOPE }

When a SPLIT containing a PARALLELSPLIT is evaluated from the log, if it is valid, the evaluation state is set to put the granule to be at the new location, and the old location data is overwritten for the times marked. The application of a PARALLELSPLIT is shown in Figure 6-1.

A PARALLELSPLIT is really just a special case of a SERIALSPLIT where both of the CHRONOTOPEs have the same owner, as shown in Figure 6-2. Although it presents no new behavior, this is a common scenario so it deserves special mention.



```
g ← GRANULE { 1 };
l ← DO-LOCATEGRANULE(g);
Assert(LOOKUP(l, TIMESPAN {4:00PM, 11:59PM}) is BULB {Carol});
s ← SPLITDATA { g
, PARALLELSPLIT
{ g , CHRONOTOPE { BULB { Alice }, TIMESPAN { 4:00PM, 11:59PM } }
} };
o ← CUSTODIANAUTHORIZATION { s, SIGN { Carol's SECRETKEY, s }};
p ← response from sending perfect time oracleTIMESTAMPREQUEST{o};
Assert(p.time is 3:00PM);
Send append-only log maintainer APPENDLOGREQUEST { SPLIT { o, p } };
```

Figure 6-1: The PARALLELSPLIT operation from Figure 4-3 is pictured here, along with an example pseudocode operation where a granule is first located to be in Carol's BULB from times 4:00 PM to 11:59 PM and then parallel split at 3:00PM to Alice's BULB from 4:00 PM to 11:59 PM.

```
g ← GRANULE { 1 };
b ← BULB { Alice };
ss ← SERIALSPLIT { g, CHRONOTOPE { b, TIMESPAN {4:00PM, 4:30PM} },
CHRONOTOPE { b, TIMESPAN {4:30PM, 5:00PM} } };
ps ← PARALLELSPLIT { g, CHRONOTOPE { b, TIMESPAN {4:00PM, 5:00PM}
}};
ss ∼ ps ;
```

Figure 6-2: The PARALLELSPLIT is shown here to be a special case of SERIALSPLIT.

6.1.1.2 Multi

If we allow a batch of splits to all receive the same timestamp[1] from the perfect time oracle, thus permitting for atomic batch bulb splits.

As all splits may be performed in any combination in a single atomic operation, allowing any redistribution of the valid time span. Thus we express a MultiSplit as:

immutable type : MultiSplit { List<Status> }

A MultiSplit is created such that the result of Do-LocateGranule on the granules could be achieved via a sequence of other splits. A MultiSplit is valid if and only if all the other splits that cause the same evaluation state were they to be evaluated are valid. It is applied as if all the equivalent splits were applied.

MultiSplit, like ParallelSplit, is just a convenience over a sequence of batched SerialSplits, they are not required, but it is a common paradigm so it deserved special mention.

6.1.1.3 Joins

Join operations are not needed, as they happen implicitly as two contiguously owned locations are implicitly treated as joined with respect to the IntervalMap in Do-LocateGranule for all Split operations.

[1]Theoretically speaking, we do not violate the properties of a perfect time oracle being able to well order a set of observations by allowing for some bits that are used only for well ordering and not for the validity

6.1.2 Sand

When a large number of granules are held with different expiry dates, it is more convenient to think in terms of an amount of sand decreasing over time.

SAND is defined as:

immutable type: AMOUNT{INTEGER}
immutable type: SAND { UUID, amount : AMOUNT }

SAND is simply a compact representation of an array of GRANULE UUIDs of length amount. Although simple, the SAND representation, in conjunction with the MULTISPLIT operation, allows us to more easily express aggregate operations on GRANULES. The type AMOUNT is just an INTEGER, but we wrap it with a type for clarity.

6.1.3 Discrete Flux

We can think of granules in aggregate form as SAND with a discrete flux (in granules per unit time) coming through the neck of the hourglass, rather than a single atomic unit having a time of expiry. We call this discrete flux rate expiration when it is negative and accumulation when it is positive.

Yhe properties we are looking to satisfy when making a MULTISPLIT with many GRANULEs are just that we don't utilize anything after expiry or before valid. Instead, we treat the granules as above as SAND with an instantaneous discrete flux function $F(t)$, that is a measure of granules entering per unit time where the unit of time is not further divisible. We can express $F(t)$ such that

$$F(t) = I(t) - O(t)$$

where $I(t)$ is the flux in and $O(t)$ is the flux out at each time step. $F(t)$ is in terms of Amount, but we denote it ΔAmount for clarity that it is a differential quantity. Given an initial value (in a bulb b_0) $(v_{b_0})_0$ we can write the recurrence

$$[v_{b_0}]_{i+1} = [v_{b_0}]_i + F_{b_0}(i)$$

or more concisely

$$[v_{b_0}]_t = [v_{b_0}]_0 + \sum_{i=1}^{t} F_{b_0}(i)$$

to express the instantaneous amount held in a bulb at any time.

We can describe splits from the owner of b_0 to the set of bulbs $b_{>0}$ in two equivalent ways, as shown in Equations (6.1d) and (6.1e), given a set of bulbs (see Equation (6.1a)), a set of flux functions (see Equation (6.1b)), and a set of amounts at bulbs at a given time t (see Equation (6.1c))[2].

$$\{b_0, b_1, \ldots, b_n\} :: \{\text{Bulb}\} \tag{6.1a}$$

$$\{F_{b_0}(t), F_{b_1}(t), \ldots, F_{b_n}(t)\} :: \{(\text{Time}) \mapsto \Delta\text{Amount}\} \tag{6.1b}$$

$$\forall t.\, \{[a_{b_0}]_t, [a_{b_1}]_t, \ldots, [a_{b_n}]_t\} :: \{\Delta\text{Amount}\} \tag{6.1c}$$

$$\forall t. [a_{b_0}]_t \geq \sum_{i=1}^{n} [a_{b_i}]_t \tag{6.1d}$$

$$\forall t. [a_{b_0}]_0 + \sum_{i=1}^{t} F_{b_0}(i) \geq \sum_{x=1}^{n} \left([a_{b_x}]_0 + \sum_{i=1}^{t} F_{b_x}(i) \right) \tag{6.1e}$$

In other words, we are specifying that the outputs of the MultiSplit never

[2]We expect the implementation of Time to be compatible with natural numbers for these equations.

exceed the inputs. As long as this property is satisfied, the properties of the kernel Hourglass are also satisfied.

We can make this restriction slightly easier to view by doing the following (shown in Equation (6.2)): First, reordering the sum; then subtracting $\sum_{x=1}^{n}(a_{b_x})_0$ from all sides; then, because $t = 0$ restricts $v_{b_0} \geq \sum_{i=1}^{n} v_i$, assuming the $\geq$ is a strict equality[3]; reordering the sums; then lastly, and lastly canceling the outermost sum.

$$\forall t.[a_{b_0}]_0 + \sum_{i=1}^{t} F_{b_0}(i) \geq \sum_{x=1}^{n}[a_{b_x}]_0 + \sum_{x=1}^{n}\left(\sum_{i=1}^{t} F_{b_x}(i)\right) \tag{6.2a}$$

$$\forall t.[(a_{b_0}]_0 - \sum_{x=1}^{n}[a_{b_x}]_0) + \sum_{i=1}^{t} F_{b_0}(i) \geq \sum_{x=1}^{n}\left(\sum_{i=1}^{t} F_{b_x}(i)\right) \tag{6.2b}$$

$$\forall t. \sum_{i=1}^{t} F_{b_0}(i) \geq \sum_{x=1}^{n}\left(\sum_{i=1}^{t} F_{b_x}(i)\right) \tag{6.2c}$$

$$\forall t. \sum_{i=1}^{t} F_{b_0}(i) \geq \sum_{i=1}^{t}\sum_{x=1}^{n} F_{b_x}(i) \tag{6.2d}$$

$$F_{b_0} \geq \sum_{x=1}^{n} F_{b_x} \tag{6.2e}$$

Equation (6.2e) expresses that the discrete flux through a bulb must be preserved under splits.

This discrete flux model of Hourglass is completely equivalent to the kernel description given in Chapter 5. In the forward direction, many granules can be modeled as having a flux function that accounts for the valid time span of each individual granule at each time step, and in the reverse direction the discrete flux function can be decomposed into individual granules with different time spans.

[3]assuming strict equality is reasonable because it could be ensured by the owner of b_0 without restricting their actions, i.e. they could just spend to a non-owned bulb.

6.2 Virtual Reserve

We consider the initial granules (those defined as the initial state in DO-LOCATEGRANULE) to be created via virtual reserve controlled by Captain Misson without expiration (i.e., Captain Misson has a bulb where they are located initially for a contiguous infinite span of times). This creates a fundamental constant in the system. It is a virtual granule because it is impossible to actually hold these infinite spanned granules unless the user is Captain Misson. Instead, they serve as a reference point for balancing equations, and we restrict transactions in and out of this bulb.

This system wide parameter on expiration can be arbitrarily set by Captain Misson, but it should be well under what the postulated optimal[4] rate is. We assume that it is possible to guess such a rate conservatively (for instance, if the optima is expected to be somewhere between 3 and 10 percent, a rate of 1 percent is clearly under, which allows for the real world usage to approach the optima).

If we denote this constant as $\mathcal{C}$, and all valid STATUSes as $\mathcal{S}$, and all GRANULEs as $\mathcal{G}$, we can thus write the equation:

$$\forall t.\mathcal{C} = \sum_{g \in \mathcal{G}} 1 \geq \sum_{s \in \mathcal{S}} \begin{cases} 1 & \text{if s.ct.timespan.enter} < \text{t} < \text{s.ct.timespan.leave} \\ 0 & \text{otherwise} \end{cases}$$

Or rather, over all time at any given time there are at most $\mathcal{C}$ assets.

Importantly, this constant $\mathcal{C}$ is just the maximum value, $\mathcal{C}$ may or may not be in circulation at any given point depending on the behavior of Captain Misson.

[4]Optimality as defined in Section 2.2, allocating resources in the most efficient manner possible.

6.3 Flowback Recipient

When a Split is appended that splits the timespan of validity of a granule from $[0, t]$ to $[0, t - \delta]$, the owner can also specify new granule locations from times $[t - \delta, t]$. This property also carries onto the higher level discrete flux description, where the negative of the flux out $(O(t))$ goes to a future recipient.

Therefore, we can think of sand as flowing to one of many targets, or flowback recipients. As it is restricted that no party may own the virtual reserve token, there could be some amount redistributed in the system over time or permanently expired. Possible motivations for such targets are to serve as a validation incentive (miner reward or fee), or to return to the sender of a split. It could also be permanently burned/reissued to Captain Misson.

Chapter 7

Concrete Implementation

In the concrete implementation we describe Hourglass as a series of patches to existing cryptocurrency technology. This chapter assumes a basic familiarity with Bitcoin's design and general Blockchain infrastructure. If such familiarity is not had, the author recommends the original bitcoin whitepaper [42], a tutorial by Michael Nielsen [2], and a recent systematization of knowledge paper [9].

7.1 Assumptions

The concrete implementation assumes a much weaker set of assumptions than the kernel and refined implementation, requiring only a system similar to Bitcoin. As a result of breaking these assumptions, several guarantees must also be relaxed. Specifically, we relax the perfect append-only ledger and perfect time oracle to their approximate versions.

7.1.1 Approximate Append Only Ledger

Bitcoin's blockchain is an approximate append-only ledger. Blocks of data appended are not guaranteed to be appended, only given a probabilistic guarantee[1] of being appended that increases as more blocks are appended. By waiting for large number of blocks to be appended, Bitcoin could be said to emulate a perfect append-only log [42].

Stellar's Consensus Protocol also emulates a perfect append-only log. Stellar's core principle is that as long as the graph of the consensus network without malicious actors is still connected, then the network can order events. Obviously, this property may fail, hence only giving an approximate guarantee based on the behavior and arrangement of the actors in the network [38].

The other way that these approximate constructions are inexact is that they have potential censorship problems for admitting data to the network. The potential for censorship arises because in order for an event to be recorded in one of these approximate append-only logs, the event must be submitted to a network that may have some incentive to prevent such events, in contrast to the perfect append-only log where any event attempted to be written gets written.

7.1.2 Approximate Time Oracle

The existence of a perfect time oracle is not a trivial assumption. There are numerous references in the literature for not-perfect-but-reliable time oracles that can be used to emulate a perfect time oracle. This subsection describes a few such systsm.

CommitCoin by Clark et al. uses the blockchain to insert timestamps, serving as a somewhat reliable way to timestamp documents. As long as the blockchain is

[1]Technically, a game theoretic guarantee.

append only within some bound, this scheme is reliable [13].

Google Spanner's TrueTime demonstrates a reasonable construction of a fault-tolerant timestamping service based on GPS that provides a range of times that contains the true time. TrueTime is a construction of a bounded error time oracle that can be used to emulate a perfect time oracle by outwaiting the error bound. Google Spanner's use of a system with bounded error via GPS time is a reliable-but-relaxed consensus algorithm. If GPS has faults, then this scheme is unreliable. Google had determined that GPS was a more reliable system than what they had built on top of it, and were therefore comfortable using it from a risk perspective [14].

Intel's Proof of Elapsed Time uses trusted hardware enclaves to run provably fair timing code to order events. Proof of Elapsed Time is broken if Intel's hardware is broken, or if Intel releases backdoored versions of the software, but many users already trust Intel hardware [15].

Therefore it is reasonable to substitute any of these approximate time oracles for a perfect time oracle without changing the analysis done in Chapter 5. In Chapter 7) we discuss the realistic constructions of a bounded time oracle to emulate a perfect time oracle.

7.2 Sandscript

Sandscript is the component that drives the flow rate in Hourglass. It is a serialization format for the sum of the discrete flux function presented in Chapter 6.

The core of Sandscript is an Opcode that is a triplet of iterations, delay, and rate.

In other words, the type signature is:

```
type: ITERS{ p : INTEGER}
type: DELAY{ d : ΔTIME}
type: RATE{ r : ΔAMOUNT}
type: OPCODE{iters : ITERS, delay : DELAY, rate : RATE}
```

The units of each are given in Table 7.2.

Parameter	**Units**
TIME	Seconds
AMOUNT	Number of granules
ITERS	Dimensionless
DELAY	Seconds
RATE	Granules per application

Table 7.1: Units of Sandscript Opcodes

Each OPCODE o, when fully applied, causes $o.iters \cdot o.rate$ discrete flux out, and accounts for $o.delay \cdot o.iters$ amount of time.

A script is defined as a list of Opcodes, i.e.:

```
immutable type: SANDSCRIPT is a LIST<OPCODE>
```

To compute the effective flow over a script $\mathcal{S} = [s_1, \ldots, s_n]$ at time t since publication, one computes the application (potentially partial) of each OPCODE as demonstrated in EVALSANDSCRIPT (Algorithm 7.1). To compute the amount remaining in a bulb with initial amount α as a result of a Sandscript $\mathcal{S}$ at a certain time τ we take EVALSANDSCRIPT$(\mathcal{S}, \tau, \alpha)\}$.

Sandscript does not restrict the expressivity of the discrete flux function, but it prioritizes small serializations for simpler flux functions. For instance, expiring 1 granule per second is expressed as SANDSCRIPT[OPCODE{n times, 1 second, 1 granule per application}], where n is the number of granules.

```
Procedure $\textsc{EvalSandScript}$(script : $\textsc{SandScript}$, time : $\textsc{Time}$, amount : $\textsc{Amount}$) $\rightarrow$ $\textsc{Amount}$ is
    foreach $\textsc{Opcode}$ $\{iters, delay, rate\} \in script$ do
        if $delay \cdot iters < time$ then
            $amount \leftarrow amount - rate \cdot iters$;
            $time \leftarrow time + iters \cdot delay$;
        else
            $amount \leftarrow amount - rate \cdot \lfloor time/delay \rfloor$;
             break ;
        end
    end
    return $\max\{0, amount\}$;
end
```
Algorithm 7.1: $\textsc{EvalSandScript}$(s : $\textsc{SandScript}$, time : $\textsc{Time}$, amount : $\textsc{Amount}$) evaluates a Sandscript up to a certain time given an initial amount. We do not allow the amount to become negative.

When performing an operation that sets a new Sandscript for a set of granules, care must be taken to ensure Equation (6.2e). In SandScript, the Equation (7.1) is equivalent to Equation (6.2e).

$$\forall t.\ \textsc{EvalSandScript}(\mathcal{S}_{old}, t) \geq \ \textsc{EvalSandScript}(\mathcal{S}_{\text{new}}, t) \tag{7.1}$$

This can either be checked exhaustively, or can be ensured implicitly by expressing SandScript transactions as a change script $\mathcal{S}_{\Delta}$ on top of the old script, as demonstrated in Equation (7.2)

$$\forall t.\textsc{EvalSandScript}(\mathcal{S}_{new}, t) = \ \textsc{EvalSandScript}(\mathcal{S}_{old}, t) + \ \textsc{EvalSandScript}(\mathcal{S}_{\Delta}, t) \tag{7.2}$$

7.3 Target Selection

Each transaction output can specify an pre-expiry owner and a post-expiry owner. The post-expiry owner may be another transaction output in the current transaction. Thus this allows for an arbitrary division of sand by time among users, like the MultiSplit operation.

7.4 Bitcoin Compatibility

The mechanisms presented above can be implemented directly in Bitcoin as a hard fork. A hard fork is a set of modifications that are not already rules-compliant with Bitcoin as is, that is, an honest participant in Bitcoin could not already be following these rules unbeknownst to other participants, these changes require a major restart. Hard forks are desirable because all participants know the knew rules. This section assume familiarity with Bitcoin scripting. The downside of a hard fork is that there is a lot of friction in causing such a change.

In order to implement Hourglass, we redefine the Bitcoin transaction output[2] data as follows:

sandscript length::**var_int**	
sandscript::**uint8_t**[sandscript length]	
script length::**var_int**	script::**uint8_t**[script length]

Enforcement of VerifyOutputSandScripts, as shown in Algorith 7.2, on each spend attempt is sufficient to ensure all properties of the Hourglass split model.

[2]The original transaction format is given in Appendix B.

```
Procedure VerifyOutputSandScripts(inputs : (Amount,
List<SandScript>), outputs : List<SandScript>, time : Time) →
Bool is
    for $i \in time \ldots$ do
        $input\_sum \leftarrow$ new 0;
        $output\_sum \leftarrow$ new 0;
        foreach $(amount, script) \in inputs$ do
            $input\_sum \leftarrow$EvalSandScript(script, time, amount);
        end
        foreach $(amount, script) \in outputs$ do
            $output\_sum \leftarrow$EvalSandScript(script, time, amount);
        end
        if $output\_sum = 0$ then
            return True;
        end
        if $output\_sum > input\_sum$ then
            return False;
        end
    end
end
```

Algorithm 7.2: VerifyOutputSandScripts(inputs : (Amount, List<SandScript>), outputs : List<SandScript>, time : Time) → Bool exhaustively evaluates input and output SandscriptS to verify Equation (7.1).

7.5 Hourglass Specific Attacks and Remedies

There are many attacks that are already possible on the Bitcoin network as is. This section focuses on attacks that become possible with the introduction of Hourglass. As these attacks mostly have to do with the criticality of time to the Hourglass protocol, after introducing two attacks and we simultaneously lay out several methods which may help mitigate both these attacks.

7.5.1 Attacks

7.5.1.1 Censorship

In Hourglass if the time-stamping of a Split can be delayed or otherwise censored, then the granules expire, causing the owner to lose value unfairly. The potential for censorship to cause loss of value is a major concern.

7.5.1.2 Reorganization Safety

In Bitcoin, a reorganization event occurs when a series of blocks have been committed to, and then a fork of the chain overtakes that series. Bitcoin is engineered such that during a reorganization event caused by “honest actors” (i.e., caused by chance and not malicious intent), all transactions could be included in both chains. There are two measures we present that can be taken to prevent this.

However, consistent ordering is not guaranteed during a fork. This is an issue if a a valid sequence of transaction’s validity can be impacted by permuting the transactions, because it means that reorganization of the sequence of events is not necessarily safe.

7.5.2 Remedies

7.5.2.1 Delay Modulation

Increasing the delay parameter in each Opcode may be used to mitigate censorship and reorganization safety concerns. The delay parameter should be increased such that if the approximate append-only log can be said to provide a publication guarantee with Δt latency (that is, censoring cannot be sustained for more than Δt), larger that $2\Delta t$ delay parameters in Sandscript Opcode’s make it difficult for a censoring party to meaningfully censor and requiring a reorganization event to fork a longer chain.

7.5.2.2 Time Commitments

Including in the block header the root of a Merkle Tree[3] of transaction commitments well before opening the commitments may reduce effects of censorship and eliminate Hourglass specific reorganization concerns. These transaction commitments are not revealed to the network, and on appending the transaction to the append-only log, the inclusion of a proof of existence in that root can provide proof of an earlier occurrence rate. Even though the transaction commitment mechanism itself is subject to foul play, the contents of the commitment are hidden from the censoring agent and a reorganization would have to fork back to the commitment point. There is not a strong potential for abuse as transaction commitments require knowledge of the transaction in advance. Of course, it is possible for a miner to try to censor the publication of the final transaction with the proof, but it is futile for them to delay the timestamp the transaction actually gets. A similar method is explored and shown to be effective for real-world use case in CommitCoin by Clark et al [13].

7.5.2.3 Relative Time

One such method is to utilize the same mechanism that Bitcoin uses for its block time field – times are valid within a range, hence the global time is not guaranteed to proceed monotonically. This means that the earliest time of a transaction within a block can be earlier in a later block. The main drawback of this method is transactions no longer being totally ordered.

[3]A Merkle Tree is an efficient method to commit to a ordered set of commitments and be able to individually prove membership in the committed set. The root is simply the top level commitment [40].

7.5.2.4 Network Time Independence

Another method is to make the transaction independent of network time, placing the burden of accepting or rejecting a transaction onto the payer and payee (Alice and Bob) rather than network. As it is network time independent, this eliminates much of the concern of either a reorganization event or censorship.

Proofs of clock synchronization agreement between Alice and Bob could be exchanged before submitting a transaction, and the network could validate only that payer and payee agreed on time[4]. To be precise, we arrange submitting transaction from Alice to Bob to the network as follows:

1. Alice and Bob agree on a the details of a transaction (e.g., amount).
2. Alice generates and signs without revealing the signed transaction to Bob. We denote this signed transaction τ.
3. Alice signs a time t and commitment to the transaction concatenated with that time, i.e., Alice signs $(t, C(\tau || t))$.
4. Alice sends the signature to Bob.
5. If the time is close to his local clock, Bob signs that signature concatenated with the hash of a block 100 blocks back from the current head, and a timeout denoted in a number of blocks.
6. Bob sends the signature to Alice.
7. Alice publishes the transaction with the signature from Bob to the network.

[4]Perhaps the network might also filter transaction attempts that are obviously far in the past.

8. The network considers the transaction valid if and only if the signatures matched the sender and recipient of the transaction, the transaction is not timed out according to the schedule Bob set, and the transaction was otherwise valid.

This scheme provides reasonable fairness via the following properties: Alice cannot send to Bob without confirming times; Bob cannot spend the transaction without providing Alice a proof; Alice cannot delay the opening of the commitment past the timeout; Alice cannot send Bob less than Bob expects without Bob finding out by the timeout.

The main drawback of this scheme is it requires all participants to be online for a payment to occur. Requiring recipient to be online is not necessarily a drawback, as the recipient would likely want to know as soon as they acquire an expiring asset. The requirement for agreement on time between sender and recipient could be relaxed to allow for transactions that are well ahead of the network consensus time.

7.6 Prunability

In Bitcoin, transactions that can be "proven" to be never utilizable can be safely deleted. In Hourglass, after a granule has expired it can no longer be split. Thus, it is possible to delete old split records via two mechanism, universal expiry and local expiry. This manifests as a potential mechanism that can increase the scalability of Hourglass.

7.6.1 Universal Expiry

If transactions are serialized in a way such that a transaction's time and expiration information is separate from the standard Bitcoin transaction data, it is possible to prune the majority of an operation's data based purely on this time information.

7.6.2 Local Expiry

If Bitcoin transactions in a block are ordered by their Hourglass expiration time, then a single transaction can be used as an expiration proof pointer. This works because all transactions with earlier expiration dates must be expired as they are located before the pointer, therefore are known to have expired. This relies heavily on knowing the transactions are indeed sorted in the block, if this is not true it is possible to run into potentially incorrect behavior.

7.7 Verfication and Mining

Verification and Mining can be done similarly to any cryptocurrency protocol, such as Bitcoin, Ethereum, or Stellar. There is more information to verify with respect to the time information, but Bitcoin is already equipped to verify time constraints via CheckLockTimeVerify and CheckSequenceVerify [54][10]. Furthermore, as noted in Section 7.5 the validity of times can be made the responsibility of sender and recipient to verify and mostly external to the network-wide verification process.

Chapter 8

Solutions to Real World Problems

Having shown the progression from analogy to theoretical kernel framework to concrete system with decentralized utilization incentives in Chapters 4, 5, 6, and 7, we discuss some specifics of how applications of Hourglass could solve the real world problems laid out in Chapter 3. We explore the details of some of these applications in more detail than others.

We focus the description of the applications on the high level behavior defined in Chapters 5 and 6 rather than the more concrete description given in Chapter 7. Several of these applications are similar to the user-issued assets mentioned in the Freicoin Freimarkets paper [36], although the Freimarkets paper does not go into detail on the types of custom assets desirable nor do they permit the time span splitting as in Hourglass.

8.1 Applying Hourglass to Renewable Resources with Phase-Delays

A cryptocurrency with decentralized utilization incentives can be used to create a reliable market for renewable-but-limited-availability goods that have a phase-delay restricting when they can be produced.

Here, we sketch out the solar example from Section 3.1.

First, we assign Captain Misson the duties of registering photo-voltaic cells and assigning them a Bulb with a Custodian that the owner of the panel has access to. Every four days, Captain Misson issues a set of Granules to be in located in the Chronotope of that Bulb when there is sunlight expected to be available (e.g., daylight, not too cloudy). Users can purchase a Granule from another user in another region to get access to electricity during that time.

As mentioned in Section 3.1, there is a fundamental challenge in physically transporting electricity. It is much simpler and cheaper to transport and verify production capacity for information as a result of computation. In this application, there is little to no need for Captain Misson. Production capacity could be monitored by a challenge response based Proof-of-Work scheme that establishes computational availability at epochs throughout the day.

8.2 Applying Hourglass to Spectrum Allocation

In general, Hourglass's system of utilization incentives provides a nice model to process spectrum allocations. Hourglass Granules could be given a UUID representing a slice of spectrum. Leases of spectrum takes the form of a Split directing a Granule to be in a Chronotope that the user is the Custodian of. As leases expire and can be subdivided, it provides a clean way for different frequencies to be doled out to many parties. For instance, one could envision a cellular network where upon connecting, leases of a channel can be purchased that allow for broadcast messages to be authenticated with the current owner's keys. On disconnect, remaining lease duration could be sold to other users.

Hourglass presents a possible asset in the design of the machine learning coordination problems posed by the DARPA SC2 challenge. As Hourglass can be deployed on top of a cryptocurrency like Bitcoin, it is possible to guarantee the atomicity of multiple transactions. Thus, spectrum leases can be trustlessly exchanged between these (potentially) mistrustful radio devices. Furthermore, the divisible lease structure avoids many of the problems of strict pre-allocation, the allocations can be balanced and planned for the future by these actors.

8.3 Applying Hourglass to Food Stamps

Hourglass could be used in the foodstamp distribution process by allocating granules to individuals rather than stamps or other cards. Admittedly, such a system only needs to utilize a limited subset of Hourglass's features. Individuals being able to further subdivide their assets by time is undesirable, so expiring GRANULEs could be issued to them as non spendable, e.g. a SPLIT where the CUSTODIAN records them as the owner, but does not allow them to process further SPLITs. The hope is that Hourglass could lower the administration costs of these programs to increase the availability while deterring certain types of fraud.

Chapter 9

Conclusion

In this chapter we conclude by comparing Hourglass to Bitcoin and discussing future work.

9.1 Comparison of Hourglass with Bitcoin

In this section, we compare and contrast Hourglass with Bitcoin. We focus on the economic differences, but make brief comment on other topics where Hourglass diverges from Bitcoin.

9.1.1 Economic and Social Robustness

The economics and social robustness of Hourglass are substantially different from those of Bitcoin while remaining as general in application.

9.1.1.1 Generality of Applications

We do not explicitly advocate an general currency application of Hourglass – e.g., a currency that is intended to be used in all markets inherently like Bitcoin. Although it may be possible to do, as Bitcoin has done, such an application is too broad for detailed study. However, Hourglass can be used in applications to account for the custody of usage rights such as solar energy credits or computation. Given that energy or computation are universally useful and valuable, we make the argument that a temporal right to energy and computation can be used as a currency. Thus, the scope of an Hourglass based economy, even when restricted to solar energy or computation, is as general as Bitcoin.

9.1.1.2 Market Growth and Stability

Hourglass promotes both market growth and stability. Market growth is promoted in Hourglass because expiration rates incentivize participants to use their assets before they expire. This makes economic activity more favorable for participants than inactivity. In contrast, Bitcoin is deflationary, which implies that participants likely get more value from continuing to hold the asset than using it to do something. Market stability is encouraged in Hourglass because participants can control during which period other participants they transact to have assets. This means that participants can attempt to ensure that markets do not become saturated or depleted, which may help with stability.

9.1.1.3 Non-Coercive, Participatory, and Decentralized

Hourglass is more non-coercive, participatory, and decentralized than Bitcoin because all sub-communities have the freedom to develop local economic norms around rates of expiration. In contrast, Bitcoin distributes permanently spendable assets with few mechanisms for users of the system to ensure that this is economically fair.

9.1.2 Security

The security of Hourglass is worse than that of Bitcoin because the time dependence of operations creates opportunity. However, in Section 7.5 we detail several mechanism which decrease the impact of time dependence on security. Hourglass, implemented as a fork of Bitcoin, is comparable to Bitcoin's security on all other categories.

9.1.3 Scalability

Hourglass has a scalability improvement over Bitcoin. In Hourglass it is possible to delete records from the log which pertain to expired granules. In Section 7.6, we discuss mechanisms for deciding which log entries to prune. The scalability improvement is dependent on many factors, chiefly the expiration selected by users.

9.2 Future Work

9.2.1 Real World Testing

A hypothetical design and construction of decentralized utilization incentives has been shown through Hourglass.

However, Hourglass has not been shown rigorously to have any effect whatsoever on utilization. Although it might be possible to construct game theoretic models on how users of Hourglass might behave, as Hayek might caution, models without evidence have low predictive power, the only way to know is by trying.

Thus, there is potential for future work in implementing Hourglass and testing it in a real economy to see if it actually works.

9.2.2 Scalability

In Chapter 7, we briefly address some scalability benefits which could be conferred by the Hourglass design. Further exploration of how expiration affects scalability could yield interesting results.

9.2.3 Expected Time

In Hourglass a TimeSpan field is a differential quantity, a start time and a finish time. A ProbabilisticTimeSpan could be given as a start time and a probability p of ending after after every time step. The amount of time required before expiry in this system is (assuming a fixed probability) T and expected value yields $E[T] = \frac{1}{p}$[1]. Equivalently, we may write that if we wish for an expiry around time T, we select $p = \frac{1}{T}$.

The implications of expected time are non trivial, and are a very interesting direction for study in game theory. In Hourglass, actors have perfect information on

[1]We may express T as $T = p \cdot 1 + (1 - p)(1 + T)$. Therefore:

$$\begin{aligned}
E[T] &= E[p \cdot 1 + (1 - p)(1 + T)] \\
E[T] &= E[p \cdot 1] + E[(1 - p)(1 + T)] \\
E[T] &= p + (1 - p)(1 + E[T]) \\
E[T] &= p + 1 + E[T] - pE[T] - p \\
pE[T] &= 1 \\
E[T] &= \frac{1}{p}
\end{aligned}$$

when they must use their assets to avoid expiration. As probabilistic expiration may occur at any time, the incentive to utilize may be perceived to be greater. Different probability distributions over time may also drive interesting effects.

Technically, probabilistic expiration also presents interesting challenges. Most straightforward implementations rely on generating a random number to control the expiration. There are protocols which are guaranteed to select a random number relative to any of the participants as seen in Blum's *Coin Flipping by Telephone a Protocol for Solving Impossible Problems* [8], but in the case of exogenously driven expiration it is unclear who the participants are. Some protocols source their random numbers from the blockchain itself, Pierrot and Wesolowski provide analysis on how an attacker might be able to alter this entropy [48].

9.2.4 First-Class Design Principles

Further exploring how expiry affects scalability, as noted in Subsection 9.2.2, is not sufficient. Were we to explore scalability in that manner, we would be guilty ourselves of the same provocation given in Chapter 1 – we would not be studying scalability as a first-class design goal of an electronic cash scheme, rather layering it on as an afterthought. Understanding scalability as a first-class design goal of an electronic cash scheme could yield novel results.

Furthermore, using this style of emphasis on other second-class goals, such as usability, hardware security, or even formal verification could yield many interesting and useful results.

Questioning why the three typical goals presented in Chapter 1 are the goals that are commonly addressed is important for the future development of this technology.

Appendix A

United States Frequency Allocations

UNITED STATES FREQUENCY ALLOCATIONS

THE RADIO SPECTRUM

RADIO SERVICES COLOR LEGEND

🟦	AERONAUTICAL MOBILE	🟨	INTER-SATELLITE	🟨	RADIO ASTRONOMY
🟦	AERONAUTICAL MOBILE SATELLITE	🟦	LAND MOBILE	🟧	RADIODETERMINATION SATELLITE
🟫	AERONAUTICAL RADIONAVIGATION	🟦	LAND MOBILE SATELLITE	🟨	RADIOLOCATION
🟩	AMATEUR	⬜	MARITIME MOBILE	🟨	RADIOLOCATION SATELLITE
⬜	AMATEUR SATELLITE	🟩	MARITIME MOBILE SATELLITE	🟩	RADIONAVIGATION
🟦	BROADCASTING	🟩	MARITIME RADIONAVIGATION	🟨	RADIONAVIGATION SATELLITE
🟩	BROADCASTING SATELLITE	⬜	METEOROLOGICAL	🟥	SPACE OPERATION
🟧	EARTH EXPLORATION SATELLITE	🟫	METEOROLOGICAL SATELLITE	🟥	SPACE RESEARCH
🟥	FIXED	⬜	MOBILE	⬜	STANDARD FREQUENCY AND TIME SIGNAL
🟪	FIXED SATELLITE	🟪	MOBILE SATELLITE	⬜	STANDARD FREQUENCY AND TIME SIGNAL SATELLITE

ACTIVITY CODE

🟥	GOVERNMENT EXCLUSIVE	⬛	GOVERNMENT/NON-GOVERNMENT SHARED
🟩	NON-GOVERNMENT EXCLUSIVE		

ALLOCATION USAGE DESIGNATION

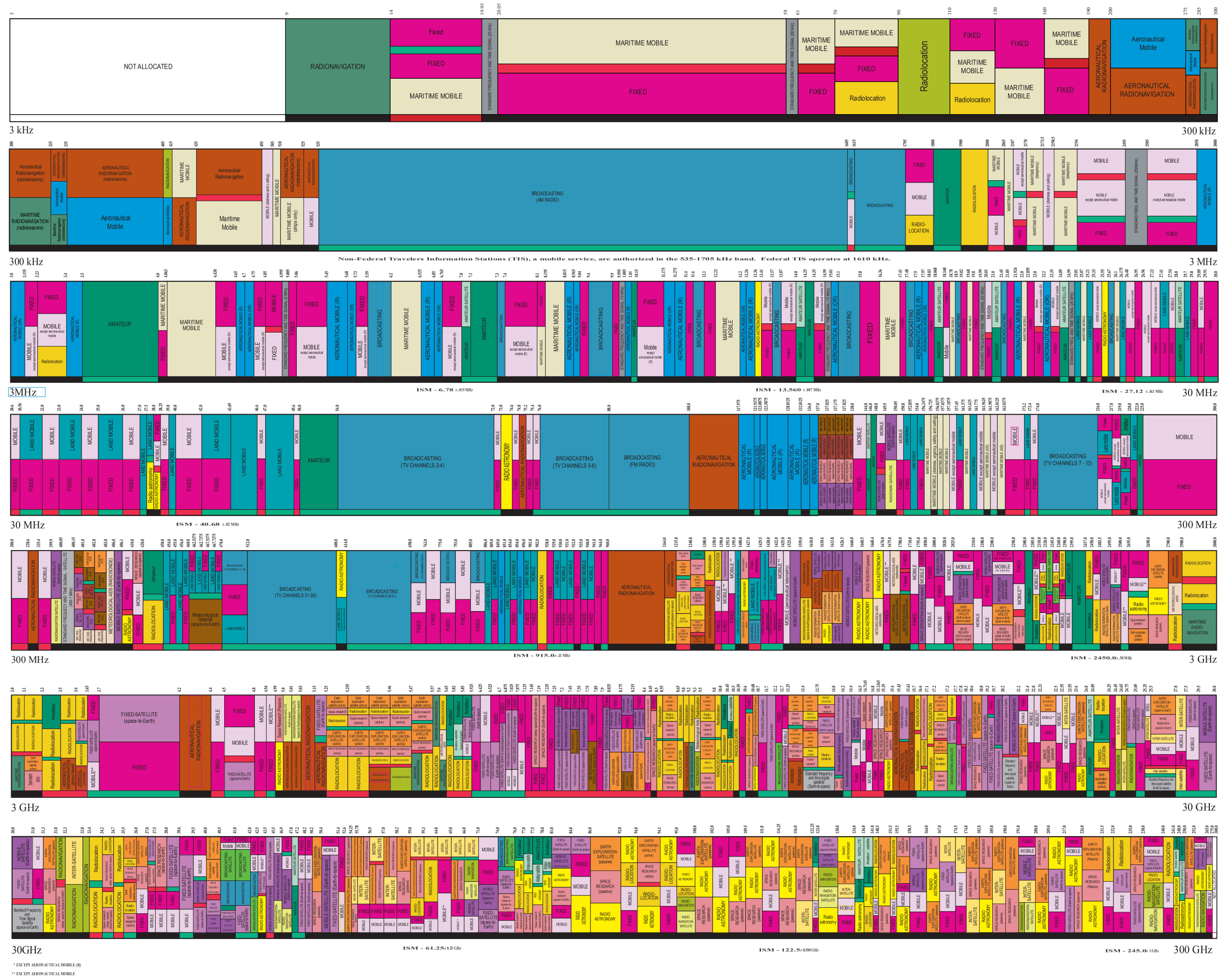
SERVICE	EXAMPLE	DESCRIPTION
Primary	FIXED	Capital Letters
Secondary	Mobile	1st Capital with lower case letters

This chart is a graphic single-point-in-time portrayal of the Table of Frequency Allocations used by the FCC and NTIA. As such, it does not completely reflect all aspects, i.e. footnotes and recent changes made to the Table of Frequency Allocations. Therefore, for complete information, users should consult the Table to determine the current status of U.S. allocations.

U.S. DEPARTMENT OF COMMERCE
National Telecommunications and Information Administration
Office of Spectrum Management

August 2011

For sale by the Superintendent of Documents, U.S. Government Printing Office
Internet: bookstore.gpo.gov Phone toll free (866) 512-1800; Washington, DC area (202) 512-1800
Facsimile: (202) 512-2250 Mail: Stop SSOP, Washington, DC 20402-0001



* EXCEPT AERONAUTICAL MOBILE (R)
** EXCEPT AERONAUTICAL MOBILE

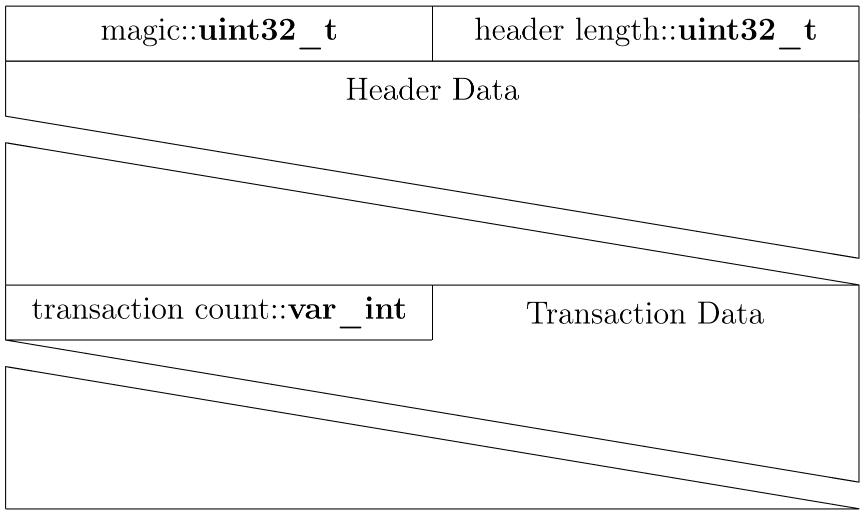
PLEASE NOTE: THE SPACING ALLOTTED THE SERVICES IN THE SPECTRUM SEGMENTS SHOWN IS NOT PROPORTIONAL TO THE ACTUAL AMOUNT OF SPECTRUM OCCUPIED.

Appendix B

Bitcoin Data Format Serialized Network Format

A brief overview of the network format is included in this appendix for reference.

B.1 Block



B.2 Header

version::**uint32_t**	previous block::**uint8_t[32]**
Merkle root::**uint8_t[32]**	time stamp::**uint32_t**
difficulty::**uint32_t**	nonce::**uint32_t**

B.3 Transaction Input

spends::OutPoint	transaction index::**uint32_t**
script length::**var_int**	script::**uint8_t**[script length]
sequence ::**uint32_t**	

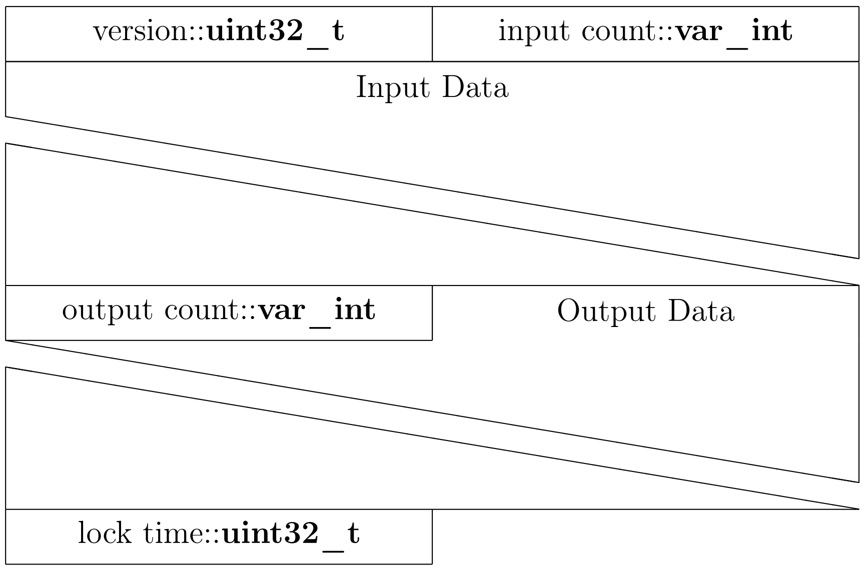
B.4 Transaction Outpoint

transaction id::**uint8_t[32]**	output index::**uint32_t**

B.5 Transaction Output

value ::**int32_t**	script length::**var_int**
script::**uint8_t**[script length]	

B.6 Transaction



Bibliography

[1] Supplemental Nutrition Assistance Program (SNAP). `http://www.fns.usda.gov/pd/supplemental-nutrition-assistance-program-snap`.

[2] How the Bitcoin protocol actually works. `http://www.michaelnielsen.org/ddi/how-the-bitcoin-protocol-actually-works/`, December 2016.

[3] pay-to-sudoku. `https://github.com/zcash/pay-to-sudoku`, February 2016.

[4] Luke Dashjr Mark Friedenbach Gregory Maxwell Andrew Miller Andrew Poelstra Jorge Timón Adam Back, Matt Corallo and Pieter Wuille. Enabling blockchain innovations with pegged sidechains.

[5] Adam Back. Hashcash - A denial of service counter-measure. Technical report, 2002.

[6] Simon Barber, Xavier Boyen, Elaine Shi, and Ersin Uzun. Bitter to better – how to make Bitcoin a better currency. In *in Financial Cryptography 2012*, pages 399–414, 2012.

[7] Edward M. Bernstein. Back to the gold standard? *The Brookings Bulletin*, 17(2):8–12, 1980.

[8] Manuel Blum. Coin flipping by telephone a protocol for solving impossible problems. *SIGACT News*, 15(1):23–27, January 1983.

[9] Joseph Bonneau, Andrew Miller, Jeremy Clark, Arvind Naryanan, Joshua A. Kroll, and Edward W. Felten. SoK: Bitcoin and second-generation cryptocurrencies. In *IEEE Security and Privacy 2015*, May 2015.

[10] Eric Lombrozo BtcDrak, Mark Friedenbach. CHECKSEQUENCEVERIFY. `https://github.com/bitcoin/bips/blob/master/bip-0065.mediawiki`, 08 2014.

[11] Rodrigo A. Cerda. On Social Security financial crisis. *Journal of Population Economics*, 18(3):509–517, 2005.

[12] David Chaum. Blind signatures for untraceable payments. In *Advances in Cryptology: Proceedings of CRYPTO '82*, pages 199–203. Plenum, 1982.

[13] Jeremy Clark and Aleksander Essex. CommitCoin: Carbon dating commitments with Bitcoin*. 2012.

[14] James C. Corbett, Jeffrey Dean, Michael Epstein, Andrew Fikes, Christopher Frost, Jj Furman, Christopher Taylor, and Google Inc. Spanner: Google's globally-distributed database.

[15] Intel Corporation. Introduction – Distributed ledger latest documentation. `https://intelledger.github.io/introduction.html`, 2016.

[16] James Bradford DeLong. Monday smackdown: Back to David Graeber's "Debt: The first 5000 mistakes". `http://www.bradford-delong.com/2014/11/monday-smackdown-in-the-absence-of-high-quality-delong-smackdowns-back-to-david-graeber.html`, 2013.

[17] James Bradford DeLong. The very last David Graeber post... `http://www.bradford-delong.com/2013/01/the-very-last-david-graeber-post.html`, 2013.

[18] James Bradford DeLong. Monday reading David Graeber's debt smackdown: Chapter 11 in chapter 11, part ii. `http://www.bradford-delong.com/2014/06/monday-reading-david-graebers-debt-smackdown-chapter-11-in-chapter-11-part-ii.html`, 2014.

[19] James Bradford DeLong. April foolsf́estival final day: A change of pace. `http://www.bradford-delong.com/2015/04/thursday-idiocy-i-see-david-graeber-is-back.html`, 2015.

[20] Freicoin Developers. How Freicoin works. `http://freico.in/how/`.

[21] The Dogecoin Project. Dogecoin. `http://dogecoin.com/`. Accessed: 2015-3-6.

[22] Ericsson. Ericsson mobility report. `http://www.ericsson.com/mobility-report`.

[23] Eli Ben-Sasson et al. Zerocash: Decentralized anonymous payments from Bitcoin. In *2014 IEEE Symposium on Security and Privacy*, 2014.

[24] Cyrus Farivar. Dogecoin to allow annual inflation of 5 billion coins each year, forever. `http://arstechnica.com/business/2014/02/dogecoin-to-allow-annual-inflation-of-5-billion-coins-each-year-forever/`. Accessed: 2015-3-6.

[25] Silvio Gessel. *Die Natürliche Wirtschaftsordnung (The Natural Economic Order)*. 1949.

[26] Paul Glover. Creating community economics with local currency. `http://www.ithacahours.com/`.

[27] Philip Gosse. *The Pirates' Who's Who*. Burt Franklin, 1924.

[28] David Graeber. *Debt: The First 5000 Years*. Melville House, 2011.

[29] Joseph Halevi. The Argentine crisis. *Monthly Review*, 53, 2002.

[30] Caroline Humphrey. Barter and economic disintegration. *Man*, 20(1):48–72, 1985.

[31] J. H. Jones. The gold standard. *The Economic Journal*, 43(172):551–574, 1933.

[32] Neeraj Kaushal and Qin Gao. Food stamp program and consumption choices. Working Paper 14988, National Bureau of Economic Research, May 2009.

[33] John Maynard Keynes. *General Theory of Employment, Interest and Money*. Cambridge University Press, Cambridge, 1936.

[34] Leslie Lamport. The implementation of reliable distributed multiprocess systems. *Computer Networks (1976)*, 2(2):95 – 114, 1978.

[35] Leslie Lamport. Paxos made simple. 2001.

[36] Jorge Timón Mark Friedenbach. Freimarkets: Extending bitcoin protocol with user-specified bearer instruments, peer-to-peer exchange, off-chain accounting, auctions, derivatives and transitive transactions. `http://freico.in/docs/freimarkets-v0.0.1.pdf`, August 2013.

[37] Alex Mayyasi. The early days of Bitcoin. `http://priceonomics.com/the-early-days-of-bitcoin/`, March 2014.

[38] David Maziéres. The Stellar consensus protocol: A federated model for Internet-level consensus. `https://www.stellar.org/papers/stellar-consensus-protocol.pdf`. Draft Version: 2/25/2016.

[39] Katherine Meckel. Is the cure worse than the disease? Unintended consequences of fraud reduction in transfer programs. `https://economics.stanford.edu/sites/default/files/kmeckel_jmp.pdf`, April 2015.

[40] R.C Merkle. Protocols for public key cryptosystems. In *Proc. 1980 Symposium on Security and Privacy, IEEE Computer Society*, pages 122–133. IEEE Computer Society, 1980.

[41] Silvio Micali and Ronald L. Rivest. Micropayments revisited. Technical report, Laboratory for Computer Science, Massachusetts Institute of Technology, Cambridge, 2002.

[42] Satoshi Nakamoto. Bitcoin: A peer-to-peer electronic cash system, 2008.

[43] Arvind Narayanan. "private blockchain" is just a confusing name for a shared database. `https://freedom-to-tinker.com/blog/randomwalker/private-blockchain-is-just-a-confusing-name-for-a-shared-database/`, September 2015.

[44] Larry Niven. YET ANOTHER MODEST PROPOSAL: The Roentgen Standard. `http://www.larryniven.net/stories/roentgen.shtml`.

[45] outreach@darpa.mil. New DARPA grand challenge to focus on spectrum collaboration. `https://web.archive.org/web/20160324000428/http://www.darpa.mil/news-events/2016-03-23`, March 2016.

[46] Thomas I. Palley. The economics of Social Security: An old Keynesian perspective. *Journal of Post Keynesian Economics*, 21(1):93–110, 1998.

[47] John Papola and Russ Roberts. "fear the boom and bust" a hayek vs. keynes rap anthem. `https://youtu.be/d0nERTFo-Sk`.

[48] Cecile Pierrot and Benjamin Wesolowski. Malleability of the blockchainâĂŹs entropy. Cryptology ePrint Archive, Report 2016/370, 2016. `http://eprint.iacr.org/`.

[49] Joseph Poon and Thaddeus Dryja. The Bitcoin Lightning network: Scalable off-chain instant payments. Technical report, Technical Report (draft 0.5.9.2). https://lightning. network, 2015.

[50] Jana Randow and Simon Kennedy. Negative interest rates, Less than zero. `http://www.bloomberg.com/quicktake/negative-interest-rates`, March 2016.

[51] Nancy Lynch Seth Gilbert. Brewer's conjecture and the feasibility of consistent, available, partition-tolerant web services. Accessed: 2015-4-6.

[52] TIME Staff. A brief history of salt. `http://time.com/3957460/a-brief-history-of-salt/`, 1982.

[53] Stellar Development Foundation. Stellar. `https://www.stellar.org/`. Accessed: 2015-3-6.

[54] Peter Todd. OP_CHECKLOCKTIMEVERIFY. `https://github.com/bitcoin/bips/blob/master/bip-0065.mediawiki`, 1 2014.

[55] Anonymous translated by Ian Grigg. How DigiCash blew everything. `https://cryptome.org/jya/digicrash.htm`, January 1999.

[56] Peter van Oudheusden and Jan Sonnenschein. Number of bank account owners worldwide grows by 700 million. `http://www.gallup.com/poll/182420/number-bank-account-owners-worldwide-grows-700-million.aspx`, 2015.

[57] Freidrech August von Hayek. *The Denationalization of Money.* Institute of Economic Affairs, London, 1976.

[58] Freidrech August von Hayek. *Law, Legislation and Liberty, Volume 3: The Political Order of a Free People.* University of Chicago Press, Chicago, 1979.

[59] Freidrech August von Hayek. *The Fatal Conceit: The Errors of Socialism.* University of Chicago Press, Chicago, 1988.

[60] Eleanor Warnock and Mayumi Negishi. Japan's negative-rate experiment is floundering. `http://www.wsj.com/articles/japans-negative-rate-experiment-is-floundering-1460644639`, April 2016.

[61] Bitcoin Wiki. Deflationary spiral — Bitcoin wiki, 2012. [Online; accessed 17-April-2016].

[62] Wikipedia. Pepper. `http://www.ancient.eu/Pepper/`, December 2010.

[63] Gavin Wood. Ethereum: A secure decentralised generalised transaction ledger.