

UNDERSTANDING AND IMPROVING THE SECURITY OF  
COLLABORATIVE DISTRIBUTED SYSTEMS

---

DISSERTATION

Submitted in Partial Fulfillment of  
the Requirements for  
the Degree of

DOCTOR OF PHILOSOPHY (Computer Science)

at the

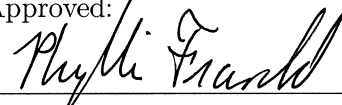
NEW YORK UNIVERSITY  
TANDON SCHOOL OF ENGINEERING

by

Luqin Wang

January 2017

Approved:



Department Chair Signature

12/20/2016

Date

ProQuest Number:10252666

All rights reserved

INFORMATION TO ALL USERS

The quality of this reproduction is dependent upon the quality of the copy submitted.

In the unlikely event that the author did not send a complete manuscript and there are missing pages, these will be noted. Also, if material had to be removed, a note will indicate the deletion.



ProQuest 10252666

Published by ProQuest LLC (2017). Copyright of the Dissertation is held by the Author.

All rights reserved.

This work is protected against unauthorized copying under Title 17, United States Code  
Microform Edition © ProQuest LLC.

ProQuest LLC.  
789 East Eisenhower Parkway  
P.O. Box 1346  
Ann Arbor, MI 48106 – 1346

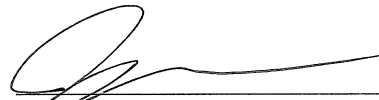
Approved by the Guidance Committee:

Major: Computer Science



**Yong Liu**  
Associate Professor  
Electrical and Computer Engineering

12 / 20 / 2016  
Date



**Justin Cappos**  
Assistant Professor  
Computer Science and Engineering

12 / 20 / 2016  
Date



**Lisa Hellerstein**  
Professor  
Computer Science and Engineering

12 / 20 / 2016  
Date



**Damon McCoy**  
Assistant Professor  
Computer Science and Engineering

12 / 20 / 2016  
Date

Microfilm or copies of this dissertation may be obtained from:

UMI Dissertation Publishing  
ProQuest CSA  
789 E. Eisenhower Parkway  
P.O. Box 1346  
Ann Arbor, MI 48106-1346

## VITA

**Luqin Wang** was born in Shanghai, China, on May 1, 1983. He received a Bachelor of Science degree in Information Countermeasure Technology from Hangzhou Dianzi University, in Hangzhou, Zhejiang, China in May 2006. After coming to the United States to study at NYU Tandon School of Engineering (formally NYU Polytechnic School of Engineering), Brooklyn, New York, he completed three Master of Science degrees, in Electrical Engineering, Telecommunication Networks, and Computer Science in 2006, 2008, and 2016, respectively. Since January 2011, he has been a Ph.D. candidate in the Department of Computer Science and Engineering at NYU Tandon, under the supervision of Prof. Yong Liu and co-advisor Prof. Justin Cappos. In this capacity, he conducts research in the fields of distributed systems and network security.

*To my parents and grandparents,  
for their endless love and support.*

## ACKNOWLEDGEMENT

First and foremost, I would like to gratefully and sincerely thank my Ph.D. advisor Prof. Yong Liu, and my co-advisor Prof. Justin Cappos, for their guidance, inspiration, and support throughout my six-year Ph.D. career. They gave me freedom to shape my own research pursuits and always encouraged me to work on projects which meshed with my personal interests. Professors Liu and Cappos have also been actively involved in detailed discussions of all the projects I conducted, and always gave me invaluable suggestions and comments. Their deep insights about research have had a great influence on my vision of how academic studies should be conducted and evaluated and contributed greatly to my professional development.

I would also like to thank Trishank Kuppusamy and Ghada Almashaqbeh, who worked with me closely on many projects. Any discussions with them have always been inspiring and helpful.

Furthermore, I am deeply indebted to Prof. Lisa Hellerstein and Prof. Damon McCoy for spending time on this dissertation and serving on my dissertation committee. I offer many thanks for their input, encouragement, and valuable suggestions.

I also want to acknowledge all the members of the Wanlab and my other friends, including Dr. Pei Liu, Dr. Shu Luo, Dr. Zizhong Cao, Tingting Lu, Dr. Sha Hua, Dr. Guibin Tian, Dr. Xiwang Yang, Dr. Yang Xu, Dr. Yuan Ding, Dr. Hao Hu, Dr. Zhengye Liu, Dr. Yanming Shen, and others, who have made my days at New York University cheerful and memorable.

Last but not least, I would like to extend my gratitude to my family. Without

the endless love and care of my beloved parents, my grandparents, and my wife, I could never have gotten this far. They are always my most valuable sources of encouragement and support.

All the work described here were supported in part by the National Science Foundation under Grant No. 0966187.



**ABSTRACT****UNDERSTANDING AND IMPROVING THE SECURITY  
OF COLLABORATIVE DISTRIBUTED SYSTEMS**

by

**Luqin Wang****Advisor: Yong Liu****Co-Advisor: Justin Cappos**

Submitted in Partial Fulfillment of the Requirements for  
the Degree of Doctor of Philosophy (Computer Science)

**January 2017**

Today's Internet-based applications, such as online file sharing and video streaming services, have created exponential growth in network demand over the past decades. As portable electronic devices (smartphones, tablets, etc.) grow in popularity compared to PCs, remote Internet services become more accessible to people everywhere, thus imposing an even larger traffic burden on the Internet. In addition, sharing information across an increasingly sophisticated technical net-

work composed of hundreds of thousands of devices raises the security threat to user privacy, as well as the possibility of these devices being vulnerable to targeted attacks.

Collaborative distributed systems have emerged that offer multiple network resources to jointly fulfill common tasks for end users. As a result, users should be entitled to securer, more reliable and much faster service. Yet, for all the promise of these systems, they all have limitations that can make it hard to balance three key factors: performance, security, and the necessary incentives to keep participants honest and reliable.

This dissertation describes our research initiatives in building and adapting a few distinct collaborative distributed systems that can help maintain this balance. Taken together, our findings also offer solutions to meet ever-increasing privacy and network requirements. We start with Bitcoin, a peer-to-peer network-based decentralized cryptocurrency. Driven by the price surge of the Bitcoin network, users are pooling their computational resources to create new blocks that can lead to higher profits. Our research began by characterizing how the productivity, computational power, and transaction activity of miners have evolved over time. By considering the Bitcoin price and the computational race between miners, we were able to build a simple economic model to explain the evolution of Bitcoin mining practices.

To better understand network security and privacy, we developed an information-theoretic multi-block private information retrieval (PIR) scheme that significantly reduces client communication and computation overhead by downloading multiple data blocks at a time. Our work demonstrates that a multi-block PIR scheme can be optimized to simultaneously achieve low communication and computation overhead, comparable to even non-PIR systems, while maintaining a high level of

privacy.

Lastly, to see how incentives could be incorporated into security strategies in practice, we proposed a cryptocurrency system called CacheCash that provides a distributed content delivery service on top of a currency exchange medium. We developed a reference implementation of CacheCash and evaluated its performance in terms of computation and bandwidth requirements. The benchmark results demonstrate that our system can quickly and efficiently deliver content to clients, and can scale well to meet huge network traffic demands.

Taken as a whole, the research presented in this dissertation suggests practical paths to harnessing the strong potential of collaborative networks.

# List of Publications

1. **Luqin Wang** and Yong Liu. "Exploring Miner Evolution in Bitcoin Network." In *International Conference on Passive and Active Network Measurement*, pp. 290-302. Springer International Publishing, 2015.
2. **Luqin Wang**, Trishank Karthik Kuppusamy, Yong Liu, and Justin Cappos. "A Fast Multi-Server, Multi-Block Private Information Retrieval Protocol." In *2015 IEEE Global Communications Conference (GLOBECOM)*, pp. 1-6. IEEE, 2015.
3. Ghada Almashaqbeh, **Luqin Wang**, Allison B Lewko, and Justin Cappos. "CacheCash: A Cryptocurrency-based Decentralized Content Delivery Service." In review.
4. Yuan Ding, Yuan Du, Yingkai Hu, Zhengye Liu, **Luqin Wang**, Keith Ross, and Anindya Ghose. "Broadcast Yourself: Understanding YouTube Uploaders." In *2011 ACM SIGCOMM Conference on Internet Measurement Conference*, pp. 361-370. ACM, 2011.

# Contents

<b>1</b>	<b>Introduction</b>	<b>1</b>
1.1	Background, Motivation and Research Strategy . . . . .	2
1.1.1	Bitcoin Network . . . . .	2
1.1.2	Private Information Retrieval . . . . .	6
1.1.3	Distributed Data Service . . . . .	9
1.2	Dissertation Outline . . . . .	13
<b>2</b>	<b>Exploring Miner Evolution in Bitcoin Network</b>	<b>14</b>
2.1	Survey of Bitcoin Network . . . . .	16
2.1.1	Account and Transaction . . . . .	16
2.1.2	Block and Blockchain . . . . .	17
2.1.3	Bitcoin Mining . . . . .	18
2.2	Methodology . . . . .	20
2.2.1	Data Collection . . . . .	20
2.2.2	Solo Miner Analysis . . . . .	21
2.2.3	Pool Miner Analysis . . . . .	22
2.2.4	Simple Economic Model for Miners . . . . .	23
2.2.5	Limit of Computational Race . . . . .	24
2.3	Characterization Results . . . . .	26

2.3.1	Solo Miners . . . . .	26
2.3.2	Pool Mining . . . . .	29
2.3.3	Economic Considerations . . . . .	30
2.4	Summary . . . . .	33
<b>3</b>	<b>A Fast Multi-Server, Multi-Block Private Information Retrieval Protocol</b>	<b>34</b>
3.1	Architecture and Threat Model of Private Information Retrieval . . . . .	36
3.1.1	Architecture . . . . .	36
3.1.2	Threat model . . . . .	37
3.2	Single-Block PIR Scheme . . . . .	38
3.2.1	Notations . . . . .	39
3.2.2	Single-Block PIR . . . . .	39
3.3	Binary Multi-Block PIR (BMB-PIR) . . . . .	42
3.3.1	Multi-Block Download Scheme . . . . .	42
3.3.2	Construction of $k$ -safe Binary Matrix . . . . .	45
3.4	Multi-Block PIR over a Finite Field (FMB-PIR) . . . . .	49
3.4.1	Construction of $k$ -safe Binary Matrix . . . . .	49
3.4.2	Multi-Block Download Scheme . . . . .	50
3.4.3	Computational Cost . . . . .	52
3.5	Robustness against Byzantine Failures . . . . .	54
3.6	Performance Evaluation . . . . .	56
3.6.1	Implementation . . . . .	56
3.6.2	Setup . . . . .	57
3.6.3	Results . . . . .	58
3.7	Summary . . . . .	63

<b>4</b>	<b>CacheCash: A Cryptocurrency-based Decentralized Content Delivery Service</b>	<b>64</b>
4.1	Related Work . . . . .	67
4.2	CacheCash Design Overview . . . . .	70
4.2.1	Design Challenges . . . . .	70
4.2.2	Network Model . . . . .	73
4.2.3	Cryptographic Primitives . . . . .	74
4.2.4	System Overview . . . . .	75
4.3	Incentivized Data Service Protocol . . . . .	81
4.3.1	Batching Client Requests . . . . .	81
4.3.2	Data Co-location Puzzle . . . . .	84
4.3.3	Data Blocks Retrieval . . . . .	87
4.3.4	Proof-of-cheating Processing . . . . .	90
4.4	Performance Evaluation . . . . .	91
4.4.1	Methodology . . . . .	91
4.4.2	Content Provider . . . . .	93
4.4.3	Cache . . . . .	104
4.4.4	Client . . . . .	107
4.5	Summary . . . . .	108
<b>5</b>	<b>Conclusion</b>	<b>110</b>
5.1	Summary of Contributions . . . . .	110
5.2	Future Work . . . . .	111

## List of Figures

2.1	Monthly BTC statistics . . . . .	26
2.2	CDF of miners' annually earnings . . . . .	27
2.3	CDF of miners' annually earning . . . . .	27
2.4	Transfer lag distribution . . . . .	29
2.5	F2Pool miner statistics . . . . .	29
2.6	F2Pool miner hash rate vs. pool hash rate. . . . .	30
2.7	Daily mining profit rate and break-even time. . . . .	32
3.1	Diagram of a Chor-PIR scheme to retrieve one data block . . . . .	41
3.2	In BMB-PIR, as the number of employed mirrors increases, the communication overhead decreases. However, the privacy level also decreases. . . . .	48
3.3	Communication overhead . . . . .	59
3.4	Data block retrieval time . . . . .	61
4.1	CacheCash network model. . . . .	73
4.2	CacheCash operation. . . . .	75



4.3	Lottery draw example: the lottery draw time is 40, the ticket hash is taken over the committed fields and compared with the hash of the block with an index that equals the lottery draw time. The redeem time of this ticket is after Block 40 is confirmed until time 60. After that the ticket expires. Lottery over $tk_{L2}$ proceeds in an analogous way. . . . .	79
4.4	Commitment to lottery tickets example. . . . .	80
4.5	Batching client requests. . . . .	82
4.6	Data blocks retrieval from caches. . . . .	87
4.7	Puzzle generation speed (warm cache). . . . .	95
4.8	Puzzle generation speed (cold cache). . . . .	95
4.9	Puzzle generation speed for warm and cold cache ( $R_{puzzle} = 2$ ). . . . .	96
4.10	Average batch count. . . . .	99
4.11	Average data block reply size (warm cache). . . . .	100
4.12	Average data block reply size (cold cache). . . . .	101
4.13	Content provider throughput. . . . .	102
4.14	Content provider goodput (warm cache). . . . .	102
4.15	Content provider goodput (cold cache). . . . .	103
4.16	Goodput per content provider per byte. . . . .	103

## List of Tables

2.1	Fraction of frozen miners and average transfer lag of active miners	28
2.2	Sustainable computational power under current BTC prices . . . .	31
3.1	Lowest overhead achieved by Chor-PIR at different privacy levels	54
3.2	Lowest overhead achieved by BMB-PIR at different privacy levels	54
3.3	Lowest overhead achieved by FMB-PIR at different privacy levels	54
3.4	BMB/FMB PIR mirror goodput (Mbps) at different privacy levels	60
4.1	Netflix internet connection speed recommendations . . . . .	92
4.2	Netflix ISP speed index . . . . .	93
4.3	Worst case puzzles solve time in seconds . . . . .	107

# Chapter 1

## Introduction

As the Internet and the devices that access it advance in sophistication and reliability, the opportunities for employing multiple network resources to jointly fulfill common tasks for end users continue to grow. These so called collaborative distributed services leverage computational power and bandwidth across multiple network nodes to enhance overall performance, scalability, reliability, and security. For example, miners in a cryptocurrency system volunteer to pool their computational resources to solve proof-of-work puzzles and share the payout with lower variance. Applications like BitTorrent [1] and Gnutella [2] – provide peer-to-peer file sharing services to exchange users’ multimedia files. Another example is the use of multi-server private information retrieval (PIR) systems [3,4], which allow clients to acquire desired content from multiple servers, none of which actually know the nature of the clients’ queries [5].

However, these collaborative projects also bring with them new challenges. To tweak and improve one single component may compromise other features. Thus, each specific system needs to carefully balance each metric, particularly when it comes to three key factors: performance, security, and the necessary incentives to

keep participants honest and reliable.

This dissertation describes our research initiatives into maintaining that balance in a few distinct collaborative distributed systems. We first look into the most popular decentralized cryptocurrency, namely Bitcoin. By analyzing the public ledger, we discuss the incentives for mining and characterize the evolution in the behavior of Bitcoin miners. In our second work, we improve the efficiency of the classical multi-server PIR protocol. By leveraging binary and finite-field  $k$ -safe matrix construction, our scheme can significantly lower communication overhead and increase server goodput. Finally, we propose a secure bandwidth verifiable CDN service called CacheCash, which is integrated with cryptocurrency as a native payment system. The performance evaluation results show CacheCash is fast, efficient, and scalable. Collectively, all three research initiatives suggests practical paths to maximize the strong potential of collaborative networks.

## 1.1 Background, Motivation and Research Strategy

### 1.1.1 Bitcoin Network

In recent years, Bitcoin [6], a peer-to-peer network based digital cryptocurrency, has attracted a lot of attention from the media, academia, and the general public. Unlike traditional currencies, which are issued by central banks, Bitcoin is not regulated by any monetary authority. It is used in a peer-based network, where every peer is entitled to monitor the creation of Bitcoins (BTCs) and verify transactions. Every BTC is tied to a Bitcoin address, which provides no information as to the owner's identity. Therefore, all transactions along the Bitcoin

network are anonymous and can bypass bank regulation. Investors seeking the huge potential of the value of Bitcoin, have speculated over a number of years to profit from the volatile exchange market. The exchange price of BTC surged to more than 1,200 USD in late 2013, and now fluctuates between 300 and 700 USD. Recognized as the first decentralized digital currency in the world, since it was deployed in 2009, Bitcoin's market cap has grown to more than 11 billion US dollars. Since that time, numerous decentralized cryptocurrencies, e.g., Ethereum [7], Ripple [8], Litecoin [9], have flooded the market. Some have made small improvements on Bitcoin's anonymity, security, reliability, or other properties. However, none have gained as much public attention or expanded market capitalization as Bitcoin. Thus, the success of Bitcoin has raised significant research interest in cryptocurrencies, and influenced many related fields.

In terms of security, Goldfeder et al. [10] proposed a new threshold signature scheme to secure Bitcoin wallets. Becker et al. [11] estimated the typical cost structures in a Bitcoin network, and discussed the possibility of a denial of service attack. Baqer et al. [12] conducted an empirical study to identify spam transactions and proposed changes to transaction fees that could mitigate the effectiveness of spam DoS attacks. Eyal and Sirer [13] and Kroll et al. [14] discussed the vulnerabilities of a Bitcoin network if powerful adversaries could potentially manipulate mining mechanisms. Huang et al. [15] studied how malware can steal users' computational power to mine Bitcoins, and to deduce the amount of money a number of mining botnets have made.

To measure activity and explore the limits of anonymity that a Bitcoin network provides, Freid and Harrigan [16] analyzed the Bitcoin transaction flow in sampled data and showed that publicly announced addresses can be used to link and identify Bitcoin users. Meiklejohn et al. [17] was able to cluster Bitcoin accounts owned

by the same user by grouping input addresses from the same transaction. Ron and Shamir [18] examined the entire transaction graph of the Bitcoin network to study the pattern of abnormal transactions, while Androulaki et al. [19] designed and implemented a simulator for Bitcoin to evaluate its privacy implications.

While these security and privacy related aspects of Bitcoin and other digital currency have been studied at length, the incentive for Bitcoin mining and its economic soundness are rarely analyzed. As we conducted our study, we looked into the Bitcoin network from a new angle by focusing on changes in mining and transaction behaviors by Bitcoin miners over time.

Understanding these behaviors starts with understanding how Bitcoin differs from regular currency. There is no central bank or authority that decides how many Bitcoins are to be issued and distributed. In addition, according to the Bitcoin protocol, this currency is finite. Other than buying Bitcoins from others, the only way for a user to acquire them is to contribute computational resources to pack and verify new transactions. We call this process Bitcoin mining, and the users who participate in it Bitcoin miners. The Bitcoin protocol is designed so that new Bitcoins are mined at a steady rate until all are mined. The surge of Bitcoin pricing motivates miners to invest in more and more powerful hardware for faster mining. Due to the dramatic growth in both the number of miners and the computational power of their hardware, it has become increasingly difficult to mine Bitcoins. For an individual miner, even with powerful hardware, it now takes a very long time if she mines by herself, a behavior called solo mining. Similar to pooling money to buy a lottery ticket, the majority of miners choose to pool their computational resources to mine Bitcoins together. This is known as pool mining, and it gives individual miners steadier payouts than solo mining.

A Bitcoin network is a P2P system from which peers can obtain direct finan-

cial incentives for contributing their computational resources. While the Bitcoin price is constantly driven by various economic, political and legal factors, we are interested in finding out how Bitcoin's price evolution has driven miners' behaviors.

Towards this goal, we conducted a measurement study by analyzing the complete transaction blockchain of the network from January 3, 2009 through March 11, 2014. We first characterized how the productivity, computational power and transaction activity of miners evolved over time. We downloaded the entire Bitcoin blockchain, where we parsed and identified 290,089 blocks and 34,646,076 transactions. Using this information, we counted the Bitcoins(BTCs) each miner extracted each month within the network, and we correlated these BTCs to their U.S. market value. In addition, we also estimated the overall computational power of miners over time. We found in the early stages that computational power was evenly distributed. Comparing the miners' BTC generation time and their first transaction after mining, we used the time interval as a lower bound of miners' cash out log. From this result, we concluded that many early miners lost their bitcoins because they did not realize their potential value.

Our next step was to conduct an in-depth study of the largest Bitcoin mining pool, F2Pool [20]. The majority of these miners are located in the US and China. F2Pool is a China-based mining pool where the payout rules are clear. From all incoming BTC generation transactions and outgoing payout transactions of this pool, we identified all pool miners from the F2Pool. It showed that the number of pool miners increases with the rise of the BTC exchange rate. We also found that the overall computation power of F2Pool grows at a similar rate as the BTC exchange value.

Finally, we built a simple economic model that explains the evolution of miner

behavior by considering the Bitcoin price and the computational race between miners. We first evaluated the profit rate of two Bitcoin mining devices released in 2011 and 2013, respectively. The key factors shown to dominate the profit rate are electricity prices, Bitcoin network difficulty, and computation-over-power efficiency. Using electricity prices from all over the world, and the most powerful current mining device, we estimated the power upper bound of the Bitcoin network, and concluded that the current Bitcoin network still has room for growth before it reaches saturation.

### 1.1.2 Private Information Retrieval

In the past decades, the growth of Internet-based services has provided a variety of new products and experiences that have helped people improve their quality of life. Whenever a user requests such a service online, for example, buying from an online store, she needs to send personal information, such as name, address, and a credit card number to an online server. Until recently, these user profiles were considered individual property, which the server could not keep or use for its own purposes. However, this assumption did not stand long. Large websites now claim the right to store user information for the stated aim of recommending new products, based on previous shopping choices.

At the same time, online users are now more conscious of the need to protect their privacy. They do not want their personal preferences used for any purpose other than their desired ones. In regard to network applications, a user often desires to retrieve information from a server without revealing exactly what she wants to download. For example, an inventor may want to query a patent database without revealing which patents she wants to access [21], or a trader may want to



get specific stock quotes without revealing the companies in which she is interested [22], or a client may want to download a security update without revealing which unpatched vulnerability the update addresses [23]. Private Information Retrieval (PIR) allows users to retrieve queries from a database without revealing what they seek to anyone or anything, including the database server itself.

One way to keep the server ignorant of the nature of a query is for the client to download the entire database. However, this scheme incurs huge communication overhead. In the original PIR protocol, proposed by Chor et al. [3], lowering the communication overhead was accomplished by deploying multiple servers, each with a replicated copy of the database. A client requests a partial masked secret from different servers, and reconstructs the desired data. The overall bandwidth between the client and servers is strictly less than the size of the database. We call such protocol *non-trivial* PIR. The security property lies in the assumption that the number of servers that communicate with each other is no more than threshold  $k$ . Even if the adversary has unlimited computational power, in theory, the assumption guarantees the privacy unbreakable. Thus we call such a protocol information-theoretic PIR.

Information-theoretic PIR relies on deployment of multiple databases. Chor et al. [3] shows that it is impossible to achieve a single-database *non-trivial* PIR. However, with the assumption that a quadratic residuosity problem is hard, a PIR scheme with one copy of the database is viable to meet the *non-trivial* requirement. We call such a protocol a computational PIR [24–27]. Unfortunately, due to communication complexity, these PIR protocols are known to be impractical [28] and there is no real-life deployment. Thus, information-theoretic PIR becomes a better choice to bring the zero knowledge query to practical use. However, the high communication overhead is still a barrier, since a client may need

to expend much more download time to retrieve the content she wants. If the content is time sensitive, or its size is extremely large, such a PIR protocol is not feasible. Henry et al. [29] proposed a multi-block scheme for Goldberg’s [30] design by encoding multiple data block requests into a single PIR query. However, in order to tackle Byzantine failure [31] and enhance robustness, their scheme applies both computationally-expensive operations over a finite field, with error-correcting codes. The resulting system incurs high communication overhead and slows client decoding time. Demmler et al. [32] developed a multi-block scheme that further reduced the communication overhead, but compromised the flexibility to retrieve any query result.

Our goal is to develop a PIR protocol that significantly reduces the client download time, thus lowering communication overhead, and increasing the server goodput. In our work, we propose the first matrix-based information-theoretic PIR scheme that combines multiple block requests, while using fast XOR operations instead of more computationally-expensive operations. Using fast XOR operations allows us to reuse the same high performance PIR mirror infrastructure that has been shown to have similar goodput to FTP on realistic datasets and deployment environments [23]. However, using multiple block requests has benefits over this existing PIR scheme. By leveraging multiple block requests, our proposed scheme can further increase performance and reduce communication overhead by a factor of up to three.

We also developed a finite-field based PIR scheme to further reduce the communication overhead. Our basic finite-field scheme assumes there is no Byzantine mirror in the system and focuses on minimizing the communication overhead. By sacrificing the mirror computation load and goodput, the finite-field scheme achieved even lower communication overhead than the binary matrix scheme by

a factor of up to 14 in extreme cases. We then augmented the basic finite-field scheme with a simple, yet robust, detect-retransmit mechanism to handle Byzantine failures, following the design philosophy of “*make the common case fast and make the uncommon case correct.*”

Finally, we built prototypes to validate the benefits of binary and finite-field schemes in practical environments. Through extensive experiments, we showed that both binary and finite-field multi-block PIR schemes have much lower communication overhead than classic PIR schemes. While maintaining a high-level of privacy, the mirror goodput is even comparable to systems that do not offer privacy, such as HTTP and FTP.

### 1.1.3 Distributed Data Service

Over the past few decades, Internet traffic has grown exponentially due to the acceleration of network-based services, which make numerous demands and require extensive bandwidth. Many of these services were originally offline. Traditionally, users who wanted to share their media content have had to physically exchange storage media with each other. In the past, movie lovers needed to rent movies from a local video store in order to watch with their friends and family.

Nowadays, Internet-based services have developed at such a fast pace that service providers can offer users faster and cheaper services with easy access. Online file sharing services, e.g., BitTorrent and Gnutella, offer users the opportunity to exchange data over a peer-based network that partitions the workload to boost the transferring speed and improve reliability. Online video streaming services, such as Netflix [33] and Amazon Video [34], offer high quality movies and TV shows for smart TV, PC, and smartphone users to watch at anytime anywhere. Simply by

clicking a mouse, one online user can transfer files to another or instantly watch online streaming. Users enjoy the service, and leave it to the internet and their service provider to take care of the rest.

All of these applications rely on distributed services that collaboratively offload workloads to multiple network nodes, thus improving overall performance. Nonetheless, with tremendous traffic demands, a multiple-connection infrastructure incurs extensive communication overhead. Such an overhead brings both security and reliability issues that largely increase the risk of exchanging data over the network, while lowering the efficiency of bandwidth utilization as well. In the last decade, content delivery services have grown 45% - 50% per year. With regard to these aspects, researchers are challenged to improve the speed, security, and reliability of data exchange among devices.

Content delivery networks (CDNs) have emerged as an attractive solution to distribute the workload [35–37]. The core idea is to replicate the media content and disburse it among geographically distributed servers, also called caches, to serve clients on behalf of the original content providers. Content providers may construct their own CDN or buy the service from a third party, such as Akamai [38]. However, infrastructure based CDNs are considered expensive, due to the costs of deployment and continuous maintenance. Inspired by peer-to-peer (P2P) networks, peer-assisted CDNs evolved as a low overhead and decentralized solution that relies on exploiting end users' bandwidth to distribute data [39, 40]. However, this flexibility in allowing any user to be part of the network raises several challenging questions: What motivates peers to use their bandwidth to serve others? Can we trust any party to distribute the correct content? And is there a guarantee that these parties will follow the protocol?

Incentive-based paradigms have been introduced to motivate collaboration be-

tween peers [41]. An early example is the tit-for-tat protocol in Bittorrent [42], where one serves data to get data. This mechanism, though, suffers from the free-riders problem of peers consuming more resources than they provide. Monetary incentives proved to be more effective since they create a market for trading available bandwidth [41, 43]. While this strategy has clear advantages, involving monetary incentives in large-scale systems can complicate payment processing. An early approach was to have a trusted party track payments for all participants, e.g. [39]. Such an approach is simple and secure, as this trusted entity can solve any dispute that may arise. Nevertheless, it introduces a single point of trust and a potential bottleneck for efficiency. This conflicts with the goal of having a distributed, flexible, and scalable content distribution system.

As an alternative to this centralized model, Bitcoin [6] has emerged as the first successful and widely-used cryptocurrency system. Bitcoin combines the use of basic cryptographic primitives, a proof of work concept, a publicly verifiable blockchain, and a distributed consensus protocol to provide a decentralized currency exchange medium. In addition, several systems have evolved to support extra services on top of Bitcoin's basic functionality [44–46], which create new opportunities to support decentralized monetary incentives in peer-assisted CDNs.

With a knowledge of these technologies, we set out to design a practical system to provide content delivery with low deployment costs. Users are incentivized to serve as caches to help content providers offload network traffic for a fair price. The system integrates cryptocurrency as a monetary payment method that relies on a decentralized consensus protocol to monitor and verify transactions. In addition, the system we design should be secure due to assumptions on certain threats.

Towards this goal, we propose a system called CacheCash that combines the best features of CDNs and Bitcoin. CacheCash is a cryptocurrency based stor-

age/bandwidth market where users can rent their storage and sell their extra bandwidth to content providers. It adopts the layered structure from CDNs in which content providers own the material, but delegate the task of serving clients to caches, which distribute replications of this content. Like a P2P network, CacheCash allows anyone to join the system as either a content provider or as a cache. Content providers pay caches to serve their clients. Hence, CacheCash enables the building of dynamic CDNs with lower overhead than infrastructure-based networks, but in a more organized way than in P2P networks. To ensure fair payments and accountability, CacheCash provides a publicly-verifiable service that uses a blockchain as a trusted log to make cheating detectable, and thus, unprofitable.

We conducted a thorough study to understand the performance of CacheCash. Our assessment asked the following questions:

- How quickly does CacheCash serve content?
- How efficiently can clients retrieve this content?
- Can CacheCash scale to handle larger demands?

We set about answering these questions by implementing a CacheCash data service protocol and evaluating the efficiency of the various modules of the system against several performance measures. We show that CacheCash benchmarks demonstrate its efficiency in terms of computational and bandwidth requirements, and, that it can easily scale to handle large scale CDN dissemination, such as distributing Netflix content.

## 1.2 Dissertation Outline

This thesis is organized as follows:

Chapter 2 starts with a short survey of a Bitcoin network. Then we explain our methodology and investigate the blockchain of such a network. Finally, we analyze our measurement results and discuss and characterize how mining behaviors have changed over time.

In Chapter 3, we present the PIR system architecture and threat model. Then we show two novel information-theoretic PIR schemes: binary matrix-based PIR and finite-field based PIR. Finally, we evaluate the performance of the proposed multi-block PIR schemes.

In Chapter 4, we first present an overview of our proposed cryptocurrency-based content delivery service system. We then present the details of the data service protocol, and conduct a thorough performance evaluation of the system.

In Chapter 5, I summarize my contributions to academic research while pursuing my Ph.D. degree in Computer Science, and point out directions for my future work. The chapter also offers some concluding thoughts about my work as a whole.

## Chapter 2

# Exploring Miner Evolution in Bitcoin Network

This work represents a collaboration with my thesis advisor Yong Liu.

Bitcoin [6] is known as *the first decentralized digital currency in the world* [47]. As a modern cryptocurrency supported by a peer-to-peer network, Bitcoin users can conduct instant, secure, and anonymous financial transactions without service fees. Unlike traditional currency, which is issued and regulated by a sovereign bank, Bitcoin is not controlled by any institution or country. It circulates globally without boundary and is free from financial regulation systems due to its decentralized P2P accounting and transaction design. Bitcoin debuted in 2009 and after five years of development, its exchange price has risen from nothing to more than \$100 per coin through mid 2013. It surged to a peak of \$1,242 on November 2013, and is now wobbling between \$350 and \$600. Till September 2014, the market capitalization of Bitcoin has increased to around 6 billion US dollars; and the Bitcoin network runs more than 60,000 transactions daily. Along with Bitcoin network's capitalization and volume, a number of derivative services



have been developed and regulated. Exchange markets, such as Coinbase [48] and Bitstamp [49], allow users to buy and sell Bitcoins using regulated currencies globally. Online merchants, e.g., Dell and Overstock, are now accepting Bitcoin as a payment method. Governments of several countries, such as Canada and Thailand, have approved fully-legal Bitcoin exchange and issued tax guidance on Bitcoin transactions.

Different from a regulated currency, there is no central bank or authority who decides how many Bitcoins are to be issued and distributed. According to the Bitcoin protocol, there are only a finite amount of Bitcoins. In addition to buying Bitcoins from others, the only way for a user to acquire Bitcoins is to contribute her computation resources to pack and verify new transactions. We call this process *Bitcoin mining*, and the users who participate as *Bitcoin miners*. The Bitcoin protocol is designed so that new Bitcoins are mined at a steady rate until all Bitcoins are mined. The surge of Bitcoin price motivates Bitcoin miners to invest in more and more powerful hardware for faster mining. Due to the dramatic growth in both the number of Bitcoin miners and the computational power of their hardware, it has become increasingly difficult to mine Bitcoins. For an individual miner, even with powerful hardware, it now takes a very long time to get Bitcoins if she mines by herself, as a so-called *solo miner*. Similar to pooling money to buy lottery tickets, the majority of miners choose to pool their computational resources to mine Bitcoins together. *Pool mining*, as it is called, gives individual miners steadier payouts than solo mining.

The Bitcoin network is a P2P system in which peers can obtain direct financial incentives by contributing their computation resources. While the Bitcoin price is constantly driven by various economic, politic and legal factors, we are interested in finding out how the fluctuations in Bitcoin value drives the miners'

behaviors. Towards this goal, we conduct a study on the evolution of Bitcoin mining practices by analyzing the complete transaction blockchain of the Bitcoin network from its very first transaction in 2009 to March 2014. We first characterize how the productivity, computational power and transaction activity of miners have developed over time. We then conduct an in-depth study of the largest mining pool F2Pool [20]. We characterize how it grows and how its computational power is distributed among its heterogeneous members. Finally, we build a simple economic model to explain the connection between the evolution of the mining behavior shifts and the fluctuation of Bitcoin prices and the computational power of the miners.

## 2.1 Survey of Bitcoin Network

### 2.1.1 Account and Transaction

The Bitcoin network is a peer-to-peer network without a central authority. A Bitcoin account is simply a pair of public/private keys. An account ID is derived from its public key. The private key is used to generate a digital signature for authentication. There is no cost to create a Bitcoin account, and each user can create as many accounts as she wishes. A transaction is a mechanism for users to transfer Bitcoins to each other. A transaction consists of a set of senders and a set of receivers (denoted by their account IDs), the amount from each sender, and the amount to each receiver. For example, if Alice wants to send 3 Bitcoins (BTCs) to Bob, she might send from two of her accounts. One account  $A_1$  has 1 BTC and the other account  $A_2$  has 2 BTCs. If Bob has only one account,  $B_1$ , this transaction is simply: 1 BTC from  $A_1$ , 2 BTCs from  $A_2$ , and 3 BTCs to  $B_1$ . All senders will

sign the transaction with their private keys, and the signed transaction will be broadcast to the entire Bitcoin network. Any user who receives this transaction will first verify whether the senders have the amount of BTCs indicated in the transaction. Different from the traditional banking systems, there is no central database to maintain the Bitcoin balance of each account. Instead, the whole Bitcoin network stores and verifies all the transactions using a shared blockchain. Any user can check the balance of any account by backtracking the blockchain to retrieve all transactions associated the account. Invalid transactions will be discarded, and valid ones will be stored in memory to be packed and appended to the blockchain.

### **2.1.2 Block and Blockchain**

The blockchain is a public ledger shared by the whole Bitcoin network. As the name suggests, the blockchain contains a chain of chronologically ordered blocks, each of which contain a generation transaction indicating which account packed this block, within a time window of ten minutes. Each user downloads and synchronizes a copy of the blockchain in her local machine to verify incoming transactions. All newly confirmed transactions are packed into a new block, which will be broadcast to the whole network. Whenever a user receives a block, she will validate all the transactions in this block using the current blockchain. If any transaction is invalid, she will discard the block. Otherwise, this block will be confirmed and appended to the current blockchain.

### 2.1.3 Bitcoin Mining

The Bitcoin network depends on the computational resources of users to maintain the integrity of the blockchain. Each user can volunteer to verify and pack new transactions to blocks. While a lot of users are doing the verification and packing work simultaneously, only the newest valid block will be confirmed by all users and appended to the current blockchain. The user (miner) who created this block will be rewarded with BTCs (the current reward is 25 BTCs/block). To achieve this, a proof-of-work mechanism is introduced. When packing new transactions to a block, a miner first generates a special transaction indicating that the network sends her the mining reward. Along with all other transactions, she repeatedly generates a random number nonce, puts them together and runs a hash function. If the hash value is below a target value, the user claims she created the block and broadcasts the block and the nonce. Other users can easily perform the same hash function with the published nonce to verify the block.

According to Nakamoto's protocol [6], the total number of BTCs that can be mined is 21 million and the last BTC to be mined is in block #6,929,999 near year 2140. By default, a new block is created approximately every ten minutes, no matter how much aggregate computational power is in the network. To control the new block creation speed, a *difficulty value* is introduced. The target value for block hash calculation is inversely proportional to the difficulty value. As a result, the higher the difficulty value, the more hash calculations each miner has to conduct to find a hash value below the target. As detailed in [50], at a given difficulty value  $D$ , for a miner with the computational power of  $H$  hashes per

second, the expected time for the miner to generate a new valid block is:

$$E[T] = \frac{D \times 2^{32}}{H} \text{seconds.} \quad (2.1)$$

The difficulty value is updated every 2,016 blocks, based on the speed at which the past 2,016 blocks were generated. The difficulty value is stored in each block. Knowing how many BTCs are generated in the whole network in one day, given the difficulty value, we can also calculate the total hash rate of the system.

### Solo and Pool Mining

In the early days of Bitcoin, miners worked alone. We call this approach *solo mining*. The advantage of solo mining is whenever a block is created by the miner, she gets all the rewards. However, as more and more computational resources are injected into the Bitcoin network, the difficulty value to control the new block creation speed increases significantly. Now it takes a powerful miner years to create a block. As an alternative, pool mining is a way for miners to combine their resources together to obtain steady payouts. A pool assigns a lower difficulty value to each of its members. It becomes easier for each miner in the pool to solve the hash problem and prove their work. Notice that if a higher difficulty is given by the network, miners who have tiny hash rates have less chance to solve the hash problem and it is harder to prove their work. Each pool miner submits her own hash values under the pool target value (called shares) to the pool for verification. If a share is under the network target value, a block will be claimed by the pool and the pool operator will distribute the reward to every pool miner. The most popular payout approach for pool mining is pay-per-share, in which miners are rewarded proportionally to the number of shares they submitted to

the pool. With pool mining, the expected payout for a miner is the same as solo mining, but the variance of payout is largely reduced.

## 2.2 Methodology

### 2.2.1 Data Collection

As described in Section 2.1, all transactions in Bitcoin network are stored in the blockchain. When a user wants to make a transfer, she must first connect to the Bitcoin network and synchronize with the current blockchain. To begin our investigation, we ran the Bitcoin client in our local machine to get the latest blockchain. We then parsed it to blocks and transactions. Each block contains its hash value, height (block ID), hash value of the previous block, generation time (in UTC timestamp), the amount of new BTCs created, target difficulty, nonce, and all transactions. For each transaction, inputs include the previous transaction hashes of the senders and the associated signature scripts; outputs include the receiver account IDs and their corresponding amounts. We use the previous transaction hash to retrieve the transaction history and the balance of each sender by iteratively backtracking the blockchain. We synchronized the complete blockchain in March 2014 and parsed the data. The raw data includes all blocks and transactions from 2009/01/03 (the very first Bitcoin block created) to 2014/03/11. We retrieved 290,089 blocks and 34,646,076 transactions. We then parsed all blocks and transactions field-by-field and stored all the parsed information into a MYSQL database.

### 2.2.2 Solo Miner Analysis

According to the blockchain, pool mining only started on December 16, 2010 [51]. All previous miners were solo miners. In pool mining, each group uses one unique ID to mine Bitcoins. We first assume each unique Bitcoin ID who successfully created a block as a solo miner. As a result, we treated pools as solo miners for now. Using block timestamps, we counted the number of BTCs mined each month in the network. In addition, using Bitcoin exchange market data [48] we calculated the monthly USD generated in the network, on the assumption BTCs were exchanged for USD at market price immediately after they were mined. Moreover, we also obtain the distribution of how many BTCs each miner mined over time.

Besides the earnings, we can also estimate the aggregate computational power of all miners. With a given difficulty value of  $D$ , if  $N$  blocks were mined in a day, based on (2.1), the aggregate daily hash rate of the entire Bitcoin network can be estimated as:

$$H_{total} = \frac{N \times D \times 2^{32}}{86,400} \quad (2.2)$$

Similarly, we can estimate a miner's daily hash rate by replacing  $N$  with the number of blocks mined in a day.

We are also interested in when the miners cashed out their BTCs after mining. However, it would be hard to collect IDs of all Bitcoin exchange markets in order to track all transactions. Instead, for each miner we track the interval between the time she mined some new BTCs and the time when her next transaction was issued. This time interval serves as a lower bound for her cash out lag.

### 2.2.3 Pool Miner Analysis

To study how the Bitcoin mining pool evolved, we collected pool data from our database. We choose miner IDs with top hash rates in the network and manually classified them. Most of these IDs belong to well-known mining pools, according to Blockchain.info, an online Bitcoin statistics website. To analyze pool mining behaviors, we choose F2Pool, a China-based mining pool whose payout rule is clear and payout transactions are easy to obtain. In our data up to March 2014, F2Pool ranked 7th in terms of the total computational power in the network. Newer statistics [52] from September 2014 show F2Pool has become the largest mining pool, with more than 25% of the overall computation power.

We query transactions having F2Pool's account ID and classify them as input or output. For transactions having F2Pool ID as the only receiver, we identify whether they are block generation transactions. For a transaction having it as the only sender, we validate whether the transaction is used to distribute payouts to pool miners. Pools have different approaches to payouts. The simplest way is to send out all payouts in one transaction immediately after each block is created. However, none of the ten pools we checked use this approach. Some pools use a binary tree-like iterative payout approach, which pays only one pool miner and transfers the remaining balance to a new ID at each iteration. And some pools randomly choose a number of miners to pay in one transaction and transfer the remaining balance to a new ID, and then distribute the remaining balance in subsequent transactions. When F2Pool mines a block, it will send out the payouts in the next day. It used to send out payouts to all members using a single transaction, but changed to two transactions recently. Knowing the payout mechanism, we can calculate how many BTCs each pool miner earns each day



from pool payout transactions.

## 2.2.4 Simple Economic Model for Miners

To become a miner, a user first needs to invest in hardware, ranging from conventional computers in the early days of Bitcoin, to graphics card, GPUs, and specially designed ASIC chips, and incurs a *capital cost*. After she joins the network for mining, she needs to pay the bill for electricity, air conditioning, housing and maintenance etc., and incurs an *operational cost*. Since miners are driven by profits derived from the mined Bitcoins, the economic question is *whether and how soon their revenues can cover their capital and operational costs?* To answer, we build a simple economic model. For a hardware with hash rate  $H$ , based on Equation (2.1), and assuming the hardware works 24 hours per day, the expected number of BTCs it can mine daily is:

$$N(t, H) = \frac{H \times 86,400}{D(t) \times 2^{32}} R, \quad (2.3)$$

where  $D(t)$  is the difficulty value in day  $t$ , and  $R$  is the number of BTCs rewarded for each block. If the hardware's power consumption is  $P$  *kw*, and the electricity price is  $\eta(t)$  per *kwh*, the daily electricity bill is  $24P\eta$ . Given the Bitcoin exchange price of  $\rho(t)$  in day  $t$ , if we only consider electricity operational cost, the daily profit rate  $r(t)$  for the hardware with hash rate  $H$  and power consumption  $P$  is:

$$r(t, H, P) = N(t, H)\rho(t) - 24P\eta(t). \quad (2.4)$$

Obviously, a miner prefers places with low electricity price  $\eta(t)$ , and will shut down her hardware whenever the profit rate becomes negative. Based on (2.3)

and (2.4), to maintain a positive profit rate, the hardware's computation-over-power efficiency should satisfy:

$$\frac{H}{P} > K \frac{\eta(t)D(t)}{R\rho(t)}, \quad (2.5)$$

where  $K$  is a constant. As  $D(t)$  increases, hardware with low computation-over-power efficiency will be quickly kicked out of the mining game.

To obtain high profit rate, miners should go for specialized mining hardware with high computation-over-power efficiency. That hardware comes at a high price, though. If the miner purchased a piece of expensive hardware at day  $t_0$  with price  $C$ , the time  $\tau$  it takes her to recover the capital cost should satisfy:

$$\int_{t_0}^{t_0+\tau} r(t, H, P) \times I[r(t, H, P)] dt = C, \quad (2.6)$$

where  $I[x]$  is the indicator function which equals 1 if  $x > 0$ , and 0 otherwise.

### 2.2.5 Limit of Computational Race

According to Equation (2.4), miners are highly incentivized to increase their computational power to obtain higher profit margin. The Bitcoin network has witnessed exponential computation power growth in the past few years. But, at the same time, the number of Bitcoins that can be mined each day is deliberately set by the network to a fixed value. If the Bitcoin price is kept flat, the total profit that miners can obtain from the network is a constant. All miners are essentially playing a zero-sum computational race game: if each miner increases her computational power, then the total computational power in the network increases. Consequently, the system increases the difficulty value  $D(t)$  to main-

tain a steady Bitcoin creation speed, which, in turn, reduces the Bitcoin mining rate of individual miners, according to Equation (2.3). This is an unfortunate and unavoidable *tragedy-of-the-commons* phenomenon that has been observed in the Bitcoin network. Such a race will automatically end when the profit margin hits zero. We can predict the equilibrium point by extrapolating from our simple economic model in the previous section. Namely, let  $\xi_0$  be the highest computation-over-power efficiency (in units of hash-per-second/kilowatt) that the future computation technology can achieve,  $\eta_0$  be the lowest electricity price in the world, and  $\rho_0$  be the steady state exchange price of Bitcoin, we could then immediately calculate the maximum sustainable computational power  $\mathcal{H}$  in the whole Bitcoin network as:

$$\frac{\mathcal{H}}{\xi_0}\eta_0 = 6R\rho_0, \quad \Rightarrow \quad \mathcal{H} = \frac{6R\xi_0\rho_0}{\eta_0}, \quad (2.7)$$

where the left-hand side of the first equation is the minimum electricity charge for one-hour of mining with the most efficient mining hardware, and the right-hand side is the expected hourly total payout in the whole network at the target mining rate of one block every ten minutes. If the total computational power goes above  $\mathcal{H}$ , the expected profit margins of all miners become negative, and some of them will start to drop out of the mining race. As this occurs, the profit margin comes back to positive.

So far we ignored the capital cost and other operational costs. Therefore Equation (2.7) gives us an upper bound on the maximum sustainable computation power at any fixed Bitcoin price  $\rho_0$ , given the highest feasible computational efficiency  $\xi_0$  and the lowest electricity price  $\eta_0$ .

## 2.3 Characterization Results

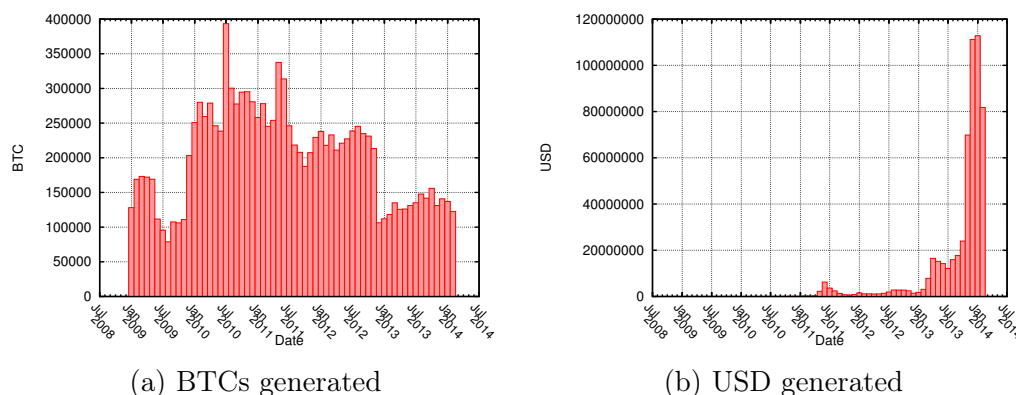


Figure 2.1: Monthly BTC statistics

Figure 2.1a plots the total Bitcoins generated each month on the blockchain. In 2009, the numbers were not stable because the Bitcoin client was newly released and the size of the network was relatively small. In December 2012, the reward  $R$  for each block was reduced from 50 BTCs to 25 BTCs, resulting in the number of BTCs being reduced by half in the latter months. Figure 2.1b shows how many USDs are generated monthly, according to the daily Bitcoin to USD exchange price.

### 2.3.1 Solo Miners

Figure 2.2 illustrates the distribution of solo miners' annual earnings. Before August 2010, there is no market data and the estimation of BTC value is \$0. The earnings in latter years are tremendously greater than in the earlier years due to the rapid rise in the exchange price. In addition, the top miners became more and more computationally powerful and were responsible for the most blocks created. Similar to (2.3), we estimated the hash rate for each solo miner, based on the

number of blocks she created.

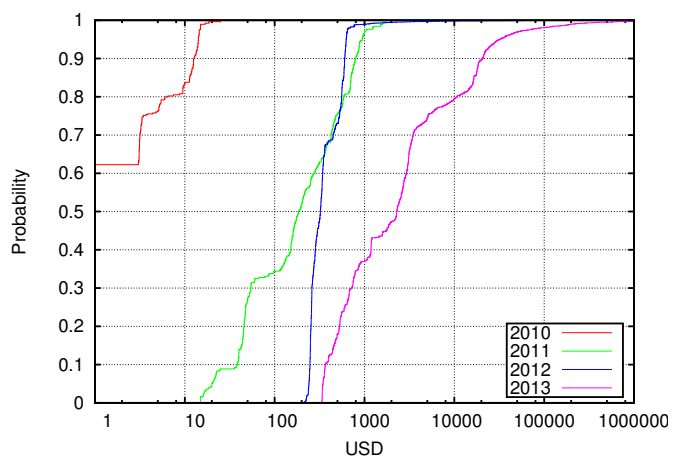


Figure 2.2: CDF of miners' annually earnings

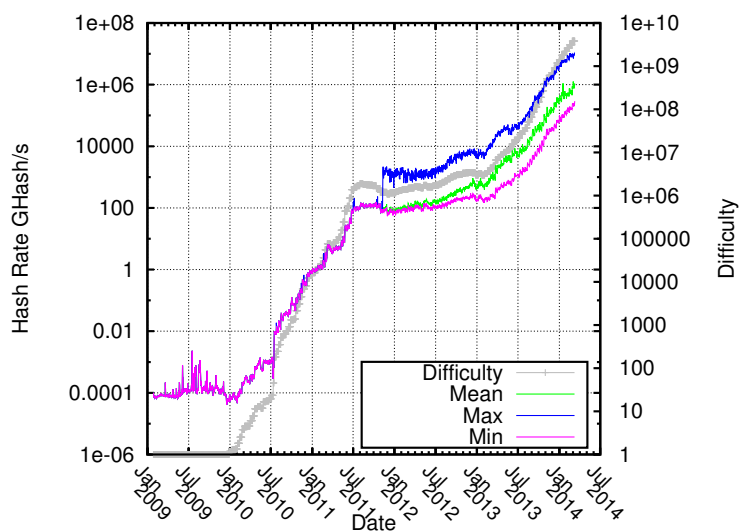


Figure 2.3: CDF of miners' annually earning

Figure 2.3 shows the minimum, maximum and mean hash rate of solo miners together with the system regulated difficulty value in logarithm scale. It shows that the computational power is evenly distributed among miners at the the early stage, then becomes highly skewed towards a small number of very powerful miners

as the Bitcoin network evolves. As will be studied next, the top mining figures indeed represent mining pools.

We now examine how quickly miners transfer out mined Bitcoins. We measured the time lag between when a miner claimed a block and her next transaction. If a miner had no subsequent transaction in our trace, we tagged the miner as frozen. For active miners, we calculated the average and distribution of their transfer lags.

	Frozen Miners	Active Transfer Lag
2009	66.36%	138 Days
2010	20.13%	102 Days
2011	1.89%	19 Days
2012	0.49%	7 Days
2013	0.96%	1.5 Days

Table 2.1: Fraction of frozen miners and average transfer lag of active miners

As shown in Table 2.1, a large fraction of early miners were frozen and never touched their mined Bitcoins, even after the Bitcoin price surge in 2013. Our conjecture is that those early miners were casual early adopters of Bitcoin as a fun technology, and were not motivated by its potential financial value. When Bitcoin became valuable, they might have, unfortunately, lost their account IDs, so that they couldn't cash out. This suggests that lots of Bitcoins mined in the first two years might have been lost permanently! Things changed completely in 2011. Not surprisingly, this is in sync with the value increase of Bitcoins. Not only are almost all miners active, but the lag for transfer gets shorter and shorter. The slight increase in frozen ratio from 2012 to 2013 is due to the artifact that our trace ends in March 2014.

Figure 2.4 further illustrates the decreasing trend of transfer lags as time evolves. This suggests that later miners were explicitly driven by profits and

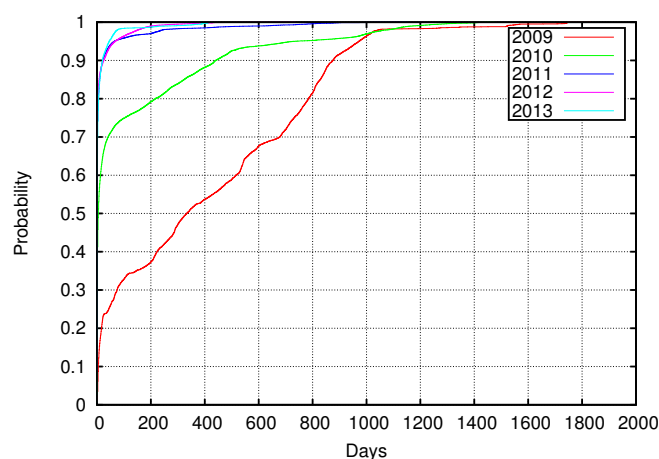
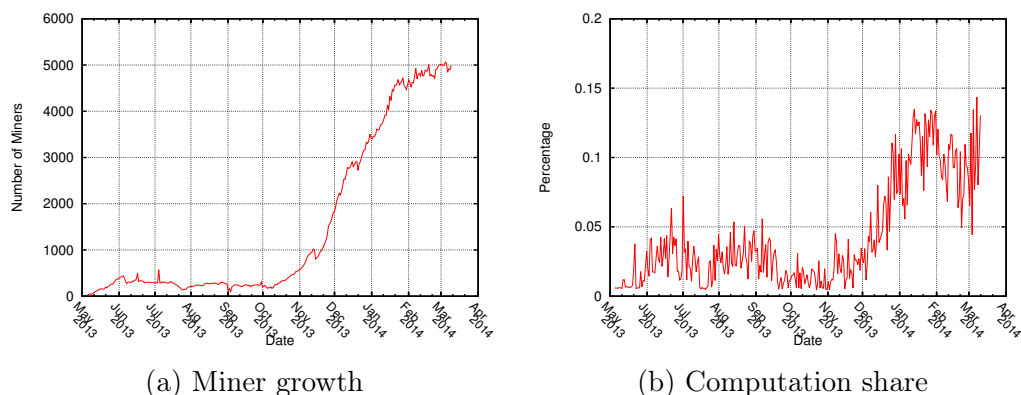


Figure 2.4: Transfer lag distribution

diligently transferred out mined Bitcoins.

## 2.3.2 Pool Mining



(a) Miner growth

(b) Computation share

Figure 2.5: F2Pool miner statistics

Figure 2.5a shows the increase in the number of miners in F2Pool. We can see that from May to October 2013, the number of pool miners is relatively stable. This is due to the stable Bitcoin price of around \$120 in that period. Figure 2.5b plots the ratio between F2Pool's computational power over that of the whole network. The ratio is also relatively stable from from May to October 2013.

Starting from November 2013, motivated by the price surge of Bitcoin, the number of miners increased by more than ten times up to March 2014. As illustrated in Figure 2.5b, F2Pool’s computational share also increased dramatically. This indicates that more miners chose to join pool mining in the face of increasingly tense competition between miners.

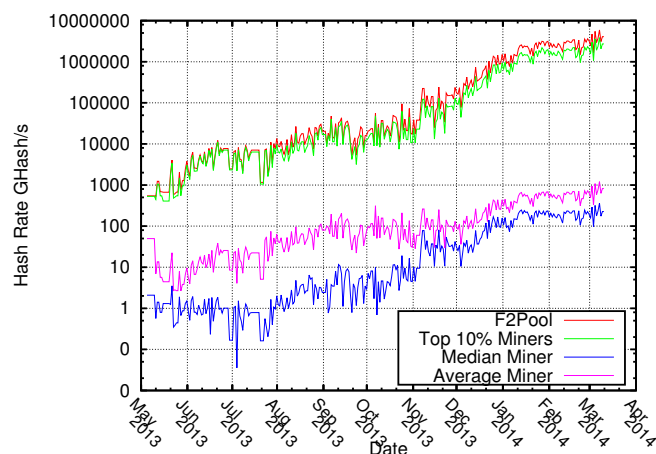


Figure 2.6: F2Pool miner hash rate vs. pool hash rate.

In Figure 2.6, we estimate the mean and median hash rate of F2Pool miners, and how much computational power is controlled by the top 10% of pool miners. The mean is larger than the median and the top 10% miners dominate the computational power of the pool. This is because the hash rates of the top pool miners are significantly larger than the low-end miners. Since a miner’s earning in a pool is proportional to her hash rate, the earnings distribution among miners in a pool conforms to the power law wealth distribution in the real world.

### 2.3.3 Economic Considerations

Curious about whether miners can earn their investment back, we chose two mining hardwares released in 2011 and 2013, respectively. The first one is a MSI



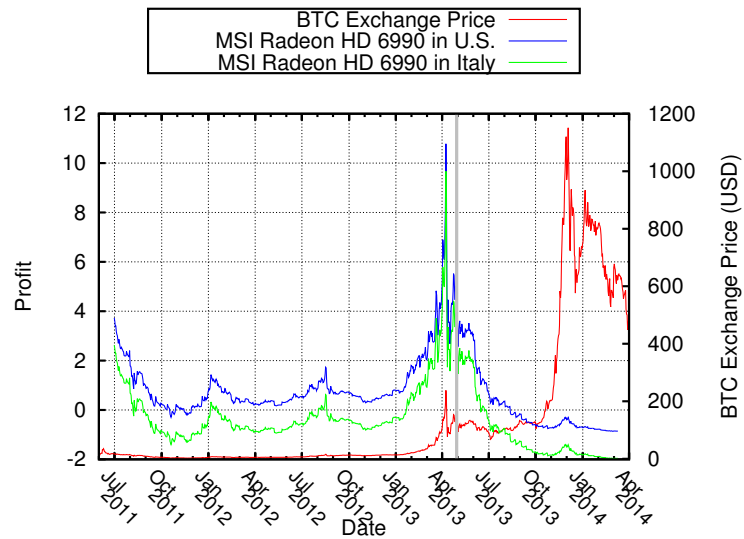
Radeon HD 6990 graphics card with 750 MHash/sec and 410 Watt. The price for this card at release was \$699. By 2010, mainstream miners were using graphics cards to do mining instead of CPUs. We set a starting date of 07/01/2011, and calculated the card's profit rate according to Equation (2.4) using the real Bitcoin price and electricity prices in the US and Italy, respectively. As shown in Figure 2.7a, this card generates positive profits in the US, breaks even (earns \$699 back) on April 30, 2013, and continues to make money until September 2013. Then the daily earning become negative even though the Bitcoin price kept increasing. This is because, as more miners joined the system, the difficulty value increased at a faster pace than the Bitcoin price. According to (2.5), the card's computation-over-power efficiency can no longer sustain a positive profit rate. Meanwhile, due to a higher electricity price (see Table 2.2), mining in Italy seldom made a profit. There is no way for the miner to recover her capital cost.

In late 2012 and early 2013, powerful ASIC hardware entered the mining market. We estimated the cost of BFL SC 5G/s mining cube, a 5,000 MHash/sec and 30 Watt advertised ASIC chip for just \$274. We found that if it were purchased on July 1, 2013, whether in the US or Italy, it would have broken even in less than one month. The major reason is that the computation-over-power efficiency of this new card is about one hundred times higher than the MSI Radeon HD 6990 graphics card.

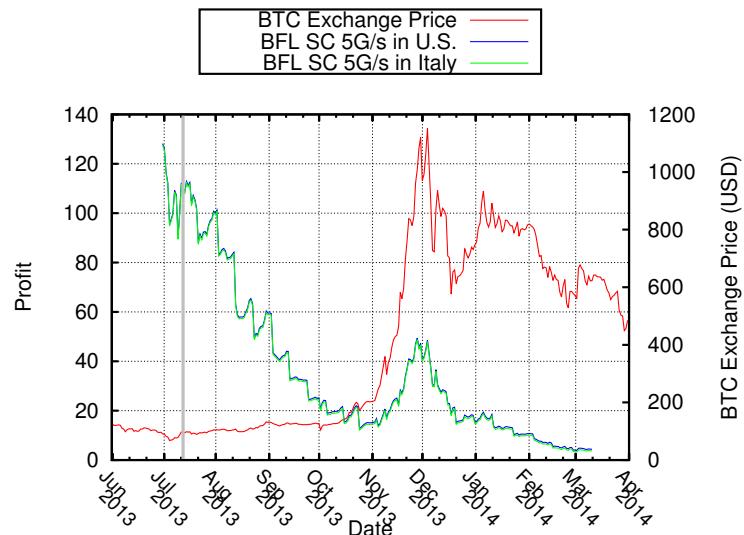
Country	Italy	UK	Belgium	US	Sweden
Average electricity price in 2013 (cent per kwh)	20.56	13.61	11.77	9.33	8.25
Computational power bound (THash/sec)	473,325	715,031	826,812	1,043,041	1,179,584

Table 2.2: Sustainable computational power under current BTC prices

Finally, we estimated the computational power upper bound of the Bitcoin



(a) Graphics card in 2011



(b) ASIC card in 2013

Figure 2.7: Daily mining profit rate and break-even time.

network according to Equation (2.7). We use the current Bitcoin price and the average electricity prices in different countries [53] to estimate mining costs. We chose the current best hardware SP35 YUKON ASIC chip, which has 6 THash/sec and 3,500 Watt. Table 2.2 shows as electricity prices vary, the network computational power upper bound can differ by a factor of 2.5. The current Bitcoin

network has a computational power of 248,116 THash/sec. There is still room for growth. Since on average the network computational power doubles every two months, our conjecture is that the network will saturate in about half a year, given that the Bitcoin price and mining hardware efficiency stay still.

## 2.4 Summary

To summarize our work, we characterized the evolution of Bitcoin miners' productivity, computation power and transaction activity by analyzing the full blockchain in the Bitcoin network. We showed how the largest mining pool in Bitcoin grows over time and how computational power is distributed among its miners. We also built a simple economic model that explains the evolution of mining hardware and predicts the limit of the computational race between miners.

## Chapter 3

# A Fast Multi-Server, Multi-Block Private Information Retrieval Protocol

For this work, I served as principal investigator, along with Trishank Kuppusamy, Yong Liu, and Justin Cappos. My research contributions to this project included the design of both binary and finite-field based  $k$ -safe matrix construction schemes, and the development of the two multi-block PIR schemes. I also conducted the performance evaluation. Trishank helped to set up the experiment and obtained the benchmark results used in the evaluation. Professors Liu and Cappos contributed insights and suggested research directions as the project evolved.

In many network applications, a client would like to retrieve information from a server without revealing exactly what it wants to download. For example, an inventor may want to query a patent database without revealing which patents she wants to retrieve [21], a trader may want to get specific stock quotes without

revealing the investment she may have or want to make [22], or a client may want to download a security update without revealing what unpatched vulnerability it addresses [23]. Private Information Retrieval (PIR) allows users to retrieve information from a database without revealing the query to anyone – including the database server itself. Computational PIR [24–27], which leverages, a single database server, is known to be impractical [28]. However, recent results have shown that information-theoretic PIR [3], which uses multiple mirrors containing copies of the server data, can, in some cases, perform in a similar manner to a non-PIR systems [23, 54].

In classic multi-server information-theoretic PIR schemes, each mirror keeps a replicated copy of the database. In order to retrieve one data block, a client needs to send multiple random binary coefficients as requests to different mirrors, and decode from those received mixed blocks [3]. While binary calculation can be fast on mirrors, privacy comes at the price of greatly increased communication overhead. More recently, Henry et al. [29] showed that if a client requests multiple data blocks, it is possible to *reuse randomly mixed data blocks across multiple requests*. Although this reduces communication overhead while maintaining the same level of privacy, the use of an error-correcting code results in a constant communication overhead, that cannot be further reduced. Moreover, the above work we discussed leverage computationally-expensive encoding and decoding operations that substantially decrease the throughput of the resulting systems.

This work includes three major contributions. First, we present the first matrix-based information-theoretic PIR scheme that combines multiple block requests with the use of fast XOR operations instead of more computationally expensive operations. Using fast XOR operations allows us to reuse the same high performance PIR mirror infrastructure that has been shown to have similar good-

put to FTP on realistic datasets and deployment environments [23]. However, using multiple block requests has added benefits. By leveraging multiple block requests, our proposed scheme can improve performance and significantly reduce communication overhead.

Secondly, we develop a finite-field based PIR scheme to further reduce the communication overhead. Our basic finite-field scheme assumes there is no Byzantine mirror in the system and focuses on minimizing the communication overhead. By sacrificing the mirror computation load and goodput, the finite-field scheme achieves even lower communication overhead than the binary matrix scheme (by a factor of up to 14 in extreme cases). We then augment the basic finite-field scheme with a simple yet robust detect-retransmit mechanism to handle Byzantine failures, following the design philosophy of *make the common case fast and make the uncommon case correct*.

Thirdly, we build prototypes to validate the benefits of both these schemes in practical environments. Through extensive experiments, we show that both binary and finite-field multi-block PIR schemes have much lower communication overhead than the classic version. While maintaining a high-level of privacy, the mirror goodput is even comparable to systems offering no privacy protection, such as HTTP and FTP.

## 3.1 Architecture and Threat Model of Private Information Retrieval

### 3.1.1 Architecture

A PIR system typically has three components.

- **Vendor:** A vendor produces a database containing blocks of data desired by clients. This database is public and can be read by any party. The vendor builds a manifest for this database that describes the secure hashes of each block. The vendor is also responsible for maintaining a list of correctly-operating mirrors.
- **Client:** A client requests one or more blocks of data from the database. In order to retrieve data, a client first contacts the vendor to get the list of mirrors and the manifest. The client then makes requests to the mirrors to retrieve content.
- **Mirror:** A mirror obtains a copy of the database and provides blocks of data to the client. When a mirror gets a request from a client, it generates a response according to the request (described below) and sends back a signed response to the client.

### 3.1.2 Threat model

To understand the scope of the issues that our work will address, we use the following threat model which comes from prior work [3, 23].

- The vendor is trusted to produce a valid database that the client wishes to retrieve. The vendor is largely trusted but wishes to reduce its bandwidth consumption by offloading client download requests to mirrors.
- Non-malicious mirrors may fail at any time, and will not respond to client queries.
- A malicious party may operate one or more mirrors. Therefore, the adversary may see all communications and decode any encrypted messages for

their mirrors. Furthermore, these mirrors may share or publicize any information they receive. However, for most of this chapter (until Section 3.5), we assume a malicious mirror is honest-but-curious. In other words, it will not corrupt or modify content, but it may collude with others and reveal information that could pose a threat.

- In Section 3.5, we relax the prior assumption and assume that a mirror may act in a Byzantine manner, including modifying content.

To retrieve information privately from potential honest-but-curious mirrors, a client sends multiple requests to multiple mirrors. Some requests are for randomly mixed blocks to “confuse” the malicious mirrors, and the rest of the requests are carefully crafted so that, after collecting all the responses, the client is able to decode and get all the data blocks she wants. In the scenario described here, we assume all the working mirrors generate correct responses and the client receives the responses correctly.

A PIR scheme protects client queries against collusion by malicious mirrors. We measure the privacy of a PIR scheme using a threshold model called  $k$ -safe PIR.

**Definition 1.** *A client information retrieval scheme is  $k$ -safe if it does not reveal any information about the client’s query as long as the number of malicious mirrors is no greater than  $k$ .*

## 3.2 Single-Block PIR Scheme

In this section, we formally develop our multi-block PIR scheme, which retrieves multiple data blocks with low communication overhead. We focus on PIR



schemes that use only the binary XOR operation. We show that Chor-PIR, a  $k$ -safe PIR scheme for downloading a single block, can be extended to download multiple blocks with significantly reduced communication overhead, while maintaining the same information retrieval privacy against  $k$  malicious mirrors.

### 3.2.1 Notations

For clarity of presentation, we will use the following notations throughout the paper:

- $D$  is the database containing  $N$  equal-sized data blocks,  $D = [B_1 \ B_2 \ \dots \ B_N]$ , with each data block being a bit string with  $S$  bits.
- $e_l = [0 \ 0 \ \dots \ 1 \ \dots \ 0]^T$  is the position vector with  $|e_l| = N$  and only the  $l$ -th bit is one.
- $C_i$  is the block encoding the coefficient vector to be sent to the  $i$ -th mirror.  $C_i$  is a column vector with dimension  $N$ .
- $E_i$  is the linearly encoded data block sent back by the  $i$ -th mirror to the client.  $E_i = D \times C_i$ , where addition and multiplication are defined in a finite field.
- $k$  is the number of colluders. The client's query will not be revealed as long as no more than  $k$  mirrors collude.

### 3.2.2 Single-Block PIR

PIR of a single block has been studied extensively in previous works [21, 25, 26, 55, 56]. Using the Chor-PIR protocol, which achieves privacy with up to  $k$

colluding mirrors, the client has to first download  $k$  randomly mixed blocks from  $k$  different mirrors, and then download the desired block (mixed with the  $k$  randomly mixed blocks) from the mirror  $k + 1$ . More precisely, to download block  $l$ :

1. The client generates  $k$  random bit strings  $\{\xi_i, 1 \leq i \leq k\}$ , with  $|\xi_i| = N$ , and sends  $\xi_i$  to mirror  $i$  as the block encoding coefficients:

$$C_i = \{\xi_{ij}, 1 \leq j \leq N\}, 1 \leq i \leq k.$$

2. Mirror  $i$  returns to the client the encoded block:

$$E_i = D \times C_i \triangleq \bigoplus_{j=1}^N \xi_{ij} B_j,$$

where the string operation is bit-wise, with XOR  $\oplus$  addition and binary multiplication. Equivalently, the mirror works in the two-element finite field  $GF(2)$ .

3. The client sends to the mirror  $k + 1$  the encoding coefficient vector:

$$C_{k+1} = \bigoplus_{i=1}^k \xi_i \oplus e_l.$$

4. Mirror  $k + 1$  returns the encoded block:

$$E_{k+1} = D \times (\bigoplus_{i=1}^k \xi_i \oplus e_l).$$

5. Finally, the client decodes the data block  $l$ :

$$\begin{aligned}
 \bigoplus_{i=1}^{k+1} E_i &= D\xi_1 \oplus \cdots \oplus D\xi_k \oplus D(\bigoplus_{i=1}^k \xi_i \oplus e_l) \\
 &= D \times (\xi_1 \oplus \cdots \oplus \xi_k \oplus (\bigoplus_{i=1}^k \xi_i \oplus e_l)) \\
 &= D \times e_l = B_l.
 \end{aligned}$$

Figure 3.1 shows a simple example of using Chor-PIR to privately retrieve a data block from two mirrors with  $k = 1$  and  $l = 4$ .

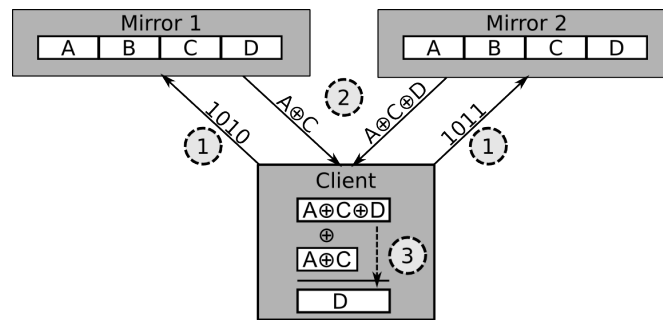


Figure 3.1: Diagram of a Chor-PIR scheme to retrieve one data block

Due to the random bit strings  $\{\xi_i, 1 \leq i \leq k\}$ , the privacy of the client's query is preserved unless  $k + 1$  mirrors collude. The privacy comes at the price of downloading  $k + 1$  mixed data blocks to decode one original data block. If we assume that each mirror is malicious independently with probability  $\hat{p}$ , then the probability that the data query will never be revealed is:

$$Privacy\_Chor(\hat{p}, k) = 1 - \hat{p}^{(k+1)}. \quad (3.1)$$

The communication overhead, or the number of extra blocks needed to retrieve one data block, is:

$$Overhead\_Chor(\hat{p}, k) = k. \quad (3.2)$$

According to Equation (3.1), given a malicious mirror probability  $\hat{p}$ , one has to choose a large  $k$  to achieve a high level of privacy, leading to high communication overhead.

### 3.3 Binary Multi-Block PIR (BMB-PIR)

When a client needs to download multiple blocks privately, one naïve way is to download each block independently using Chor-PIR. Then each block incurs a communication overhead of  $k$ . To reduce the communication overhead, one can try to reuse the randomly mixed data blocks from the first  $k$  mirrors, and then download another data block (say  $t$ ) from mirror  $k + 2$  by sending a bit string  $C_{k+2} = \oplus_{i=1}^k \xi_i \oplus e_t$ . Unfortunately, if mirror  $k + 1$  and mirror  $k + 2$  collude, then they only need to add the encoding coefficient vectors from the client,

$$C_{k+1} \oplus C_{k+2} = (\oplus_{i=1}^k \xi_i \oplus e_l) \oplus (\oplus_{i=1}^k \xi_i \oplus e_t) = e_l \oplus e_t,$$

to eliminate all the random strings, and discover that the client wants to download blocks  $l$  and  $t$ . A more refined download scheme is needed to reuse the randomly mixed blocks and reduce the communication overhead.

#### 3.3.1 Multi-Block Download Scheme

We propose a scheme for multi-block PIR based on the binary XOR operation to achieve privacy when up to  $k$  mirrors collude.

**Definition 2.** *k-Safe Binary Matrix:* we call a binary matrix  $R$  *k-safe* if any  $k$  columns of  $R$  are linearly independent under XOR addition.

If we can generate a  $k$ -safe binary matrix of the form:

$$R_{n \times m}^{(k)} = \left[ \begin{array}{cccc|ccc} 1 & * & \dots & * & v_{1,n+1} & \dots & v_{1,m} \\ 0 & 1 & \ddots & \vdots & v_{2,n+1} & \dots & v_{2,m} \\ \vdots & \ddots & \ddots & * & \vdots & \dots & \vdots \\ 0 & \dots & 0 & 1 & v_{n,n+1} & \dots & v_{n,m} \end{array} \right]_{n \times m} \quad (3.3)$$

$$= [RL_{n \times n}^{(k)} | RR_{n \times (m-n)}^{(k)}], \quad (3.4)$$

then we can download  $m - n$  data blocks by reusing  $n$  randomly mixed data blocks and achieve privacy when up to  $k$  mirrors collude. Note that  $RL_{n \times n}^{(k)}$  is an upper diagonal matrix. Here is the client downloading strategy:

1. The client generates an  $N \times n$  random binary matrix:

$$F \triangleq [\xi_1, \xi_2, \dots, \xi_n],$$

where  $\xi_i$  is the  $i$ -th column, corresponding to a random binary string with length  $N$ .

2. The client sends mirror  $i$  the encoding vector:

$$C_i = F \times RL_{n \times n}^{(k)}(i), \quad 1 \leq i \leq n,$$

where  $RL_{n \times n}^{(k)}(i)$  is the  $i$ -th column of matrix  $RL_{n \times n}^{(k)}$ .

3. Mirror  $i$  sends back to the client an encoded block:

$$E_i = D \times C_i.$$

4. The client decodes the  $n$  randomly mixed blocks by computing  $D\xi_i$  as:

$$D\xi_i = [E_1, \dots, E_n] \times RL^{-1}(i), \quad 1 \leq i \leq n,$$

where  $RL^{-1}$  is the inverse matrix of  $RL_{n \times n}^{(k)}$ , and  $RL^{-1}(i)$  is its  $i$ -th column.

5. The client sends to mirror  $n + j$  the encoding vector:

$$C_{n+j} = F \times RR_{n \times (m-n)}^{(k)}(j) \oplus e_{p_j}, \quad 1 \leq j \leq m - n,$$

where  $p_j$  is the index of the  $j$ -th data block the client wants to download.

6. Mirror  $n + j$  returns to the client the encoded block:

$$\begin{aligned} E_{n+j} &= D \times \left( F \times RR_{n \times (m-n)}^{(k)}(j) \oplus e_{p_j} \right) \\ &= [D\xi_1, \dots, D\xi_n] \times RR_{n \times (m-n)}^{(k)}(j) \oplus De_{p_j} \\ &= \bigoplus_{i=1}^n v_{i,n+j} D\xi_i \oplus B_{p_j}. \end{aligned}$$

7. The client decodes block  $p_j$  as:

$$B_{p_j} = E_{n+j} \bigoplus_{i=1}^n v_{i,n+j} D\xi_i, \quad 1 \leq j \leq m - n.$$

Since  $R_{n \times m}^{(k)}$  is  $k$ -safe, then by definition, any subset of up to  $k$  mirrors cannot cancel out the random strings  $\{\xi_1, \xi_2, \dots, \xi_n\}$  by manipulating their received coding strings  $C_i$  from the client. So the client can download  $m - n$  data blocks by first downloading  $n$  randomly mixed data blocks, and achieve privacy with up to  $k$  colluding mirrors. If the client wants to download more than  $m - n$  blocks, it has to repeat the process.

### 3.3.2 Construction of $k$ -safe Binary Matrix

The dimensions of a  $k$ -safe binary matrix determine the number of randomly mixed blocks the client needs to download and the number of data blocks it can thereafter retrieve. Now the challenge is to construct  $R_{n \times m}^{(k)}$  with small download overhead  $\frac{n}{m-n}$ . The single-block Chor-PIR is a special case of the multi-block scheme with

$$R_{k \times (k+1)}^{(k)} = \begin{bmatrix} 1 & 0 & \dots & 0 & 1 \\ 0 & 1 & \dots & 0 & 1 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & \dots & \dots & 1 & 1 \end{bmatrix}_{k \times (k+1)},$$

which we call the basis  $k$ -safe matrix. It is easy to check if any of the  $k$  columns are linearly independent. Unfortunately, the communication overhead introduced by the basis  $k$ -safe matrix is  $k$ , which is too high. Now we propose an iterative algorithm to grow the basis  $k$ -safe matrix to reduce the communication overhead. By duplicating matrix  $R_{k \times (k+1)}^{(k)}$  from left to right, adding a  $\lfloor k/2 \rfloor$ -safe matrix to the bottom right and filling zeros in the bottom left, we have:

$$R_{d(k, 2k+2) \times (2k+2)}^{(k)} \triangleq \begin{bmatrix} R_{k \times (k+1)}^{(k)} & R_{k \times (k+1)}^{(k)} \\ 0 & R_{d(\lfloor k/2 \rfloor, k+1) \times (k+1)}^{(\lfloor k/2 \rfloor)} \end{bmatrix}, \quad (3.5)$$

where  $R_{d(\lfloor k/2 \rfloor, k+1) \times (k+1)}^{(\lfloor k/2 \rfloor)}$  is a  $\lfloor k/2 \rfloor$ -safe matrix with  $k+1$  columns. The number of rows is determined by a function  $d(k, m)$ . For the basis  $k$ -safe matrix, we have  $d(k, k+1) = k$ .

**Proposition 3.3.1.** *The binary matrix  $R_{d(k, 2k+2) \times (2k+2)}^{(k)}$  constructed in (3.5) is  $k$ -safe.*

*Proof.* Since any  $k$  columns of  $R_{k \times (k+1)}^{(k)}$  are linearly independent, if we apply Gaussian column elimination with XOR addition to any  $k$  columns of the expanded matrix  $R_{d(k,2k+2) \times (2k+2)}^{(k)}$ , the only way to cancel out the upper portion of  $k$  columns of  $R_{d(k,2k+2) \times (2k+2)}^{(k)}$  is to take exactly the same  $\lfloor k/2 \rfloor$  vectors from the left and right halves. However, since the bottom-right matrix is  $\lfloor k/2 \rfloor$ -safe, the bottom portion of those  $k$  columns will never be canceled out. So  $R_{d(k,2k+2) \times (2k+2)}^{(k)}$  is indeed  $k$ -safe.  $\square$

According to the expansion process, we have

$$d(k, 2k + 2) = d(k, k + 1) + d(\lfloor k/2 \rfloor, k + 1).$$

By switching columns, we can convert  $R_{d(k,2k+2) \times (2k+2)}^{(k)}$  into a  $k$ -safe matrix of the form in (3.3), with  $m = 2k + 2$ ,  $n = d(k, 2k + 2)$ . The left upper triangular property ensures that the left side square matrix is invertible.

More generally, given a  $k$ -safe matrix  $R_{d(k,m) \times m}^{(k)}$  with  $m$  columns and  $d(k, m)$  rows, one can expand it into a  $k$ -safe matrix with  $2m$  columns using a similar process:

$$R_{d(k,2m) \times 2m}^{(k)} = \begin{bmatrix} R_{d(k,m) \times m}^{(k)} & R_{d(k,m) \times m}^{(k)} \\ 0 & R_{d(\lfloor k/2 \rfloor, m) \times m}^{(\lfloor k/2 \rfloor)} \end{bmatrix},$$

with  $d(k, 2m) = d(k, m) + d(\lfloor k/2 \rfloor, m)$ . The proof is a straightforward extension of Proposition 3.3.1.



For example, if we set  $k = 4$ , then the 4-safe matrix (without expansion) is:

$$R = \begin{bmatrix} 1 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 & 1 \end{bmatrix}_{4 \times 5} .$$

After one expansion, we see that:

$$R = \begin{bmatrix} 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 \end{bmatrix}_{7 \times 10} .$$

The overhead and privacy of Chor-PIR are analyzed in Equation (3.1) and (3.2) respectively. For BMB-PIR based on a  $k$ -safe binary matrix  $R_{d(k,m) \times m}^{(k)}$ , we use  $m$  mirrors to download  $m - d(k, m)$  data blocks in a  $k$ -safe manner. If each mirror is malicious with probability  $\hat{p}$ , then the query privacy is preserved if no more than  $k$  mirrors are malicious. Therefore, the privacy of BMB-PIR can be calculated as:

$$Privacy\_BMB(k, m, \hat{p}) = \sum_{i=0}^k \binom{m}{i} \hat{p}^i (1 - \hat{p})^{m-i} . \quad (3.6)$$

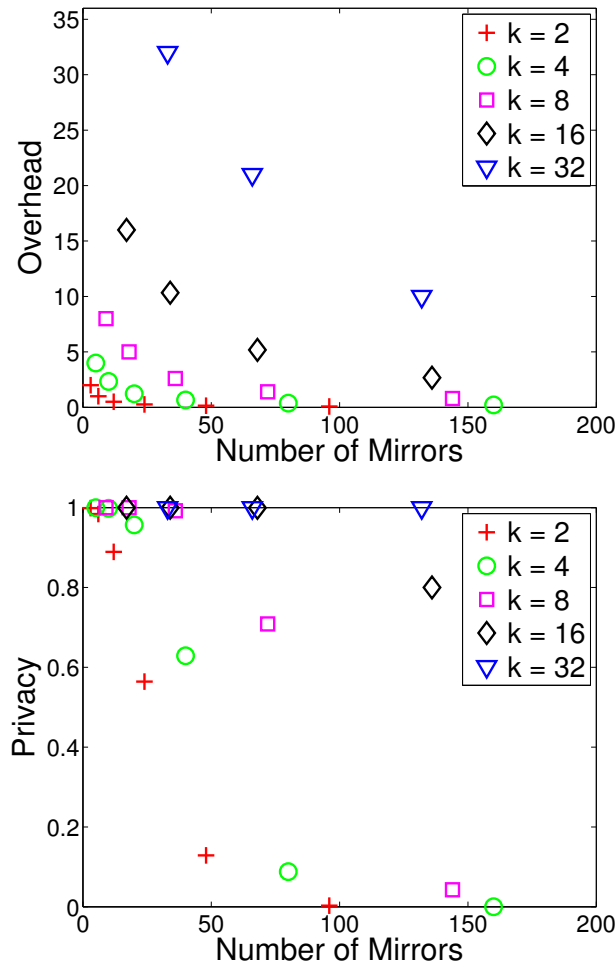


Figure 3.2: In BMB-PIR, as the number of employed mirrors increases, the communication overhead decreases. However, the privacy level also decreases.

And the communication overhead is:

$$Overhead\_BMB(k, m) = \frac{d(k, m)}{m - d(k, m)}. \quad (3.7)$$

For the 4-safe example considered in this section, we can see that the overhead is 4 without expansion. After one round of expansion, the overhead becomes 2.3333, a reduction of nearly 40%.

### 3.4 Multi-Block PIR over a Finite Field (FMB-PIR)

For a given dimension  $n$ , there are only a limited number of linearly independent binary vectors. As a result, a  $k$ -safe binary matrix will have large dimensions, leading to high communication overhead. To further reduce this overhead, we can work with a matrix in which each entry takes a value from a larger finite field. The binary multi-block PIR we studied in the previous section is a special case of a finite field with two elements. A finite field with higher cardinality enables us to find more  $k$ -safe vectors without expanding the dimensions of the matrix.

**Definition 3.**  *$k$ -Safe Matrix in Finite Field:* We say that a matrix  $R$  in a finite field  $\mathcal{F}$  is  $k$ -safe if any  $k$  columns of  $R$  are linearly independent under the addition and multiplication operations of  $\mathcal{F}$ .

#### 3.4.1 Construction of $k$ -safe Binary Matrix

It is convenient to find  $k$ -safe matrices in a finite field  $\mathcal{F}$  using the Vandermonde matrix of the form:

$$V_{k \times m} = \begin{bmatrix} 1 & 1 & \dots & 1 \\ x_1^1 & x_2^1 & \dots & x_m^1 \\ \vdots & \vdots & \vdots & \vdots \\ x_1^{(k-1)} & x_2^{(k-1)} & \dots & x_m^{(k-1)} \end{bmatrix}, \quad (3.8)$$

where  $m \geq k$  and  $x_i \in \mathcal{F}$ ,  $1 \leq i \leq m$ . It is well-known that the determinant of a square Vandermonde matrix ( $m = k$ ) is:

$$|V_{k \times k}| = \prod_{1 \leq i < j \leq k} (x_j - x_i). \quad (3.9)$$

If all  $x_j$  in  $V_{k \times m}$  are distinct, then according to (3.9), any  $k \times k$  sub-matrix of  $V_{k \times m}$  has a non-zero determinant; i.e., any  $k$  columns of  $V_{k \times m}$  are linearly independent. Therefore,  $V_{k \times m}$  as constructed in (3.8) is a  $k$ -safe matrix as long as all  $x_j$  are distinct. We partition the  $k$ -safe matrix into two submatrices:

$$V_{k \times m} = [VL_{k \times k} | VR_{k \times (m-k)}].$$

Vandermonde matrices have been widely used in constructing erasure codes (such as the Reed-Solomon error-correcting code) to detect and correct errors and erasures. They are also used in Shamir's secret sharing in cryptography. PIR protocols based on Shamir's secret sharing were developed in [29, 57]. As discussed, those protocols are designed to be robust against Byzantine mirrors. However, we use finite field arithmetic here to reduce the communication overhead of multi-block PIR.

### 3.4.2 Multi-Block Download Scheme

Now we describe our multi-block PIR over general finite fields. For clarity of presentation, we use  $GF(2^8)$  as the underlying finite field (but the process holds true over other finite fields). With  $GF(2^8)$ , each data block  $B_i$  in database  $D$  is divided into 8-bit chunks. The pseudocode of the FMB-PIR protocol is presented in Algorithm 1. The input parameters are:  $N$ , the total number of data blocks in

---

**Algorithm 1:** The FMB-PIR protocol
 

---

**Input:**  $N, k, m$ , and  $\{e_{p_1}, e_{p_2}, \dots, e_{p_{(m-k)}}\}$   
**Output:**  $\{B_{p_1}, B_{p_2}, \dots, B_{p_{(m-k)}}\}$

- 1 Client Initialization:
- 2 create Vandermonde matrix:  $V_{k \times m}$ ;
- 3 create random matrix in  $GF(2^8)$ :  $F_{N \times k} = [\xi_1, \xi_2, \dots, \xi_k]$ , where the  $i$ -th column  $\xi_i = \{\xi_{ij} \in GF(2^8), 1 \leq j \leq N\}$
- 4 calculate request coefficient matrix:  $W_{N \times m} = F_{N \times k} \times V_{k \times m}$
- 5 **for**  $i = 1$  **to**  $k$  **do**
- 6     client sends request vector  $C_i = W_{N \times m}(i)$  to mirror  $i$
- 7     client gets encoded data block  $E_i = D \times C_i$  from mirror  $i$
- 8 **end**
- 9 client decodes  $D \times F = [E_1, \dots, E_k] \times VL_{k \times k}^{-1}$
- 10 **for**  $j = 1$  **to**  $m - k$  **do**
- 11     client sends  $C_{k+j} = W_{N \times m}(k + j) + e_{p_j}$  to mirror  $k + j$ ;
- 12     client gets encoded data block  $E_{k+j} = D \times (F \times V_{k \times m}(k + j) + e_{p_j})$  from mirror  $k + j$
- 13     client decodes block  $p_j$  as  $B_{p_j} = E_{k+j} - [D\xi_1, \dots, D\xi_k] \times V_{k \times m}(k + j)$
- 14 **end**

---

$D$ ;  $m$ , the total number of mirrors;  $k$ , the maximum number of malicious mirrors; and  $\{e_{p_i}\}$ , the position vectors corresponding to the blocks the client wants to download. The output of the algorithm is the data blocks the client wants. The client generates a  $k \times m$  Vandermonde matrix, and an  $N \times k$  random matrix, and then uses them to derive a request coefficient matrix. In the first **for** loop, the client sends coding coefficient vectors to  $k$  mirrors and downloads  $k$  encoded data blocks. Then the client uses the received data blocks to decode the  $k$  randomly mixed blocks according to  $F$ . In the second **for** loop, the client contacts an additional  $m - k$  mirrors, and retrieves one data block from each mirror using the requesting and decoding strategy given in Algorithm 1.

Since  $V_{k \times m}$  is  $k$ -safe, by definition, any subset of up to  $k$  mirrors cannot cancel out the random vectors  $\{\xi_1, \xi_2, \dots, \xi_k\}$  by manipulating their received coding strings  $C_i$  from the client. So the client can download up to  $m - k$  desired data

blocks by first downloading  $k$  randomly mixed data blocks to achieve query privacy against up to  $k$  colluding mirrors. In  $GF(2^8)$ , the maximum number of distinct elements is 256, so given  $m \leq 256$  mirrors, the largest number of data blocks that can be downloaded using the Vandermonde matrix constructed above is  $m - k$ . The following is an example of a 5-safe matrix in  $GF(2^8)$  with 10 vectors:

$$R = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 \\ 0 & 1 & 4 & 5 & 16 & 17 & 20 & 21 & 64 & 65 \\ 0 & 1 & 8 & 15 & 64 & 85 & 120 & 107 & 58 & 115 \\ 0 & 1 & 16 & 17 & 29 & 28 & 13 & 12 & 205 & 204 \end{bmatrix}_{5 \times 10} .$$

### 3.4.3 Computational Cost

With FMB-PIR, each mirror simply multiplies the database  $D_{S \times N}$  with the request vector  $C_{N \times 1}$ . This takes less than  $\frac{S}{8}N$  additions and exactly  $\frac{S}{8}N$  multiplications of  $GF(2^8)$ . On the client end, to prepare the randomly mixed blocks the client must generate the  $k$  request vectors  $C_1, \dots, C_k$ , compute an inverse matrix, and multiply the responses by the inverse matrix. The resulting number of additions is less than  $k^2N + \frac{S}{8}k^2 + k^3$ , in which  $k^3$  is the cost of matrix inversion, which will not be significant, since a reasonable  $k$  will not be very large. The number of multiplications is also less than  $k^2N + \frac{S}{8}k^2 + k^3$ . The computational cost of getting  $m - k$  data blocks is the cost of generating the  $m - k$  request vectors  $C_{k+1}, \dots, C_m$ , (less than  $k(m - k)N + (m - k)N$  additions and  $k(m - k)N$  multiplications) and the cost of decoding the data blocks ( $\frac{S}{8}(km - k^2 + m - k)$  additions and  $\frac{S}{8}(km - k^2)$  multiplications). There are in total

$(k(m-k)+m-k)N + \frac{S}{8}(km-k^2+m-k)$  additions and  $k(m-k)N + \frac{S}{8}(km-k^2)$  multiplications. Hence, we get  $k^2N + \frac{S}{8}k^2 + k^3 + (k(m-k)+m-k)N + \frac{S}{8}(km-k^2+m-k)$  additions and  $k^2N + \frac{S}{8}k^2 + k^3 + k(m-k)N + \frac{S}{8}(km-k^2)$  multiplications on the client end. In practice,  $k$  and  $m$  are small numbers (e.g. 5 and 8). Given  $k$  and  $m$ , the number of additions and multiplications is  $O(S+N)$ , where  $S$  is the size of each data block and  $N$  is the number of data blocks in the database.

For a  $k$ -safe FMB-PIR with  $m$  mirrors, the privacy level remains the same as equation (3.6), but the communication overhead becomes:

$$\text{Overhead\_FMB}(k, m) = \frac{k}{m-k}. \quad (3.10)$$

Given a probability  $\hat{p} = 0.1$  that a mirror is honest-but-curious, Tables 3.1 through 3.3 report for the Chor-PIR, BMB-PIR and FMB-PIR protocols the lowest communication overhead that each can achieve at target privacy levels  $\bar{p}$  ranging from 0.9 to 0.99999, and the corresponding best  $\langle k, m \rangle$  settings. We see that when the number of mirrors increases, BMB-PIR achieves nearly 33% overhead reduction over Chor-PIR. FMB-PIR has a significant improvement on overhead when we choose a large  $m$ . For example, at  $m \leq 64$ , the overhead is about 10 times less than Chor-PIR. Although the computational cost in  $GF(2^8)$  of basic operations such as addition and multiplication is much higher than the cost of these operations in  $GF(2)$  (as will be seen in Section 3.6), the overall performance of FMB-PIR is generally very good.

Table 3.1: Lowest overhead achieved by Chor-PIR at different privacy levels

Privacy	$\langle k, m \rangle$	Overhead
0.9	$\langle 1, 2 \rangle$	1
0.99	$\langle 1, 2 \rangle$	1
0.999	$\langle 2, 3 \rangle$	2
0.9999	$\langle 3, 4 \rangle$	3
0.99999	$\langle 4, 5 \rangle$	4

Table 3.2: Lowest overhead achieved by BMB-PIR at different privacy levels

Privacy	$m \leq 8$		$m \leq 16$		$m \leq 32$		$m \leq 64$		$m \leq 128$		$m \leq 256$	
	overhead	$\langle k, m \rangle$	overhead	$\langle k, m \rangle$	overhead	$\langle k, m \rangle$	overhead	$\langle k, m \rangle$	overhead	$\langle k, m \rangle$	overhead	$\langle k, m \rangle$
0.9	0.3333	$\langle 1, 4 \rangle$	0.3333	$\langle 1, 4 \rangle$	0.3333	$\langle 1, 4 \rangle$	0.3333	$\langle 1, 4 \rangle$	0.3333	$\langle 1, 4 \rangle$	0.3333	$\langle 1, 4 \rangle$
0.99	1	$\langle 1, 2 \rangle$	1	$\langle 1, 2 \rangle$	1	$\langle 1, 2 \rangle$	1	$\langle 1, 2 \rangle$	1	$\langle 1, 2 \rangle$	1	$\langle 1, 2 \rangle$
0.999	2	$\langle 2, 3 \rangle$	2	$\langle 2, 3 \rangle$	2	$\langle 2, 3 \rangle$	1.9091	$\langle 15, 64 \rangle$	1.9091	$\langle 15, 64 \rangle$	1.9091	$\langle 15, 64 \rangle$
0.9999	3	$\langle 3, 4 \rangle$	2.2	$\langle 7, 16 \rangle$	2.2	$\langle 7, 16 \rangle$	2.2	$\langle 7, 16 \rangle$	2.2	$\langle 7, 16 \rangle$	2.2	$\langle 7, 16 \rangle$
0.99999	4	$\langle 4, 5 \rangle$	4	$\langle 4, 5 \rangle$	3.8	$\langle 11, 24 \rangle$	3.8	$\langle 11, 24 \rangle$	3.4138	$\langle 31, 128 \rangle$	3.4138	$\langle 31, 128 \rangle$

Table 3.3: Lowest overhead achieved by FMB-PIR at different privacy levels

Privacy	$m \leq 8$		$m \leq 16$		$m \leq 32$		$m \leq 64$		$m \leq 128$		$m \leq 256$	
	overhead	$\langle k, m \rangle$	overhead	$\langle k, m \rangle$	overhead	$\langle k, m \rangle$	overhead	$\langle k, m \rangle$	overhead	$\langle k, m \rangle$	overhead	$\langle k, m \rangle$
0.9	0.3333	$\langle 2, 8 \rangle$	0.2222	$\langle 2, 11 \rangle$	0.1852	$\langle 5, 32 \rangle$	0.1668	$\langle 9, 63 \rangle$	0.1524	$\langle 16, 121 \rangle$	0.1396	$\langle 31, 253 \rangle$
0.99	0.6	$\langle 3, 8 \rangle$	0.4	$\langle 4, 14 \rangle$	0.2917	$\langle 7, 31 \rangle$	0.2308	$\langle 12, 64 \rangle$	0.1961	$\langle 20, 122 \rangle$	0.1689	$\langle 37, 256 \rangle$
0.999	1	$\langle 4, 8 \rangle$	0.6	$\langle 6, 16 \rangle$	0.3913	$\langle 9, 32 \rangle$	0.2917	$\langle 14, 62 \rangle$	0.2308	$\langle 24, 128 \rangle$	0.1907	$\langle 41, 256 \rangle$
0.9999	1.6667	$\langle 5, 8 \rangle$	0.7778	$\langle 7, 16 \rangle$	0.5	$\langle 10, 30 \rangle$	0.3478	$\langle 16, 62 \rangle$	0.2626	$\langle 26, 125 \rangle$	0.211	$\langle 44, 252 \rangle$
0.99999	2.5	$\langle 5, 7 \rangle$	1	$\langle 8, 16 \rangle$	0.6	$\langle 12, 32 \rangle$	0.4	$\langle 18, 63 \rangle$	0.2929	$\langle 29, 128 \rangle$	0.2304	$\langle 47, 251 \rangle$

### 3.5 Robustness against Byzantine Failures

The previous analysis assumes the mirrors work correctly. However, if some mirrors act in a Byzantine manner or fail, the client still can correctly retrieve the data using Algorithm 2. The idea is to repeatedly download  $k + 1$  blocks ( $k$  mixed blocks plus 1 data block) until the data block is decoded successfully. The remaining  $m - k - 1$  data blocks, will be downloaded and decoded one by one. If the client fails to decode, she simply switches to another random mirror and downloads the data block until it is successfully decoded.

For the clarity of presentation, we only show the communication overhead of FMB-PIR here. Communication overhead of BMB-PIR can be similarly derived. Assume that a mirror fails with probability  $\bar{p}$ . In order to successfully decode the



---

**Algorithm 2:** Detect and retransmit protocol
 

---

**Input:**  $k, m,$  and  $\{C_1, C_2, \dots, C_{m-k}\}$ 
**Output:**  $\{B_{p_1}, B_{p_2}, \dots, B_{p_{(m-k)}}\}$ 

```

1 Client Initialization:
2 for  $i = 1$  to  $k$  do
3   | client randomly chooses a mirror, sends request vector  $C_i$  and gets
   | encoded data block  $E_i$ 
4 end
5 client randomly chooses a mirror, sends request vector  $C_{k+j}, j = 1$  and gets
   encoded data block  $E_{k+j}$ 
6 client decodes  $B_{p_1}$ 
7 if client fails to decode  $B_{p_1}$  then
8   | go back to Step 2;
9 end
10 for  $j = 2$  to  $m - k$  do
11   | client randomly chooses a mirror, sends request vector  $C_{k+j}$  and gets
   | encoded data block  $E_{k+j}$ 
12   | client decodes  $B_{p_j}$ 
13   | if client fails to decode  $B_{p_j}$  then
14     | go back to Step 11;
15   | end
16 end

```

---

first data block, the client needs to download a number of blocks as per:

$$\begin{aligned}
 & (k+1) \times \sum_{L=1}^{\infty} L[1 - (1 - \bar{p})^{k+1}]^{(L-1)} (1 - \bar{p})^{k+1} \\
 &= \frac{k+1}{(1 - \bar{p})^{k+1}}
 \end{aligned} \tag{3.11}$$

To decode the remaining  $m - k - 1$  data blocks, the client needs to download a number of blocks as follows:

$$(m - k - 1) \times \sum_{L=1}^{\infty} L \bar{p}^{(L-1)} (1 - \bar{p}) = \frac{m - k - 1}{1 - \bar{p}} \tag{3.12}$$

Hence, the overall communication overhead under mirror Byzantine failure prob-

ability  $\bar{p}$  is:

$$\frac{k + 1 + (m - k - 1)(1 - \bar{p})^k}{(m - k)(1 - \bar{p})^{k+1}} - 1 \quad (3.13)$$

## 3.6 Performance Evaluation

To determine whether our proposed solutions are practical, we conducted a performance evaluation. In particular, we focused on several key aspects of system performance, notably the communication overhead, the mirror goodput, and client-perceived retrieval time.

### 3.6.1 Implementation

We implemented BMB-PIR and FMB-PIR in a mix of C and Python code. The majority of the functionality is written in Python, with operations requiring high efficiency operations programmed in C. The former allowed us to keep our implementation simple and easy to test, while delegating to the latter, more expensive tasks (e.g. finite field arithmetic) to more optimized code.

In BMB-PIR and FMB-PIR, the mirror computation time and data block download time can overlap, allowing for higher performance. As a result, the implementation concurrently requests the data blocks from multiple mirrors.

Both BMB-PIR and FMB-PIR generate an associated set of  $k$ -safe matrices between every  $m - n$  data block request. As a result, up to  $m - n$  data blocks can be fetched and decoded in parallel. Note that it would be possible to prepare more than one set of associated  $k$ -safe matrices ahead of time, perhaps in the background. This could be used to decode up to  $m - n$  data blocks without the usual setup time.

### 3.6.2 Setup

We tested each protocol on a 2GB database consisting of 1,024 2MB data blocks. We ran the PIR mirrors for each protocol on ten Amazon EC2 instances [58] located in the United States (one each in Virginia, California and Oregon), the European Union (two in Ireland), South America (two in São Paulo) and Asia (two in Tokyo and one in Singapore). This means that our mirrors were distributed around the world, much like existing software mirrors or Internet clients, so that we could test the protocols on realistic network conditions.

We chose each instance to be an *m1.large* machine, which meant that it had two 64-bit virtual cores with two EC2 Compute Units each, 7.5 GiB of memory and moderate I/O performance. This meant that we had sufficient CPU cores for reasonable concurrency and more than sufficient memory to keep our database in main memory. (Note that these specifications are common hardware for software mirrors.) The client was a computer with an Intel Core i7-2600 Processor (4 cores, 8M Cache, up to 3.80 GHz) and 11 GB of main memory located in the Eastern United States.

In order to balance the network requests, a new set of mirrors of the desired number were randomly sampled on every set of data block requests. However, since the total number of mirrors is limited to ten, we reused the mirrors whenever we needed more than ten of them. This helped to keep the experiments manageable and allowed us to simulate reasonably busy mirrors. Given that our implementation fetches data blocks concurrently, we believe that reusing mirrors has little impact on our results.

Unless otherwise specified, we assume the probability that a mirror will maliciously disclose information to be  $\hat{p} = 0.1$ . We set the target privacy  $\bar{p}$  to be

in  $\{0.9, 0.99, 0.999, 0.9999, 0.99999\}$  and the total number of mirrors  $m$  to be in  $\{8, 16, 32, 64, 128, 256\}$ . For each protocol of interest, we obtained a set of values for  $k$  and  $m$  that provide the lowest communication overhead as determined by equations (3.2), (3.7) and (3.10).

We analyzed the performance of the BMB-PIR and FMB-PIR protocols as compared to upPIR [23], a highly optimized implementation of the Chor-PIR protocol adapted for private yet efficient software updates. We chose upPIR for comparison because BMB-PIR is a generalization of the foundational Chor-PIR protocol, while FMB-PIR is a generalized extension of BMB-PIR.

For upPIR, we retrieved ten data blocks for each setting of  $k$  and  $m$ . For FMB-PIR, we retrieved  $m - k$  data blocks for each setting of  $k$  and  $m$ . (This means that the number of fetched data blocks varied over each setting.) Similarly, for BMB-PIR, we retrieved  $m - d(k, m)$  data blocks for each setting of  $k$  and  $m$ . For each PIR scheme and each parameter setting, we repeated the experiment three times and reported the statistics. With these statistics, we are able to explore several key aspects of performance.

### 3.6.3 Results

We found that the observed communication overhead for FMB-PIR and BMB-PIR precisely matched the expected overhead. We noticed that the wall clock time spent on network operations (such as data marshaling, and more importantly, waiting for the mirror to produce the encoded data block) dominated the time spent on local computation (such as generating  $k$ -safe matrices, and encoding or decoding a block). Of course, as the values for  $k$  and  $m$  increased, then so did the average client computing time, but the network wall clock time was always the

dominating cost. Concurrently fetching the data blocks proved to yield significant gains in the average time to fetch a data block.

We found that upPIR was more efficient than BMB-PIR in terms of communication overhead for small values of  $k$  and  $m$ , but less efficient than FMB-PIR. Moreover, the expected overhead quickly reaches the point for large  $k$  and  $m$  where upPIR is worse than BMB-PIR and FMB-PIR protocols because the communication overhead for every data block request proved to be sufficiently prohibitive. An important finding is that, in many cases, a computationally efficient PIR protocol will not sufficiently compensate for its communication overhead.

### Communication Overhead

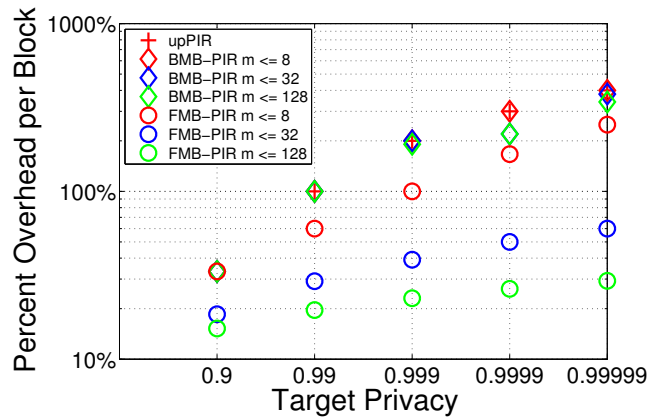


Figure 3.3: Communication overhead

For every setting of  $\bar{p}$  and  $m$  over each protocol, we computed the  $k$  and  $m$  that yielded the target privacy closest to  $\bar{p}$  with the lowest communication overhead. The results are shown in Tables 3.1 through 3.3. For example, if we choose  $\bar{p} = 0.9999$  and  $m \leq 32$ , then  $\langle k = 3, m = 4 \rangle$  gives the lowest overhead of 3 for upPIR, whereas  $\langle k = 7, m = 16 \rangle$  gives the lowest overhead of 2.2 for BMB-PIR, and  $\langle k = 10, m = 30 \rangle$  gives the lowest overhead of 0.5 for FMB-PIR.

So while BMB-PIR's savings over upPIR are significant, FMB-PIR is far superior to either.

Figure 3.3 shows the measured percentage of overhead (the percentage of extra blocks downloaded for every data block) given target privacy for our protocols of interest. As the target privacy increases, upPIR has to employ more mirrors. Consequently, its overhead increases multiplicatively. upPIR only downloads one data block at a time. Meanwhile, both BMB-PIR and FMB-PIR can concurrently download multiple data blocks by reusing the common set of randomly mixed preparation blocks. At each target privacy level, as the number of mirrors increases, BMB-PIR and FMB-PIR are able to increase download concurrency and decrease overhead. Even with 0.99999 privacy, the overhead ranges from 60% to 23% when at least 32 mirrors can be used.

## Mirror Goodput

Table 3.4: BMB/FMB PIR mirror goodput (Mbps) at different privacy levels

Privacy	$m \leq 8$		$m \leq 16$		$m \leq 32$		$m \leq 64$		$m \leq 128$		$m \leq 256$	
	BMB	FMB	BMB	FMB	BMB	FMB	BMB	FMB	BMB	FMB	BMB	FMB
0.9	9.32	2.91	9.32	3.17	9.32	3.27	9.32	3.32	9.32	3.36	9.32	3.40
0.99	6.22	2.42	6.22	2.77	6.22	3.00	6.22	3.15	6.22	3.24	6.22	3.32
0.999	4.14	1.94	4.14	2.42	4.14	2.79	4.27	3.00	4.27	3.15	4.17	3.26
0.9999	3.11	1.45	3.88	2.18	3.88	2.59	3.88	2.88	3.88	3.07	3.88	3.20
0.99999	2.49	1.11	2.49	1.94	2.59	2.42	2.59	2.77	2.82	3.00	2.82	3.15

We then measured the goodput, (i.e., the amount of useful data a mirror is able to produce in a second). Goodput is an important metric in evaluating a PIR protocol because what matters is not simply how much data a mirror is able to produce in a second, but rather how much of it is actually valuable to clients. For example, if a mirror is able to produce 5 blocks in a second, of which 4 are mixed blocks and 1 is a data block for a client, then the goodput must factor in the additional time to produce the additional 4 mixed blocks in the time to produce 1

data block. Therefore, the mirror goodput is 1 out of 5 blocks per second in this case.

Given the same number of mirrors and a target privacy level, Table 3.4 compares the mirror goodput with BMB-PIR and FMB-PIR. With only a few mirrors or a low target privacy level, BMB-PIR can produce higher goodput than FMB-PIR. In both cases, the lower communication overhead of FMB-PIR was insufficient to win over the faster block production of BMB-PIR. However, as the number of mirrors or targeted privacy level increase, we find that the performance gap between these two will decrease once a certain threshold is reached. FMB-PIR starts to outperform BMB-PIR. As the number of mirrors increases, the goodput of FMB-PIR will increase more rapidly than BMB-PIR.

### Block Retrieval Time

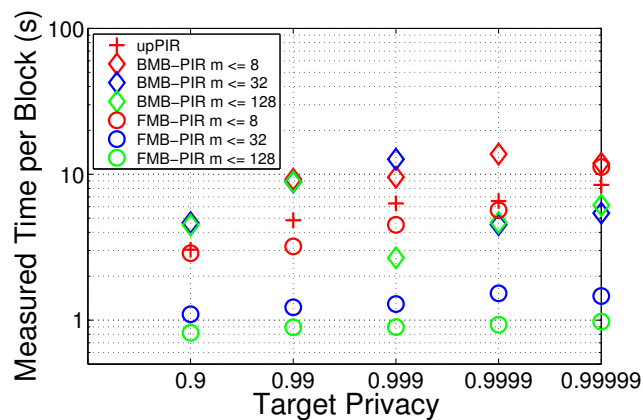


Figure 3.4: Data block retrieval time

Figure 3.4 displays the average time measured in seconds to retrieve a data block given a target privacy. For both FMB-PIR and BMB-PIR, it is the total time to concurrently fetch and decode  $m - n$  data blocks (including the production

of  $k$ -safe matrices and  $n$  preparation blocks) averaged over the total number of fetched data blocks.

FMB-PIR shows a substantial reduction in download time compared to both upPIR and BMB-PIR, especially as the number of mirrors or target privacy increases. BMB-PIR is faster in mirror operations (like upPIR), but has higher overhead; consequently, it has better performance when the number of mirrors is small (and thus the mirror computation time is a more important consideration). With FMB-PIR, the reduction in communication overhead when 32 or more mirrors are available eventually outweighs the mirror's penalty of finite field arithmetic. As a result, even with target of 0.99999 privacy, FMB-PIR with 64 or more mirrors available can retrieve a block a second (a goodput of about 2MB per second).

### Future Optimizations

We could further optimize our algorithms to improve their performance. In the current BMB-PIR implementation, an associated  $k$ -safe matrix is lazily generated for each group of  $m - n$  data block requests. This means that only up to  $m - n$  data blocks are concurrently fetched. Due to the low computation overhead on mirrors, we can concurrently download multiple groups of data block requests, with each group consisting of  $m - n$  requests and associated with a  $k$ -safe matrix. This will significantly reduce the block retrieval time for BMB-PIR, and equalize it with the current implementation of FMB-PIR. The current implementation of FMB-PIR is able to download more data blocks at once with a  $k$ -safe matrix simply because it has a much lower overhead. BMB-PIR performance can be further improved by concurrently generating  $k$ -safe matrices in the background while the client is busy fetching data blocks; this would amortize the computational cost so that it



becomes insignificant compared to the communication cost.

### 3.7 Summary

In this work, we described a fast multi-block PIR scheme that is capable of efficiently and privately downloading multiple data blocks. Specifically, we first showed that Chor's PIR scheme can be extended to download multiple data blocks at a time by a recursive construction to produce larger  $k$ -safe matrices from smaller ones. This reduces the communication overhead of multi-block PIR retrieval and has excellent mirror goodput. We also demonstrated that, by working in a finite field, a multi-block PIR scheme can achieve significantly reduced overhead, albeit at the cost of mirror goodput. Experiments over real world Internet hosts demonstrate that multi-block PIR schemes are computationally fast while significantly outperforming existing PIR schemes in multiple ways. As a result, these protocols can be used in practice to retrieve both small files, such as security updates, and large files, such as disk images, privately and efficiently.

## Chapter 4

### CacheCash: A

# Cryptocurrency-based Decentralized Content Delivery Service

This chapter describes work done as part of a joint research initiative conducted with Ghada Almashaqbeh, Allison Bishop Lewko, and Justin Cappos. Discussion in this chapter is limited to the parts of the work I was most involved with: the design of a data service protocol and an evaluation of the system's performance. Ghada and I worked independently of each other in formulating CacheCash. She developed the threat model, and conducted the security, and economic analyses. We collaborated in the design of the data service protocol. Professors Lewko and Cappos provided analysis, feedback, and guidance on our efforts.

Online content delivery has witnessed a large growth in the last decade and is expected to grow at a rate of 40% - 45% per year [59]. This content is generated by various centralized content providers (e.g. Netflix, Amazon, etc.) and by end users through media distribution applications and social networks (e.g. Youtube,

Twitter, etc.). Content providers are challenged to handle this huge workload while balancing service quality and cost concerns.

Content delivery networks (CDNs) have emerged as an attractive solution to distribute the workload [35–37]. The core idea is to replicate the media content and disburse it among geographically distributed servers (or caches), which serve clients on behalf of the original content providers. Content providers may construct their own CDN or buy the service from a third party, such as Akamai [38]. Unfortunately, infrastructure-based CDNs are considered expensive due to deployment and continuous maintenance costs. Due to this expense, peer-assisted CDNs, inspired by peer-to-peer (P2P) networks, have evolved as a low overhead and decentralized solution. These peer-assisted CDNs rely on exploiting the end users' bandwidth to distribute data [39, 40]. However, such a network requires allowing any user to be a part of it, which raises several challenging questions: What motivates peers to use their bandwidth to serve others? Can we trust any party to distribute the correct content? And is there a guarantee that these parties will follow the protocol?

In this work, we propose a system called CacheCash that combines the best features of CDNs and Bitcoin. CacheCash is a cryptocurrency-based storage/bandwidth market where users can rent their storage and sell their extra bandwidth to content providers. It adopts the layered structure from CDNs in which content providers own the material, but delegate the task of serving clients to caches, which distribute replications of this content. Like a P2P network, CacheCash allows anyone to join the system as either a content provider or as a cache. Content providers pay caches to serve their clients. Hence, CacheCash enables the building of dynamic CDNs with lower overhead than infrastructure-based networks, but in a more organized way than in P2P networks. To ensure fair payments and account-

ability, CacheCash provides a publicly-verifiable service that uses a blockchain as a trusted log to make cheating detectable, and thus, unprofitable.

In reconfiguring these existing technologies, the CacheCash system is able to make two novel claims. The first is that it supports a realistic distributed environment that can offer both efficient and secure service without the need for a trusted authority/party. And the second is that it can secure the system, in part, by making cheating much less profitable. Content providers achieve a high degree of efficiency by batching responses to clients' requests, and by offloading, as one response, all the needed information to proceed in any service session. Furthermore, CacheCash utilizes a customized probabilistic micropayment scheme that is able to reduce the number of transactions that need to be logged on the blockchain. This scheme features a modified version of the Bitcoin protocol that supports the additional logic and transaction types needed to handle payments, verify the correct service, and detect cheating. All transactions are made in the basic currency unit, Cachecoin, which can be exchanged with any other altcoin or fiat currency.

CacheCash is designed to work well in practice, addressing both security and performance concerns. Threats that may result from deviant behaviors of selfish participants are neutralized either cryptographically or economically. The former is achieved by employing various cryptographic primitives/protocols to ensure confidentiality, integrity, and unforgeability. The latter is achieved by introducing a game theoretic model of the economic aspects of the system, which operates under the assumption of rational participants. Finally, benchmarks of CacheCash demonstrate its efficiency in terms of computational and bandwidth overhead, and show that it can easily scale to handle large scale CDN dissemination, such as distributing Netflix content.

## 4.1 Related Work

In this section we reviewed previous work that are related to two fundamentals of CacheCash: peer-assisted CDNs and cryptocurrencies.

Using monetary incentives to attract participants [39, 41, 60, 61] were initially proposed to overcome the free-riders problem of people reaping the benefits of the CDN without contributing. One early system, KARMA [60], tracks payments in the form of scores that increase whenever a peer consumes resources to serve other clients, and decrease once she receives a service. Floodgate [61] uses micropayments to pay for delivered service, making the content provider the trusted party who tracks payments. Dandelion [39], on the other hand, is a hybrid incentive-based system that employs both monetary and bandwidth exchange incentives. Hinent [41] provides a more flexible design by allowing peers to set price agreements with clients, and thus better prioritize service.

The main drawback of the aforementioned proposals is that they assume the existence of a centralized trusted party to track these payments and resolve disputes. CacheCash does not rely on any central authority to issue payments or monitor the free-rider issues in its peer-based network. To secure the payments, CacheCash applies a cryptocurrency-based probabilistic micropayment scheme. The free-rider problem is resolved by running a data co-location puzzle game to check caches' bandwidth utilization.

Earlier examples of cryptocurrency proposals followed satoshi-style blockchain techniques, such as Bitcoin, Litecoin [9], Monero [62], and Zerocoin [63]. These proposals all leverage computationally-expensive processes, namely, proof-of-work (POW) scheme to reach consensus on transactions and deter certain types of attacks. However, the criticism of these approaches is that they waste computational

power without adding any intrinsic value. To address such concerns, recent cryptocurrencies have sought to provide some type of added service. For example, Primecoin [64] rewards finding long chains of prime numbers, so-called Cunningham chains. Nooshare [65] proposes substituting POW with scheduled Markov-Chain Monte-Carlo simulations. Namecoin [66] provides a private, censorship-resistant domain name service by paying peers periodically to maintain the registered domains.

A few cryptocurrencies aim to provide distributed services as well, especially cloud storage services. Some propose their own cryptocurrency networks. Others offer frameworks that run on top of existing blockchains. These approaches utilize a modified POW scheme, such as a smart contract [67] or a proof-of-retrievability (POR) formula [68], to achieve peer consensus and secure transactions.

Ethereum [7] created its own blockchain and currency market. Its open blockchain platform allows users to build decentralized applications to execute codes, facilitated by propagating transactions in the form of smart contracts. To create new blocks, miners have to verify these contracts and run the codes. Thus, network applications, such as storage service, can be implemented on top of a smart contract and be executed in a decentralized routine. However, every node in Ethereum has to execute the contract if it is interested, raising a computational arm race problem. In addition, Ethereum brings unnecessary computational complexity to a pure monetary transaction, thus reducing the liquidity of the currency.

Filecoin [46], Permacoin [69], and Retricoin [70] incorporate distributed data storage services with their own blockchains. Unlike Bitcoin and other traditional cryptocurrencies, these altcoins require miners to invest not only computational resources, but also storage. Filecoin allows users to upload and download their own files to the network, while the other two aim to maintain a large static data

pool among all miners to allow for recoverability, even after a catastrophic failure. In Filecoin, currency is awarded for storing files, and is transferred in transactions, similar to Bitcoin. Similarly, Permacoin and Retricoin reward miners for storing encoded shares of the large archive. In addition, Retricoin claims to have lower storage overhead and network bandwidth requirements than that of Permacoin, and provides pool mining options as well. All three of these altcoins rely on POR techniques to verify the data stored by miners locally in order to reach consensus. If a miner claims to create a new block, she needs to submit proof that she possesses the data locally in a way that can be verified publicly by the entire network. However, for Permacoin and Retricoin, it is arguable whether it is economically worthwhile to mobilize so many users to contribute both computational resources and storage for the sole purpose of maintaining a static file.

Instead of announcing new cryptocurrencies, Metadisk [44], also known as Storj, offers a distributed storage service framework that runs on top of existing blockchains. Different from Permacoin and Retricoin, Metadisk users can choose peers to store their content by signing contracts. Both parties in Metadisk can negotiate the price and time to store the content. Once an agreement is reached, a user sends an encrypted shard of her file to this peer. Metadisk also leverages a POR scheme to verify the integrity of the data stored by miners. It offers a pay-as-you-go model where peers are paid with some frequency by the users. Since users only upload encrypted files to peers, the security and recoverability of the file rely on the encryption/decryption key pairs and the erasure code technique. Other projects, such as Sia [71] and MaidSAFE [72], provide similar services with a few tweaks, such as integrating a smart contract and improving the efficiency of operations.

Compared to the proposals mentioned above, CacheCash has several funda-

mental differences. Firstly, CacheCash does not invent its own blockchain. Instead, it incorporates its technology with other existing cryptocurrencies. Miners still follow the original POW process, but need to verify additional types of transactions. Secondly, CacheCash aims to provide a decentralized content delivery service, not storage. Although correctly storing the content is one crucial part, CacheCash does not periodically audit caches, since this can be done implicitly in the content delivery process. Thirdly, different from most projects that endorse a POR, CacheCash proposes its own data co-location puzzle game that can check multiple data blocks at once, and therefore ensures the integrity of its data service. Lastly, the incentive for caches participating in CacheCash lies in the economic fact that any entity should act honestly in order to get its fair share, and everyone's profit can be maximized.

## 4.2 CacheCash Design Overview

This section presents a high level description of the design and operation of CacheCash. We start with the main challenges and issues that guided the system design.

### 4.2.1 Design Challenges

It is challenging to build a system where all parties (even malicious ones) are economically incentivized to act toward a common goal. Enforcing security and trust is a necessity. Yet, efficiency is also crucial for the system to be of actual use. Content providers seek trusted networks of caches that provide wide coverage but at low cost. Users want to work as caches in these networks, but without the additional cost of dedicated hardware. Thus, both are trying to maximize profits



while minimizing overhead. On the other hand, clients are just interested in retrieving content and are not concerned with this profit dilemma. They usually are willing to either pay indirectly (e.g., watch advertisements) or directly (e.g., subscribe to Netflix) for this content. To meet all these interests, the following principles directed the design decisions of CacheCash:

- Content providers begin with content that they make available for caches to serve. No prior knowledge about these content providers/caches is available. Participants are identified using public keys that can be easily changed.
- The purpose of using CDNs is to reduce the amount of interaction between clients and the content provider by delegating expensive portions of the client service to caches. Thus, much like what happens on popular sites, such as YouTube or Netflix, some small communication happens with the content provider (e.g., logging in or issuing search requests), but the bulk of the data actually comes from caches. In this configuration, clients send content requests to the content provider and get back replies that enable them to fully retrieve the data from caches. This enables content providers to reply to a larger number of clients, and it enables clients to get faster service, as no further delays are incurred while waiting for additional responses from content providers.
- Under the assumption of rational participants, content providers aim to serve the maximum number of clients while paying the minimum service fee to caches. Caches want to collect the maximum possible payments while providing the least amount of resources (specifically bandwidth). Malicious content providers may try to avoid paying caches, and malicious caches may try to be paid without working. The system must provide a guarantee that

honest caches will be paid, and honest content providers will have their clients served. Thus, payments and service must be intertwined in a way that forces both parties to be honest until the end of the service session.

- It is highly desirable to have both the service delivery and the payment process publicly verifiable. Anyone should be able to verify that: a content provider has a database with specific content, a content provider is able to pay caches, clients are served correctly, and caches are paid. This feature has the advantage of making many types of cheating detectable, and thus, with proper economic penalties, it can be made unprofitable.
- It is tempting to achieve public verifiability by logging everything on the blockchain. However, this will explode its size and increase the workload of the miners to process the huge number of messages/transactions. Consequently, we need techniques and primitives that provide succinct representation and selectivity of what will be published on the blockchain.

There are issues related to the cryptocurrency system that must be considered as well, such as the stability of the currency value, and how to encourage miners to hold the stake of the system, and to make users trust the new currency enough to join the system. These issues are beyond the scope of this work and will be introduced in a successive work that deals with the financial aspects. The goal of this research is to introduce the service-related issues, and argue about the system's security and efficiency levels.

#### 4.2.2 Network Model

CacheCash's operation is based on forming a dynamic network of content providers, clients, and caches who can join and leave the system at any point

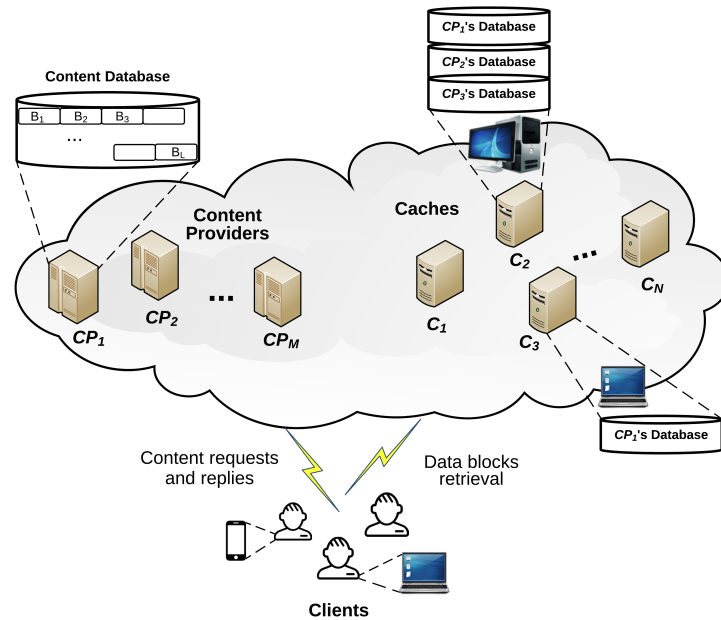


Figure 4.1: CacheCash network model.

in time. As depicted in Figure 4.1, the network model consists of  $M$  content providers  $CP_1, \dots, CP_M$ ,  $N$  caches  $C_1, \dots, C_N$ , and an arbitrary number of clients  $CL_1, \dots$ . Each  $CP_j$  produces and maintains a content database composed of  $L$  equally-sized data blocks, denoted as  $B_1, \dots, B_L$ . A copy of this database is shared with all caches who are willing to work with  $CP_j$ .  $CP_j$  publishes data block manifests and a signed hash of her database publicly to enable caches to verify the correctness of the shared content. Any cache  $C_i$  may concurrently work with several content providers if it owns sufficient storage/bandwidth (e.g. cache  $C_2$  in Figure 4.1).

For efficiency reasons, the system operation is divided into batches. During each batch the content provider aggregates all clients' requests and replies to them at once. Similarly, the micropayment scheme operation is divided into rounds. Each round is the time needed to mine a block on the blockchain, and it is defined

by the index of a confirmed block on the blockchain. Note that a batch period (e.g., milliseconds) is much smaller than a round period (e.g., minutes) since content providers must reply to clients in a timely manner.

### 4.2.3 Cryptographic Primitives

CacheCash employs several cryptographic primitives to secure its operation. Based on the protocol design, as will be shown shortly, each retrieved data block is protected by two layers of encryption. We refer to the outer and inner encryption layers as  $E_{outer}$  and  $E_{inner}$ , respectively, and both are implemented using symmetric key encryption algorithms. The inner/outer session encryption keys used by cache  $C_i$  are denoted as  $k_{in,i}$  and  $k_{out,i}$ , respectively. The session keys are refreshed for each service session and generated using a pseudorandom function (PRF). This PRF is keyed with master keys that are generated/exchanged in advance. There are two sets of master keys:  $K_{M,CP,C_i}$  which are shared between the content provider and cache  $C_i$  and used to generate  $k_{in,i}$ , and  $K_{M,C_i}$  known only to  $C_i$  and used to generate  $k_{out,i}$ . We let  $\sigma$  denote a digital signature on a specific message for which each party maintains a private/public key pair to sign/verify the issued messages. The public key is used as the ID that identifies any party within the system.

### 4.2.4 System Overview

At the core of CacheCash is an incentivized data service protocol that ensures data delivery to clients and payments to caches. Caches are paid per each data block they serve. As expected, this will result in a large number of small value payments that, if performed naively, could overwhelm the blockchain. To

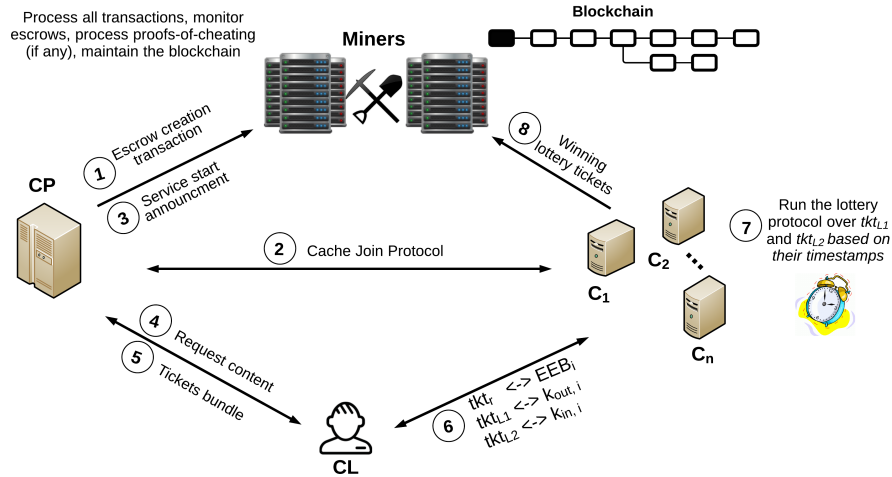


Figure 4.2: CacheCash operation.

solve this problem, we devise a decentralized probabilistic micropayment scheme, conceptually similar to prior work [73,74], that is compatible with the CacheCash operation. Namely, we utilize the lottery concept in which payments take the form of lottery tickets and winning tickets only result in currency transfer transactions. In this chapter, we give only a brief overview of the micropayments system. However, full details, including transaction types and processing, and the format of the lottery tickets, will be discussed in a follow-up work.

Another issue related to payments is when to pay for the delivered service. Paying caches in advance is not reasonable, as malicious ones may get payments without serving clients. On the other hand, paying caches after the service makes sense only when sending encrypted data blocks. In this case, caches do not send the decryption keys unless they receive valid lottery tickets. Nevertheless, malicious caches may send corrupted data, and get the lottery tickets without sending the decryption keys. Thus, an honest content provider would lose currency, while her clients get corrupted content. To reduce the loss to honest parties, we divide

payments into two parts. Each cache gets two lottery tickets, denoted as  $tk_{L1}$  and  $tk_{L2}$ , for every data block it serves. The cache must first commit bandwidth before getting these tickets, which minimizes the benefit of a cache that accepts tickets without serving clients. In addition, we compute the currency value of a winning  $tk_{L1}$  based on how many winning  $tk_{L2}$  a cache has. This is done to encourage caches to be loyal and complete the service session to the end, as they will be interested in collecting both types of tickets.

Having two lottery tickets means that caches need two layers of encryption to protect the served data blocks. Hence, each lottery ticket is traded with the decryption key of each encryption layer. One key,  $k_{out,i}$  is known to caches only and is traded with  $tk_{L1}$  while  $k_{in,i}$  is known to both the content provider and caches and is traded with  $tk_{L2}$ . Hence, if a malicious content provider does not want to pay, the delivered data to the client is not useful, since only caches know  $k_{out,i}$ . On the other hand, if a cache stops prematurely after retrieving  $tk_{L2}$  the data is still useful since the content provider can send the missing keys to the client.

The system operation is depicted in Figure 4.2. As shown, both the content provider and caches must complete a setup phase before starting the service. A content provider who wishes to join CacheCash creates an escrow using a special transaction denoted as  $EscTr$ . This transaction is verified by the miners and published on the blockchain.  $EscTr$  provides a guarantee for caches that the content provider can pay them for the service. In addition, it contains information on how to reach the content provider and a signed hash of her content database.

Similarly, caches register with a content provider to join the system. As shown in **step 2** in Figure 4.2, this is done by exchanging a set of messages in what we call a cache join protocol. During this protocol, a cache retrieves a copy of the content

provider's database, verifies its correctness, and proves to the content provider that it has fully retrieved the database. Once a reasonable number of caches join the content provider's network, she completes the setup phase. This event is announced to the miners (**step 3** in Figure 4.2) and logged on the blockchain.

To start a service session, a client contacts a content provider asking for specific content, such as a video, which is defined in terms of data block indices. A set of  $n$  caches is selected to serve clients, with each cache  $C_i$  assigned a specific data block  $B_i$ . We let the content provider select this set using any mechanism of her choice. The content provider generates a reply to the client that contains a request ticket (denoted as  $tkr$ ), in addition to lottery tickets for caches. All these tickets, along with other information, comprise the content provider's reply to the client (illustrated as **steps 4** and **5** in Figure 4.2), and is called a tickets bundle. As mentioned earlier, the content provider replies to all requests on a batch basis. Hence, one signature covers all tickets bundles that belong to the same batch.

On its side, the client continues the service session by sending each request ticket  $tkr_i$  to its designated cache.  $C_i$  replies with a double encrypted data block denoted as  $EEB_i$ . The decryption keys are sent to the client upon reception of the lottery tickets, as mentioned earlier. As such, **step 6** in the figure is a multi-step interaction between the client and each  $C_i$ .

Caches keep their lottery tickets and run a lottery protocol to claim payments using the winning tickets as depicted by **steps 7** and **8** in Figure 4.2. The lottery protocol is distributed and publicly verifiable. Furthermore, it guarantees with high probability that all valid winning lottery tickets will be honored by the content provider, and that a ticket's winning probability cannot be manipulated. For this purpose, we tie the lottery with the confirmed blocks on the blockchain. In detail, each lottery ticket contains three timestamps that specify the issue time

of the ticket, the lottery draw time, and the redeem time. A winning ticket is defined as the one with the least  $u$  bits of its hash matching the corresponding bits of the hash of a block on the blockchain that has an index equal to the lottery draw time. For example, consider the lottery ticket and the blockchain status in Figure 4.3 with  $u = 8$  bits. This ticket is a winning one since there is a hash match with the block that carries the same index as the lottery draw time found inside the ticket.

Two subtle issues arise here. First, a malicious content provider may issue more lottery tickets than she can afford. Second, a malicious content provider may manipulate the lottery winning probability by printing winning tickets after seeing the hash of the block used in the lottery draw. We solve these issues by computing the maximum number of tickets a content provider is allowed to issue based on the escrow balance, and by making him commit to the issue time of these tickets in advance. The latter is done by making the content provider commit to the sequence numbers of the tickets that she can issue per round. The content provider includes all this information in *EscTr*. Thus, anyone is able to verify its correctness. For this purpose, additional fields are added to each lottery ticket including a sequence number denoted as *tseq*, and the IDs of the destined cache, the content provider, and the escrow to which this ticket is tied.

An example of lottery tickets commitment is shown in Figure 4.4. *EscTr* includes the total number of tickets that can be issued using the locked currency value. Based on the expected tickets issue rate, this number is divided among the rounds that follow the confirmation of *EscTr*. Each round number is the issue time of the tickets with the sequence numbers range assigned to the round. In Figure 4.4, a content provider can afford 3000 tickets and she is expecting to issue 1000 tickets per round. Her *EscTr* is included in block 10 on the blockchain



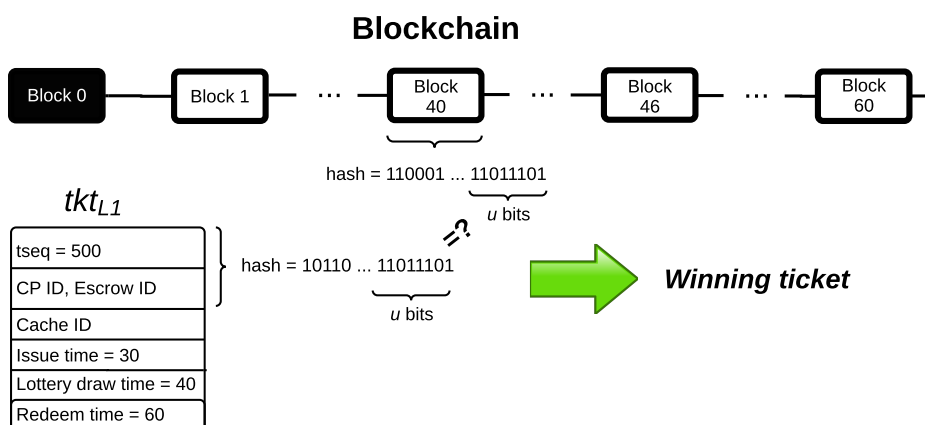


Figure 4.3: Lottery draw example: the lottery draw time is 40, the ticket hash is taken over the committed fields and compared with the hash of the block with an index that equals the lottery draw time. The redeem time of this ticket is after Block 40 is confirmed until time 60. After that the ticket expires. Lottery over  $tk_{L2}$  proceeds in an analogous way.

and confirmed at the time of block 16 (assuming that a block is confirmed when an additional 6 blocks are added to the blockchain on top of it). Thus, she is allowed to issue tickets for 3 rounds, starting at round 17, with contiguous sequence number ranges. All tickets that are issued in the same round enter the lottery at the same time. Once the time of those rounds passes, the content provider cannot use this escrow to issue lottery tickets. Thus, she has to create a new escrow to continue serving her clients.

Anyone can validate the issue time of a lottery ticket by checking its  $tseq$  with what is found in  $EscTr$  of the same escrow ID found in the ticket. Hence, in the previous example, a lottery ticket with an issue time of 19 and  $tseq = 450$  is invalid. Since the content provider commits to the escrow ID, her ID, and  $tseq$  in advance, we make the lottery hash cover those fields only, as shown in Figure 4.3.

Miners check the residual balance in the escrow after each payment transaction. If it falls below a threshold value, miners break the escrow and distribute the residual balance as shares among the content provider and her set of caches.

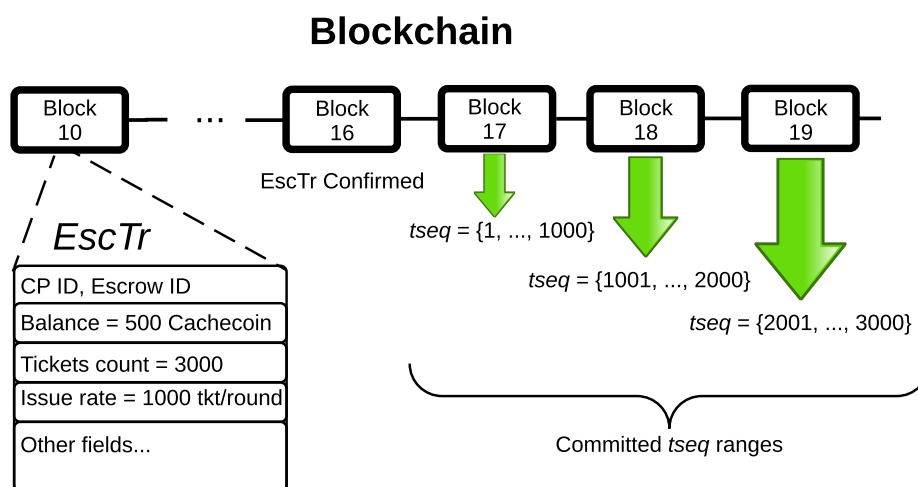


Figure 4.4: Commitment to lottery tickets example.

Hence, caches are rewarded for storing unpopular content that has a low request rate from clients. On the other hand, a content provider may not get a large number of client requests. Hence, after redeeming all winning tickets, her escrow will still contain currency. To avoid burning currency in the system, a content provider sends a request to miners to return the remaining escrow balance. This is possible as all lottery tickets expire after their redeem time.

One final issue is related to cheating detection in the system. If an entity detects someone cheating during the data service protocol, it sends a proof-of-cheating to the miners that contains all the needed evidence. Miners verify the cheating incident and are able to economically punish the cheating party based on the cheater's role in the system, as will be shown later.

## 4.3 Incentivized Data Service Protocol

### 4.3.1 Batching Client Requests

As mentioned earlier, a service session starts with a content request sent by clients ( $CL$ s). For each request  $CL$  generates a sequence number  $rseq$  that is used to check the freshness of the request, and to identify the service session. A content provider ( $CP$ ) aggregates all requests received within a batch and replies to them all at once, as depicted in Figure 4.5.  $CP$  replies to each  $CL$  with a tickets bundle that includes the following fields:

- Caches' contact information, such as their IP addresses and port numbers.
- Group of request tickets  $tkr_r$ .
- Group of lottery tickets 1 ( $tkr_{L1}$ ).
- One masked  $tkr_{L2}$  denoted as  $mtkr_{L2}$  (masking details are found in the next subsection).
- Data co-location puzzle related fields (see the next subsection).
- $CP$ 's batch signature over all ticket bundles in the current batch.

The field  $tkr_r$  is a set of  $n$  request tickets, each destined to a single cache  $C_i$  and denoted as  $tkr_{r,i}$ . Each  $tkr_{r,i}$  is used to request one data block from  $C_i$  and contains  $C_i$ 's ID,  $CL$ 's ID, the requested data block index, a hash of  $k_{in,i}$  that will be used by  $C_i$  during this session, and  $rseq$ :

$$tkr_{r,i} = ID_{C_i} || ID_{CL} || index(B_i) || h(k_{in,i}) || rseq$$

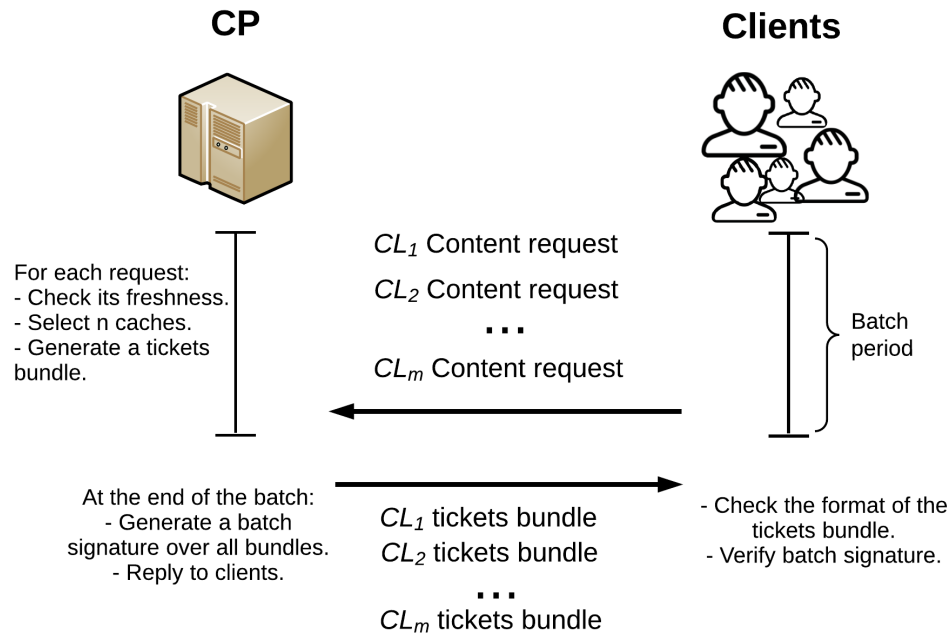


Figure 4.5: Batching client requests.

The inclusion of  $h(k_{in,i})$  enables  $C_i$  to check that its  $k_{in,i}$  is identical to what is used by  $CP$ . Thus, she cannot pretend that  $C_i$  uses an invalid key to accuse it of cheating.

The field  $tk_{L1}$  is also a set of  $n$  individual lottery tickets each denoted as  $tk_{L1,i}$  and destined to  $C_i$ . On the other hand,  $tk_{L2}$  is a single ticket for all caches (all caches get a copy of the same ticket) and it is not sent in the clear as  $tk_{L1}$ . Instead, it is masked using the results of the data co-location puzzle. This is done to ensure that  $CL$  has retrieved all data blocks from caches before they received  $tk_{L2}$ . The purpose of having a single  $tk_{L2}$  is to address the case of a malicious  $CL/CP$  who tries to send this ticket to malicious caches only to lower the value of payments to honest caches.

For computational efficiency reasons, we make  $CP$  produce one signature over all tickets bundles that belong to the same batch. In the signing process  $CP$

hashes the tickets bundles in several stages before signing. First, she hashes the various fields in a tickets bundle individually. We refer to this set of hashes as  $h_{bundle}$ . Second,  $CP$  computes the hash of a single tickets bundle as the hash of its  $h_{bundle}$ . Lastly,  $CP$  computes the hash of all tickets bundles in the current batch by computing a Merkle hash tree of their hashes, which we call  $mtree_{bundles}$ . The batch signature  $\sigma_{CP,batch}$  is simply  $CP$ 's signature over the root of this tree.  $CP$  adds the following to each tickets bundle:  $root(mtree_{bundles})$ , its membership path in this tree, and  $\sigma_{CP,batch}$ . For simplicity, in what follows once we mention  $\sigma_{CP,batch}$  we view it as a structure that contains  $h_{bundle}$ , the membership path in  $mtree_{bundles}$ ,  $root(mtree_{bundles})$ , and  $CP$ 's signature over this tree root.

As for signature verification,  $CL$  first hashes the tickets bundle fields and compares them with the received  $h_{bundle}$ , checks the given membership path of the hash of  $h_{bundle}$ , and then verifies  $CP$ 's signature over the given root. Caches use a slightly different method since they do not receive the full tickets bundle. Instead, each cache  $C_i$  gets individual tickets at different stages within the service session, and receives  $\sigma_{CP,batch}$  with the first ticket it gets from  $CL$ . Consequently,  $C_i$  hashes the received  $h_{bundle}$  and continues in a similar way to  $CL$ . However, to check that a ticket is covered by this signature,  $C_i$  hashes the ticket and looks for an identical hash in  $h_{bundle}$ . If found, this means that  $\sigma_{CP,batch}$  covers the ticket and it is accepted.

Before signing and sending the tickets bundle,  $CP$  has to compute the data co-location puzzle over the raw data blocks requested by  $CL$ . This is needed to produce  $mtkt_{L2}$ .

### 4.3.2 Data Co-location Puzzle

As mentioned previously, caches receive two lottery tickets,  $tk_{L1}$  and  $tk_{L2}$ , for each data block they serve. There is a collusion risk between  $CL$  and caches in which  $CL$  (who is not interested in the content) sends the lottery tickets to caches without retrieving any data (i.e. performs a slacking attack). To address this collusion case, we rely on the fact that  $tk_{L2}$  gives value to  $tk_{L1}$ . For this purpose,  $tk_{L2}$  is masked using a puzzle computed by  $CP$  over the requested data blocks. Hence,  $CL$  needs to receive the data blocks first, compute this puzzle, and then unmask  $tk_{L2}$  before sending it to caches.

To ensure efficiency, we want this puzzle to be computationally light for  $CPs$  to ensure fast response time when replying to clients. On the other hand, we want it to be heavier for  $CLs$  to make computing it in a malicious way expensive. Thus, this puzzle is designed in a way that computing it by retrieving all data blocks locally is cheaper than any malicious strategy. Thus, rational  $CL$  and caches will choose to act honestly as it is more profitable.

The core idea is to encrypt and hash small pieces of the data blocks at random locations to produce a single hash digest to be used in masking. For short, we refer to this puzzle as the hash-encryption puzzle, abbreviated as  $HE_{puzzle}^{CP}$  and  $HE_{puzzle}^{CL}$  for  $CP$  and  $CL$ , respectively. The input of  $HE_{puzzle}^{CP}$  is  $B_1, \dots, B_n$ , while for  $HE_{puzzle}^{CL}$  it is  $EB_1, \dots, EB_n$ . Nevertheless, both  $CL$  and  $CP$  produce the same results. For this purpose,  $CP$  encrypts the data block pieces in the same way used by caches, i.e., using the encryption algorithm and same  $k_{in,i}$  used by each cache.

To enable such random encryption of data pieces, a parallelizable encryption mode is needed. We selected AES in the counter mode (AES-CTR) to handle this functionality. The initial value of the CTR mode counter for each block  $B_i$ , called

$ctr_{i,initial}$ , is generated using any method that makes both  $CP$  and  $C_i$  produce the same value non-interactively. Moreover, this mechanism must produce a different value for each session, even if it is over the same  $B_i$ . For example, one may use encryption of some pieces of  $B_i$  and use the ciphertext as the counter value.

---

**Algorithm 3:**  $HE_{puzzle}^{CP}$  computation

---

**Input:** Data blocks  $B_i$  for  $i = 1, \dots, n$   
**Output:**  $HE_{puzzle}^{CP}$  result and stopping criteria

- 1 /\*Initialization\*/:
- 2 **for**  $i = 1$  **to**  $n$  **do**
- 3     Generate  $k_{in,i}$
- 4     Compute  $ctr_{i,initial}$
- 5 **end**
- 6 Select  $m \in \{1, \dots, n\}$  randomly.
- 7 Select  $sB_1$  randomly from  $B_m$
- 8 Set  $offset(sB_1) = \frac{address(sB_1) - address(B_m)}{size(sB)}$
- 9 Set  $location(sB_1) = h(m || offset(sB_1))$
- 10 /\* $i$  is the data block index\*/
- 11 Set  $i = m$
- 12 **for**  $j = 1$  **to**  $(n * R_{puzzle} - 1)$  **do**
- 13     /\*Compute  $ctr$  and encrypt  $sB_j$ \*/
- 14      $ctr_j = offset(sB_j) + ctr_{i,initial}$
- 15      $c_j = E(k_{in,i}, ctr_j, sB_j)$
- 16     /\*Compute location and offset of  $sB_{j+1}$ \*/
- 17      $location(sB_{j+1}) = h(location(sB_j) || c_j)$
- 18      $offset(sB_{j+1}) = location(sB_{j+1}) \bmod \frac{size(B_i)}{size(sB_j)}$
- 19      $i = (i \bmod n) + 1$
- 20 **end**
- 21 Set  $u = n * R_{puzzle}$
- 22 Set  $result = location(sB_{u-1})$
- 23 Set  $last = location(sB_u)$
- 24 **return**  $result, last, R_{puzzle}$

---

The details of  $HE_{puzzle}^{CP}$  computation are depicted in Algorithm 3. In this algorithm, each  $B_i$  is divided into aligned sub-blocks of equal sizes, each denoted as  $sB$ .  $R_{puzzle}$  is the number of puzzle rounds,  $size(.)$  returns the size of its input

in bytes,  $address(.)$  returns the starting address in memory,  $offset(.)$  returns the index of  $sB$  within a data block  $B_i$  (the first sub-block has offset 0, the second one has offset 1 and so on),  $location(.)$  returns some random value generated based on its input, and  $h(.)$  is a hash function. Finally,  $result$  is the puzzle output, and the stopping criteria include  $R_{puzzle}$  and  $last$ , which is the  $sB$ 's location that comes immediately after  $result$ .

$CP$  starts by generating all the needed encryption keys and counters values. Then, she picks a data block randomly and a starting point  $sB_1$  randomly inside this block. For this starting point,  $CP$  computes its offset and location. The former is deterministic and computed based on  $sB$  address distance from  $B_i$  starting address. The latter can be computed using any method that produces a random value, given that both  $CP$  and  $CL$  produce the same value non-interactively. After that,  $CP$  computes the location and offset of the next  $sB$  in the next  $B_i$  and repeats that for the required number of iterations as found in lines 14 - 19 in the above algorithm. In this loop,  $CP$  works on the data blocks in a round robin fashion. The purpose is to touch all data blocks in an approximately uniform way. As shown, the puzzle output is the second to last computed  $sB$  location, while the stopping criterion is the location of the last  $sB$ . The field of the data co-location puzzle-related information inside a tickets bundle includes  $Puzzle_{start}$ ,  $R_{puzzle}$ , and  $last$ .

The computation of  $HE_{puzzle}^{CL}$  is slightly different since the input is the inner layer encrypted data blocks, not the raw ones. Since  $CL$  works on  $EB_1, \dots, EB_n$ , there is no need to encrypt  $sB_i$ , and  $CL$  only computes the accumulated hash. In addition,  $CL$  is given a set of possible starting points for the puzzle, denoted as  $Puzzle_{start}$ . Thus, she recomputes the puzzle for every starting point until she finds the correct result. In fact, this is what makes the puzzle computation



more extensive on  $CL$ 's side. The result correctness check is simply to test if the value of  $last$ , which she receives from  $CP$ , equals  $location(sB_{n*R_{puzzle}})$ , which she computes in the last iteration of the algorithm.

### 4.3.3 Data Blocks Retrieval

$CL$  continues the service session by contacting caches to retrieve the data blocks to fulfill her content request. In what follows, we describe this interaction using notation similar to that used by Sirivianos et al. [39]. A pictorial illustration is found in Figure 4.6.

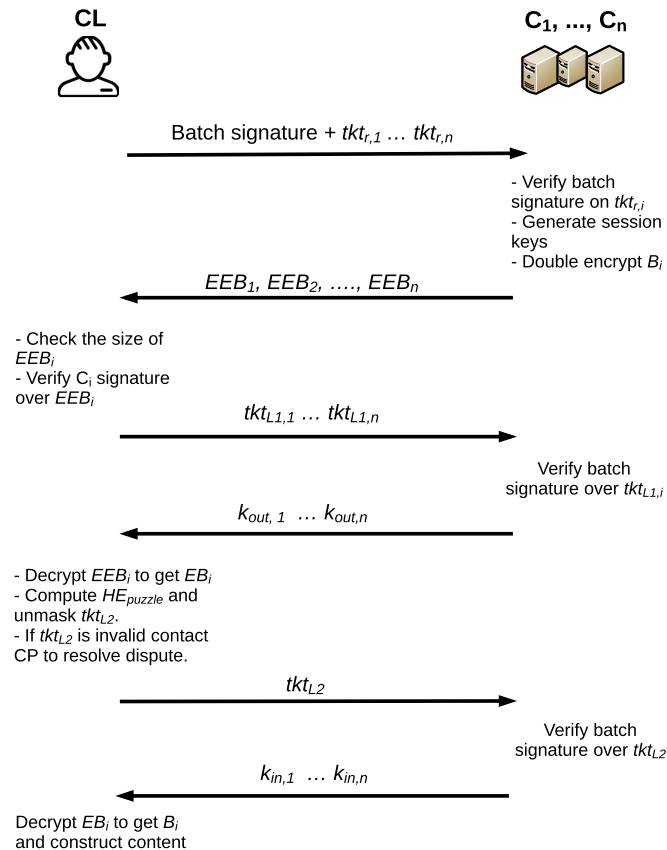


Figure 4.6: Data blocks retrieval from caches.

**Step 1:**  $CL$  forwards  $tk_{r,i}$  along with  $\sigma_{CP,batch}$  to caches as follows:

$CL \rightarrow C_i : tkt_{r,i} || \sigma_{CP,batch}$  for  $i = 1, 2, \dots, n$

**Step 2:**  $C_i$  verifies  $CP$ 's signature over  $tkt_{r,i}$  and checks the freshness of  $rseq$ . Then, it generates the needed session keys in the same way used by  $CP$  to generate  $k_{in,i}$ , while it uses  $k_{M,C_i}$  to generate  $k_{out,i}$ . Those keys are used to double encrypt  $B_i$  (i.e.  $EEB_i = E_{k_{out,i}}(E_{k_{in,i}}(B_i))$ ) and the result is sent back to  $CL$ :

$C_i \rightarrow CL : EEB_i$  for  $i = 1, 2, \dots, n$

**Step 3:**  $CL$  checks that the size of  $EEB_i$  is comparable to the size of  $B_i$  and verifies  $C_i$ 's signature over  $EEB_i$ . If valid, she forwards  $tkt_{L1,i}$  to every cache as an implicit request for  $k_{out,i}$ :

$CL \rightarrow C_i : tkt_{L1,i}$  for  $i = 1, 2, \dots, n$

**Step 4:**  $C_i$  validates  $tkt_{L1,i}$  as follows: verify  $\sigma_{CP,batch}$  over this ticket, check that the ticket sequence number  $tseq$  is within the allowed range of the corresponding escrow and has the correct issue time. If anything goes wrong  $C_i$  drops  $CL$ 's request. However, in case that  $tseq$  is out of range,  $C_i$  issues a proof-of-cheating against  $CP$ . If everything is correct,  $C_i$  sends  $k_{out,i}$  to  $CL$ :

$C_i \rightarrow CL : k_{out,i}$  for  $i = 1, 2, \dots, n$

**Step 5.a:**  $CL$  decrypts the outer layer of all  $EEB_i$ s. She uses  $EB_i$  to compute  $HE_{puzzle}^{CL}$  and un.masks  $tkt_{L2}$ . A correct unmasked  $tkt_{L2}$  is sent to caches to complete the service payments and to request  $k_{in,i}$ :

$CL \rightarrow C_i : tkt_{L2}$  for  $i = 1, 2, \dots, n$

**Step 5.b:** An invalid unmasked  $tkt_{L2}$  means that either a cache(s) has sent cor-

rupted data, or a cache(s) has sent invalid  $k_{out,i}$ , or  $CP$  has sent invalid  $HE_{puzzle}$  stopping criteria to avoid paying caches  $tk_{L2}$ .  $CL$  cannot identify the cheater, hence, she contacts  $CP$  to investigate the situation by sending all the received  $EEB_i$  and  $k_{out,i}$  from caches. If  $CP$  is not the cheater, she identifies the malicious cache(s) and issues a proof-of-cheating to inform the miners.  $CP$  has  $k_{in,i}$  used by each  $C_i$  and she sent its hash earlier in  $tk_{tr}$ . Since caches replied with  $EEB_i$  this means that they agree that  $k_{in,i}$  is identical to what they used. Otherwise, they were supposed to drop the  $CL$  request.  $CP$  decrypts  $EB_i$  received from  $CL$  and checks the resulting data block by comparing its hash to the one found in the escrow creation transaction. If it does not match, this means that  $C_i$  has sent corrupted data or invalid  $k_{out,i}$ .  $CP$  proves the cheating to the miners by sending  $C_i$ 's responses, which include  $EEB_i$ ,  $k_{out,i}$ , along with  $k_{in,i}$  and the original  $tk_{tr}$  that contains the hash of this key. Since  $C_i$  signs all the sent messages, it cannot deny cheating or blame  $CP$  for anything. In addition, all these messages contain  $rseq$  to bind them to the service session. Miners will check everything by verifying signatures on the responses, and decrypting  $EB_i$  in addition to comparing data blocks hashes. The punishment of this cheating cache is discussed in the next subsection. (The case of a cheater  $CP$  is handled by the economic analysis of the system, as will be shown later.) Meanwhile,  $CP$  directs  $CL$  to contact other caches to retrieve valid copies of the corrupted data blocks:

$CL \rightarrow CP : EEB_i, k_{out,i}$  for  $i = 1, 2, \dots, n$

$CP \rightarrow miners : Proof-of-cheating\ against\ C_i\ (if\ any)$

$CP \rightarrow CL : new\ tickets\ bundle$

**Step 6:** Caches validate  $tk_{L2}$  as in **Step 4** with one difference: they have to mask this ticket before verifying the batch signature. The puzzle result is a field

inside  $tk_{L2}$  and is used to produce  $mtkt_{L2}$ . The response of a correct ticket is to send  $k_{in,i}$  to  $CL$ :

$C_i \rightarrow CL : k_{in,i}$  for  $i = 1, 2, \dots, n$

**Step 7:** Upon receiving  $k_{in,i}$ ,  $CL$  fully decrypts the data blocks and constructs the requested content. In case of missing  $k_{in,i}$ ,  $CL$  contacts  $CP$  to get the missing keys that complete the service session:

*if* ( $isReceived(k_{in,i})$ ) for  $i = 1, 2, \dots, n$

$CL$  constructs content.

*else*

$CL \rightarrow CP : \text{request missing } k_{in,i}$

#### 4.3.4 Proof-of-cheating Processing

Effective proof-of-cheating processing mechanisms are part of the financial implications of CacheCash and thus not discussed at length in this work. A proof-of-cheating is processed by miners as follows. First, they verify that there is cheating using the provided information inside the proof that documents the incident. Then, they issue a punishment to compensate the honest parties that have been harmed, and to discourage malicious entities from cheating again. For this purpose, the proof must be irrefutable and undeniable (beside guaranteeing its integrity).

The only punishment that has been well defined in CacheCash up until now is breaking the escrow of a malicious  $CP$  without giving him any share of its currency balance. The punishment of caches, on the other hand, is a system design parameter. One option is to remove this cache from all  $CP$ s cache lists. Finally,

detecting a malicious *CL* does not impact our system since clients are not involved in the payment process. They neither pay nor get paid. Thus, caches/*CP* may have additional policies to handle these situations, such as ignoring or delaying requests originated by misbehaving clients.

## 4.4 Performance Evaluation

In this section we evaluate CacheCash against several performance measures. To better understand the performance and efficiency of our protocol, we want to answer three important questions:

- How quickly does CacheCash serve content?
- How efficiently can clients retrieve this content?
- Can CacheCash scale to handle larger demands?

We set about answering these questions by implementing and conducting thorough microbenchmarks on the CacheCash data service protocol. In what follows, we describe the methodology of our test, and discuss the results obtained.

### 4.4.1 Methodology

In order to establish our microbenchmarks, we built reference implementations in C of CacheCash modules, namely, a content provider, a client, and a cache. For our microbenchmarks we isolated the constructions needed to compute the performance metrics we wished to measure.

For the caches and content providers, our experiments were conducted using a modest 64-bit desktop machine with a 4.0 GHz 4-core CPU, 16GB DDR 3 RAM, and Linux Mint 17.3 Rosa of kernel version 3.19.0-32.

Required broadband connection speed	0.5 Mbps
Recommended broadband connection speed	1.5 Mbps
Recommended for SD quality	3.0 Mbps
Recommended for HD quality	5.0 Mbps
Recommended for Ultra HD quality	25 Mbps

Table 4.1: Netflix internet connection speed recommendations

For the clients, we also employed a low-end five year old smartphone. The 2012 ZTE V887 model has 1GHz Dual-core Cortex-A9 CPU and 512 MB memory and runs the Android OS 4.0.4 operating system.

To put our performance measures into context, we then compared our benchmark results to those of a large-scale, time-sensitive online video streaming service. This service, the popular content provider Netflix, has stricter bandwidth and network latency criteria than typical file sharing services. It currently claims more than 83 million subscribers in 190 countries, enjoying more than 125 million hours of TV shows and movies per day, supported by more than 10,000 globally deployed ISP/IXP servers [75]. In a recent paper [76], researchers identified 4,669 Netflix ISP/IXP servers in 243 locations around the world. In North America, the company alone accounted for more than 36% of downstream demands in 2015 [77]. We want to understand if and how CacheCash can scale to meet the tremendous bandwidth demands of a content provider like Netflix.

The recommended Internet connection speed for end-users of Netflix is shown in Table 4.1. The amount of traffic per server varies greatly, but most generate 100 Mbps to 1 Gbps traffic volume. The entire Netflix network generates 3.12 Tbps on average, and 4.88 Tbps at peak time. Table 4.2 shows lists of the prime time bitrate for Netflix content streamed to Netflix members during a particular month in different countries. In the US, the high, average, and low speed are 3.62 Mbps, 3.28 Mbps, and 1.88 Mbps, respectively.

	US	UK	France	Japan	India	Brazil	Australia
High (Mbps)	3.62	3.75	3.75	3.73	2.21	3.09	3.41
Average (Mbps)	3.22	3.46	3.24	3.41	1.78	2.57	2.85
Low (Mbps)	1.88	3.19	2.87	2.63	1.04	1.94	2.4

Table 4.2: Netflix ISP speed index

We use the statistics listed above as baseline facts to evaluate the performance of CacheCash in the following sections.

#### 4.4.2 Content Provider

When a service provider wants to deliver its content to end users, she can join CacheCash as a content provider. According to the data service protocol, content providers do not communicate with each other, which means each content provider serves her clients independently. CacheCash encourages content providers to join the system, and there is no limit on how many of them can join. In the following paragraphs, we measure and analyze the performance of one single content provider.

We are interested in knowing how quickly a content provider can generate data and serve clients. As described in Section 4.3, a content provider is responsible for receiving data block requests from clients, generating replies, and sending replies back. In order to evaluate the performance of a content provider, we first benchmark the puzzle game generator, the most computationally-expensive module on the content provider end. Then we measure the average batch count and reply size of data block replies. Finally, we estimate the throughput and goodput of the content provider.

## Puzzle Game Generator

To prevent malicious caches cheating on the content provider by collocating data, a client needs to verify the integrity of the received data by solving a puzzle game and unlocking payment ticket 2. The puzzle game result is pre-generated by the content provider when producing the data block reply to the client.

We are curious about how quickly a content provider can generate puzzle game results. In order to generate one valid puzzle game output, a content provider needs to perform a certain number of AES encryptions (AES-CTR-128 on 16-byte data) and secure hash functions (SHA-256 on 16-byte data). The number of operations is the number of caches per request times the number of puzzle iteration rounds. According to Algorithm 3, each puzzle iteration round should be at least 1 round such that collocating data among malicious caches would not be beneficial. In our experiments, we set the range of number of caches per request from 1 to 16, and the puzzle iteration rounds as 1, 2, 4, and 8, respectively. To utilize all CPU cores, we spawn one puzzle game generating thread on each core. We label two options for content database storage. With the warm cache option, the entire database fits into RAM memory. With the cold cache option, the database is retrieved from the hard disk. In the experiment, to simulate a perfect cold cache, we exchangeably read different portions of the database from the hard disk between any two adjacent puzzle games. To benchmark the puzzle game generator, we assume the bandwidth is unlimited and let each individual experiment solve 1 million puzzle games in sequence and record the time.

Figure 4.7 and 4.8 shows the number of puzzle game results produced per second by the content provider in warm and cold caches, respectively. Single thread results are shown on the left side, and four-thread results are shown on



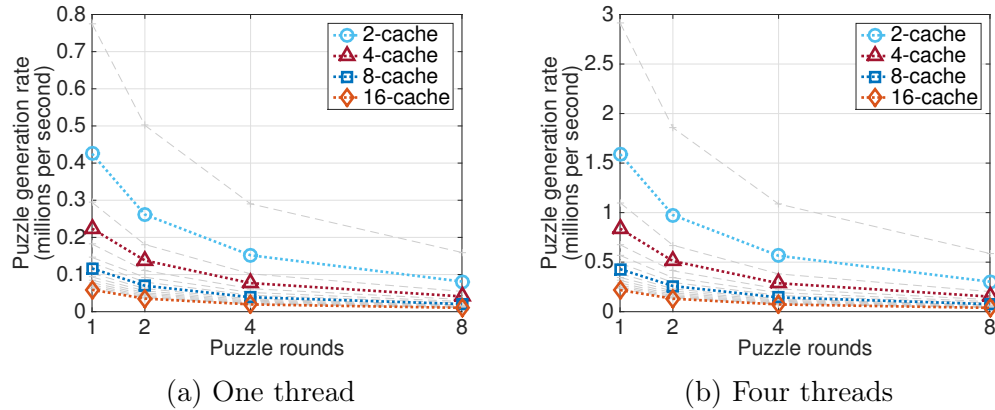


Figure 4.7: Puzzle generation speed (warm cache).

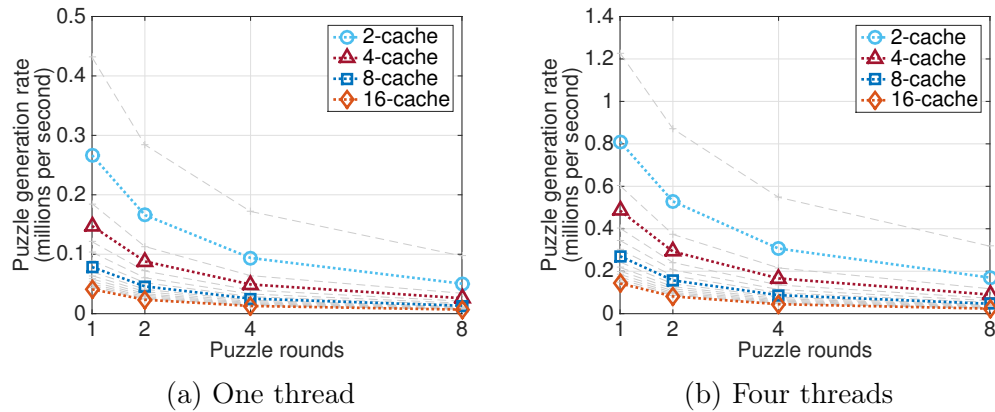
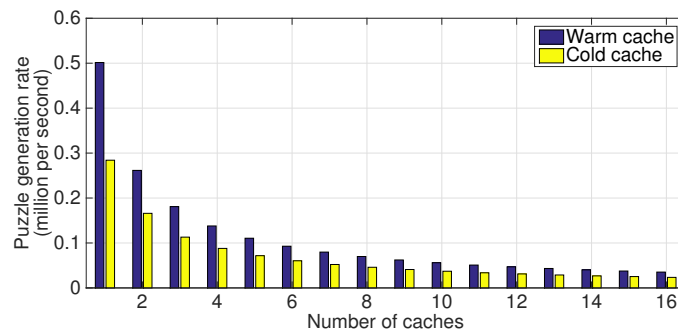


Figure 4.8: Puzzle generation speed (cold cache).

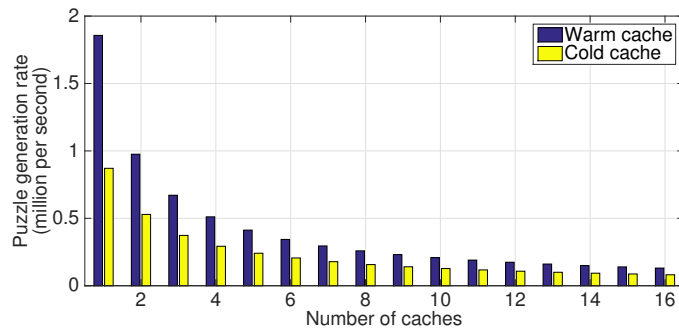
the right side. We highlight the results when the number of caches per request is at 2, 4, 8, and 16 by using four different colors. The rest of the results - those taken at other numbers of caches per request (1, 3, 5, 6, ...) - are colored in light grey. Each curve plots the number of puzzle games generated in millions per second. The number of puzzle games generated falls gradually when either puzzle iteration rounds or number of caches per request is increased. This result is expected, based on the cost of puzzle game result computation described in the previous paragraph.

To take a closer look at the results from two puzzle iteration rounds, we com-

pare the warm and cold cache outputs in Figure 4.9. In both single and four-thread cases, warm cache results are 30 – 40% higher than cold cache. When a content provider has a large amount of content to serve, she usually places the most popular content into memory and leaves less popular files on the hard disk. The odds are only a small percentage of clients would request unpopular content, which means cold cache situations are less frequent. Hence, the following analysis will mainly focus on warm cache outputs from 2 puzzle iteration rounds.



(a) One thread



(b) Four threads

Figure 4.9: Puzzle generation speed for warm and cold cache ( $R_{puzzle} = 2$ ).

A primary task for the content provider is to generate replies to client requests. Upon receiving such a request, the content provider takes the following steps: parses the request, generates puzzle game result, constructs a ticket bundle, and signs and attaches batch information (discussed in Section 4.4.2). As a result, a data block reply is created and sent back to the client. In our microbenchmark,

we implement the whole process to produce data block replies. Without a doubt, the puzzle game generator is the most computationally expensive module among all these steps. The benchmark results show that the speed to generate data block replies is mainly determined by the puzzle game generator (other factors only incur an additional 0.05% cost). Therefore, we use the speed of puzzle game generation as the metric of content provider **service rate**.

To put our results in context, we decided to test CacheCash performance using a much in demand piece of content from Netflix.com. According to Netflix, 6.5% of its 83 million subscribers watched at least one episode of the third season of its political thriller “House of Cards” within a month of its release [78]. Based on these numbers, we made an extreme assumption and proposed that all of the users ( $\sim 5.2$  million) watched the first episode immediately after its release. Given the 0.375 MBps Netflix SD quality video rate and 1 MB data block size, users would have generated requests for  $\sim 2$  million data blocks per second. As shown in Figure 4.9b, a content provider in CacheCash generates  $\sim 1$  million tickets bundles per second for the 2 caches case, which translates to  $\sim 2$  million data blocks per second. This means that one single CacheCash content provider would be able to handle this peak load of concurrent users. HD quality video rate (1.85MBps) is about 5 times higher than SD quality rate. Therefore, in this extreme case, 5 CacheCash content providers are capable of generating the required bandwidth for their clients.

### Throughput and goodput

Knowing the content provider’s ability to generate data block replies, we want to determine how much data a content provider can produce per second, or what we call content provider throughput.

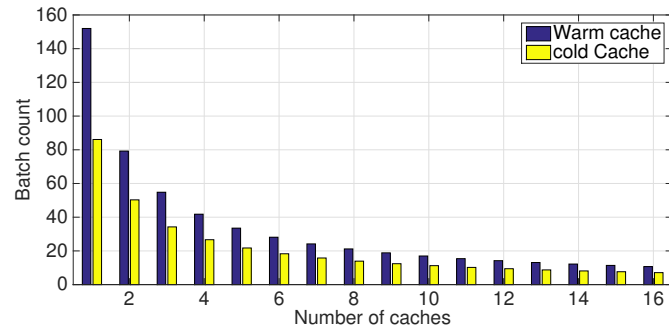
It is straightforward that the total amount of data produced by a content provider per second is the number of replies generated times the average reply size. However, the content of these replies is not the content the client wants. Instead, it is the block requests, payment tickets, and checksums that direct clients to download the real content from caches.

We also want to know the total amount of content that a content provider produces indirectly per second, which is called the goodput. This value tells us precisely how quickly CacheCash can generate data.

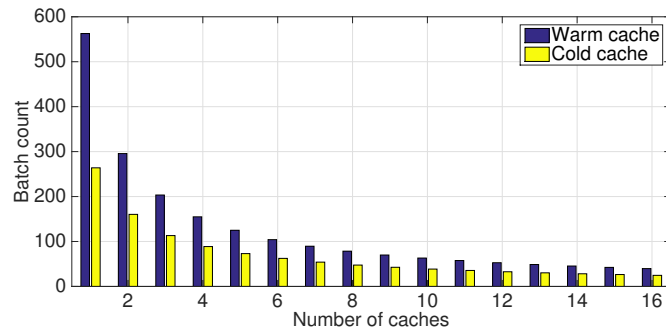
By comparing these two results, we also know how efficiently the content provider can serve clients.

### **Batch count and reply size**

As we just mentioned, suppose the content provider is not bound by the bandwidth. The total data she could generate in one second would depend on the size of the data block reply. When the content provider generates a reply, she needs to wait for the ticket bundles to be signed in batch, and attaches the batch information to the reply. In our experiment, we create one signing thread to utilize 100% usage of one CPU core to sign batch signatures. More specifically, in the data block reply generation process, a reply generator adds its valid puzzle game result from the puzzle game generator to the ticket bundle. It then pushes the bundle hash into a circular buffer and waits for the signature. Meanwhile, the signing thread continuously pops all available hashes from the buffer, calculates the merkle tree path, signs the batch signature, and returns the useful information to each reply generator. In this manner, the reply generator waits the minimum time to get the batch signature. Our signing thread can produce on average 3,568 ECDSA signatures per second. With merkle tree path calculation, the speed falls



(a) One thread



(b) Four threads

Figure 4.10: Average batch count.

nearly 8% to about 3,300 queries per second. Therefore, we get the average batch size of each reply. According to the specification, the size of batch information is:

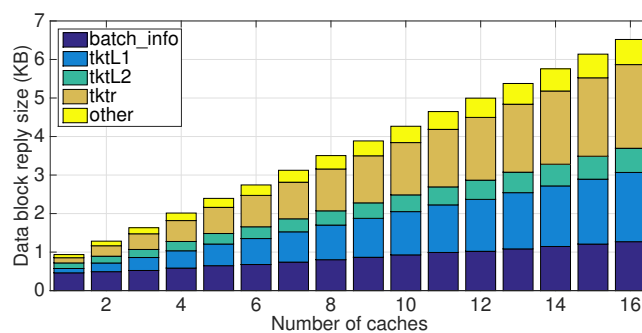
$$size_{batch\_info} = \#cache \times 64 + \lceil \log_2(batch\_count) \rceil \times 33 + 178 \quad (4.1)$$

, where  $\#cache$  is the number of caches per request. The size of a reply is

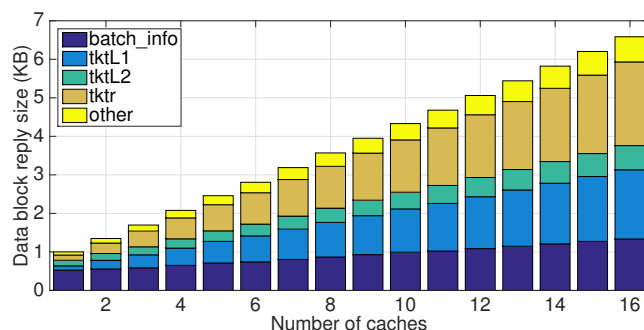
$$size_{reply} = \#cache \times 390 + \lceil \log_2(batch\_count) \rceil \times 33 + 375 \quad (4.2)$$

Throughout the composition, batch count does not have a major impact, because merkle tree construction reduces the batch information logarithmically.

Figure 4.10 shows the average batch count, while Figure 4.11 and Figure 4.12



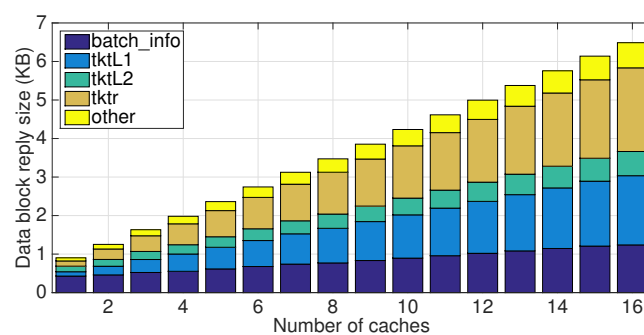
(a) One thread



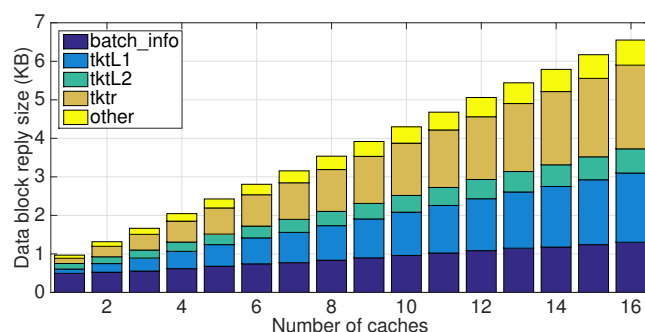
(b) Four threads

Figure 4.11: Average data block reply size (warm cache).

show the data block reply size, given that the number of caches per reply ranges from 1 to 16, and includes both warm and cold caches. With a higher number of caches per reply, the size of batch information slightly increased, which is expected. Comparing the single thread case with the four-thread case, even though the four-thread has about 4 times higher batch counts than the single thread, the size of the batch information does not increase linearly. In the four-thread case, the size of batch information is only 5–14% more than what it is in the single thread case. Therefore, the increment in data block reply size from one thread to four-thread is insignificant.



(a) One thread



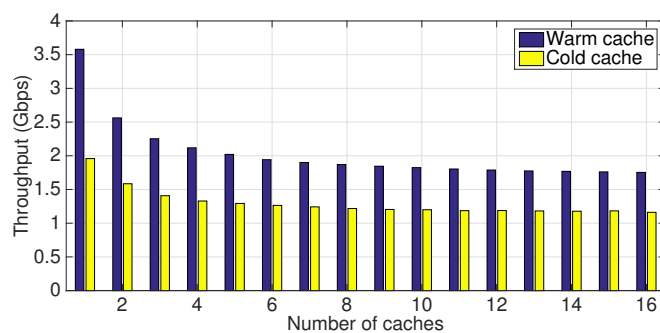
(b) Four threads

Figure 4.12: Average data block reply size (cold cache).

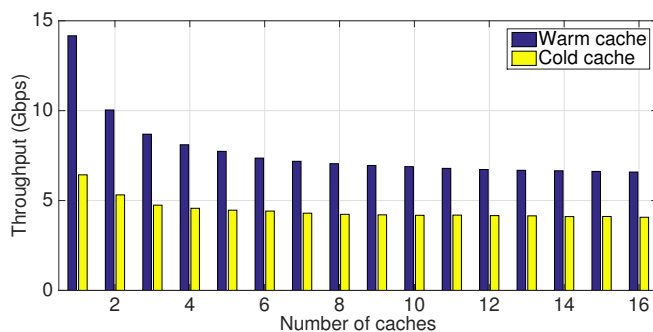
## Throughput

The throughput is the amount of data that a content provider produces in a second. Assuming it is unbounded by the bandwidth, we show the content provider throughput in Figure 4.13 based on the content provider service rate and data block reply size given in Section 4.4.2. We can see that the content provider throughput is always no lower than 1.7 Gbps in a single thread.

Compared to Netflix, throughput from a single CacheCash content provider always exceeds that of the top-rated Netflix server. However, this throughput does not represent the amount of real content generated. Instead, these results imply the bandwidth requirement for a content provider when fully loaded.



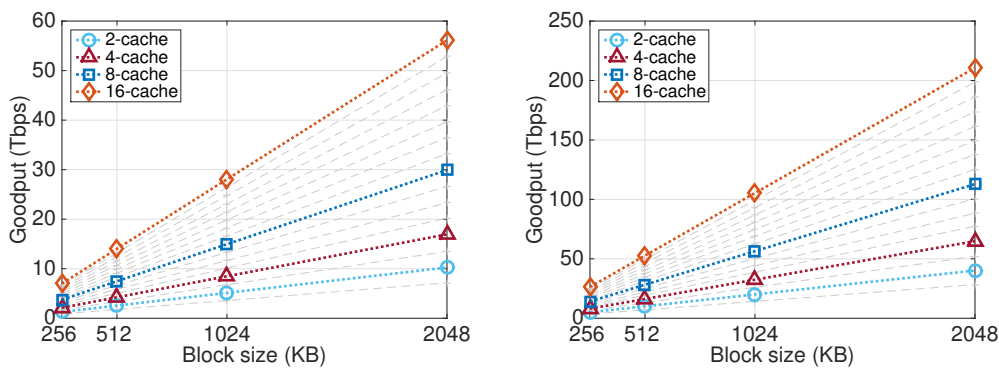
(a) One thread



(b) Four threads

Figure 4.13: Content provider throughput.

### Goodput



(a) One thread

(b) Four threads

Figure 4.14: Content provider goodput (warm cache).

The goodput is the total amount of data assigned by a content provider to all caches to serve clients in one second. In other words, it represents the amount of



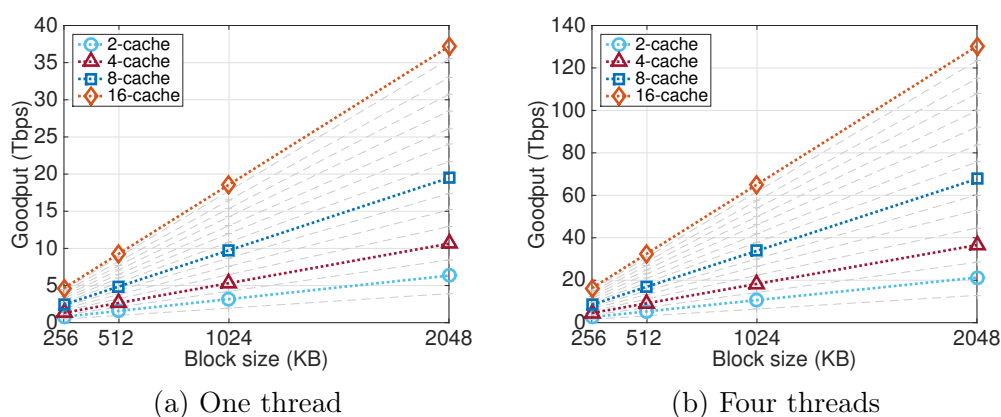


Figure 4.15: Content provider goodput (cold cache).

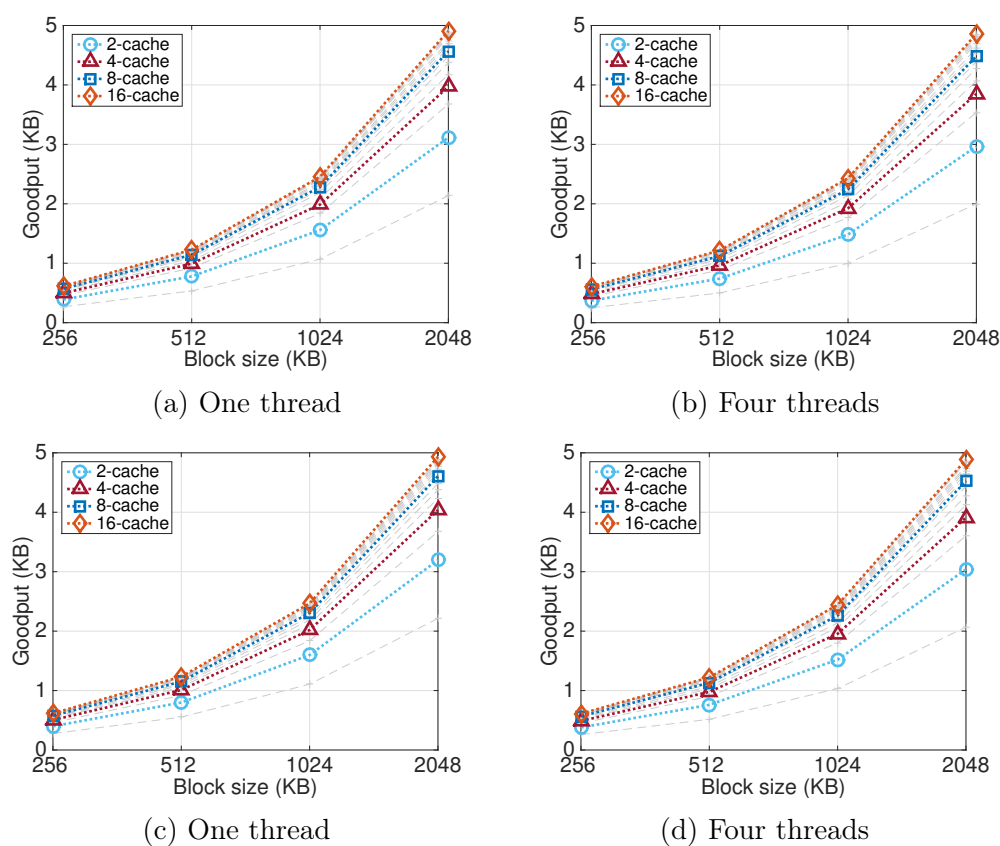


Figure 4.16: Goodput per content provider per byte.

actual content that a content provider produces per second. More specifically,

$$\text{Goodput} = \text{num\_reply\_per\_sec} \times \text{num\_cache\_per\_reply} \times \text{block\_size} \quad (4.3)$$

Equation 4.3 indicates goodput is proportional to block size, which is shown in Figures 4.14 and 4.15. With four caches per reply and a 1,024 KB block size, the content provider can produce more than 8 Tbps goodput in a single thread, and 30 Tbps with four threads. Compared to Netflix's 4.88 Tbps entire network peak traffic volume, CacheCash can dexterously achieve such a rate with a considerably lighter system load.

Figure 4.16 shows how much real content it creates for each byte the content provider generates. For example, a four-cache data block reply size is 2 KB. With a 1,024 KB block size, a client can download 4,096 KB content, which is 2,048 times more than the reply size. In this manner, whenever a client fails to retrieve content from caches, she can simply send a new data block request to the content provider. Therefore, we claim that the content provider is remarkably efficient in term of communication overhead.

### 4.4.3 Cache

The cache plays an essential role in the CacheCash system. Unlike the content provider, whose duty is to generate data block replies as auxiliary information to help clients retrieve content, the cache directly serves the client with encrypted content. In return, the cache gets paid by the content provider. The CacheCash system encourages end users to contribute their spare computational power and bandwidth as caches.

When a cache receives a block request from a client, she doubly encrypts the data block using the AES-128-CTR mode with inner and outer keys. These keys are shared with the content provider. Then she sends the encrypted data back to

the client. After that, she gives the outer and inner keys to the client in exchange for payment 1 and 2.

We are interested in the following questions:

- How fast can a cache generate encrypted content?
- How many clients can a cache serve in parallel?
- How many caches does CacheCash need to support a saturated content provider?

In our experiment, we set up a cache and tested its strength to do AES-128-CTR encryption. We ran it twice to benchmark the double encryption rate. The results show that, with a single thread, a cache can produce approximately 700 Mbps encrypted data, while with four-threads, it can push the rate up to 2,800 Mbps. This is the same rate that a home user can produce with a mid-end desktop. Compared to Netflix ISP/IXP servers' current peak rates, CacheCash cache can be 2.7 to 7 times faster.

To evaluate cache performance with practical bandwidth allocation, we check the major US ISP bandwidth for home users. Broadband cable Internet services [79], such as Optimum and Time Warner, provide 5 to 35 Mbps uploading bandwidth. Fiber optic Internet services [80], such as Verizon Fios and Google Fiber support 50 to 1,000 Mbps uploading bandwidth. According to Table 4.1 and Table 4.2, cable users can support up to 10 concurrent users, albeit uploading bandwidth can create a bottleneck. Fiber optic users can take advantage of higher bandwidth and can accommodate about 275 users in parallel. In this way, a cache set up by a fiber optic user can essentially have the same performance as a top ISP server for Netflix.

In order to output huge network traffic volume, like that of Netflix, CacheCash would need about 146,000 fully-loaded broadband caches, or 5,100 fiber optic caches to provide 4.88 Tbps of throughput. However, home users could be incentivized to join CacheCash as caches in order to make money. And to set up a cache, a user needs only to run a simple piece of software. As peers keep merging into the network, there is no reason why the CacheCash community can not grow in a similar manner to that of a typical BitTorrent tracker site. Therefore, we are confident that, after a cold startup, CacheCash will gradually grow to the size of a typical BitTorrent site, such as the Pirate Bay (TPB), which has attracted more than 5 million active users since 2007 [81]. If CacheCash has TPB's scale, the throughput of CacheCash would exceed 3,500% of Netflix's peak rate.

Under the above assumption, that the CacheCash network of caches would number about 5 million, we wanted to know if it could store the entire 3.14 PB Netflix database, and how many clients it could serve. The naïve method is to distribute video episodes to each cache uniformly, regardless of their popularity. This would mean every cache would store and run a 1 GB database, with 16 copies of every video episode. As such, for any given episode, CacheCash, running in the warm cache case, could support about 1,700 users watching concurrently. A better database deployment scheme could optimize storage for more efficient distribution of content by ensuring that popular content has more duplicates in caches to accommodate higher demands, while less popular items are carried in lesser quantities.

#### 4.4.4 Client

When a client wants to download content, she first sends a data block request to the content provider and receives a data block reply in return. The client then contacts multiple caches to download doubly encrypted content. In order to decrypt and retrieve the content, the client needs to play a lightweight version of the puzzle game. A range of possible starting points are given by the content provider in the data block reply. The client tries to solve the puzzle game from these starting points one by one. If from one starting point she gets a puzzle result for which the hash value matches one given by the content provider, this result is the key to unlock payment ticket 2.

The client performance metric is related to how fast a client is able to retrieve the requested data blocks after receiving the data block reply. It is a measure driven by the interaction between the client and caches, though we are concerned only with the client's part in this process. Solving the puzzle game and decrypting the data blocks are considered the primary operations that may incur delays on the client's side. Hence, we measure the speed of these two processes as representative of this metric.

	256 KB	512 KB	1024 KB	2048 KB
#Cache = 1	0.57	1.14	2.30	4.59
#Cache = 2	1.03	2.07	4.14	8.25
#Cache = 3	1.50	3.01	6.07	12.17
#Cache = 4	1.95	3.90	7.82	15.63
#Cache = 5	2.42	4.91	9.76	19.70
#Cache = 6	2.91	5.82	11.71	23.44
#Cache = 7	3.36	6.14	13.59	27.15
#Cache = 8	3.80	7.65	15.36	30.81

Table 4.3: Worst case puzzles solve time in seconds

Table 4.3 shows the worst case time, run using all possible starting points, for

a client to solve one puzzle game. We see the time used to solve the puzzle game increases proportionally to the block size and number of caches per request as well. Take the 1,024 KB block size and four-cache case, for example. The average time to solve a puzzle game is 3.96 seconds, which means a client needs less than 1 second to solve the puzzle to retrieve 1 MB of encrypted content.

The client also needs to double decrypt the encrypted content. We conducted an experiment on the smartphone and a single thread outputs a rate of 126 Mbps, which is more than adequate for high definition streaming.

We also needed to consider CacheCash's impact on power consumption, a critical issue for smartphones. For this purpose, we ran microbenchmarks on the smartphone to measure to what extent puzzle solving and double decryption drain battery life. In order to meet the bandwidth requirement for average US Netflix users, a client would need to spend 5.54 mW on decryption and 86.29 mW on puzzle solving. The Netflix App consumed about 592 mW on our smartphone. Consequently, only 13.24% of energy overhead must be added to watch online streaming. The extra computation is much less energy intensive than the display and network costs for the video.

## 4.5 Summary

We have presented CacheCash, a new cryptocurrency system that provides a distributed content delivery service on top of a currency exchange medium. CacheCash borrows the layered distinction of content providers, caches, and clients from traditional CDNs, but combines these roles and relationships with the flexibility of P2P networks. This enables content providers to create dynamic CDNs at a low deployment cost. Most importantly, it utilizes both technical and eco-

conomic strategies to ensure security. As a storage/bandwidth market, monetary incentives in CacheCash motivate users to work in a collaborative and honest way to achieve high revenues. The exhibited benchmark results provide insights about the efficiency level of CacheCash. The system has outstanding content generation speed. Its communication overhead is extremely low, thus making data service remarkably efficient. Beyond that, the CacheCash system can scale to accommodate huge network traffic demands and handle the load with few content providers.

# Chapter 5

## Conclusion

### 5.1 Summary of Contributions

In this dissertation, we studied three key factors in the operation of collaborative distributed systems: performance, security, and the necessary incentives to keep participants honest and reliable. We revealed the limitations of current PIR schemes and CDN networks, and proposed efficient solutions, based on their unique requirements, to exploit the full potential of these practical distributed systems.

For the Bitcoin network, we conducted a thorough study on its public ledger: the blockchain. In this measurement study, we characterized the behavior of both solo and pool miners. Based on the BTC to USD exchange rate and miners' transaction frequencies, we observed that many early miners who worked individually with less-than-powerful devices, lost control of the BTCs they earned. With the growth of the Bitcoin network, miners became more likely to join pools and collaboratively mine BTCs for a steady payout. In addition, they were more eager to cash out their newly mined BTCs immediately to take a profit. We also estimated



the upper bound of computational power for the Bitcoin network. According to the economic model we proposed, the computational race among miners will continue and miners can persist in making a profit, until the overall network hashrate reaches the sustainable bound.

For private information retrieval, we first revealed the limitations of current works as described in the literature. Then we extended the classic multi-server PIR protocol to make it more efficient by reusing randomly mixed data blocks across multiple requests. The key to achieving more efficient performance is by using faster XOR operations, which are shown to have similar goodput to FTP on realistic datasets and deployment environments as more computationally-expensive approaches. We also developed a finite-field based PIR scheme to further reduce the communication overhead of our proposed binary multi-block PIR scheme.

Lastly, we proposed CacheCash, a cryptocurrency-based CDN network that allows content providers to offload traffic using low cost cache deployment. Caches are incentivized to serve clients honestly if they want to get paid. Both the decentralized consensus protocol and probabilistic micropayment scheme ensure fair payments. We developed a reference implementation for CacheCash and evaluated its performance in terms of computational and bandwidth requirements. The benchmark results demonstrate that our system scales well with huge network traffic demands, and can quickly and efficiently deliver content to clients.

## 5.2 Future Work

Throughout this dissertation, we have focused on the analysis and design of various collaborative distributed systems. While a lot of theoretical work was proposed and extensive numerical simulations have been carried out for evaluation,

these protocols, and algorithmic designs still await verification through real-world implementations and evaluation of those results. For instance, the growth of the Bitcoin network has experienced both highs and lows in terms of BTC exchange prices. We conducted the measurement study during its growth period. Studying current mining and transaction patterns, and using this data to validate our economic forecast model is encouraged. In this way, we can further understand miners behavior as the Bitcoin network grows.

A network application may have different components that perform inconsistently within different operating systems. Therefore the theoretical analysis in terms of computational complexity may not truly reflect the performance of a practical system. In other words, an algorithmic design may be sound but when building a practical system the performance might not be desirable. Thus, the implementation and performance evaluation of a practical system becomes undeniably important. With CacheCash, the benchmark has shown its strength in both content generation and its low overhead. Building an implementation prototype and evaluating it on a large scale will help us understand and further optimize the system.

# Bibliography

- [1] “BitTorrent,” <http://www.bittorrent.com>.
- [2] “Gnutella,” <https://en.wikipedia.org/wiki/Gnutella>.
- [3] B. Chor, E. Kushilevitz, O. Goldreich, and M. Sudan, “Private information retrieval,” *Journal of the ACM (JACM)*, vol. 45, no. 6, pp. 965–981, 1998.
- [4] W. Gasarch, “A survey on private information retrieval,” in *Bulletin of the EATCS*. Citeseer, 2004.
- [5] U. Feige, A. Fiat, and A. Shamir, “Zero-knowledge proofs of identity,” *Journal of Cryptology*, vol. 1, no. 2, pp. 77–94, 1988.
- [6] S. Nakamoto, “Bitcoin: A peer-to-peer electronic cash system,” 2008, <http://www.bitcoin.org/bitcoin.pdf>.
- [7] G. Wood, “Ethereum: A secure decentralised generalised transaction ledger,” *Ethereum Project Yellow Paper*, 2014.
- [8] “Ripple,” <https://ripple.com>.
- [9] “Litecoin,” <https://litecoin.info>.

- [10] S. Goldfeder, R. Gennaro, H. Kalodner, J. Bonneau, J. A. Kroll, E. W. Felten, and A. Narayanan, “Securing bitcoin wallets via a new dsa/ecdsa threshold signature scheme,” 2015.
- [11] J. Becker, D. Breuker, T. Heide, J. Holler, H. P. Rauer, and R. Böhme, “Can we afford integrity by proof-of-work? scenarios inspired by the bitcoin currency,” in *The Economics of Information Security and Privacy*. Springer, 2013, pp. 135–156.
- [12] K. Baqer, D. Y. Huang, D. McCoy, and N. Weaver, “Stressing out: Bitcoin “stress testing”,” in *Workshop on Bitcoin and Blockchain Research*, 2016.
- [13] I. Eyal and E. G. Sirer, “Majority is not enough: Bitcoin mining is vulnerable,” *arXiv preprint arXiv:1311.0243*, 2013.
- [14] J. A. Kroll, I. C. Davey, and E. W. Felten, “The economics of bitcoin mining, or bitcoin in the presence of adversaries,” in *Proceedings of WEIS*, vol. 2013, 2013.
- [15] D. Y. Huang, H. Dharmdasani, S. Meiklejohn, V. Dave, C. Grier, D. McCoy, S. Savage, N. Weaver, A. C. Snoeren, and K. Levchenko, “Botcoin: Monetizing stolen cycles,” in *Proceedings of the Network and Distributed System Security Symposium (NDSS)*, 2014.
- [16] F. Reid and M. Harrigan, *An Analysis of Anonymity in the Bitcoin System*. Springer, 2013.
- [17] S. Meiklejohn, M. Pomarole, G. Jordan, K. Levchenko, D. McCoy, G. M. Voelker, and S. Savage, “A fistful of bitcoins: Characterizing payments among

- men with no names,” in *Proceedings of the 2013 conference on Internet measurement conference*. ACM, 2013, pp. 127–140.
- [18] D. Ron and A. Shamir, “Quantitative analysis of the full bitcoin transaction graph,” in *Financial Cryptography and Data Security*. Springer, 2013, pp. 6–24.
- [19] E. Androulaki, G. O. Karame, M. Roeschlin, T. Scherer, and S. Capkun, “Evaluating user privacy in bitcoin,” in *Financial Cryptography and Data Security*. Springer, 2013, pp. 34–51.
- [20] F2Pool.com, <https://www.f2pool.com>, accessed September 10, 2014.
- [21] D. Asonov, “Private Information Retrieval - An Overview And Current Trends,” in *GI Jahrestagung (2)*, 2001, pp. 889–894.
- [22] E. Y. Yang, J. Xu, and K. H. Bennett, “A fault-tolerant approach to secure information retrieval,” in *Reliable Distributed Systems, 2002. Proceedings. 21st IEEE Symposium on*. IEEE, 2002, pp. 12–21.
- [23] J. Cappos, “Avoiding theoretical optimality to efficiently and privately retrieve security updates,” in *Financial Cryptography and Data Security 2013 (FC 2013)*, 2013.
- [24] A. Beimel, Y. Ishai, E. Kushilevitz, and J.-F. Raymond, “Breaking the  $O(n^{1/(2k-1)})$  barrier for information-theoretic private information retrieval,” in *Foundations of Computer Science, 2002. Proceedings. The 43rd Annual IEEE Symposium on*. IEEE, 2002, pp. 261–270.
- [25] C. Cachin, S. Micali, and M. Stadler, “Computationally private information retrieval with polylogarithmic communication,” in *Proceedings of the 17th in-*

- ternational conference on Theory and application of cryptographic techniques*, ser. EUROCRYPT'99. Berlin, Heidelberg: Springer-Verlag, 1999, pp. 402–414.
- [26] D. Asonov and J.-C. Freytag, “Almost optimal private information retrieval,” in *Privacy Enhancing Technologies*. Springer, 2003, pp. 209–223.
- [27] A. Ambainis, “Upper bound on the communication complexity of private information retrieval,” in *Automata, Languages and Programming*. Springer, 1997, pp. 401–407.
- [28] R. Sion and B. Carbunar, “On the Computational Practicality of Private Information Retrieval,” in *In Proceedings of the Network and Distributed Systems Security Symposium (NDSS)*., 2007.
- [29] R. Henry, Y. Huang, and I. Goldberg, “One (Block) Size Fits All: PIR and SPIR Over Arbitrary-Length Records via Multi-block PIR Queries,” in *In Proceedings of the Network and Distributed Systems Security Symposium (NDSS)*., 2013.
- [30] I. Goldberg, “Improving the robustness of private information retrieval,” in *Proceedings of the 2007 IEEE Symposium on Security and Privacy*, ser. SP '07. Washington, DC, USA: IEEE Computer Society, 2007, pp. 131–148.
- [31] A. S. Tanenbaum and M. Van Steen, *Distributed systems*. Prentice-Hall, 2007.
- [32] D. Demmler, A. Herzberg, and T. Schneider, “Raid-pir: Practical multi-server pir,” in *Proceedings of the 6th edition of the ACM Workshop on Cloud Computing Security*. ACM, 2014, pp. 45–56.

- [33] “Netflix,” <http://www.netflix.com>.
- [34] “Amazon video,” <https://www.amazon.com/gp/video/getstarted>.
- [35] A. Vakali and G. Pallis, “Content delivery networks: Status and trends,” *IEEE Internet Computing*, vol. 7, no. 6, pp. 68–74, 2003.
- [36] G. Pallis and A. Vakali, “Insight and perspectives for content delivery networks,” *Communications of the ACM*, vol. 49, no. 1, pp. 101–106, 2006.
- [37] T. Bektas, O. Oguz, and I. Ouveysi, “Designing cost-effective content distribution networks,” *Computers & Operations Research*, vol. 34, no. 8, pp. 2436–2449, 2007.
- [38] Akamai, <http://www.akamai.com>.
- [39] M. Sirivianos, X. Yang, and S. Jarecki, “Robust and efficient incentives for cooperative content distribution,” *IEEE/ACM Transactions on Networking (TON)*, vol. 17, no. 6, pp. 1766–1779, 2009.
- [40] P. Wendell and M. J. Freedman, “Going viral: flash crowds in an open CDN,” in *Proceedings of the 2011 ACM SIGCOMM conference on Internet measurement conference*. ACM, 2011, pp. 549–558.
- [41] D. F. Kassa and K. Nahrstedt, “Hinent: Quick content distribution with priorities and high incentives,” in *Consumer Communications and Networking Conference (CCNC), 2013 IEEE*. IEEE, 2013, pp. 22–30.
- [42] B. Cohen, “Incentives build robustness in BitTorrent,” in *Workshop on Economics of Peer-to-Peer systems*, vol. 6, 2003, pp. 68–72.

- [43] T. Mori, N. Kamiyama, S. Harada, H. Hasegawa, and R. Kawahara, “Improving deployability of peer-assisted CDN platform with incentive,” in *Global Telecommunications Conference, 2009. GLOBECOM 2009. IEEE*. IEEE, 2009, pp. 1–7.
- [44] S. Wilkinson, T. Boshevski, J. Brandoff, and V. Buterin, “Storj a peer-to-peer cloud storage network,” 2014.
- [45] “Syscoin,” <http://www.syscoin.org>.
- [46] “Filecoin,” <http://www.filecoin.io>.
- [47] bitcoin.org, “Frequently asked questions,” <https://bitcoin.org/en/faq>, accessed September 10, 2014.
- [48] Coinbase.com, <https://www.coinbase.com>, accessed September 10, 2014.
- [49] Bitstamp.com, <https://www.bitstamp.net>, accessed September 10, 2014.
- [50] Bitcoinwiki, “Bitcoin difficulty,” <https://en.bitcoin.it/wiki/Difficulty>, accessed September 10, 2014.
- [51] bitcoin.cz, “World’s first mining pool celebrates 3rd year with 0% fee,” <https://mining.bitcoin.cz/news/2013-12-16-pool-celebrates-3rd-anniversary>, accessed September 10, 2014.
- [52] blockchain.info, “An estimation of hashrate distribution amongst the largest mining pools,” <https://blockchain.info/pools/>, accessed September 10, 2014.
- [53] statista.com, “Felectricity prices in selected countries in 2013 (in u.s. dollar cents per kilowatt hour),” <http://www.statista.com/statistics/263492/electricity-prices-in-selected-countries/>, accessed September 10, 2014.



- [54] F. G. Olumofin and I. Goldberg, “Revisiting the computational practicality of private information retrieval,” in *Financial Cryptography*, 2011, pp. 158–172.
- [55] K. S. Narayanam, “A Novel Scheme for Single Database Symmetric Private Information Retrieval,” in *Financial Cryptography*, 2006.
- [56] M. Layouni, “Accredited symmetrically private information retrieval,” in *IWSEC 2007. LNCS*. Springer, 2007, pp. 262–277.
- [57] C. Devet, I. Goldberg, and N. Heninger, “Optimally robust private information retrieval,” in *Proceedings of the 21st USENIX conference on Security symposium*, ser. Security’12. Berkeley, CA, USA: USENIX Association, 2012, pp. 13–13.
- [58] “AWS Instance Types,” <http://aws.amazon.com/ec2/#instance>.
- [59] C. Labovitz, S. Iekel-Johnson, D. McPherson, J. Oberheide, and F. Jahanian, “Internet inter-domain traffic,” *ACM SIGCOMM Computer Communication Review*, vol. 41, no. 4, pp. 75–86, 2011.
- [60] V. Vishnumurthy, S. Chandrakumar, and E. G. Sirer, “Karma: A secure economic framework for peer-to-peer resource sharing,” in *Workshop on Economics of Peer-to-Peer Systems*, vol. 35, 2003.
- [61] S. K. Nair, E. Zentveld, B. Crispo, and A. S. Tanenbaum, “Floodgate: A micropayment incentivized p2p content delivery network,” in *Computer Communications and Networks, 2008. ICCCN’08. Proceedings of 17th International Conference on*. IEEE, 2008, pp. 1–7.
- [62] “Monero,” <https://getmonero.org/home>.

- [63] I. Miers, C. Garman, M. Green, and A. D. Rubin, “Zerocoin: Anonymous distributed e-cash from bitcoin,” in *Security and Privacy (SP), 2013 IEEE Symposium on*. IEEE, 2013, pp. 397–411.
- [64] S. King, “Primecoin: Cryptocurrency with prime number proof-of-work,” 2013.
- [65] A. Coventry, “Nooshare: A decentralized ledger of shared computational resources,” Tech. Rep., 2012, [http://web.mit.edu/alex\\_c/www/nooshare.pdf](http://web.mit.edu/alex_c/www/nooshare.pdf).
- [66] “Namecoin,” <http://www.namecoin.info>.
- [67] N. Szabo, “Formalizing and securing relationships on public networks,” *First Monday*, vol. 2, no. 9, 1997.
- [68] A. Juels and B. S. Kaliski Jr, “Pors: Proofs of retrievability for large files,” in *Proceedings of the 14th ACM conference on Computer and communications security*. Acm, 2007, pp. 584–597.
- [69] A. Miller, A. Juels, E. Shi, B. Parno, and J. Katz, “Permacoin: Repurposing bitcoin work for data preservation,” in *Security and Privacy (SP), 2014 IEEE Symposium on*. IEEE, 2014, pp. 475–490.
- [70] B. Sengupta, S. Bag, S. Ruj, and K. Sakurai, “Retricoin: Bitcoin based on compact proofs of retrievability,” in *Proceedings of the 17th International Conference on Distributed Computing and Networking*. ACM, 2016, p. 14.
- [71] D. Vorick and L. Champine, “Sia: Simple decentralized storage,” 2014.
- [72] G. Paul, F. Hutchison, and J. Irvine, “Security of the maidsafe vault network,” in *Wireless World Research Forum Meeting 32 (WWRF32)*, 2014.

- [73] S. Micali and R. L. Rivest, “Micropayments revisited,” in *Cryptographers’ Track at the RSA Conference*. Springer, 2002, pp. 149–163.
- [74] R. Pass, “Micropayments for decentralized currencies,” in *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*. ACM, 2015, pp. 207–218.
- [75] “About Netflix,” <https://ispspeedindex.netflix.com>.
- [76] T. Böttger, F. Cuadrado, G. Tyson, I. Castro, and S. Uhlig, “Open connect everywhere: A glimpse at the internet ecosystem through the lens of the Netflix CDN,” *arXiv preprint arXiv:1606.05519*, 2016.
- [77] “Netflix demands,” <http://fortune.com/2015/10/08/netflix-bandwidth>.
- [78] “House of cards statistics,” <http://www.cnn.com/2016/03/04/how-many-people-will-be-watching-house-of-cards.html>.
- [79] “Broadband speed,” <https://www.optimum.net/pages/speed.html>.
- [80] “Fiber optic speed,” <https://fiber.google.com/cities/kansascity/plans/>.
- [81] “The pirate bay,” [https://en.wikipedia.org/wiki/The\\_Pirate\\_Bay](https://en.wikipedia.org/wiki/The_Pirate_Bay).