Cryptocurrencies

> We have proposed a system for electronic transactions without relying on trust.
>
> *Satoshi Nakamoto, "Bitcoin: A Peer-to-Peer Electronic Cash System"*

Since the launch of the Bitcoin network in 2009, the market cap of cryptocurrencies has risen into the hundreds of billions of dollars. The Bitcoin protocol has spawned a wave of innovation and experimentation in the domains of distributed systems, cryptography, and economics.

Some claim that the ideas behind blockchains will be integral to the future of money, governments, and the Internet. Whether or not those claims turn out to be true, it's clear that cryptocurrencies are an important area of technological innovation. In this course we'll discuss them, examine in detail how they work, and build one for ourselves.

This course is designed to be an introduction to the theory and mechanics behind cryptocurrencies. It is intended for professional software engineers, and assumes a basic knowledge of programming and computer science. Beyond merely blockchains, the study of cryptocurrencies is inherently multidisciplinary, touching concepts from computer science, security, economics, and history. Though our primary focus will be on computer science, we'll also delve into each of those disciplines where appropriate.

There will be required reading and homework assignments in this class. Note that the readings may be a mixture of videos and written material where necessary.

Bitcoin and Cryptocurrency Technologies: A Comprehensive Introduction by Arvind Narayanan, Joseph Bonneau, Edward Felten, Andrew Miller, and Steven Goldfeder, is the recommended textbook for the class. They provide free access to a prepublication draft here, but we recommend purchasing the final textbook if possible. Note that there are also video lectures (part of a Coursera course that you can watch instead if you prefer—they generally cover the same material).

PROJECTS AND EXERCISES

Because this course is designed for professional software engineers, we will be exploring the material by writing software. We will be building out a Bitcoin-esque cryptocurrency and all of its requisite components (a blockchain, cryptography, P2P networking, and proof-of-work). We'll also build some Ethereum smart contracts using Solidity and analyze them for potential attacks.

ASSUMED KNOWLEDGE

For this course, we assume that students have some familiarity with algorithmic analysis ("Big O") as well as linked lists and binary search trees. Our suggested first resource for these is Bradfield's /algos lecture notes, available freely online. For more, we suggest Steven Skiena's videos and book, *The Algorithm Design Manual*.

For networking, students should understand the basic responsibilities and differences between TCP and UDP, and have a general understanding of what constitutes an IP packet, and how they are routed. For the transport layer (TCP and UDP) we suggest working through chapters 3.3 and 3.5 of *Computer Networking: A Top Down Approach* by Kurose and Ross, or unit 2 of Stanford's online Introduction to Computer Networking course. For the internet layer (IP, routing) we suggest chapter 4.4 of the Kurose and Ross book, or unit 1 of the Stanford course.

Students should have a general understanding of the mechanism and purpose of public key cryptography. The Khan Academy Cryptography course has a section "Modern cryptography" that covers this, or we suggest chapters 8.2 and 8.5 of the Kurose and Ross book, or unit 8 of the Stanford course.

Ideally, students would also have had some exposure to models of consistency in distributed systems, as well as mechanisms and degrees of fault tolerance (including Byzantine fault tolerance). For this, we suggest sections 7.1-7.3 and 8.5-8.6 of Maarten van Steen and Andrew Tanenbaum's *Distributed Systems: 3rd Edition available for free online, or chapter 5 of Martin Kleppmann's* Designing Data-Intensive Applications*.

None of these are strictly required to derive value from the classes, but some rudimentary background will help you make the most of your time. Please let us know if you'd like any further suggestions or guidance.

Classes

1 CYPHERPUNKS, DIGITAL MONEY, AND THE PRECURSORS TO BITCOIN

"The Times 03/Jan/2009 Chancellor on brink of second bailout for banks" — Bitcoin genesis block's metadata

In October 2008, a pseudonymous programmer by the name of Satoshi Nakamoto published a whitepaper to the Cypherpunks mailing list, in which he described a protocol for a decentralized digital currency. He called this protocol Bitcoin. Two years later Satoshi disappeared without a trace, but the currency he created would go on to change the world.

Bitcoin is now a global phenomenon. But where did Bitcoin come from? What is its lineage, and how did it succeed where its predecessors failed?

We'll start our exploration of Bitcoin with a history lesson. We'll learn about the cypherpunks, the previous attempts at building digital currencies, and the problems that plagued them. And we'll get a hint of what it was about Bitcoin that it was able to overcome these problems.

Required reading

- From *Bitcoin and Cryptocurrency Technologies*, read the foreword and sections 7.4 ("The Roots of Bitcoin") and 7.5 ("Governments Notice Bitcoin").

Further resources

- A Cypherpunk's Manifesto by Eric Hughes (1993)
- b-money by Wei Dai (1998)
- Bit Gold by Nick Szabo (2005)
- History of Bitcoin, Cypherpunk (Wikipedia)
- Bitcoin History: From the Cypherpunk Movement to JPMorgan Chase (video) from the UC Berkeley Cryptocurrency Decal

2 DECENTRALIZATION, P2P NETWORKS, AND GOSSIP PROTOCOLS

"Governments are good at cutting off the heads of a centrally controlled networks like Napster, but pure P2P networks like Gnutella and Tor seem to be holding their own." — Satoshi Nakamoto

The key difference between Bitcoin and previous digital currencies is Bitcoin's decentralization. Bitcoin is decentralized not just in that does not rely on a central bank, but also in its network topology.

To create a truly decentralized and censorship-resistant monetary system, you must be resilient to any node being attacked or leaving the system. By tracing the history of P2P networks and gossip protocols, we'll learn how Satoshi arrived at the network design that's now shared by almost all cryptocurrencies.

In this class, we'll look at the history and design of P2P networks, including Napster, Gnutella, and BitTorrent, and their designs inspired the Bitcoin network. We'll then implement our own version of a gossip protocol as an in-class project, to be completed as homework.

Required reading

- From *Bitcoin and Cryptocurrency Technologies*, read section 3.5 ("The Bitcoin Network")
- Read descriptions of the Napster architecture and Gnutella P2P architectures, from *Peer-to-Peer Networking and Applications*, by Xuemin S. Shen
- Read through the design of BitTorrent

Further resources

- Design of Tor (Wikipedia)
- Gossip Protocols (video) from Urbana Champaign's MOOC on Cloud Computing

3 CRYPTOGRAPHY 101: PUBLIC-KEY CRYPTOGRAPHY, DIGITAL IDENTITIES, HASHING, AND MERKLE TREES

"With e-currency based on cryptographic proof, without the need to trust a third party middleman, money can be secure and transactions effortless." — Satoshi Nakamoto

Cryptography is the basis of how cryptocurrencies work, and it is where cryptocurrencies get their name. The most important primitives are public-key cryptography (also known as asymmetric cryptography), cryptographic hashing functions, and Merkle Trees.

In this class, we'll discuss all of the aforementioned primitives. We'll learn how these primitives can be used to create digital identities and proof-of-work puzzles. As our in-class assignment we'll be creating proof-of-work solvers and constructing Merkle Trees.

Required reading

- From *Bitcoin and Cryptocurrency Technologies*, read all of Chapter 1 ("Cryptography and Cryptocurrencies")

Further Reading

- RSA (video) for learning how RSA works, the first and simplest public key crypto system
- ECDSA for elliptic curve signatures, which is the public-key crypto used in most cryptocurrencies
- Merkle proofs for proofs of existence
- Mining Bitcoin by hand by Ken Shirriff

4 BYZANTINE FAULT-TOLERANCE, NAKAMOTO CONSENSUS, AND BUILDING A BLOCKCHAIN

In this paper, we propose a solution to the double-spending problem using a peer-to-peer distributed timestamp server to generate computational proof of the chronological order of transactions. The system is secure as long as honest nodes collectively control more CPU power than any cooperating group of attacker nodes. — Bitcoin: A Peer-to-Peer Electronic Cash System, Satoshi Nakamoto

It is not enough for a digital currency to be decentralized and fault-tolerant—it must also be resilient to attackers. This elusive property is known as Byzantine fault-tolerance, and it has a long history of academic research.

In this class we'll look at Byzantine fault-tolerance and the lineage of consensus algorithms that preceded Bitcoin. Then we'll examine Nakamoto Consensus, the chief consensus algorithm and technical innovation of Bitcoin.

As our class exercise, we'll take everything we've learned and use it to build a toy cryptocurrency. You will begin the currency in class and continue it as homework, to be presented in the subsequent class.

Required reading

- From *Bitcoin and Cryptocurrency Technologies*, read all of Chapter 2 ("How Bitcoin Achieves Decentralization")
- At last: read the Bitcoin white paper! It's only 8 pages, and you should have enough background by now to understand all of it.

Further reading

- The Byzantine Generals Problem by Leslie Lamport, Robert Shostak, and Marshall Pease, which first formalized Byzantine fault tolerance (1982)
- Practical Byzantine Fault Tolerance by Barbara Liskov and Miguel Castro, the first practical BFT algorithm from which most non-PoW algorithms are derived (1999)
- The Sybil Attack by John Douceur (2002), which conceived of PoW as a security model
- Let's build a blockchain! by me—a tech talk I gave in which I live code a mini-cryptocurrency. Feel free to watch this if you're getting stuck on some implementation details, or just want to review concepts.

5 CRYPTOECONOMICS AND INCENTIVE DESIGN

"The root problem with conventional currency is all the trust that's required to make it work. The central bank must be trusted not to debase the currency, but the history of fiat currencies is full of breaches of that trust." — Satoshi Nakamoto

All security is economic security. In other words, we believe systems are secure because our threat model implies it would be extremely costly to break it. Even with cryptography, our security claims implicitly make an economic argument.

Cryptoeconomics is this idea taken to its logical conclusion. If you are trying to secure a cryptocurrency, you want to make it as expensive as possible to attack. But since you're designing the currency to begin with, you can sometimes engineer the right incentives directly into the protocol. By marrying game theory and cryptography, you can design a cryptocurrency that incentivizes its own security.

In this lecture, we'll explore game theory and how you can use it to model different actors in the cryptocurrency ecosystem. If you assume rational actors, then with the right mechanism design, you can create a cryptoeconomic equilibrium.

Required reading

- From *Bitcoin and Cryptocurrencies*, read sections 5.4 ("Mining Pools") and 5.5 ("Mining Incentives and Strategies")
- The Current State of Cryptoeconomics (video) by Vlad Zamfir

Further reading

- Game theory and Nash Equilibrium from Khan Academy's Microeconomics lesson
- Making sense of cryptoeconomics by Josh Stark
- Mechanism design (Wikipedia)
- A Crash Course in Mechanism Design for Cryptoeconomic Applications by Alex Evans
- Cryptoeconomic Protocols in the Context of Wider Society (video) by Vitalik Buterin

6 SMART CONTRACTS, ETHEREUM, AND DECENTRALIZED COMPUTATION

Ethereum is open-ended by design, and we believe that it is extremely well-suited to serving as a foundational layer for a very large number of both financial and non-financial protocols in the years to come. — Vitalik Buterin, Ethereum's white paper

We've learned how blockchains and proof-of-work can achieve distributed, Byzantine fault-tolerant consensus within a peer-to-peer network. But a payments network is just one application you can run atop such a blockchain. In 2013, Vitalik Buterin, the creator of Ethereum asked: what if you used a blockchain to implement a decentralized computer?

In Ethereum, you pay miners to execute your programs on this distributed virtual machine. This means you can perform arbitrary computations, using a Turing-complete programming language (unlike Bitcoin script). This includes payments-related applications, so Ethereum enables a superset of Bitcoin's functionality, but has also birthed a renaissance of innovation in the form of "smart contracts."

In this class, we'll look to understand how Ethereum works, and the different applications it enables. We'll also look at how you can interact with the Ethereum blockchain.

Required reading

- From *Bitcoin and Cryptocurrencies*, read section 3.2 ("Bitcoin Scripts") through to the end of 3.4 ("Bitcoin Blocks"), as well as section 10.7 ("Ethereum and Smart Contracts")
- Ethereum: A Next-Generation Smart Contract and Decentralized Application Platform (white paper)

Further reading

- The Ethereum yellow paper by Gavin Wood (technical protocol specification)
- Formalizing and Securing Relationships on Public Networks by Nick Szabo (1996), in which he coins the term "smart contract" and theorizes on their possibilities
- Decentralized autonomous organizations (Wikipedia)

7 SMART CONTRACT DEVELOPMENT

"Of course, for the computer scientist code is law. And if code is law, then obviously the question we should ask is: Who are the lawmakers?" — Lawrence Lessig, *Code v2*

The foundation of Ethereum is smart contracts. They're what enable Dapps, DAOs, and much of the innovation in blockchains.

But smart contract programming is hard. They can mask themselves as normal programs, but smart contracts entail fundamentally different execution environments than normal programs run on single machines.

We're going to explore the challenges of writing Ethereum smart contracts in Solidity. We'll develop what seems like an elementary smart contract, and successively build on it using blockchain programming primitives.

Required reading

- Work through CryptoZombies programming exercises until at least the end of Chapter 2.
- Play around a bit in Remix and get familiar with it.

Further reading

- Solidity docs
- If you want to play around some more, here's a nice tutorial for building an ERC-20 token.
- Some of Solidity's design warts (HackerNews)
- Viper, an experimental new and safer smart contract programming language, designed by Vitalik Buterin

8 SMART CONTRACT SECURITY AND SCALING CRYPTOCURRENCIES

"There will be further bugs, and we will learn further lessons; there will not be a single magic technology that solves everything." — Vitalik Buterin, Thinking About Smart Contract Security

In our final class, we'll look at many of the multi-million dollar attacks that have been carried out against smart contracts. This includes the infamous DAO hack, as well as the two Parity hacks. It turns out, all of these hacks are readily understandable, and we'll be able to see exactly where in the smart contract code they occurred.

From there, we'll turn toward the future and how cryptocurrencies can scale. We'll touch on optimizations like sharding, proof of stake, and sidechains.

Required reading

- Understanding The DAO Attack by David Siegal (entertaining read!)
- From *Bitcoin and Cryptocurrencies*, read section 3.6 ("Limitations and Improvements")
- On Scaling Decentralized Blockchains by Ittay Eyal, Ari Juels, Andrew Miller, Emin Gun Sirer, et al.

Further reading

- Ethereum Smart Contract Best Practices by ConsenSys
- Analysis of the DAO exploit by Phil Daian
- Analysis of the Parity Wallet Hack by me! (also entertaining)
- Analysis of Parity Wallet Hack 2 by Matt Condon
- The Ethernaut, a wargame to practice smart contract hacks
- The Bitcoin Lightning Network: Scalable Off-Chain Instant Payments by Joseph Poon and Thaddeus Dryja
- Sharding FAQ from Ethereum's wiki
- Proof of Stake FAQ from Ethereum's Wiki
- Plasma: Scalable Autonomous Smart Contracts by Joseph Poon and Vitalik Buterin